Representing, matching, and generalising structural descriptions of complex physical objects.

by David Brian Andreae

A thesis

submitted to the Victoria University of Wellington
in fulfillment of the requirements for
the degree of Doctor of Philosophy

Victoria University of Wellington
1994

Abstract

This thesis addresses the problem of representing, matching, and generalising descriptions of complex structured physical objects, in the absence of functional and domain-specific knowledge. A system called GRAM is described, which includes a representation scheme, an instance-constructor, a matcher, and a generaliser. These components incorporate and extend ideas from a number of other structured-object learning systems, as well as introducing several new ideas.

A central contribution of this thesis is to show that descriptions of complex physical objects can be matched and generalised effectively and efficiently by exploiting their structure. GRAM does this by a number of means, such as by representing objects at multiple levels of detail; using 'neighbour relationships' to allow a more flexible traversal of object graphs during matching; explicitly distinguishing between substructure and context to allow partial matching and a simple form of disjunction; and using an explicit representation of groups to describe several similar objects as a single descriptive entity.

A second contribution is to show that complex objects can be matched without having to enforce consistency between object correspondences. This is possible partly because of the richness of physical objects, and partly because GRAM represents concepts as simple entities defined by relationships with other concepts, rather than as a complete set of subcomponents defined locally within the concept description itself. This scheme leads to greater simplicity, efficiency, and robustness.

Acknowledgements

I would like to thank my supervisor, Peter Andreae, for giving such good advice and support throughout this thesis, even when I ignored his suggestions. His feedback during the writeup stage was invaluable.

I am very grateful to my parents and my sister, Gillian, for their love and support throughout this project. Thankyou to my friends for their friendship throughout this sometimes-difficult time, and especially to my friends in the Free Daist community who have helped me to complete this thesis.

Thanks also to Pat Langley for his motivating comments when he visited this department.

I was financially supported by a University Grants Committee scholarship, an IBM scholarship, a William Georgetti scholarship, and various teaching positions in the Department of Computer Science at Victoria University. I am very grateful for this support which made my living situation considerably easier.

Contents

Chapter 1

Introduction

This thesis addresses the problem of building a system that can represent, construct, match, and generalise descriptions of complex structured physical objects, without using functional or domain-specific knowledge. The thesis describes an implemented system, called GRAM, which operates in a domain of static two-dimensional structured objects, such as those shown in Figure 1.1.

The original inspiration for building GRAM was based on a long-term vision of building an "instructable autonomous robot" that could learn to perform tasks in the physical world. Tasks such as vacuuming a room, finding and retrieving objects, or drying and putting away dishes, require the robot to have effective and efficient *classification* mechanisms for recognising the objects encountered in the world. The robot must also be able to *learn* descriptions of object categories (or 'concepts') so that it can adapt to new or changing environments. The robot should be able to learn either on its own in response to encountering a new instance of a known category, in response to explicit instruction from a teacher, or in response to demands from other components of the robot system.

A basic strategy for classifying an object consists of several steps. First, a description of the object is constructed from image data. *Indexing* mechanisms are then used to access a selection of concepts in a potentially vast memory. A *matcher* then compares the object (or *instance*) with each concept.

The central component of a learning system is a *generaliser*, which generalises an existing concept to cover an observed instance. The learning system must also be able to create and add new concepts to concept memory, and to reorganise concept memory if necessary.

GRAM provides four components to support the classification and learning tasks: a representation scheme, an instance constructor, a matcher, and a generaliser. This thesis discusses the issues in designing each of these components, and presents the new ideas and mechanisms that have been developed. Mechanisms for indexing and memory organisation, and the way in which all of the mechanisms are to be integrated into a complete classification and learning system, are the subject of future research.

There has not been a great deal of other research that encompasses all four of the above components (representation, instance construction, matching, and generalisation) in the domain

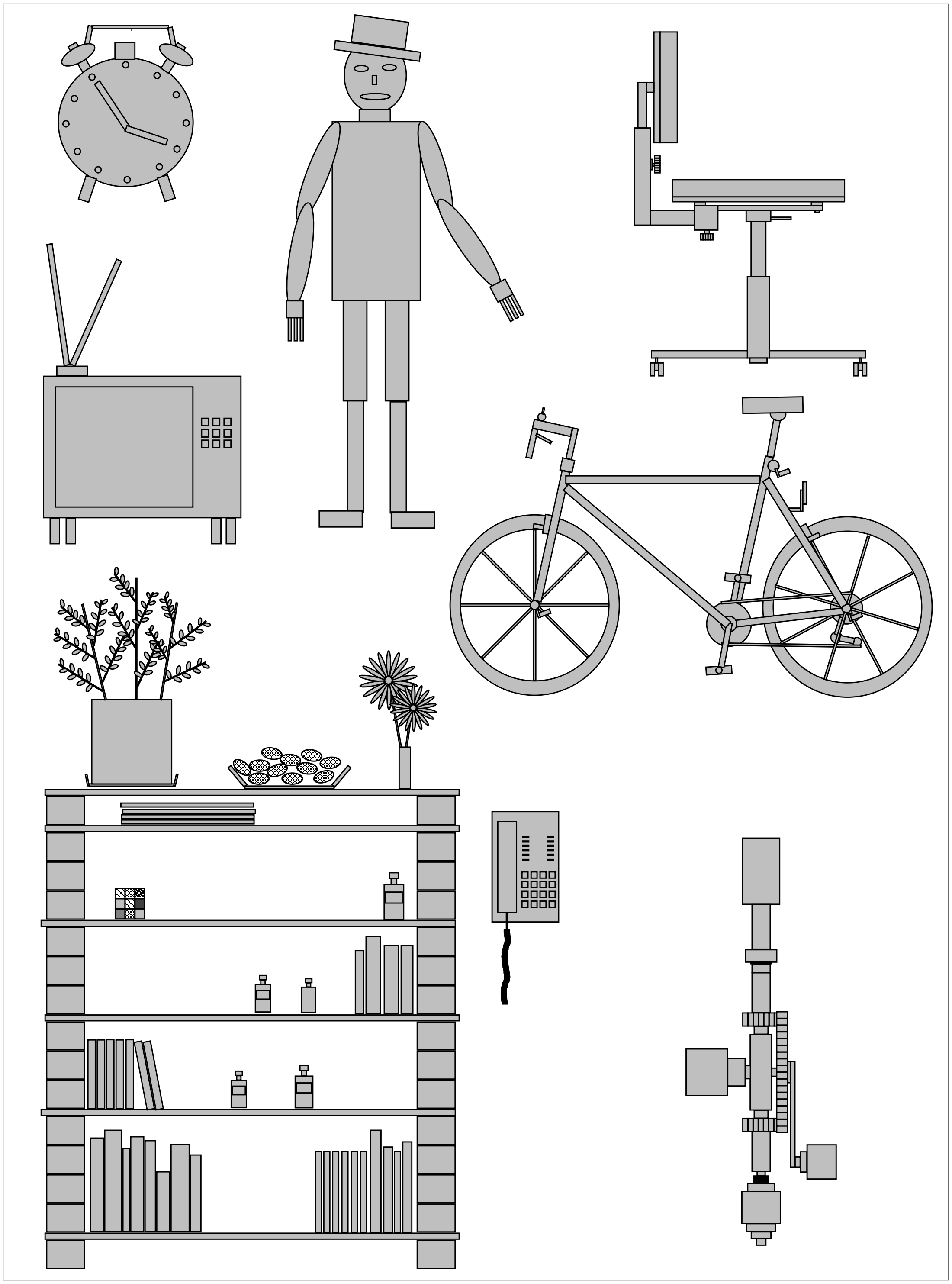


Figure 1.1: Some objects in GRAM's domain.

of complex structured objects. Most of the work in visual object representation and recognition does not address the problems of concept learning, and most of the systems developed for concept learning do not deal with or exploit the characteristics of complex structured physical objects. Therefore, although the work in this thesis draws on the research done in each of these areas, it presents some new ideas and techniques which directly support the development of an integrated classification and learning system.

One claim of this thesis is that complex structured objects can be effectively matched and generalised without functional or domain-specific knowledge. Lebowitz [Lebowitz, 1986] made a similar claim some years ago, although with less emphasis on structural objects, and certainly not addressing the complexity that GRAM deals with. GRAM achieves this by using a representation scheme that is richly expressive and allows redundancy, so that functionally important information is more likely to be implicitly embodied in the explicit structural descriptions, and less likely to be lost during the generalisation process. Representing objects at multiple levels of approximation and abstraction is an important aspect of this. Another aspect is the explicit distinction between structure and context. This enables GRAM to notice that two objects have similar isolated structure (or 'form') but different contexts (or 'role'), or vice versa, and also allows a simple form of disjunction to be represented.

In a structured domain in which scenes and objects may be composed of hundreds or thousands of components, an exhaustive or simple general-purpose matching strategy cannot provide sufficient efficiency (or even effectiveness) in a real-time system. Therefore, a second claim of the thesis is that complex objects can be matched more effectively and efficiently by exploiting the structural relationships between components of the objects to guide and constrain the search. This is a somewhat obvious claim, and other systems have also used it, primarily by representing objects in multiple levels of detail to allow top-down traversal of the objects [Wasserman, 1985], [Connell, 1985], [Marr, 1982]. However, GRAM's contribution is to show that *context* information can be exploited as well as substructure information by distinguishing between three types of inter-part relationships – parent, neighbour, and subpart relationships – each of which is itself a rich descriptive entity. Relationships can be used by the matcher to guide the search, not only down the decomposition hierarchies, but also via parent and neighbour relationships, thus providing multiple paths to the correct correspondences. This resolves the 'level hopping' problem in which corresponding components are on different levels of the decomposition hierarchies and would not be found by a strict top-down search. A further consequence is that GRAM is more robust since it does not require canonical object descriptions.

Another aspect of exploiting the structure of objects is the formation of *groups* of similar and similarly related components, where each group description summarises its members in terms of a 'typical member'. If the individual members are then removed, the overall object description is reduced in complexity, hence reducing the search required by the matcher when comparing such descriptions.

A third claim of the thesis is that descriptions of physical objects can be matched effectively and efficiently without enforcing global consistency between correspondences during the search. This approach is significantly different from the usual graph matching approach, and has surprising advantages in terms of simplicity, efficiency, and robustness: It avoids the need for a backtracking mechanism by keeping multiple competing hypotheses active simultaneously; it permits a much greater degree of parallel implementation; and it enables components of an object to play multiple roles when matched with another object.

As an example of what the implemented GRAM system [1] is able to do, the matcher was presented with descriptions of the two bicycles shown in figure 1.2, each consisting of 80 and 100 composite and primitive parts respectively, organised as a part hierarchy. GRAM correctly found 65 of the 69 desired correspondences, and then successfully generalised 57 of them. Two identical descriptions of *BIKE1* were also matched, and GRAM identified all 80 of the correct correspondences. Furthermore, a description of *BIKE1* was matched against a different description of the same bike, with significant differences between the decomposition hierarchies, thus testing GRAM's level-hopping ability. The matcher correctly found 73 of the 74 of the desired correspondences. The one incorrect correspondence was only marginally higher scoring than the correct correspondence, which GRAM also found. These results are described in more detail in chapter 7 which also presents the important result that GRAM's efficiency seems to be *linear* relative to the number of components in the objects being matched.

A methodology that is used throughout this thesis to address each problem, is to first identify the *kinds* of requirements, situations, characteristics, etc, of the problem, and only then consider the mechanisms that could be used to solve it. This helps to ensure that the solutions are fitted to the problem, rather than fitting the problem to some arbitrary solution.

Outline of this section and the thesis.

This introductory chapter begins the thesis by giving an overview of the main issues and ideas considered in the thesis. Section 1.1 describes the characteristics of the domain and task which the GRAM system supports, and which form the basis for justifying the various design decisions made throughout the development of GRAM. It presents several examples of the kinds of tasks that GRAM can perform.

Sections 1.2 to 1.6 give an overview-summary of the main issues and ideas presented in the five main chapters of the thesis.

Section 1.2 places this thesis in the context of other related research, and summarises the limitations of that research. This is explored in more detail in chapter 2.

Section 1.3 outlines the issues of how to represent complex structured physical objects, in generalised and ungeneralised form, and gives a brief overview of the key contributions of the GRAM representation scheme. This section is a summary-overview of chapter 3.

Section 1.4 briefly presents the issues of constructing a description of an observed object, based on information that is assumed to be available from a low-level vision system. Various components of this instance-construction process that have been developed for GRAM are briefly described. This includes a discussion of the various criteria used to justify the formation of

[1] The system is written in Common Lisp, and works directly from postscript data produced by a graphics package called IDRAW and a text file that specifies the part decomposition hierarchy.

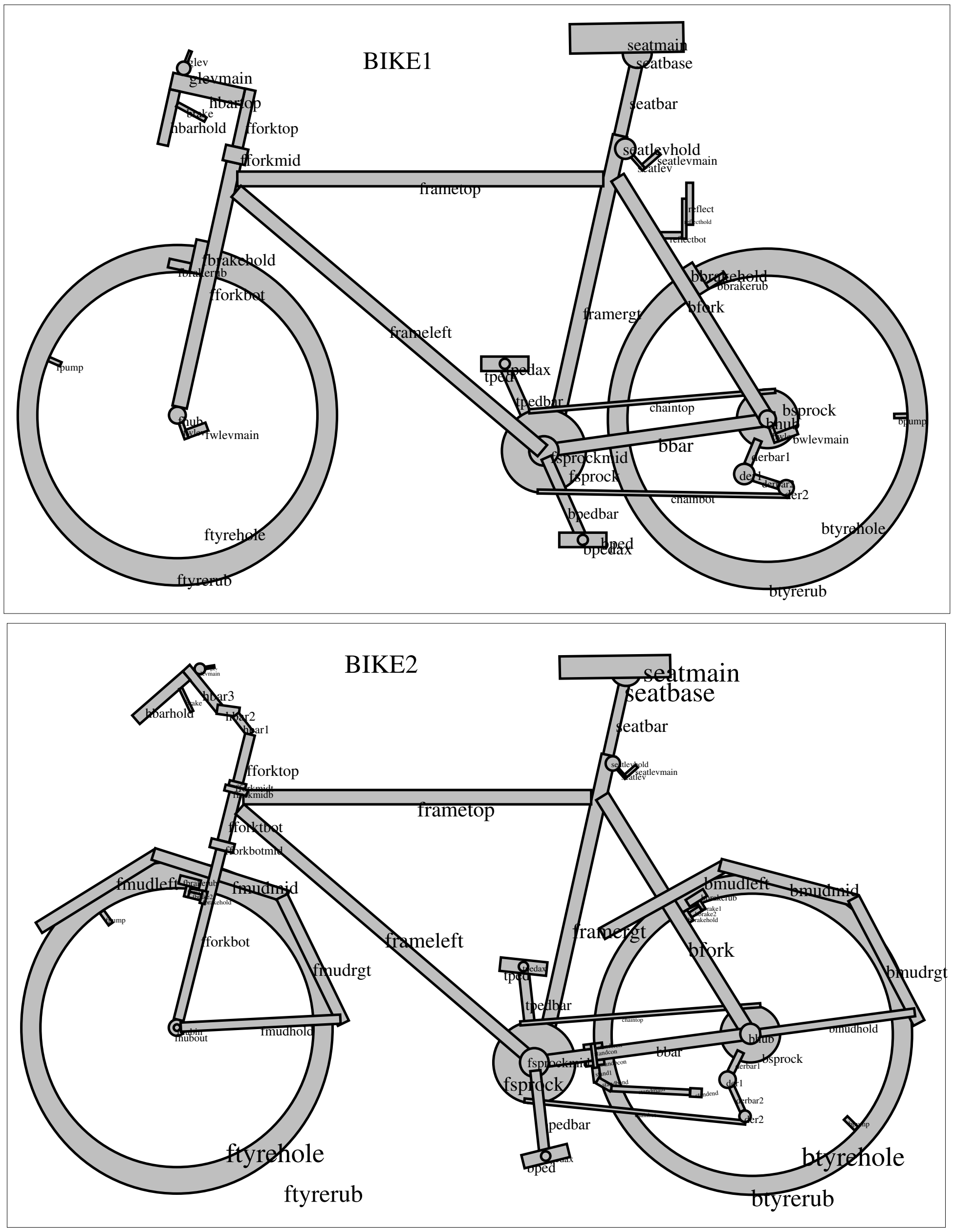


Figure 1.2: Two bicycles matched and generalised by GRAM.

composite objects within an instance description, and the criteria for forming explicit relationships between objects. Criteria and mechanisms for finding groups of similar components within a scene or object, are also discussed. This section is a summary-overview of chapter 6.

Section 1.5 introduces the main issues of how to match complex structured physical objects (both instances and concepts), and briefly describes how the GRAM matcher addresses these issues. The two main issues addressed are, firstly, how to measure the similarity between a concept and an instance, and secondly, how to search for correspondences between their components. This section is a summary-overview of chapter 4.

Section 1.6 outlines the main issues of how to generalise a concept description so that it accounts for a new instance. The main characteristics of the GRAM generaliser are also presented. This section is a summary-overview of chapter 5.

The implemented GRAM system is partially evaluated in chapter 7, and the main ideas and conclusions of the thesis are summarised in chapter 8.

An understanding of the main contributions of this thesis can be obtained by reading the introduction and conclusion chapters. The section headings throughout the thesis (and in the table of contents) can also be read as a rough summary.

1.1 The Domain and Task

Although the long term goal of GRAM is to be a component of an "instructable autonomous robot", which might be used as a household helper or a workshop assistant, this thesis is just one small contribution towards it, since such a project will also require many other mechanisms for planning, procedure learning, reasoning, language understanding, and so on, to be developed, covering almost all areas of Artificial Intelligence research.

GRAM is focussed specifically towards supporting the tasks of classifying physical objects and learning descriptions of categories of physical objects. In particular, it addresses the problem of representing, constructing, matching, and generalising object descriptions.

This section gives a brief overview of the kinds of classification and learning tasks that GRAM supports, and the characteristics of these tasks which have implications for the design of the system. It then briefly outlines the main features of GRAM's simplified two-dimensional domain and explains why this is sufficient for demonstrating the potential of GRAM to operate in a real-world three-dimensional domain.

1.1.1 Kinds of classification tasks.

There are several different kinds of classification tasks that need to be supported by GRAM. The simplest kind of classification task is of the form *"Is that an egg-beater?"*. Since the concept is given, there is no need to search concept memory. The instance is simply compared with the concept, and a measure of similarity is produced.

If the system knows about other concepts that are similar to the specified concept, then it may also need to match the instance with those. For example, if it knows that the concept *handdrill* is similar to the *egg-beater* concept, then it may need to compare the object with that, since the measure of similarity between an observed handdrill and the egg-beater concept might otherwise seem acceptable.

The task *"What is wrong with that X?"* is an extension to the *"Is that an X?"* task, because it involves identifying and reporting the key differences between the concept and the instance.

The most common classification task is of the form *"What is that?"*, and involves observing an object (such as a room or a hammer or a chair-leg), and classifying it by finding the best-matching previously-learned concept in memory. This task might also be extended to an entire scene, or to all of the components of an object, in the form "Classify the components of that scene or object."

To perform the task *"Find an X"*, the system does not need to search concept-memory, but it must search the observed scene or environment for an object that matches the specified concept.

An autonomous robot might have to perform a task such as *"Assemble those bicycle parts"*, and this may involve all of the above, since it needs to classify the parts, find particular parts (and tools) that it needs, and determine when the assembly is completed and correct. Similarly, the task *"Go to the bedroom and tidy it"* has similar requirements.

1.1.2 Kinds of learning tasks.

Learning tasks come in two forms, supervised and unsupervised. The basic form of supervised learning task occurs in response to the instruction *"That is an X"*, and involves generalising the concept features to take into account the features of the new instance. If the concept is not already known, then the instance is recorded as new concept.

Unsupervised learning occurs in response to the system's own classification of an observed object. Such learning is therefore susceptible to errors, since the classification may be incorrect. Therefore, the system could also seek confirmation from a teacher after classification, especially if the instance is unusual in some way. If an observed object is unrecognisable, then a new concept can be automatically created.

Unsupervised concept learning is made more complicated by the fact that each learned concept may have a number of subconcepts, forming a concept hierarchy, such as for different varieties of *chair* and different varieties of *office-chair*. Thus the system must be able to create, reorganise, use, and maintain such hierarchies.

In systems such as GRAM, where every component of an object is an instance of a concept, supervised learning is almost always accompanied by unsupervised learning. More specifically, if an object is given a classification by a teacher, which enables a particular concept to be generalised, the system is still responsible itself for determining the classifications of the subcomponents of the object, and for determining whether and how to incorporate the new subcomponents into concept memory. Thus a learning system in a structured domain always involves some unsupervised learning, unless its concepts are represented as complete part hierarchies rather than in terms of other concepts, or if the teacher specifies classifications for every subcomponent.

1.1.3 Characteristics of classification and learning tasks in a physical domain.

The descriptions of the classification and learning tasks above do not indicate the complexity of what is involved, and so this section discusses some of the characteristics of a real-world physical environment which must be taken into account.

Objects are composed of subcomponent objects, and concepts are therefore defined in terms of subcomponent concepts.

One of the main characteristics of dealing with a structural domain, such as a real physical environment, is that every observed instance of a concept is composed of smaller subcomponent objects which are themselves instances of other concepts. Therefore, the task "Is that an X?" not only involves comparing the object with the concept X (such as *chair*), but also involves classifying and matching its subcomponents with other concepts by which X is defined (such as *chair-leg* or *cushion*). Similarly, in order to recognise a room as being a *bedroom*, the system must recognise its main components, such as *bed*, *desk*, *etc*. Each of these concepts may exist within a hierarchy of concepts, and so the system must be able to deal with a potentially complex inter-dependence between concepts in memory.

The world can be viewed at multiple levels of detail.

Objects in a real-world domain are often recognisable from a rough level of detail, as has been discussed by [Biederman, 1985]. In many cases, classification is possible from just a few of an object's largest subcomponents perceived as rough shapes (such as cylinder, rod, cube, etc), and rough spatial relationships between them. For example, humans can recognise an object as being a bicycle without having to observe its exact shape and all of its details, which could vary considerably amongst different specialised varieties of bicycle. This suggests that concept descriptions should include abstract and approximate features to enable such classification, and also suggests that the matcher should exploit this property to enable rapid recognition.

However, finer details are also necessary for tasks such as fault-finding, which may require that the matcher takes into account every subcomponent. Finer details are also necessary for performing more specialised classifications, such as for discriminating between different kinds of cars. This implies that objects and concepts need to be represented in multiple levels of abstraction and approximation, and the classification system should exploit this.

A physical domain is characterised in terms of objects, relationships, properties, surfaces, and edges.

There are a variety of basic descriptive entities and features that characterise the physical world (for humans). The *object* is perhaps the most obvious descriptive entity, but equally important is the structural *relationship* between two objects. There are many kinds of information that humans seem to use to characterise a structural relationship, such as relative position, size, orientation, and alignment. Various forms of connectivity are also distinguished, such as fixed joins, articulated joints, contact, or 'same-piece' connections, such as between the bowl, stem, and base of a wineglass.

Objects also have a variety of different kinds of *properties*, such as shape, colour, texture, material, solidity, and so forth. Other kinds of components, such as surfaces, edges, corners, and axes, also characterise physical objects. All of these types of descriptive feature need to be taken into account by the classification and learning system.

Concepts may be highly variant.

An important characteristic of the kinds of domains in which a general-purpose autonomous robot would operate, such as a household or workshop, is that concepts can be highly variant. For example, there are many variations of the concept *hammer*, in terms of its shape, colour, structure, *etc*. This situation contrasts with that of specialised classification systems that operate in highly constrained environments, such as bin-picking robots or assembly-line quality-control robots that can assume that each object category is highly or completely invariant.

One aspect of variance is that the attributes that characterise the concept can have ranges of values, such as for length and colour. Another aspect is that components might be optional, with some measure of frequency of occurrence. For example, a television may or may not have an aerial on top of it, a chair may or may not have arms, and a door may or may not have a keyhole. Since an important aspect of a classification and learning system is the ability to make predictions, the variability of concepts suggests that concept descriptions should include probabilistic measures for the presence or absence of its sub-components and neighbouring objects.

In addition to having optional components, a concept may also be defined in terms of alternative, or disjunctive, sets of components. For example, the definition of a *door* might have to indicate that the door-handle can have any one of a variety of alternative door-handle structures. The door-handle might be described non-disjunctively at a coarse level of detail, with the disjuncts providing more detailed information to enable tasks such as fault-finding, or discriminating between types of doors, to be performed. The disjuncts could be specified simply by referring to a *door-handle* concept that has several subconcepts.

A concept may be defined in terms of its contents, independent of their arrangement.

Most object categories are defined in terms of a fairly rigid well-defined substructure, as in the case of a *hammer*, *desk*, or *vacuum-cleaner*. However, there are also some object categories of which the arrangement of their subparts (or 'contents') is highly variable, and therefore less important. For example, a bedroom typically contains a bed, desk, lampshade, wardrobe and so on, but the arrangement of these within the room is highly variable. Similarly, concepts such as *shopping-center*, *childrens-playing-area*, or *computer-lab* are also defined primarily in terms of their contents. Some of the relationships between the components may be important, but there is a great deal of variability.

Conversely, some concepts are defined primarily by their arrangement only. For example, the characteristics of the subcomponents of a *tower* or an *arch* are not particularly important, since it is the structural organisation of those subcomponents which is crucial.

Objects may be only partially visible.

An object may be partially occluded, either by other objects or just by the fact that it can only be viewed from one direction. There may also be insufficient time for more than a brief glance at the object, or the object may be observed from too far away to see more than a fuzzy blob. Therefore, the system should be able to cope with partial information, and be able to predict missing information on the basis of previously learned concept descriptions.

Object boundaries may not be available prior to classification.

Another difficulty for classification is that the distinction between component objects may not be available prior to classification. In other words, the 'objectness' of a region of a scene may not be identifiable by the low-level vision system. For example, when finding a hammer in a jumbled toolbox, the boundaries between objects can only be identified *on the basis of* classification, rather than prior to classification. Thus the process of processing an image cannot necessarily progress in a simple manner from low-level visual perception of pieces and blocks and objects and so forth, up to abstract recognition. Rather, a two-way up-and-down process may be necessary.

Concepts may be defined in terms of substructure (or 'form') ***and*** **context (or 'role').**

Although an object is often recognisable from its substructure, some concepts are defined just as much, or even more, by their surrounding *context*. For example, a *chair-leg* is defined in terms of its relationships with the *chair* concept, and other chair components. Similarly, a chair needs to be partially defined, or at least described, in terms of its typical context (such as being upright on a floor, and usually in a room). Context information can not only lead to the correct classification of an object, but can also be used in the reverse direction to predict classifications of its surroundings. For example, if a chair has been recognised from its substructure alone, then the context information in the *chair* concept can suggest that the object on top of it is a *person*, and the object next to it is a *desk*.

Scenes and objects often contain groups of similar components.

Scenes and objects often contain groups of similar and similarly related items, such as buttons on a shirt, windows on a building, fruit in a fruit bowl, or spokes on a bicycle wheel. The system should be able to exploit this to enable more compact 'summary' description, since a group can be characterised by a generalised description of its typical member, generalised descriptions of the typical relationships between members, and properties of the group as a whole. The formation of a group description, therefore, enables transfer of information between instances, since each member is effectively being generalised by replacing it by the description of the typical-member.

Groups also enable more efficient and effective matching, since two groups can be compared as whole entities, rather than attempting to find correspondences between every member of two groups, which might not be possible if the groups have different cardinalities. In fact, if the groups have different cardinalities, it is necessary for the groups to be explicitly noticeable and representable so that the generaliser can produce a generalised variable-sized group.

Another reason why the system should explicitly notice groups, is that this is a form of concept discovery. Normally a concept is learned by observing several instances over time, but a concept should also be formed if several similar objects are observed within a single scene. Thus, the creation of a typical-member description to characterise a group, is the creation of a concept. It is an unusual concept because is partially defined by relationships with itself, which denote the typical relationships between members of the group.

A concept may be a pre-condition for a robot action.

Another kind of concept that a robot system needs to learn are *action pre-conditions*. In fact, this was one of the original motivations for the GRAM project, since it was intended to fit into a procedure learning system developed by [Andreae, 1985] which required a subsystem that could learn visual pre-conditions for actions in a generalised procedure. These conditions are often partial scenes, rather than objects. For example, the concept "full shelf" could trigger a dishwashing robot to start stacking plates on the shelf above, or the concept "untidy room" would trigger a household robot to commence a tidying activity.

Objects are often characterised by the way they move, the way they interact with other objects, and their function.

There are aspects of objects other than their structural properties that are also important to the definition of a concept. For example, an important characteristic of a bicycle is that the wheels, pedals, and front fork (and many other parts) move in certain ways. Therefore, to fully capture the definition of a bicycle, this kind of information should be representable.

Similarly, the kinds of activity that a bicycle is typically involved in, such as rolling along a road from one location to another, is also important. In fact, this kind of information would comprise the definition of the *function* of a bicycle. Most man-made objects serve some function, and therefore functional knowledge is clearly important in a concept description. It should be noted that the function of an object is often partially defined by its structural context, such as the way in which a bicycle is structurally related to the parts of a human body.

Rapid classification must be possible because scenes may consist of many components.

In a real-world domain it is necessary to perform classification very rapidly because a scene or object may contain a huge number of component objects. If a robot is navigating through a room or building, such as when doing the vacuuming or searching for an object, its eyes are presented with vast quantities of information which must be chunked into large numbers of recognisable objects. This suggests that the mechanism for indexing from instances to concepts must be very efficient, as must be the process of comparing instances and concepts.

There are a vast number of concepts that characterise a physical environment.

In addition to the problem of coping with scenes that contain huge numbers of objects, big and small, there is also the problem of dealing with a vast number of different concepts. If I look around the office I am in, every single object can be considered to be an instance of a concept, such as *telephone*, *lockable-door-handle*, *space-bar*, *dust-speck*, *scratch-on-desk*, *desk-keyhole*, and *row-of-books*. Some concepts, such as telephone, have labels associated with them, while others such as *desk-keyhole* may not, since they are often referred to in speech in the form "the keyhole of the desk" or "a row of books".

A concept might also be defined solely in terms of one particular instance, such as the 'concept' *my-set-of-keys*. Therefore, concept memory may be vast, and this suggests that efficient memory access and organisation mechanisms must be available.

1.1.4 The domain and tasks of the GRAM system.

The development of a classification and learning system which takes into account all of the domain and task characteristics discussed above, is clearly not a trivial matter, let alone the problem of building a complete autonomous robot. GRAM goes some way towards extending the research in this area by focusing on three core components of such a system: a representation scheme, a matcher and a generaliser. Some aspects of instance-construction are also addressed, in particular the group-finding process. The discussion below explains in more detail what GRAM actually does, and the domain in which it operates.

GRAM **deals with a two-dimensional domain, where objects are comprised of simple 'blocks'.**

I have chosen to work with a two-dimensional domain because it simplifies the development of the system while still capturing most of the important features of a real-world domain discussed in the previous section. Most of the objects dealt with in the thesis are, in fact, very close to projections of three-dimensional objects. Recent work discussed in chapter 26 of [Winston, 1992] shows that recognition of three-dimensional objects is actually possible from just a few two-dimensional projections, without even requiring any volumetric description.

The only significant limitation of a strictly two-dimensional domain is that objects are always completely visible, and the system does not have to deal with the problem of partial views. The current system allows two-dimensional objects to overlap, but there is no notion of occlusion. However, the issue of occlusion could be addressed without having to deal with a full three-dimensional domain, by adding depth information. The problem of coping with the hidden two-(and-a-half)-dimensional objects can be considered equivalent to the problem of coping with the hidden portion of a three-dimensional object.

The input to the GRAM system is a description of an object or scene consisting of simple primitive 'blocks' which can be rectangles, ellipses, or simple polygons, such as those in Figure 1.1 above. Each block is considered to have a bounding rectangular box which defines its dimensions and orientation relative to other blocks, as illustrated in Figure 1.3. Additionally, the input also includes a set of 'fuzzy blocks' such as those indicated by the dotted boxes for the humanoid in Figure 1.4. These are visual approximations of a set of smaller blocks, at a coarser level of detail.

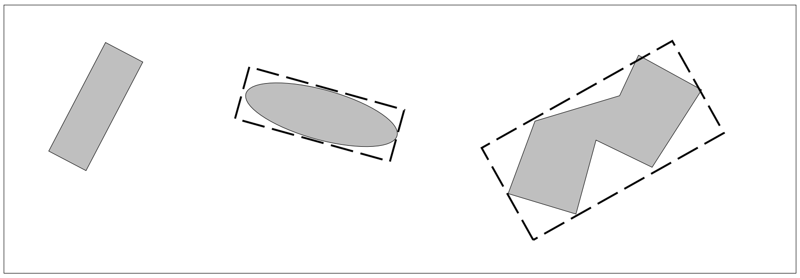


Figure 1.3: Primitive block shapes.

GRAM assumes that a low-level vision system (such as described in [Connell, 1985]) is available to produce descriptions of blocks at multiple levels of approximation, although at present the input comes directly from a graphics/drawing program. However, an issue that is not significantly addressed in this thesis is how the high-level recognition system should interact with the low-level vision system, since it is not always the case that a vision system can construct block descriptions bottom-up, without guidance based on the expectations of the high-level recognition system. However, GRAM can cope with partial information, so that if the vision system is not able to produce a full block description from an image, the matcher can still make use of what it is given. ACRONYM [Brooks, 1981] takes a different approach, in which the recognition system works more in the opposite direction, generating expected two-dimensional image features from its three-dimensional generic models, rather than producing a three-dimensional model from the image.

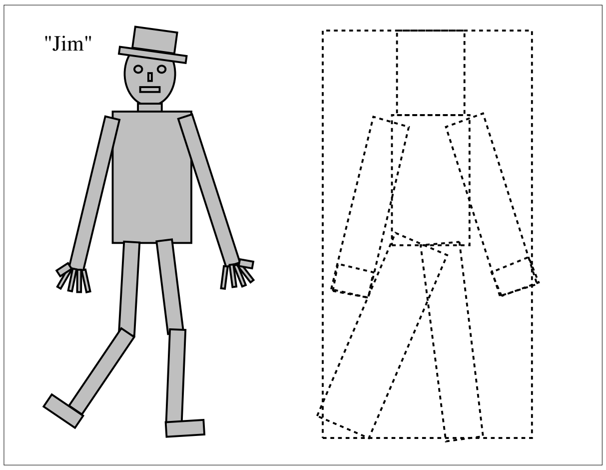


Figure 1.4: Composite parts of a humanoid.

Each block in GRAM's domain can have various properties, such as aspect-ratio, shape, number of edges, *etc*. Currently it does not include colour, texture, material, stiffness, etc, although these could easily be added. A simple extension to allow blocks to be *generalised cylinders* [2] would also enrich the domain considerably, and would only require the representation language to include a few additional properties specifying the spine and taper functions.

The system is able to obtain information about how pairs of blocks are spatially related to each other, including relative size, proximity, direction, orientation, and alignment. Currently there is no distinction between different kinds of *connection* relationship between blocks. For example, there is no distinction between blocks that are fixed together, merely touching, or have an articulated joint. Nevertheless, the description of relationships in GRAM are still sufficiently rich to give good performance, and the addition of more connection types would only improve its performance.

The simplified two-dimensional domain has made development of GRAM easier, because it is has not been necessary to develop mechanisms for processing a mass of low-level details. However, it also means that blocks are not as easily distinguishable on the basis of properties alone, as can be seen in the examples in figure 1.1. Blocks are primarily distinguished by their substructure and/or contextual relationships.

If the domain was extended to include more property and relationship distinctions, this would certainly help GRAM to classify objects more effectively, since such information would help discriminate between classes, as humans may find when looking at black and white photos. However, the argument of this thesis is that if GRAM can work reasonably effectively *without* such information, then it will certainly work even better in a domain in which more information is available, and so I do not consider the simplified domain to indicate a limitation of GRAM. The thesis is primarily focusing on dealing with rich structure, in terms of substructure and context relationships, rather than many kinds of property such as colour, texture, *etc*

This thesis claims that extending GRAM to deal with three-dimensions is straightforward: Firstly, object properties would be modified to account for the three axes of the object. For example, the *aspect-ratio* property could be split into two or three properties, specifying the ratio of the longest axis with the middle-length axis, and with the shortest axis. Secondly, relationships would need to be defined with respect to the three-dimensional coordinate frames of the objects, rather than their two-dimensional coordinate frames. Thirdly, the matcher would have to consider more alternative axis correspondences when comparing two objects. Other than these relatively minor extensions, the representation, matcher, and generaliser can remain unchanged.

Only structural (rather than functional or behavioural) knowledge is to be used.

One of the goals of this thesis was to find out whether effective concept learning can be performed by a system which deals only with syntactic *structural* descriptions, without consid-

[2] Generalised cylinders are defined by a cross-sectional area swept along a spine with some taper function. Systems such as Brooks' ACRONYM [Brooks, 1981] and Connell-and-Brady [Connell and Brady, 1985] use this representation. In two-dimensions a generalised cylinder is perhaps better called a generalised rectangle, since the cross-section is just a line-segment.

ering *functional* or *behavioural* knowledge. GRAM therefore deals with only one static scene or object at a time, without any notion of *time* between different observations, and without any knowledge of, or ability of reason about, how the object is used, or intended to be used.

One justification for this approach is that in many cases there is no functional or behavioural knowledge available for an observed object, and matching and generalisation *must* be able to manage with only static structural information. For example, if a teacher asks a robot-helper to find "one of these", while showing the robot an unfamiliar object, but does not say what it is used for, then only structural information can be used. Similarly, a robot workshop assistant should be able to learn to recognise a class of tool prior to learning what it is used for.

Another justification is that the 'function' of an object is often definable largely in terms of structure anyway. For example, the function of a chair is that it allows a person to be attached to it in a particular structural pose. Likewise, the function of a table-leg can be defined (in part) in terms of the way in which it is vertically beneath the table-top. Obviously knowledge about gravity, support, uses of tables, etc, may help to recognise an unusual table, but for more standard tables, recognition from structure alone is simpler and more efficient than having to perform functional reasoning about whether an observed object satisfies the required function. Structural descriptions could be said to 'operationalise' functional descriptions. Other arguments for the utility of structural descriptions, with no functional knowledge, have been discussed by Lebowitz in [Lebowitz, 1986].

The task is to construct, match, and generalise object descriptions.

GRAM performs three tasks. The first task is to construct a structured instance description from the set of blocks provided by a low-level vision system. This involves creating an object description for each block, by producing properties and relationships that characterise the structure and context of the object. Also, various other composite objects may be created by combining sets of smaller objects that collectively form some interesting abstract whole, such as a group of similar items, or a topologically distinct structure. This can be considered a form of *constructive induction* [Dietterich and R., 1986]. In a future GRAM system, the process of producing an instance description could also include classifying the component objects. This would enable the system to predict missing or occluded information, and to suggest the formation of additional composite objects. Thus, this first task would somewhat overlap with the second task.

The second task is to match an object description with a specified concept description. The object may be any component within an observed scene or some other enclosing object. To do this, the matcher needs to match the descriptions of the object's subcomponents and structurally related objects with the descriptions of the substructure and context of the concept. The result of the matcher is a comparison description that specifies the similarity (and dissimilarity) of the two descriptions, and the best correspondences between the components of their structure and context descriptions, each of which also has its own comparison description.

The third task is to produce a generalisation of a concept to cover an observed object, making use of the results of their comparison produced by the matcher. In doing this, the generaliser may need to generalise other concepts that define the substructure and context of the concept, to cover the object's subcomponents and context. For example, Figure 1.5 shows a number of chairs which could be shown to the GRAM system. It would create a new *chair* concept from the first chair, and then for each successive chair observed it would generalise the *chair* concept to account for the features of the new instance.

Therefore the three components of GRAM are the *instance-constructor*, the *matcher*, and the *generaliser*. The issue of classifying an instance by indexing into, or searching, concept memory, and the issue of how to reorganise concept memory in response to a new instance, are discussed briefly later in this chapter, but are not significantly addressed by the thesis.

1.2 Related Work

There has been relatively little research done in the area of structured object learning, and even less in the domain of physical objects. Chapter 2 describes some of the systems that address this problem and are most relevant to this thesis. A brief outline of the main contributions and limitations of these systems is given in this section.

Winston's learning system [Winston, 1975] (described in section 2.1) learned generalised structured descriptions from examples, represented as semantic networks in which the nodes were parts or properties. The system provided much of the motivational basis for this thesis, and some of the ideas have been adopted and extended, especially the methods for finding groups of similar objects and representing them as a single descriptive entity characterised by a description of the 'typical-member'.

An important contribution of Winston's system was the idea of using "near-miss" negative examples. GRAM, in contrast, uses only positive examples, using frequencies of observation to indicate the importance of a each feature. In fact, this thesis argues that near-miss examples (such as a chair with a missing leg) should still be classified as a chair. The missing leg is a 'fault', rather than a feature that, if missing in an instance, indicates that the instance is not a chair. This issue is discussed futher in section 8.3.5.

Winston's system operated in a "toy" blocks-world domain and did not deal with complex objects. It did not make explicit use of multiple levels of detail, such as for improving the efficiency of the match algorithm, which was not described in the paper. The system was not able to represent optional parts, or specify probabilities of the presence of a part. The system introduced several important generalisation operations for structured objects, although it did not deal with ambiguity or disjunction formation.

Brooks's ACRONYM, discussed in section 2.2, has the most expressive representation scheme of the systems discussed in this thesis. Objects are represented as a part hierarchy, and properties and relations can be described in terms of conjunctions of complex arithmetic expressions that can include variables and parameters of the parts (such as height, orientation,

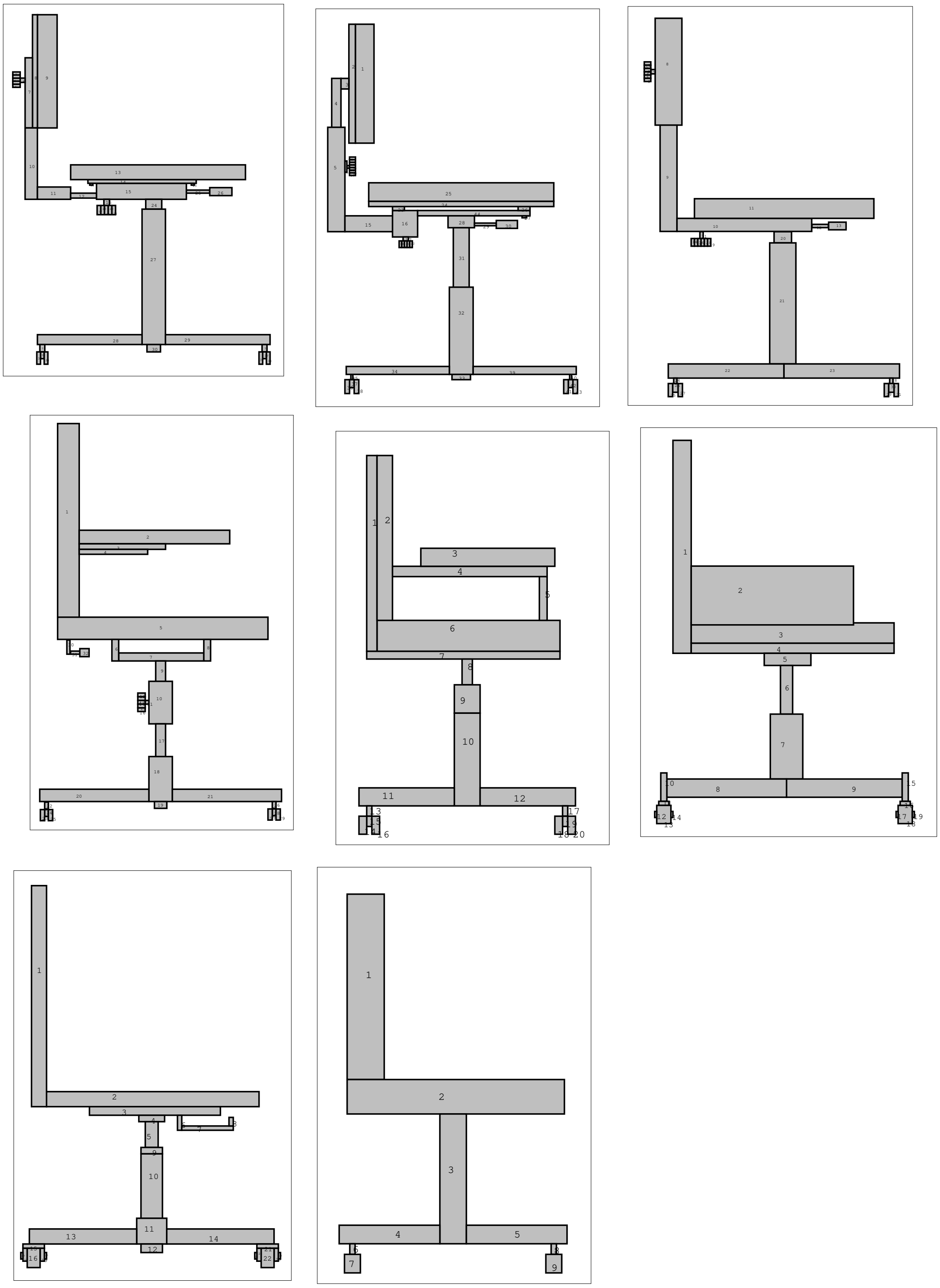


Figure 1.5: Some chairs.

etc). However, ACRONYM does not address the problem of learning, which is why the representation can be so rich, since it is not clear how the arbitrary arithmetic expressions could be generalised.

An important contribution of ACRONYM is that it addresses the problem that it is not usually possible to obtain a complete volumetric description of an observed object using a bottom-up data-driven approach. Instead, ACRONYM uses geometrical reasoning about three-dimensional model descriptions to make predictions of expected two-dimensional features.

CLUSTER/S [Stepp and Michalski, 1986a] (discussed in section 2.3) is a non-incremental unsupervised learning system that finds conceptual clusters of structured objects. The key idea of relevance here is that it converts a structured description into an attribute-vector description that can then be processed by the attribute-based CLUSTER/2 system [Stepp, 1987a]. This idea has, in some sense been adopted by Labyrinth (discussed below) and also GRAM, by representing concepts as simple entities defined in terms of a few relationships to other concepts, rather than as a complex part hierarchy defined locally to the concept.

Wasserman's MERGE (described in section 2.5) performs incremental concept acquisition and organisation for objects that have a hierarchical structure, such as physical objects or corporate management structures. MERGE distinguishes between G-trees and F-trees, where each G-tree is a subconcept hierarchy or taxonomy for a particular class of objects, and each F-tree is a part hierarchy for a particular instance object or generalised object. A G-tree is a hierarchy of F-trees, and an F-tree is defined by a set of subparts, which are in fact nodes in a G-tree. Thus every component of an object is an instance of a concept, rather than merely being an instance of a component of a concept. This approach has also been adopted by Labyrinth (below) and GRAM. The system allows inheritance, and deals to some extent with the "level hopping" problem in which corresponding components in two objects cannot be matched because they are on different levels of the part hierarchies.

The key limitation of MERGE is that an observed object and its parts are already pre-classified, by their names, as belonging to a particular G-tree. Thus the classification and matching problems are made relatively trivial.

NODDY [Andreae, 1985] (described in section 2.6) is a procedure-learning system of which GRAM was originally intended to be a component. Since the actions of a general-purpose robot system should be able to be conditional on visual input, GRAM was to be a subsystem that could build generalised descriptions of visual observations, which could then be used in the conditional statements of a generalised procedure. The process of matching and learning procedures is similar to the task of GRAM in that it involves matching and generalising structured descriptions, and forming groups of repeated components. However, procedures have a simpler sequential structure, without multiple levels of detail, and so the techniques used in Noddy are not particularly extendible to the domain of physical objects.

Connell and Brady's system [Connell and Brady, 1985] (described in section 2.7) was built with similar goals to GRAM. It learns descriptions of two dimensional objects, and was intended to form part of a "mechanic's mate" project which would assist a mechanic in various ways, such as finding a desired category of tool. It represents concepts and instances as a semantic network, with coarse details at the center of the network, and finer details nearer the fringe. The matcher works by spreading outwards through the network, and this approach is similar to that used in GRAM, except that Connell and Brady's matcher only searches from coarse details to fine details, rather than in any direction through the network. It also does not address the level-hopping problem. The representation does not support optional parts, groups, or disjunctive substructure, and the generaliser does not deal with ambiguities. An important idea of the system is the use of Gray Coding, which enables a unified matching and generalisation scheme to be employed.

Labyrinth [Thompson and Langley, 1991] (described in section 2.8) is an incremental unsupervised concept learning system for the domain of structured objects. It deals specifically with the issue of acquiring and organising multiple concepts in memory. An instance is represented as a part hierarchy, and each concept is defined by a set of subparts and the concepts of which these subparts must be instances. Relationships between the parts are also represented, and probabilities can be associated with features. To classify an instance, its subparts are classified, and then a modification of the COBWEB algorithm [Fisher, 1987a] is used to traverse a concept hierarchy to find the most similar concept. An important limitation of Labyrinth is that it does not include *context* in its concept descriptions. A more significant limitation is that its classification scheme *relies* on concepts not being defined in terms of context, since it could not otherwise classify leaf nodes of an instance part-hierarchy prior to classifying parts higher in the hierarchy, or even siblings. It also requires that every instance part is classified separately, using the modified COBWEB algorithm, rather than directly accessing a candidate concept via the expectations of other classifications.

The PARVO system [Bergevin and Levine, 1993] (described in section 2.9) performs object recognition from two-dimensional line-drawings. It does not address the problem of learning. The relevant contribution of PARVO to this thesis is that it demonstrates that physical objects can often be classified on the basis of their coarse details alone, without requiring the finer details to be matched (unless more specialised classifications are required, or for the task of fault-finding). This characteristic of the domain of physical objects also means that if the matcher is able to operate from coarse levels of detail to fine levels of detail, then the coarser levels of the description are likely to be correctly matched, and therefore will be able to guide the matching of finer details.

1.3 Representation

The representation scheme is central to GRAM, since it underlies all of its other components. The way in which concepts and instances are represented largely governs the design and the performance of the rest of the system. This section gives a very brief outline of some of the main principles and contributions of GRAM's representation language.

1.3.1 An instance is represented as an object graph, with parent, neighbour, and subpart relationships.

To describe a complex physical structure, it is necessary to represent it in multiple levels of detail. GRAM does this in the form of an object decomposition hierarchy, in which each object (except primitive objects at the bottom of the hierarchy) is an abstraction or approximation of the subpart objects beneath it. Each object can be described by its set of subparts.

However, objects also need to be represented in terms of their context, not only their substructure. The context of an object includes not only its enclosing objects (that are higher in the decomposition hierarchy), but also the objects connected to it, close to it, or otherwise interestingly related to it.

Therefore, GRAM's representation provides three types of relationship, namely *parent*, *neighbour*, and *subpart* relationships, where each relationship is a rich descriptive entity characterising how one object is structurally related to another object in terms of position, size, connectivity, alignment, orientation, *etc*. This allows objects to be represented not just as a decomposition hierarchy, but as a graph. A description of an object also includes a set of structure properties (such as aspect-ratio, shape, etc) and a set of contextual properties (such as a connectivity profile).

Figure 1.6 shows an example of an object-graph, where each node denotes an object that has explicit relationships with other objects. The solid lines denote parent or subpart relationships, and the dotted lines denote neighbour relationships. [3]

One new idea here is that the representation does not deal with arbitrary relations between nodes, such as *left-of*, *bigger-than*, etc, but instead combines the information about the relationships between objects into parent, subpart, and neighbour relationships, each of which is defined by an attribute vector consisting of both qualitative and quantitative information. Therefore, when comparing the relationships between two pairs objects, it is not necessary to deal with a multitude of separate relations. Instead, a single attribute-vector comparison is performed, giving a single overall similarity score.

Each relationship is not only a descriptive entity that partially characterises the structure or context of an object, but also acts as a direct link between objects, in any of three directions through the object graph. These links are exploited by the matcher to constrain and guide the search for correspondences between objects and learned concepts. The use of neighbour relationships for this purpose is especially significant, since it enables the matcher to cross levels of the hierarchy, rather than being restricted to a top-down search. This is discussed further in section 1.5.

1.3.2 Structure and context are explicitly distinguished.

Another contribution of the GRAM system is the explicit distinction between context (defined by parent and neighbour relationships, and various contextual properties) and structure (defined by

[3]Each line should actually be depicted as two distinct directed lines, since each object has its own set of relationships, and they are not shared by other objects. This simplifies the generaliser.

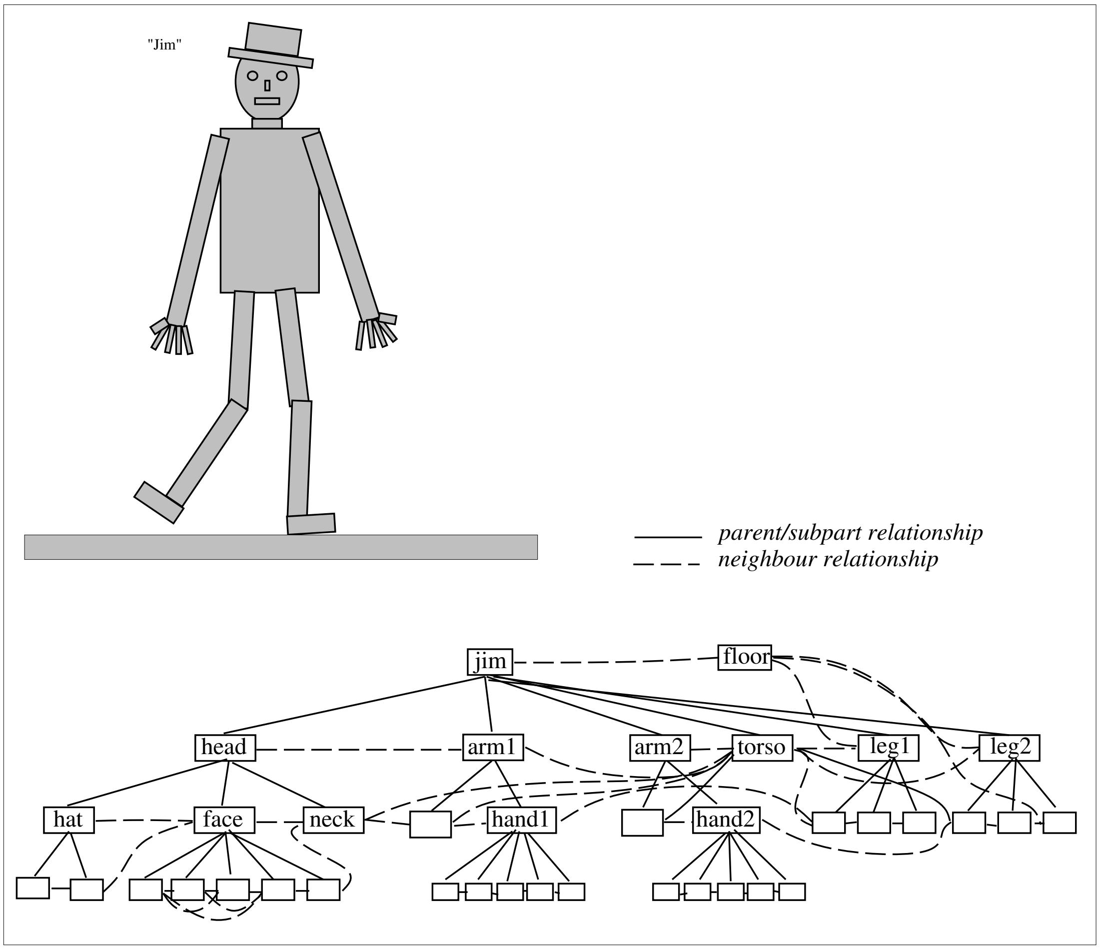


Figure 1.6: An object graph.

subpart relationships and various structural properties). This supports partial matching, since it enables the matcher to notice that two objects have similar structure (or 'form') but different contexts (or 'role'), or vice versa. The distinction also allows structure and context disjuncts to be represented.

1.3.3 Groups are represented by a multi-relationship to a typical-member concept.

A characteristic of everyday physical domains that was identified earlier is that scenes and objects often contain groups of similar items, such as rows of books, or cookies in a bowl, as illustrated in Figure 1.7. Groups should be explicitly representable for a number of reasons: to enable several objects to be represented compactly in summary form as a single entity; to enable properties of the group as a whole to be made explicit; to enable efficient matching by comparing groups as single entities, rather than comparing individual members; to enable generalisation of groups that have different cardinalities; and to provide transfer of information amongst group members.

Winston introduced the idea of representing a group in terms of a *typical member*. GRAM extends this idea by allowing the typical member to be a complex structured generalised concept. The group object has a *multi-relationship* to this typical-member concept, which is a generalised subpart relationship with a *howmany* count indicating how many instances of the concept are present in the group. In a generalised group, the *howmany* count may also be generalised.

A typical-member concept can also have neighbour relationships to itself, which represent the typical inter-member relationships, thus capturing the topology of the grouping. Various kinds of grouping topologies are possible, such as a linear chain, a grid-like array, an unstructured cluster, or a loop.

GRAM takes this notion of a multi-relationship further by allowing *any* instance or concept description to include multi-relationships to *any* concept, and this is interpreted to mean "there are *n* instances of that concept related to this in such and such a way." Thus groupings can be described without even having an explicit group-object. For example, a bedroom or bookshelf might have a multi-relationship to the concept *pot-plant*, without having to represent the collection of potplants as an explicit entity.

1.3.4 A concept is a generalised object, defined in terms of other concepts.

A crucial component of GRAM's representation scheme which is largely responsible for enabling complex structures to be dealt with in a manageable way is its representation of concepts. During the earlier stages of working on this thesis, the representation included three types of descriptive entity: the concept, the instance, and the part [Andreae, 1993]. Each concept and each instance was represented as a complete graph of parts, so that the definition of a concept consisted of an explicit set of all parts and their relationships. To compare a concept with an instance, the matcher needed to find non-conflicting one-to-one correspondences between the parts of the two graphs.

For large complex objects, such as a bicycle or bedroom, this scheme proved very problematic. One problem was that descriptions of concepts such as bicycles and bedrooms must allow disjunctions, to represent alternative and optional substructure and context. This led to very complex concept graphs which were unwieldy to match and generalise. For example, if all of the chairs in Figure 1.5 are generalised to form a single part graph that characterises all of the common and variant features, then the part graph will be very complex.

Another problem was that concepts must often be defined in terms of components that should be concepts in their own right, such as wheels, handlebars, beds, pillows, etc, and which should be recognisable directly without having to necessarily deal with an enclosing bedroom or bicycle description. This suggests that subgraphs need to be extracted out when appropriate, and the original graph somehow refer to them, perhaps via inheritance.

The complexities of this representation scheme were overcome by the development of a representation scheme in which each concept description is a small compact chunk of information

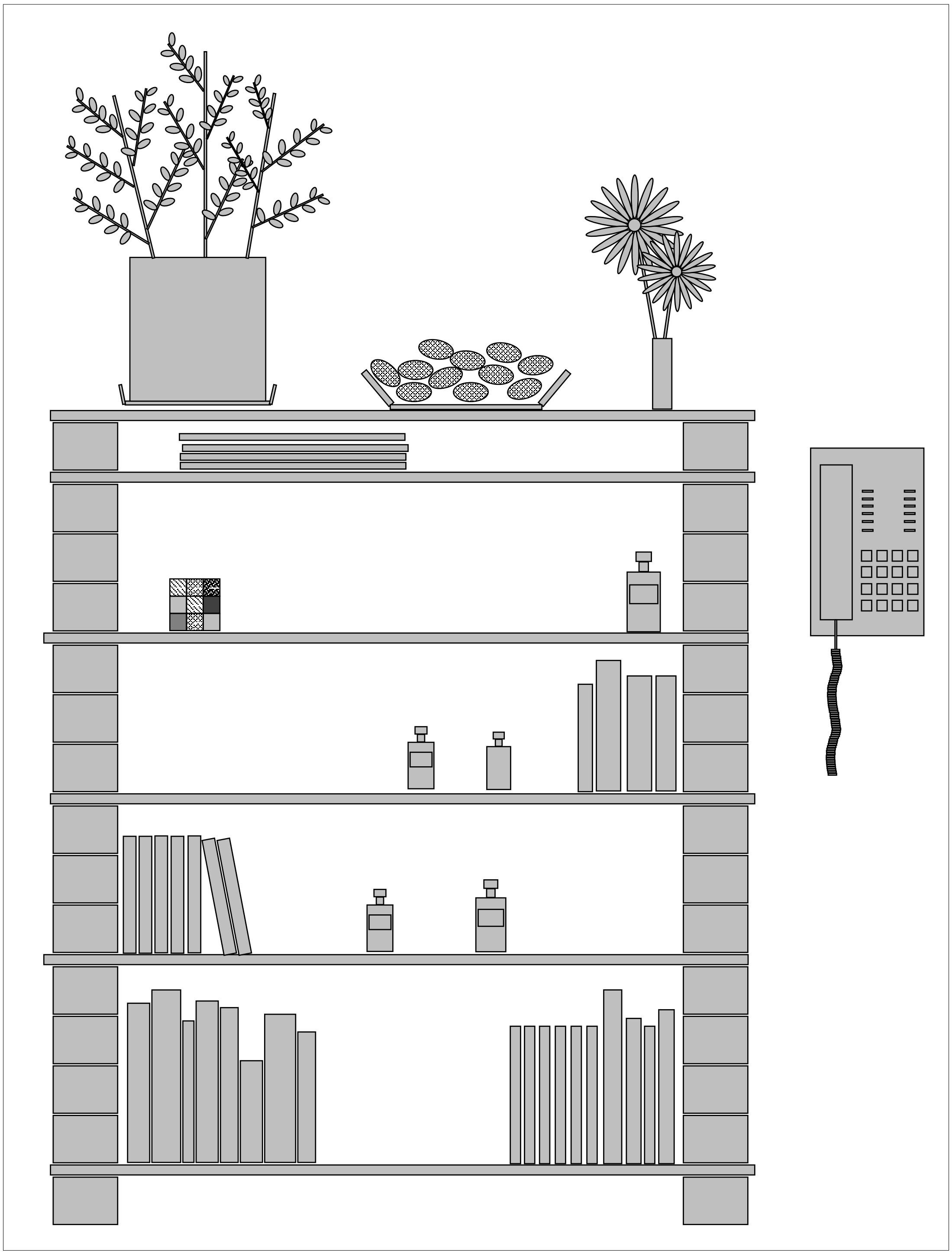


Figure 1.7: Some examples of groups in a bookshelf.

that consists of a set of properties, and a set of parent, neighbour, and subpart relationships *to other concepts*. If a concept X has a relationship to a concept Y, then this is interpreted to mean that for each instance of concept X, there exists an instance of concept Y which is related to the X instance in the specified way. This means that there is no explicit set of parts stored in a concept description, but only references to other concepts. A concept's part decomposition hierarchy and its context are now less explicit, although concepts can still capture the richness and complexity of the domain.

The simplicity and uniformity of this representation has significant implications for the matcher and generaliser. There is no need to deal with two kinds of entity — concepts and concept-parts. Instead, memory consists only of concepts, each of which is a small manageable description. Also, the matcher does not have to find a one-to-one correspondence between two nodes of two graphs. Instead, a simpler, more flexible, and more robust method is possible, as will be outlined later. Disjunction is now representable simply by defining a concept by a set of subconcepts, and any other concept that is defined in terms of such a concept is therefore implicitly disjunctive.

Labyrinth [Thompson and Langley, 1991] also used this idea to some degree, but the contribution of GRAM is, firstly, that it includes *context* in a concept description, thus making each concept a more richly described and more constrained entity. This reduces the kinds of problems that Labyrinth had due to the under-constrained nature of their concepts which were defined only in terms of subparts. Secondly, the GRAM matcher exploits parent and neighbour relationships to guide the search, allowing a multi-directional search, rather than a merely top-down tree traversal which suffers from the "level-hopping" problem when two similar objects have been decomposed into hierarchies that do not correspond level-to-level. Thirdly, GRAM includes multi-relationships and groups (by referring to a typical-member concept) giving the representation greater expressiveness, and thus improving the performance of the matcher and generaliser.

1.3.5 Structure and context can each be described disjunctively.

GRAM explicitly distinguishes between structure (or form), defined by a set of properties and a set of subpart relationships, and context (or role), defined by a set of properties and a set of parent and neighbour relationships. This distinction also means that the structure (and/or context) of a generalised concept can be described disjunctively. This is done simply by indicating that the structure (and/or context) is disjunctive, which causes the matcher and generaliser to use the structure (and/or context) of the *subconcepts* of the concept as the disjuncts. Each subconcept is a disjunct, or variant, of its parent concept.

For example, the concept *door-handle* could be defined by multiple forms that fulfill the same role. More specifically, it could be defined by a single non-disjunctive context description (consisting of a relationship to the concept 'door') and a disjunctive structure description defined by the set of structures of the subconcepts of the *door-handle* concept.

Conversely, the concept *swivel chair* could be defined by a single form but multiple roles. More specifically, it may have a single non-disjunctive structure description, and a disjunctive context description defined by the contexts of the subconcepts of the concept, each characterising a different role of a swivel chair, such as with and without a person on it.

Sometimes a concept has disjuncts of both structure and context. For example, instances of the more general concept *chair* can appear in a variety of contexts and can have a variety of substructures. In this situation, the structure and context disjuncts are not distinguished, but are both defined by the complete set of subconcepts. Some of the subconcepts may have a highly generalised context and a specific structure (such as the subconcept for a standard four-legged chair); others may have a highly generalised structure and a specific context (such as a chair with a person sitting on it); and others may have a specific structure and a specific context (such as a dentists chair). The current GRAM representation scheme is not able to explicitly distinguish between 'structure subconcepts' and 'context subconcepts', as this would complicate the concept hierarchy considerably.

1.3.6 Concept descriptions are probabilistic.

Section 1.1 stated that concepts in GRAM's domain can have highly variable properties and relationships. A generalised concept description in GRAM expresses the permissible variability of its instances in two ways, both of which have been used in other earlier systems, most notably COBWEB [Fisher, 1987a]. Firstly, each numerical attribute value (such as for size, orientation, etc) is represented as a distribution with a mean and variance, and each nominal attribute value is represented as a frequency distribution. Secondly, each parent, neighbour, and subpart relationship of the concept description has an *instance-count* which indicates how many observed instances of the concept included that relationship. The concept as a whole also has an instance-count, which is the total number of instances that contributed to the generalisation. Thus the ratio of the instance-count of a relationship over the instance-count of the concept as a whole can be interpreted as a probability or expectation of a new instance having that feature. For example, the concept 'door' *may* have a subpart relationship to the concept 'door-lock' with an instance-count ratio of 0.3, meaning that 30% of doors have door-locks.

1.3.7 Concepts can have a variety of interpretations.

GRAM allows concepts to be interpreted in several ways, depending on how the concept was formed. If a concept is formed from the parent, neighbour, and subpart relationships that are common to all or most of its instances, ignoring any atypical relationships, then the concept has a *partial* interpretation, meaning that the matcher should permit a new instance of the concept to have any additional parents, neighbours, and subparts, so long as it satisfies the concept's structure and context properties. On the other hand, if a concept is formed by taking the union of the instance's parent, neighbour, and subpart relationships, then it has a *complete* interpretation. If a new instance has additional relationships that are not present in the concept description, then these indicate a mismatch.

This distinction was made by [Stepp, 1987b][4], who pointed out that a number of learning

[4] Stepp used the terms 'contains' semantics and 'is' semantics.

systems did not explicitly take the distinction into account, and therefore suffered from semantic ambiguity. GRAM explicitly allows both interpretations, and therefore allows the generaliser to perform either intersection or union when creating a concept. GRAM also distinguishes a number of more specialised variants of these two interpretations, which are used to define how group concepts and disjunctive concepts are to be interpreted by the matcher and generaliser.

1.3.8 The richness of the representation scheme can be exploited by the matcher and generaliser.

This section has shown that GRAM's representation scheme is sufficiently rich to enable the important features of complex generalised physical objects to be explicitly represented. The scheme allows explicit context and substructure, groups, optional and alternative features, instance-counts associated with features, relationships and multi-relationships with other concepts, and several alternative interpretations. This richness enables the matcher and generaliser to exploit the structure of physical objects to achieve more efficient and effective performance.

1.4 The Instance Constructor

The instance constructor takes information obtained from a low-level vision system (which is currently simulated manually by input from a drawing program) and produces a description in GRAM's representation language. The information from the vision system specifies a set of blocks described in some visual coordinate frame. Some of these may be 'fuzzy' blocks which are approximations of several smaller blocks.

More specifically, the instance constructor must construct an object graph, at multiple levels of detail. One issue is to determine what objects should be created, where each object should be a useful abstraction or approximation of other smaller objects. Most of these can be formed directly from the blocks provided by the vision system, but some objects may be formed on the basis of other object-formation criteria, such as groupness, connectivity, symmetry, *etc*. One contribution of this thesis is to identify the kinds of criteria that justify object-formation with respect to the requirements of the matcher and generaliser. Another contribution is a set of criteria for selecting which parent, neighbour, and subpart relationships should be explicitly included in the object graph.

A more significant contribution of the instance constructor is the set of 'groupness' criteria for justifying group formation in an instance. A mechanism for searching for groups has also been developed, called *seed-expansion*, which first identifies seed groups consisting of two objects that could potentially expand into a group, and then incrementally adds new objects to the group until a clear group-boundary is reached, or until the group is abandoned. This method contrasts with another method described called *propose-and-prune*, which begins with a generous grouping, and then prunes off members until a stable group with a clear member-nonmember boundary is reached.

The work on group-finding is based on Winston's early group-finding system [Winston, 1975]. Winston seemed to use a seed-expansion approach to find sequences of similar and similarly related objects, although the algorithm was not described, and seemed to deal only with simple chains. His system also used a propose-and-prune algorithm to find groups of objects that were similarly related to some other object. The seed-expansion algorithm presented in this thesis extends Winston's work in several ways: It integrates the capabilities of Winston's two group-finding techniques; it provides a more elaborate scheme for proposing an initial seed groupings; it measures group strength and member typicality in a more general manner; and it is applicable to GRAM's richer representation.

1.5 The Matcher

The purpose of the matcher is to compare two descriptions, usually a concept description with an instance description. This involves finding and evaluating correspondences between the parent, neighbour, and subpart relationships of the two descriptions, producing various measures of similarity that indicate overall similarity, structure similarity, context similarity, and the similarity of each pair of corresponding relationships and related concepts or objects.

In an earlier version of GRAM when concepts were represented as a complete part-graph, the matcher had to find the best set of one-to-one correspondences between the parts of the concept part-graph and the parts of the instance part-graph. One approach to this is to exhaustively evaluate all possible correspondences between the components of the two descriptions. Such an approach did not use the description structure to guide or constrain the search, and was therefore computationally expensive, and even infeasible for large objects such as the bicycles in Figures 1.8 and 1.9. The part-graph for *bike-1*, shown in Figure 1.10, has about 120 composite and primitive parts, so there are roughly 120! sets of correspondences between the parts of the two bicycles. In fact this is an underestimate since it does not include correspondence sets in which some parts are left unmatched. Also, it does not consider axis-correspondences: A pair of parts may correspond in a number of ways depending on which axes are put into correspondence. Two rectangular parts can be corresponded in 4 ways (or 8 if reflection is considered). So, clearly, an exhaustive search is computationally infeasible.

Similarly, a purely top-down search that traverses subpart relationships of the two descriptions (as in the systems by Brooks [Brooks, 1981] and Wasserman [Wasserman, 1985]) is also inadequate because we cannot assume canonical part decomposition hierarchies, and because we may also want to compare the *context* of the two items. A more flexible method is necessary which is able to search upwards and outwards via parent and neighbour relationships as well as downwards via subpart relationships.

Therefore, another method for part-graph matching was developed. In this method [Andreae, 1993] the matcher first chose a seed correspondence between two parts. The best corresponding parent, neighbour, and subpart relationships were then used to propose new correspondences to

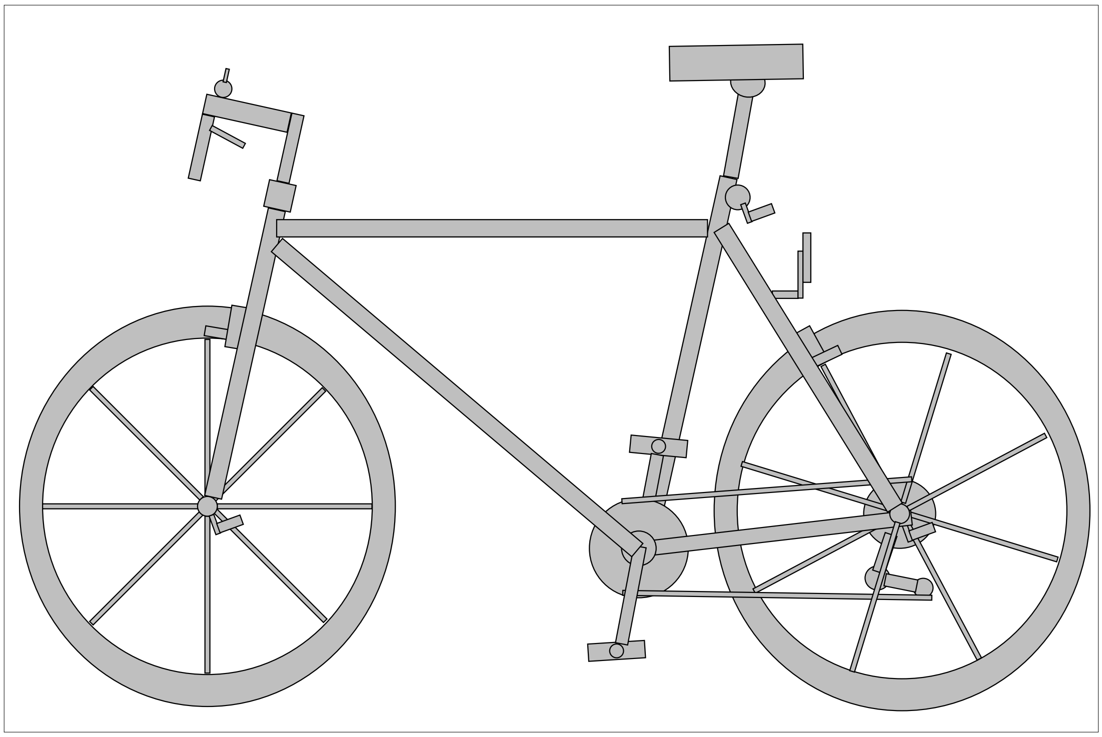


Figure 1.8: A Bicycle

be added to a pool of candidate correspondences. Then the best candidate correspondence was selected, in the manner of a "greedy" search, where 'best' meant not only having similar properties, but also most consistent with the previously selected correspondence. Then more candidate correspondences were proposed on the basis of the best corresponding parent, neighbour, and subpart relationships. This "spreading activation" process continued until the (hopefully) best globally consistent set of one-to-one correspondence bindings between all parts of the two graphs was found.

Although the basic algorithm worked efficiently and effectively for straightforward cases, extending the algorithm to cope with more complex situations proved difficult.

One aspect of the problem was that it was often necessary to compare significant portions of the substructures of two parts before they could confidently be put into correspondence, or before ambiguity between two competing correspondences could be resolved. This required that the matcher be invoked recursively to compare the two substructures, with their contextual correspondences kept 'hidden', or at least made unchangeable, during the scope of the recursive match. This meant that a new subgraph comparison description had to be created, and then later integrated into the original graph comparison description if appropriate. The recursive match sometimes needed to make use of previously-selected correspondences between surrounding objects, and cope with any previously-selected correspondences of the subparts of the subgraphs. Since there could be many levels of recursive matching, and hence nested graph comparison descriptions, the maintenance of consistency, and the integration of

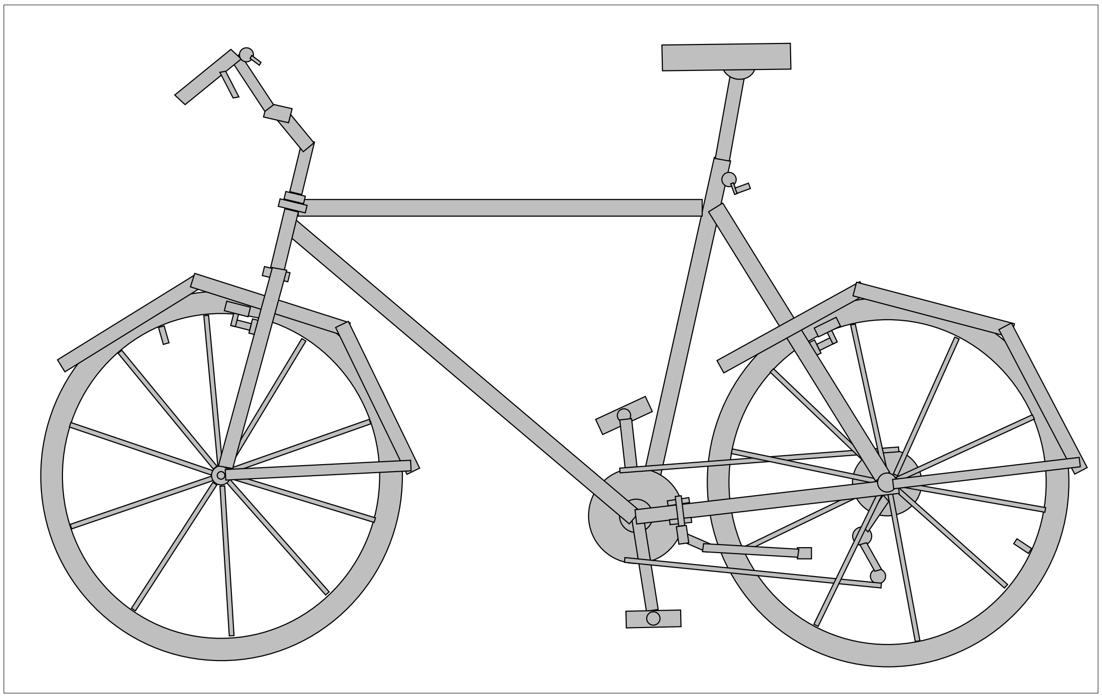


Figure 1.9: Another Bicycle

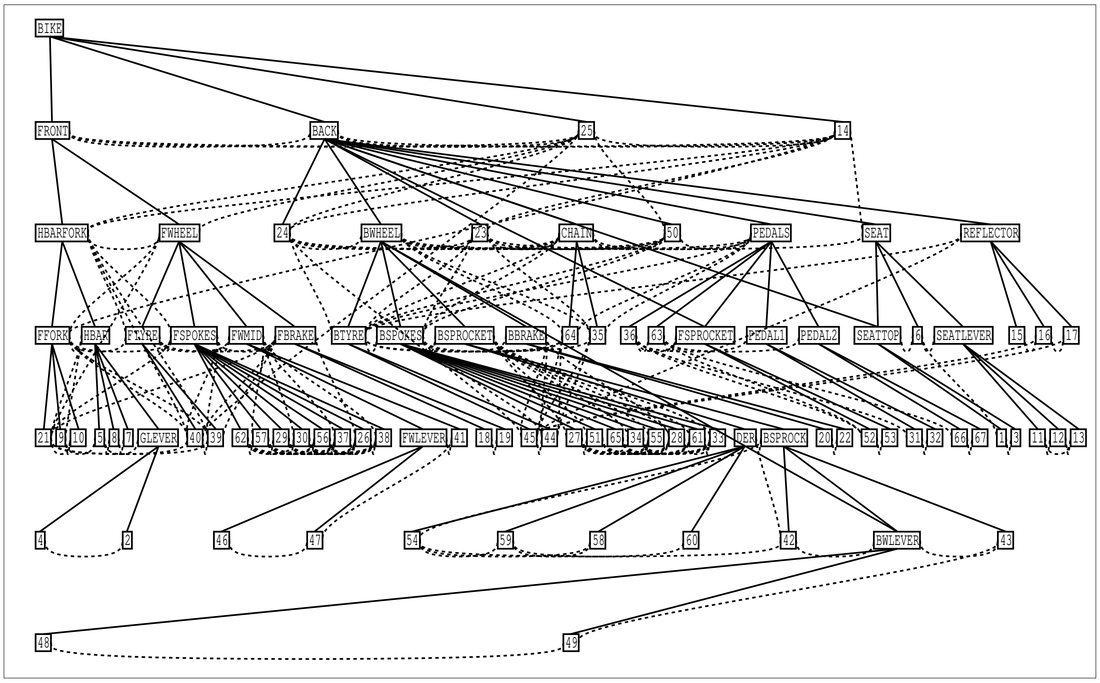


Figure 1.10: The object-graph for a bicycle

the nested comparisons, was complex.

Another problem was that a correspondence might be selected and then later found to be a bad choice, and this required some form of backtracking. The system had to determine which correspondences to unselect, and when. One difficulty was that for backtracking to be effective it was necessary to undo a *set* of correspondences, rather than just one at a time, since otherwise the matcher would tend to get stuck on local maxima of the hill-climbing search space. The system also needed to avoid cycles.

Another problem was that the representation scheme included a special kind of relationship that referred to another concept, rather than to another part within the same hierarchy. Therefore, the matcher had to deal with both kinds of relationship, and be able to recursively apply the matcher to that other concept and a subgraph of the instance, while also maintaining a globally consistent set of bindings, thus adding another level of complexity. Also, structure and context disjunctions of a concept were represented as sets of alternative substructure descriptions, and each of these had to be matched with the structure and context of the instance, in the manner of a recursive match. The problem of inheritance of complex structures, from superconcept descriptions, also needed to be addressed.

Although consistent one-to-one bindings were required, it was required that the matcher notice and record ambiguities so that groups could be created during the generalisation process. This meant that the matcher needed to maintain multiple alternative subgraph comparisons. If a group was already present in a concept description, but not in the instance description (or vice versa), then it was necessary to maintain multiple alternative subgraph comparisons between the typical member of the group, and individual parts of the instance (or vice versa).

Although a matcher for dealing with these issues could no doubt be developed, and many of the individual issues have been dealt with in other systems, the complexity and cumbersomeness of the process indicated that perhaps a different concept representation scheme was necessary. Also, the earlier scheme did not take into account the fact that most of the components of most physical objects are themselves instances of concepts, such as the buttons on a telephone, the zip of a pencil case, and the cushion on a chair. The matcher did not exploit or even account for this fact, since it was designed primarily for concepts that were represented as a complete part hierarchy.

The new representation scheme takes the complete opposite approach, representing every concept *only* in terms of other concepts, with no local part hierarchy. This means that a much simpler matcher is possible.

1.5.1 The GRAM matcher does not maintain or enforce a set of consistent correspondence bindings.

The key idea in the new GRAM matcher is that there is no requirement that a consistent set of correspondence bindings be created and maintained. This is a somewhat radical approach, but has proved surprisingly successful. Consistency seems to be implicitly maintained by the rich constraints inherent in physical objects themselves.

To determine how well a concept matches an instance, the correspondences between the concept's parent, neighbour, and subpart relationships, and the instance's parent, neighbour, and subpart relationships, are evaluated. A set of reasonable candidate correspondences are chosen, and these are then evaluated more thoroughly by recursively applying the matcher to compare the concept and instance to which each pair of relationships refers. The best correspondences are then chosen, and similarity scores for these, and similarity scores for the properties of the concept and instance, are used to compute overall measures of similarity. Similarity scores are computed using weighted averages of numerical measures of similarity of the attributes that characterise the properties and relationships.

Thus, the matcher employs a kind of backward-chaining manner, similar to the instantiation of a Prolog Horn clause, since to evaluate a concept-instance comparison, other concept-instance comparisons must be evaluated. Therefore, the matcher spreads up, out, and down through the parent, neighbour, and subpart relationships, matching instances with concepts as it goes.

The main difference between this matcher and the old matcher is that each concept-instance comparison is performed without taking into account the selection of best relationship correspondences for any other concept-instance comparisons. Only the similarity scores of other correspondences are used. There is no notion of "fixing" a correspondence, since all correspondence selections are made only locally.

This approach works surprisingly well, and has made the matcher more simple, flexible, robust, and efficient, and it can be made significantly more efficient because it is more amenable to a parallel implementation. Inconsistency amongst correspondences is not a problem, because a good classification only requires that the parent, neighbour, and subpart relationships and relatees of an instance match sufficiently well with the parent, neighbour, and subpart relationships and relatees of the concept. Consistency is implicitly enforced by the richness of the domain, since inconsistencies will tend only to occur when there really are ambiguities, in which case we *want* the matcher to produce multiple correspondences, since these can be used by the generaliser to suggest the formation of groups or multiple 'roles' in the new concept.

1.5.2 A breadth-first beam search with iterative-deepening is used.

To ensure search efficiency, the matcher applies successively increasing levels of effort to the comparison, thus preventing the matcher from spreading outwards except via the most promising correspondences. This is done by a kind of breadth-first search using iterative-deepening, pruning off poor branches as it goes. If rough and rapid matching is required, then only a low-effort comparison is necessary. If a thorough detailed comparison is required, then a high-effort comparison is performed. Also, if the match is clearly a bad match, then the comparison may be abandoned early, before it has invested much effort.

1.5.3 Neighbour relationships largely resolve the "level-hopping" problem.

An important contribution of GRAM is the use of *neighbour relationships* to explicitly characterise the structural relationship between connected, close, or interestingly related objects. One of the reasons why this is significant is that it enables the matcher to "cross levels" of the object decomposition hierarchy. Other systems such as Brooks [Brooks, 1981] and Wasserman [Wasserman, 1985] have suffered from the "level hopping" problem, since their matchers worked by a top-down traversal of the part-hierarchy. Parts in two descriptions that are not on corresponding levels, could not be matched except by using some kind of additional level-hopping techniques. In GRAM, however, the matcher can traverse in any direction through the object graph, not only up and down parent and subpart links but also along neighbour links. Thus, even if two objects are on different levels of their decomposition hierarchies, neighbour relationships from other objects will often allow their correspondence to be found.

1.5.4 Classification of a scene can begin at any seed correspondence

The multi-directional spread also means that the matcher can begin from any 'seed' correspondence, not just from the top node of an instance hierarchy. For example, suppose an observed object is partially occluded, but the system is able to index from one observed component to the concept 'bicycle-seat'. When that component is matched with the concept 'bicycle-seat', the matcher spreads outwards via parent, neighbour, and subpart relationships, and hence is able to make the prediction that the object is a bicycle. In the process, many of the surrounding components (such as the frame, wheels, gears, etc) are also classified.

The above characteristic also helps resolve the problem of not being able to identify the boundaries between objects. For example, if the task is to find a hammer in a jumbled toolbox, but the vision system cannot identify the hammer as a distinct object but *is* able to identify the hammer head as a distinct part, then the matcher could spread outwards to establish a complete match with the hammer concept.

The ability of the matcher to traverse neighbour relationships also means that the *context* of a concept and instance can be matched, and this is important for classification of concepts which are defined (at least partially) in terms of context, such as a door handle or bicycle wheel.

1.5.5 Two types of similarity scores are distinguished: Fit-scores and Proximity-scores

The results of the matcher are used in different ways by different components of the system. A consequence of this is that two different kinds of similarity scores have been distinguished: *proximity-scores* and *fit-scores*. Proximity-scores measure the absolute similarity of two objects within object-space, and fit-scores measure the typicality of an instance with respect to a concept, where typicality is measured on the basis of the ratio of feature-differences to the variance of the concept features.

Proximity-scores are used within the matcher itself for evaluating object correspondences, and within the generaliser to determine whether two objects are similar enough to justify generalisation. Fit-scores are used by the generaliser to determine whether an instance fits a concept well enough to justify modifying that concept to cover the instance, rather than creating a new concept. Fit-scores are also required for fault-finding to identify the faulty or unusual features.

1.6 The Generaliser

The generaliser is responsible for generalising an existing concept description to include an observed instance. It can either produce a new concept, or modify the existing one.

In its most basic form, the generaliser is quite straightforward, since most of the important work is performed by the matcher. The basic method for generalising two descriptions is to (a) generalise their properties, (b) generalise their best-corresponding relationships, and (c) generalise their best-corresponding relatees, by recursively applying the generaliser to each pair of relatees. Thus the generaliser spreads outwards like the matcher, but only via the best-correspondences of parents, neighbours, and subparts that were found by the matcher.

The complexities of the generaliser lie in the areas of spread-control, disjunct generalisation, ambiguity resolution, special cases that arise due to the different kinds of structure and context interpretation, and deciding when to create a new concept, or modify or copy an existing concept.

One problem of generalisation in a structured domain is that a classification may have been obtained on the basis of a low-effort comparison that only takes into account the coarser details of the concept and instance. This may be sufficient for recognition, but insufficient for determining what generalisation action should be performed. Therefore, the generaliser needs to be able to request the matcher to apply further effort to the comparison.

Another problem is determining whether generalisation should be performed before or after matching has completed. If a robot is navigating through a room it will be continually classifying what it observes, and each classification will lead to other classifications via parent, neighbour, and subpart relationships. So in effect there is no notion of 'completing a match'. Concept generalisation would have to occur concurrently with matching, which means that the system's control strategy must be very flexible and robust. In the current GRAM system I have made the simplifying assumption that a single scene or object is observed, then a seed classification is provided for one instance object, and then the remaining instance objects are matched via the spreading comparison process. Generalisation is performed only after this process has completed. However, the nature of GRAM's matcher is such that it allows a great deal of flexibility in the way it can be integrated with other components of the system. This is because each concept is just a simple description, rather than an entire complex part graph, and, more importantly, because it does not enforce global consistency during the search.

1.7 GRAM in a larger system.

The current version of GRAM only provides mechanisms for matching and generalising. This section outlines the issues that need to be addressed for extending GRAM to perform full classification and multiple-concept learning.

1.7.1 Classification

The kind of task that the classifier is responsible for is shown in Figure 1.11. On the right is an object graph for an observation, with solid lines indicating parent relationships and subpart relationships, and dotted lines indicating neighbour relationships. On the left is an illustration of concept memory, in which the thick lines are AKO links, the thin lines are subpart relationships and parent relationships, and the thin dotted lines are neighbour relationships. Similarity-links may also be included, which provide direct access between concepts that are similar. The task is to find correspondences (*i.e.* classifications) between one, some, or all of the observed objects, and the concepts in memory.

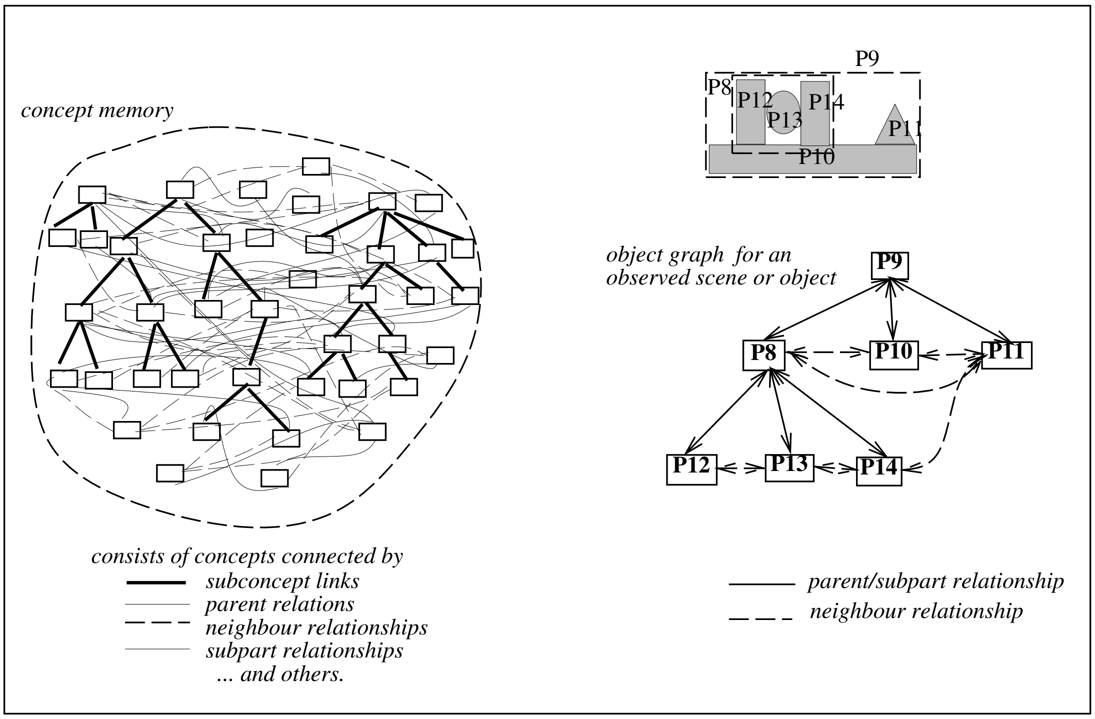


Figure 1.11: The classification task.

There are three stages in classifying an observed object. Firstly, one or more concepts in memory must be *accessed*. Secondly, each concept must be *matched* with the observation to determine the similarity. Thirdly, a *search* may be required to find a more appropriate classification, either via AKO-links to find a more abstract concept, or subconcept links to find a more specialised concept, or similarity-links to find an improved match.

There are three ways of accessing a concept to match with an observed object:

From task specifications. If the larger task requires an observed object to be matched with a specified concept, then concept access is not an issue. Such a task may be given by a teacher or by some other component of the robot system.

By direct indexing. A particular feature or combination of features of an observed object may be used to directly access one or more concepts in memory. The concepts found are then passed to the matcher for comparison with the instance. If too many concepts are found, then multiple indexing can be used, and only the concepts that are accessed by all or most of the indexes are considered. For example, if we index an object on the basis of its colour alone we may obtain many concepts, but if we take the intersection of those concepts and the concepts obtained by indexing on size or shape, then we might obtain a smaller set of hypothesised classifications.

Via the classification of a related object, during matching. When GRAM's matcher compares two descriptions, it usually spreads upwards, outwards, and downwards, comparing their parents, neighbours, and subparts. In other words, correspondences (*i.e.* classifications) may be obtained on the basis of some other comparison. For example, the comparison of an observed object with the concept chair may lead (via neighbour-relationship correspondence) to the classification of an object on top of the chair as being a 'person'.

Currently GRAM does not include a direct-indexing mechanism, and therefore relies on the first or the third of the access-mechanisms above. Although direct indexing would obviously be essential for a full system, the concept access that results from the 'spreading comparison' process is often sufficient for classifying many objects in a scene. For example, when you walk into an office, knowing that it is an office, many of the contents are often recognisable based on expectation, such as the desk, windows, telephone, chair, filing cabinet, *etc*. In other words, if there is a relationship from one concept to another, then an instance of the latter will often be classifiable on the basis of recognising an instance of the former.

Future work on the GRAM system will address the issues of *indexing*, and also of *searching* for better classifications via AKO links and similarity-links between concepts.

1.7.2 Multiple Concept Learning.

Concept learning is the overall process of building up an organised memory of concepts on the basis of observed scenes and objects. Although this thesis focuses only on the *generaliser* component of this, this section outlines some of the tasks and issues of the concept learning system to show how the generaliser fits in to it, and also to indicate the kinds of future work to be undertaken. Other systems such as Labyrinth have addressed concept learning (referred to as *concept formation* in the Labyrinth work), but without the richness of representation language, in particular the use of context relationships, groups, and disjuncts.

Figure 1.12 illustrates the kind of situation that faces the concept-learner. It is the same diagram that was shown earlier in Figure 1.11, except that some results of classification are shown by the dotted lines from each instance part to concepts in memory. The number on each line is a similarity score, and thus only the high-scoring correspondences are good

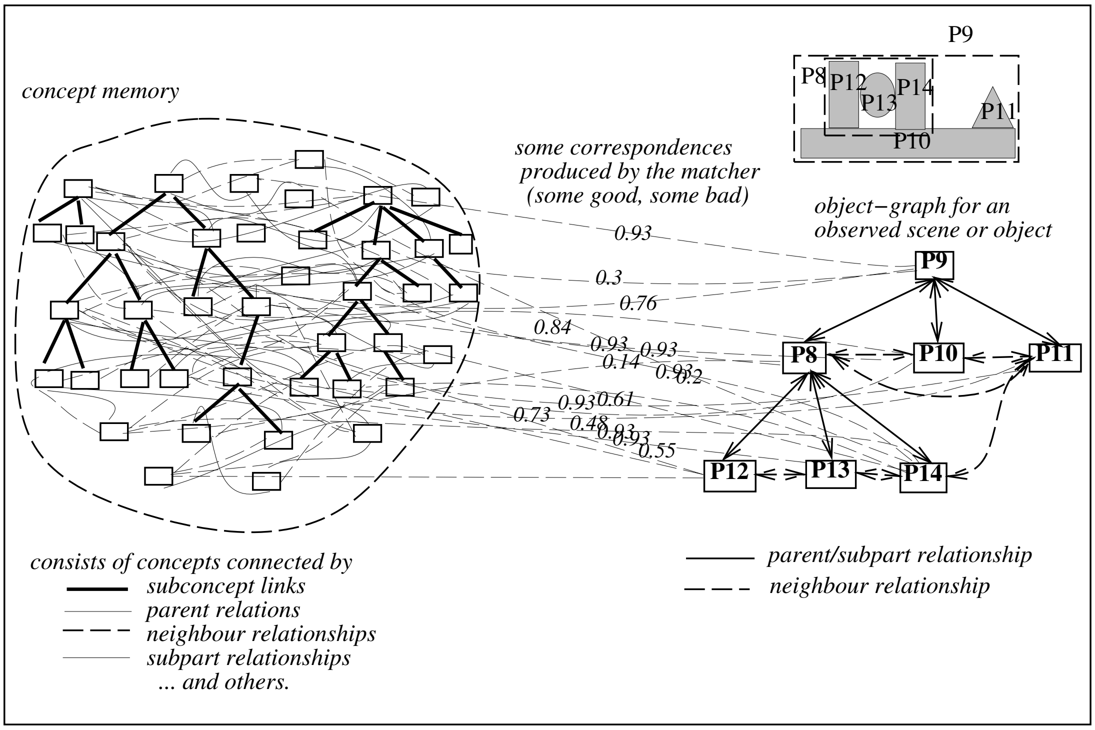


Figure 1.12: The learning task.

classifications. Each classification could either have been obtained via direct indexing, or via the spreading activation matching process, or from direct teacher or task specifications.

The concept learner must decide which of the classifications justify generalisation. If an instance fits an existing concept sufficiently well (according to the *fit-score*), then that concept could be generalised to cover that instance. On the other hand, if the instance is similar to the concept, but does not have a good fit-score (such as swivel chair with respect to the concept *four-legged-chair*) then a new parent concept could be created which has the original concept and the new instance as its two subconcepts. Various other reorganisations of the AKO hierarchy are also possible, such as merging existing concepts, adding new subconcepts, removing a no-longer-useful concept, or removing a concept and promoting its subconcepts. The COBWEB system [Fisher, 1987a] performed this kind of process in domains of unstructured objects.

In a structured domain, the creation and maintenance of an AKO hierarchy is somewhat more complex than in an unstructured domain. This is because concepts are defined in terms of other concepts, and so each change in one concept may affect many other concepts. For example, a concept may need to be removed from concept-memory if it becomes overly general, overly specialised, or otherwise not useful, but it cannot simply be deleted, because the parent, neighbour, and subpart relationships of other concepts may refer to it. Similarly, if a concept is overgeneralised, this has the effect of overgeneralising all other concepts that are defined in terms of it. Therefore, a conservative approach to generalisation is especially important in such a domain. A basic heuristic is that when in doubt, create a new concept rather than generalising an existing one. However, this can have the consequence of a complex cluttered memory. Also, under-generalisation can mean that less is learned, since the information obtained from each instance will tend not to be combined with information from other instances.

Although concept-memory must primarily be organised as an AKO hierarchy, it is useful to maintain *similarity-links* (or *difference-links*) between some pairs of concepts. This can enable the classification and concept learning systems to traverse these links to find better correspondences, on the basis of correspondences found via indexing or the matcher's spreading activation. Since each comparison is represented as a match description, these descriptions can be used as the similarity links, thus not only providing direct access between similar concepts, but also specifying the way in which they are similar.

Designing a concept learning system for a domain of structured objects is non-trivial. The "Background" chapter looks at a few systems that have addressed this problem to some degree. The Labyrinth system was the first system to address this problem using a representation where concepts were defined in terms of other concepts, and it proved not very successful in a number of ways. The enriched representation scheme of GRAM addresses some of the main limitations of Labyrinth, in particular the lack of context information, and also the inflexible matching algorithm. However, it remains for future work to develop a complete concept learning system on the basis of the work done for this thesis.

Chapter 2

Related Work

This chapter provides a background to the work in this thesis by discussing some of the most relevant systems that have already been developed for representing, matching, or generalising concepts in the domain of structured physical objects. Each section briefly describes the relevant features of the system, and then outlines the main limitations with respect to the requirements of the GRAM system discussed in chapter 1.

2.1 Winston's "Arch Learner"

Winston made the first significant attempt to represent and learn structured descriptions of physical objects. His learning system [Winston, 1975], which operates in a simple "blocks world" domain, introduced many important ideas about representing, matching, and generalising structured objects, and has been a motivational basis for much of the work on the GRAM system in this thesis (as well as for much of the other work in machine learning).

The system incrementally learns a concept by being shown individual instances by a teacher. The system does not address the problem of discovering and organising multiple concepts, but instead focuses on supervised learning of a single concept, from examples specifically chosen by a teacher.

Instances are represented using a semantic network in which a node can denote either a part of the object, or a qualitative predicate (such as *rectangular*, *small*, or *standing*). Nodes can have relations between then, such as *has-property*, *supported-by*, or *one-part-is*. A simple example of this is given in Figure 2.1.

A concept is initially formed from a single instance provided by a teacher, and is generalised by merging it with a new instance. To do this, the concept and the instance are matched to find corresponding nodes and relations, and then a comparison description is created for each correspondence. A variety of generalisation operations are available, and an appropriate one is selected for each correspondence, based on the nature of the similarity. These include

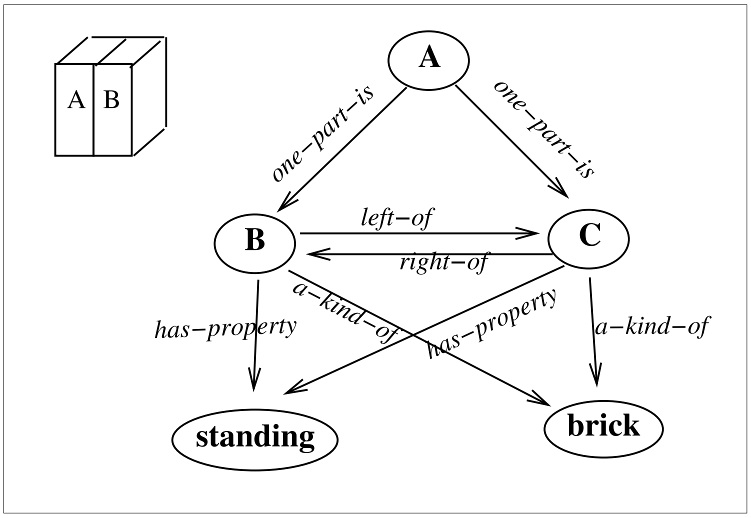


Figure 2.1: Object representation in Winston's system.

operations such as "climb the AKO hierarchy" to find a common generalised concept, "drop the feature" to remove the node or relation from the concept if it is not present in the instance, or "ignore the feature" to ignore an instance feature that is not present in the concept. An example of the use of the drop-feature operation is when a concept *television* includes an *aerial* component which is not present in a new instance. The aerial may be dropped from the concept description.

An important contribution of the system is the idea of using *near-miss* negative examples. If a near-miss negative example has a feature that is not present in the concept, then this feature is added to the concept, with the annotation "MUST-NOT-HAVE". Conversely, if a concept has a feature that is absent in the near-miss negative example, then that feature is given a "MUST-HAVE" annotation.

In the kind of domain that GRAM is to operate in, such as a house or a workshop, it seems unlikely that near-miss examples would be available. Instances of other similar concepts may serve as negative examples, or even near-miss examples, such as a stool being a near-miss for the concept *chair*. But it does not seem feasible to rely on a teacher to provide near-miss examples, perhaps by removing one leg of a chair, or misaligning the drawers of a desk. In fact, in these two examples the 'fault' does not cause the instance to be a non-member of the class, but simply indicates that a particular feature is important for the functionality of the object. A chair with a missing leg is still a chair, but it is a faulty or broken chair.

One problem with Winston's system is that the use of the drop-feature (or ignore-feature) operation implies that generalised descriptions must have a "contains" semantics, rather than an "is" semantics, as discussed by [Stepp, 1987b]. A "contains" semantics means that a concept description implicitly allows additional features that are not present in the concept description, to be present in an instance. An "is" semantics means that an instance must *not* have additional features that are not present in the concept. Therefore, Winston's system seems to necessarily have a "contains" semantics, since otherwise it would not allow an instance to have features that had been observed in previous positive instances and then ignored or dropped because they were not shared by all observations. However, this has the unsatisfactory consequence of allowing an instance to have *any* features in addition to those in the concept. For example, a chair and a person on it could, as a composite whole, be classified as a 'chair' simply because it contains all the necessary components of the chair concept.

Winston does not directly address this issue, although the use of MUST-NOT-HAVE annotations does help to constrain a concept to a small degree. However, this is not a practical way of enforcing an "is" semantics, since it requires that every non-allowable feature has to be explicitly included in the description. Also, each MUST-NOT-HAVE feature has to be specified by a teacher via near-miss negative examples.

The problem could be alleviated by not using the drop-feature or ignore-feature rule, and instead allowing each feature to have a frequency measure associated with it, to indicate how many observed instances had that feature present, and therefore to indicate the importance of that feature and the probability of it being present in a future valid instance. Even just the ability to annotate a feature as being *optional* would avoid the need for the drop-feature operation, therefore allowing a concept to have an "is" semantics, which in turn would make the need for negative examples less necessary, except perhaps for giving a especially strong emphasis to the required presence or absence of a particularly functionally significant feature.

Winston's system is not able to represent structural *disjuncts* (such as the back of a generalised chair having several alternative substructures) except perhaps by referring to another concept (such as *chair-back*) which has several subclasses. His paper does not address how such a concept hierarchy is formed, hence the only disjuncts supported are those involving predefined concept hierarchies (such as the *brick* class and its two subclasses *wedge* and *cube*).

A contribution of Winston's system which has particular significance to the GRAM system is that his representation allows groups of similar components (such as a tower of bricks) to be represented as a single entity. A group is characterised by a *typical-member*, which is a generalisation of the individual members, formed by extracting the features common to most or all of them. His representation of groups has been extended in several ways in the GRAM system, as have the algorithms for finding groups within an object. This is discussed in the "Representation" and "Instance Construction" chapters later in the thesis.

The issues involved in matching two descriptions, and the algorithm for doing it, are not discussed in his paper. The focus of the work was on identifying different kinds of similarity and the kinds of generalisation operation required for each of them. There is no mention of using the structure of descriptions to guide the matching process. In particular, although the representation includes a *one-part-is* relation, the system does not provide a way of representing large complex objects in multiple levels of detail, and of enabling the matcher to exploit the decompositional nature of physical objects.

Overall, Winston's system presents a number of important ideas, but only implemented in a simple way. The GRAM system described in this thesis shows how some of these ideas can be extended to cope with more complex objects, in particular those pertaining to group representation and group finding.

2.2 ACRONYM

Brooks's ACRONYM [Brooks, 1981] is not a concept learning system, but is a system for representing class-hierarchies of three-dimensional viewer-independent models of complex physical objects, and using these models to interpret and predict two-dimensional image data. A central contribution of the system is its geometric reasoning and constraint manipulation, which is not directly relevant to this thesis, although it might be applicable in future work in an extended GRAM.

One relevant aspect of ACRONYM for this discussion is its representation scheme, which, in many ways, is more elaborate than the GRAM representation, since ACRONYM is not constrained by the requirement that model descriptions should be *learnable*.

As in the case of many of the systems discussed in this chapter, ACRONYM represents multiple concepts in a hierarchy defined by the "subclass-of" relation, and represents individual concepts as a part hierarchy based on the "part-of" relation. Each part is represented as a generalised cone, which is the volume formed when a two dimensional planar cross-section is swept along a spine curve while being held at a constant angle to the spine and transformed according to a 'sweeping rule' (such as a change in dimensions). Each generalised cone is describable using just a few attribute values, and is therefore a simple but powerful way of representing complex objects with a wide variety of shapes. Future work on the GRAM system is likely to adopt this representation scheme, since it enriches the descriptions of objects considerably, without affecting the rest of the system at all. It only requires the addition of a few more attributes. However, it does also require a vision system which can produce generalised cone descriptions.

A feature of ACRONYM that is not present in the other systems in this chapter is that the properties and relations characterising parts can be described using conjunctions of complex algebraic expressions involving parameters of the parts (such as height, orientation, quantity, and so on). This means that models can be extremely complex, capturing almost any spatial constraint between its components. GRAM adopts a simpler, though not quite as general, approach.

Groups are representable in ACRONYM by specifying a value greater than 1 in the 'quantity' slot of an 'affixment' relation to a component. In fact, the value can be a free variable that is constrained by algebraic expressions involving other parameters of other components. As a simple example, a model could specify that there are n flanges and m motors, with the algebraic constraint that $n = (2 * m) + 3$.

A component that is optional can be represented by specifying its 'quantity' slot to be "0 or 1". Structural disjuncts are not supported, although it is possible to state that an object has either a flange or a base but not both, using an expression such as:

((flange-quantity = 0) and (base–quantity = 1))
or (flange-quantity = 1) and (base-quantity = 0))

or perhaps as: *flange-quantity = |(1-base-quantity)|*

However, the prediction and interpretation systems are not able to deal with such constraints. Such disjuncts could only be represented by referring to a special-purpose class which has two subclasses, a flange, and a base.

In addition to the representation scheme, another relevant aspect of ACRONYM is that it deals with the issue of comparing volumetric models with two-dimensional image data. Other systems in this chapter that deal with physical objects, assume that instance descriptions are in the same form as concept descriptions. That is, instance and concept descriptions are both described in part-based viewer-independent volumetric models, and can therefore be directly compared. Such systems assume that it is possible to form volumetric instance descriptions from an observed image prior to classification, using a vision system that can recognise primitive volumetric solids. Brooks argues that this may be impossible or very difficult, due to ambiguities or lack of information in the image description. ACRONYM is significant because it does not make the assumption that a volumetric model can be obtained from image data produced by low-level visual mechanisms, and instead provides a means to perform classification using two-dimensional image features. It does this by making predictions of invariant two-dimensional observable image features, computed from the three-dimensional viewer-independent concept description. These can be used to form rough hypotheses of model to image feature correspondences. These predictions can also act as instructions on how to use measurements of image features to deduce three dimensional information about the object to which it has been hypothetically matched.

ACRONYM's matching task is more complex and computationally expensive than for systems that assume volumetric instance descriptions. Therefore, it seems undesirable, and in fact unnecessary, to completely abandon the assumption that the volumetric instance descriptions can be obtained. Instead, a combination of a low-level volume-perception mechanism (such as PARVOS in section 2.9, a volumetric matcher, and an ACRONYM-like image prediction and interpretation mechanism, might be a more optimal approach. The volume-perception mechanism could produce volumetric descriptions to the extent possible by bottom-up techniques, and an ACRONYM-like system could produce classified volumetric descriptions of some of the other components using its more top-down expectation-driven mechanism. The results of these two systems could be used to classify other as-yet unclassified components, and the object as a whole, by direct volumetric matching. The volumetric matcher could be applied as soon as there are sufficient volumetric descriptions of subcomponents, produced by the other systems.

The main limitation of ACRONYM relative to the domain and task requirements of this thesis, is that it does not address the problem of *learning* object models, and the representation was not specifically designed to support learning. Models are instead input to the system manually via a graphical modelling system. Many features of an object are made implicit (via quantitative algebraic expressions) rather than as explicit qualitative descriptors. Also, it is not clear how these complex arbitrary algebraic expressions, and the free variables in them, could be learned from example objects.

2.3 CLUSTER/S

CLUSTER/S [Stepp and Michalski, 1986a] is a non-incremental unsupervised learning system that finds conceptual clusters of structured objects. It is the least relevant to the work in this thesis, but it is mentioned here because it is significant in the development of systems that deal with structured objects. The key idea presented is to convert a structured object description into an attribute-vector description so that attribute-based clustering mechanisms can be applied. CLUSTER/S uses this technique to enable the previously developed CLUSTER/2 system [Stepp, 1987a] to be applied in a structured domain.

CLUSTER/S represents objects using an annotated form of predicate calculus that was developed for the earlier INDUCE systems [Michalski, 1983], but with the addition of n-ary predicates. For example, the 'chair' object in Figure 2.2 could be represented in the following way:

```
chair(x) ==>  ∃ p1,p2,p3
          [part-of p1 X]  [part-of p2 X]
          [part-of p3 X]  [part-of p4 X]
          [color(p1)=brown]   [shape(p1)=rectangle]
          [color(p2)=brown]   [shape(p2)=rectangle]
          [color(p3)=black]   [shape(p3)=rectangle]
          [color(p4)=black]   [shape(p4)=rectangle]
          [orientation(p1)=vertical] [orientation(p2)=horizontal]
          [orientation(p3)=vertical] [orientation(p4)=vertical]
          [on(p1,p2)]
          [on(p2,p3)]
          [on(p2,p4)]
          [left-of(p3,p4)]
```

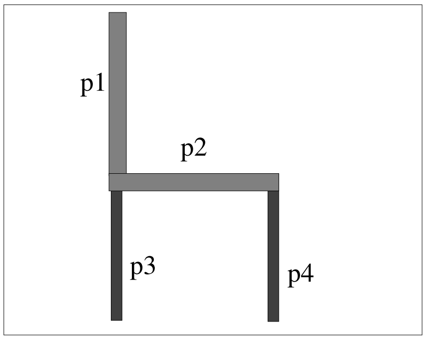


Figure 2.2:

CLUSTER/S converts the descriptions of a set of structured objects into attribute-based descriptions, which can then be clustered using CLUSTER/2. It does this by finding a 'core' description that is the common substructure of the set of objects. This common substructure enables the individual object descriptions to be converted to attribute-based descriptions, since the corresponding parts can be treated as 'named parts'. When clustering, there is no need to deal with structured object matching, since correspondences have already been found during this preliminary stage.

Since the clustering aspect of CLUSTER/S is not directly relevant to the focus of this thesis, the primary relevance of CLUSTER/S is its representation scheme, and the way in which objects are matched to produce the 'core' description.

One limitation of the representation is that it allows arbitrary predicates and attributes, which are not organised or partitioned in a way that enables the matcher to exploit the structure of the description. There is no explicit notion of representing objects in multiple levels of detail, and the matcher does not allow objects to be processed at a coarse level of detail before considering the finer details. In fact, the work on CLUSTER/S has not directly addressed the problem of dealing with large complex structured objects, but is instead based on a more general form of graph-matching, with some degree of pruning the search to prevent combinatorial explosion. However, it seems that it would be unreasonably computationally expensive for dealing with objects as complex as, for example, the bicycle in Figure 1.1 on page 2. The fact that it is non-incremental also makes it inappropriate in a domain where learning must occur incrementally in response to newly observed instances.

The representation scheme also does not support probabilistic information about the expected presence of the features of a concept and instead seems to employ the "drop-feature" generalisation operation. This means that generalisations have a "contains" semantics, which has the same consequences that were discussed earlier for Winston's system in section 2.1.

It seems that the central 'mistake' in the development of CLUSTER/S is that the problem of concept acquisition in the domain of structured objects was addressed without first considering the issue of what kind of representation would best support this task and domain. Instead, CLUSTER/S uses a representation scheme and clustering mechanisms that were originally developed for attribute-based domains, and it is not clear that these mechanisms are applicable to structured domains, since they do not explicitly exploit the structure of descriptions.

2.4 MARVIN

MARVIN [Sammut and Banerji, 1986] is a semi-supervised concept-learning system in which concepts are described in terms of other concepts, and which can deal with structured objects.

Concepts are represented as Horn Clauses, such as the example below for the chair in Figure 2.2 given earlier:

```
CHAIR(X) :- ∃ P1,P2,P3,P4,Y:
        (PARTOF P1 X) & (PARTOF P2 X)
      & (PARTOF P3 X) & (PARTOF P4 X)
      & (VERTICAL P1)   & (HORIZONTAL P2)
      & (VERTICAL P3)  & (VERTICAL P4)
      & (ON P1 P2)  & (ON P2 P3)  & (ON P2 P4)
      & (CHAIRBACK P1)  & (CHAIRSEAT P2)
      & (ROD P3)  & (ROD P4)
      & (ON X Y)  & (FLOOR Y)
```

The main idea proposed is that of iteratively learning a concept by automatically constructing instances to test the validity of, and to refine, the concept. A concept is initially created from a single instance, which is then generalised by performing a 'replacement' operation. This involves replacing one or more of the predicates on the right hand side of the concept (or clause) being learned, with the left hand side of some other previously acquired clause.

For example, BRICK(A) might be replaced by ANY-SHAPE(A) by using the clause:

ANYSHAPE(A):-BRICK(A).

Similarly, the set of predicates

BRICK(X) & BRICK(Y) & BRICK(Z) &
ON(Z,X) & ON(Z,Y) & SEPARATE(X,Y)

might be replaced by a single previously-acquired predicate ARCH(X,Y,Z).

An instance is then constructed which satisfies the new concept but does not satisfy the previous concept prior to the replacement. A teacher is asked whether the instance is valid, and if so, then the generalisation is presumed to be acceptable. If not, then the concept must be specialised by performing a further replacement which involves some of the predicates previously removed. The new concept is then generalised again, and the above steps are repeated. This process continues for all possible replacement operations.

This is similar to Winston's "near-miss" training in the sense that it involves making use of instances that are 'almost correct', and refining the concept accordingly. However, Winston relied on near-miss examples to be provided by the teacher, while MARVIN constructs these itself. A teacher is only required to verify these instance.

Although the papers describing MARVIN deal with examples in the domain of physical objects, such as 'arches', MARVIN was not developed particularly for structured objects, and therefore (as for CLUSTER/S) the system does not exploit the structured nature of the concepts it learns. All predicates are treated equivalently, and the structure of objects is not explicitly reflected in the structure of their descriptions. For example, there is no distinction made between contextual and substructural components.

MARVIN uses an unusual method of representing groups of similar components, by recursion. For example, the concept 'column' would be defined by the following clauses:

COLUMN(X) :- BRICK(X) & STANDING(X) & ON(X,Y) & COLUMN(Y).
COLUMN(X) :- BRICK(X) & STANDING(X) & ON(X,Y) & GROUND(Y).

This definition means that an individual brick is an instance of the concept 'column'. This does not seem to be a 'natural' way of representing groupings, and does not make the grouping explicit, nor allow predicates on the group as a whole to be made explicit.

MARVIN learns from a single observed instance, which is then generalised via domain knowledge and teacher feedback. It does not provide any method of incorporating newly observed instances into the concept description, and this would prevent it from being applicable to an autonomous robot that must learn and refine concepts on the basis of unsupervised observation of new instances. It is not possible for such a robot to generate trial instances for validation, except perhaps by pulling the legs of chairs or bending television aerials. It could perhaps ask verbal questions, such as "can the seat of a chair be elliptical?", or perhaps draw pictures of a test instances, but primarily it must learn from positive examples, or from discriminating between examples of similar concepts. In the domain of complex physical objects there are so many possible generalisations that could be made, and so many possible trial instances that could be proposed, that such an approach would be infeasible, unless there was sufficient reasoning ability or domain knowledge to identify the important questions or test instances.

It is not clear how well MARVIN would perform on more complex examples in a real physical domain. The system does not make use of the decompositional structure of objects to constrain and guide the classification and generalisation processes, but seems to take a more exhaustive approach, in the manner of a PROLOG interpreter. It also does not address the issue of partial structural matching, or of dealing with alternative partitioning of instances.

Despite the limitation of MARVIN for the domain of structured objects, a central idea which has also been adopted in GRAM, is that concepts are defined in terms of other concepts, rather than in terms of a complete set of components that are organised as a part graph (as is the case in Winston's learning system, Brooks's ACRONYM, and Connell&Brady's system (described later)). This means that concept memory is essentially a learned description language consisting of clauses that can be used within other clauses. A concept description does not consist of a global set of all of its components (structural and contextual), but only a local set of its directly related components. The matcher only has to establish classifications of its components, locally, rather than finding a consistent set of one-to-one correspondences between two graphs.

2.5 MERGE

Wasserman's MERGE performs incremental concept acquisition and organisation for objects that have a hierarchical structure, such as physical objects or corporate management structures.

MERGE represents an instance as a hierarchy of components, where the hierarchy is defined by some *fundamental relation* such as "part-of" or "has-boss". Such a hierarchy is called an *F-tree*. For the rest of this discussion we will assume an F-tree is a part hierarchy organised according to a *part-of* relation.

F-trees are stored in memory in generalisation hierarchies called G-trees. Each node of a G-tree is an F-tree, and the nodes below it are its subclasses, or *variants*. Since each variant F-tree is similar to its parent F-tree, MERGE avoids the need to store identical information redundantly, by supporting inheritance in several forms: a variant can be defined by specifying parts that are to be added, deleted, or substituted. All other parts are implicitly inherited from the parent F-tree, and so do not need to be explicitly included in the description of the variant, unless they are necessary for indicating the branch on which a lower-level part is to be added, deleted, or substituted.

Every kind of part has its own G-tree, which is essentially a hierarchical clusterings of instances of that kind of part. To illustrate this, figure 2.3 shows two chairs which have been represented in memory as a number of G-trees. The notation is different from that used in Wasserman's thesis, and has been chosen to enable a better comparison with the GRAM system. Each node of a G-tree is an F-tree, although it is defined in the diagram by just specifying its subparts, which are F-trees in other G-tree. The diagram only shows information relevant to this discussion.

For example, the *chair* G-tree consists of a generalisation of both chairs, at the top, and two variants beneath it. The subparts of the generalised chair are *chairback*, *chairseat*, and *chairsupport-#*, each of which is an F-tree in another G-tree. For example, *chairsupport-#* is the top node of the *chairsupport* G-tree, and has two variants, one of which is the two-legged support, and the other is the central-leg support.

The variants of the generalised chair inherit the details of their parent, although the generalised *chairsupport-#* is substituted with *chairsupport-1* or *chairsupport-2*. The hierarchy could extend to any number of levels if more instances were observed.

The representation also allows various properties to be associated with each F-tree node, and also allows *non-fundamental* relations between components, such as "left-of" or "bigger-than".

Although this representation scheme seems reasonable, there is a significant assumption made by the system which makes the task addressed by MERGE significantly different from that of GRAM. This assumption is that when a new instance is observed, and is to be incorporated into memory, each part of the instance (and the instance itself) has already been classified as belonging to a particular G-tree, by virtue of its name. For example, if a third chair is observed which is identical to the second chair, the names of its parts, such as *chairback-3*, *chairseat-3*, or *chairsupport-3*, immediately enable the system to determine which G-tree each part should be incorporated into. The only task required by the system is to determine which node of the

G-tree should be generalised, or where the new instance should be added as a variant. This is done basically by searching the G-tree, matching each F-tree with the instance F-tree to find the best correspondence.

Two F-trees are matched by applying the MATCH procedure to the root nodes of the F-trees. MATCH then recursively matches all pairings of the subparts of the nodes, to find the best set of bindings. The similarity of leaf nodes is based on two measures, firstly a measure indicating the hierarchical distance from the two nodes to their lowest common ancestor in their G-tree, and secondly, a measure of the similarity of their properties and relations. The match process is strongly constrained because such parts are only compared if they belong in the same G-tree, and this is immediately known by their name.

MERGE addresses the "level-hopping" problem, in which similar objects are represented with corresponding components on different levels of their F-trees, and thus cannot be matched using a strict top-down traversal of the hierarchy. MERGE deals with this by inserting "null nodes" into the hierarchy to test all possible adjustments of the hierarchies (by one level) in the hope that a better match may be found. This scheme does not cope with level hops of more than one level difference, and the strategy seems computationally expensive.

The assumption of named parts means that MERGE is not really able to classify objects on the basis of their structure, other than identifying which node of a given G-tree an object best corresponds to. The whole search is constrained dramatically by the pre-classification of parts by their names. This assumption was presumably made because MERGE was based on Lebowitz's work on reading patent abstracts for complex physical objects, where the parts of the objects are identifiable by their names.

Therefore, MERGE does not support the task which GRAM is addressing, which is to be able to match complex objects by comparing their unlabelled substructure and context.

It is not clear how much of the MERGE system could be extended to cope with unlabelled objects. It would require searching through all G-trees in memory, and G-trees are not organised or indexed in any manner which allows this. Also, the exhaustive recursive matcher would be unacceptable for complex objects, especially if the pre-classification assumption does not hold.

One feature of MERGE that has also been used in GRAM is the idea of representing *all* components of objects as concepts. The advantage of this is that it enables each concept to be defined directly in terms of other concepts, rather than part hierarchies, and this results in a simpler and more homogeneous representation scheme, without having to distinguish between concepts and parts of concepts, or having to decide when to extract out portions of substructure as concepts in their own right.

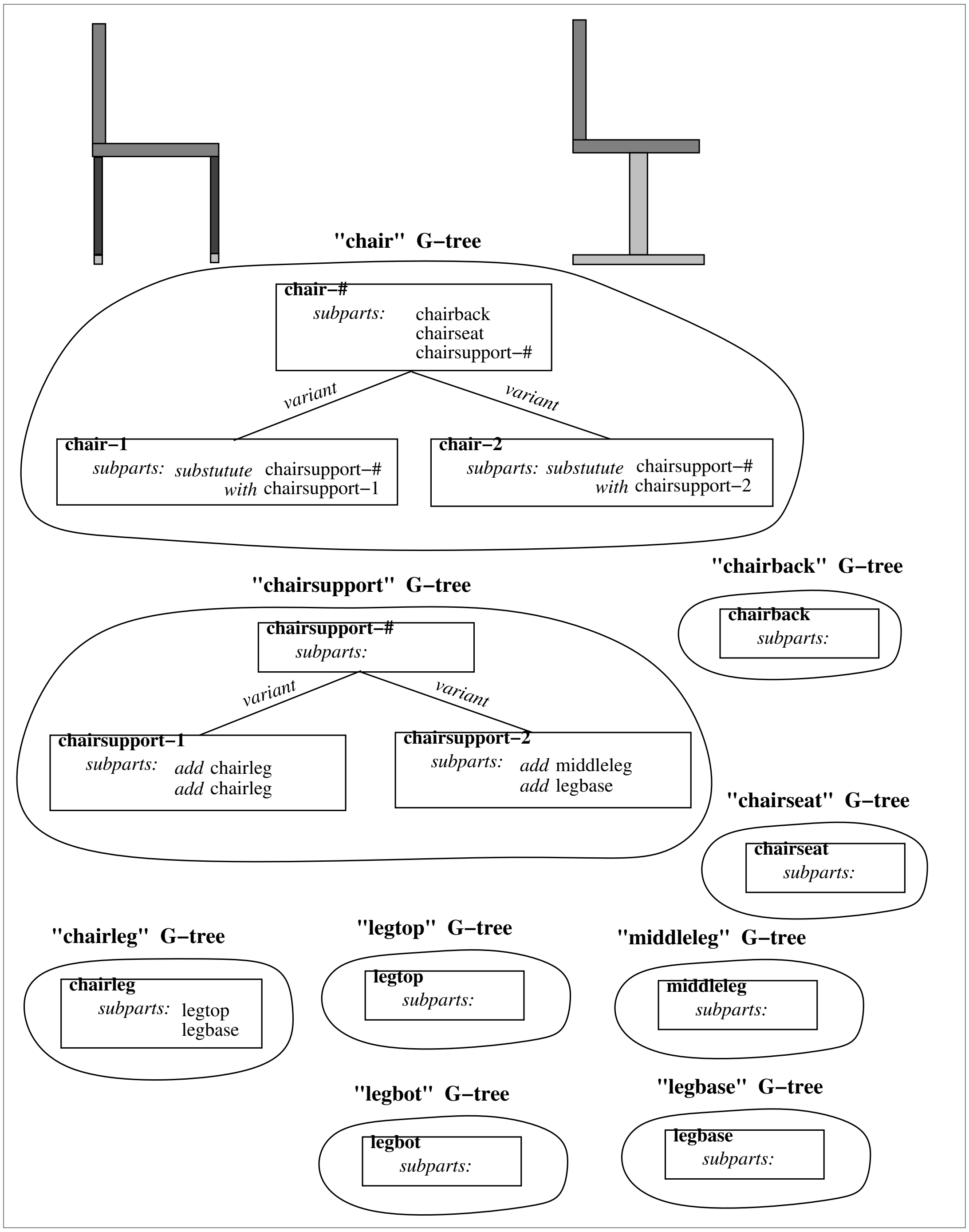


Figure 2.3: Wasserman's representation scheme.

2.6 Noddy

NODDY [Andreae, 1985] is the system of which GRAM was originally intended to be a component. NODDY incrementally learns robot procedures from examples, by matching the steps of example traces of a procedure, and generalising the existing procedure description by forming loops, conditional branches, and simple expressions of variables. Since the actions of a general-purpose robot system should be able to be conditional on visual input, GRAM was to be a subsystem that could build generalised descriptions of visual observations, which could then be used in the conditional statements of a generalised procedure.

In many ways, the procedure learning task is a similar to the physical object learning task, in the sense that both involve representing, matching, and generalising structured descriptions.

For example, loop formation is similar to group-finding, since a loop is formed to summarise a repetition of actions, and a group is formed to summarise a repetition of objects. However, NODDY only forms loops *during* the matching process to resolve ambiguity, while GRAM is also able to form groups *prior* to matching, thus avoiding ambiguity and improving the efficiency of the matcher.

Another similarity between the two domains is that procedures and physical objects can both be described in multiple levels of abstraction and detail (although NODDY deals only with 'flat' procedures), and procedures can have relationships with other procedures (*i.e.* procedure calls). NODDY deals with components described by numerical properties, but has a much simpler set of properties than GRAM.

One important difference between the two domains is that there is less complexity in the structure of procedures, since procedures are sequential, with the relationships between components basically limited to 'follows' and 'precedes'. The sequential ordering guides the matcher in a way that is not possible for physical objects.

NODDY's matcher begins by finding a few 'seed' correspondences between distinct steps of the current generalised procedure and steps of the example procedure trace (such as the first and last), and then propagates linearly from these.

2.7 Connell and Brady

Connell and Brady's system [Connell and Brady, 1985] was built with similar goals to GRAM. It learns descriptions of two dimensional objects, and was intended to form part of a "mechanic's mate" project which would assist a mechanic in various ways, such as finding a desired category of tool.

Instance descriptions are obtained from a low-level vision mechanism called the Smoothed Local Symmetries program [Brady, 1983] which breaks a two-dimensional image into segments. Such a system could also be used to provide input to the GRAM system, which assumes that a vision system is available.

Connell and Brady's system represents the results of the segmentation by a semantic net, in which nodes denote object components (such as parts, part-ends, and part-sides) and edges denote relations or properties (such as 'has', 'join', 'very long', *etc*). The network spreads outwards from a 'root' node via several kinds of relations, with larger and coarser details close to the root node, and smaller and finer details further away.

An important aspect of the representation is the use of *Gray Coding*, in which each property is described by a set of predicates, rather than a single predicate. Each predicate adds a distinction to the property, such as the predicate 'very' which can be combined with the predicate 'long'. This means that similarity can be measured simply by counting how many predicates are shared, rather than requiring hidden domain-specific similarity metrics. This helps to support Connell & Brady's requirement that semantic similarity is directly reflected in 'syntactic' similarity.

Gray Coding enables a single generalisation mechanism to be used, which simply 'drops' predicates that are not common to both descriptions. However, this does require that predicates be chosen carefully to ensure that the methods of matching and generalisation do in fact give appropriate results.

The representation does not support optional components, or disjunctive sets of alternative substructures. The only form of disjunction provided is at the level of the whole concept, which can be described in terms of a disjunctive set of *models* and *non-models*. Another limitation of the representation is that it cannot explicitly represent groupings of similar components.

The matcher works by spreading outwards from the root node of the network, extending the "match horizon" outwards via relations if the current similarity is sufficiently good. The system keeps track of the best bindings between nodes of the two descriptions being compared, and performs backtracking if necessary. Differences that are more distant from the root node are treated as being less important than differences that are close to the root node. This is based on the assumption that larger and coarser details are semantically (or functionally) more important than smaller and finer details. If this assumption is valid, then the 'syntax' of the semantic network reflects its semantics, as required.

This matcher is more similar to the GRAM matcher than any of the other systems discussed in this chapter. One difference is that it deals with concepts and instances that are complete graphs, and therefore must maintain a globally consistent set of bindings between nodes. This means that backtracking is necessary, since it is not possible to always correctly determine the best correspondences of nodes at the horizon of the match without considering details that only become available later as the matcher spreads outwards. The details of the algorithm were not specified in the paper, and so it is not clear how effective the backtracking mechanism is, since in theory it would require a complex ATMS-like system to correctly determine which correspondences (or sets of correspondences) should be unselected. [Provan, 1987] has shown that an ATMS approach is infeasible for large objects, and therefore a backtracking scheme that relies only on partial information would seem to be necessary. Also, it is not clear how well Connell&Brady's backtracking scheme would perform in situations where ambiguities are only resolvable by descriptive features that are several levels beyond the current horizon of the match.

A second difference is that the matcher does not address the level-hopping problem, and instead assumes that two objects can only be considered identical or very similar if they have been partitioned in the same way, with corresponding components being at the same level in the hierarchy.

The generaliser involves the single technique of 'ablation' in which common features are retained in the generalised description, while unshared features are dropped. If two descriptions differ significantly, then a new model can be added to the concept. Models may later be replaced by a single model if more intermediate instances are observed. A concept may also include *non-models* which are like Winston's *censors* [Winston, 1984], and achieve an effect similar to Winston's 'MUST-NOT' conditions. If an instance matches a model but also matches a non-model, then it is considered to fail the match. A non-model may also be overridden by more models, which themselves may be overridden by more non-models.

One problem with this scheme is, as has been the case for several of the systems in this chapter, that concept descriptions necessarily have a "contains" semantics, since the drop-feature generalisation operation is employed. To a small degree this is alleviated by the use of non-models, as for Winston's MUST-NOT conditions, but these non-models also suffer from the same problem. Also, the use of non-models is an expensive way to constrain a description, since they are themselves complete descriptions of a negative example, and must be matched and generalised independently.

2.8 Labyrinth and COBWEB

The Labyrinth system [Thompson and Langley, 1991] is an incremental unsupervised concept learning system for the domain of structured objects. It deals specifically with the issue of acquiring and organising multiple concepts in memory.

Labyrinth is built upon the COBWEB mechanism [Fisher, 1987a] for incorporating instances into a concept hierarchy in a way that maximises the measures of 'utility' for the concepts. The key contribution of Labyrinth is that it extends the COBWEB mechanism to deal with structured objects, rather than merely attribute vectors.

Instances are represented as a part decomposition hierarchy, in which each part in the hierarchy is linked to its subparts via the 'part-of' relation. The topmost node of the hierarchy represents the observed object as a whole, and the leaf nodes represent primitive parts that have no further decomposition. Each part is also characterised by an attribute-vector specifying information such as shape and colour. The subparts of each part can have arbitrary relations between them, such as *left-of*, or *connected-to*.

Labyrinth does not make the assumption of MERGE that the parts of an instance are already partially pre-classified by being implicitly associated with a concept hierarchy for a particular kind of object. Rather, Labyrinth stores all concepts in memory in a single concept hierarchy, and the system itself must classify the instance parts with no prior information. The concept hierarchy acts as a kind of indexing structure, since it allows the classifier to traverse the hierarchy top-down, following the branches on which the concepts best match the instance, thus obtaining a successively more specialised classification.

Concepts are represented in a manner similar to instances, except that a concept's part-hierarchy is only one level deep. The subcomponents of the concept are not decomposed further within the concept description itself. Instead, each subcomponent is defined as being an instance of some other concept. For example, a bedroom might be represented in terms of three subcomponents, x, y, and z, which are defined to be instances of the concepts 'chair', 'bed', and 'desk', respectively. This avoids the problem of dealing with concepts that are large complex part hierarchies or part graphs, since the substructure of each concept is 'hidden' in the descriptions of other concepts. This particular aspect of Labyrinth is basically the same as for MARVIN and MERGE, and is also the scheme used by GRAM.

The representation scheme supports a more precise form of feature prediction than the other systems described in this chapter, in the sense that attributes, relations, and subcomponents of a concept have probabilities associated with them, indicating how frequently they have occurred in the observed instances of the concept.

Figure 2.4[1] illustrates the kind of concept memory that might exist after observing two chairs. Each node is a concept, some of which are defined in terms of subparts, with occurrence probabilities indicated. The properties and relations between subparts have not been shown.

[1] This example was not generated by Labyrinth, but was created by hand for this chapter, and is only intended to convey the general idea of the Labyrinth system, rather than giving precise details of exactly how the two chairs would be represented in concept memory.

The names on each concept have been given only for readability, since instance parts are not given concept names (in contrast with MERGE).

An instance is classified by first classifying its subpart objects. These in turn are classified by classifying their subparts. This recursive process bottoms-out at the leaf nodes, which have no substructure, and so can be classified by the attribute-based COBWEB system. This involves searching the concept hierarchy, top-down, to find the best-matching concept.

After the leaf subparts have been classified, their parent parts at the next level up the hierarchy can be classified, since their subparts are now labelled. This is done by using a modified COBWEB algorithm that can deal with sets of subparts, each described by an attribute vector specifying its properties, and by relationships with other subparts. (The previous COBWEB only dealt with instances and concepts represented as a single attribute-vector). Each label is represented as a property of a subpart. The modified COBWEB traverses the concept hierarchy top-down, comparing concepts with the instance by comparing their overall properties, the properties of their subparts (including the *label* property), and relations between the subparts.

The technique of labelling subparts, to simplify the classification task, is similar in principal to the technique used by CLUSTER/S (described in section 2.3) to convert a structured description into a non-structured description, or, in the case of Labyrinth, to a minimally structured description.

The classifying process continues back up the instance part hierarchy until the root part (*i.e.* the object as a whole) has been classified. Hence classification of an instance involves a depth-first divide-and-conquer technique, breaking up the overall classification problem into a series of simpler classifications, one for each subtree of the instance part hierarchy. During the classification process, concept memory is also updated by generalising existing concepts, creating new concepts, or reorganising concept memory.

The modified COBWEB mechanism compares a concept and an instance by an exhaustive search of all possible sets of bindings between their labelled subcomponents. The label (*i.e.* the classification) of each subcomponent contributes to the measures of similarity. This differs from MERGE, which uses labels to directly constrain which subcomponent bindings to consider.

One significant limitation of Labyrinth (which was to be addressed in future work) is that a concept is only defined in terms of its substructure, and does not include any context, or 'role', information. This means that there is less information in the concept descriptions to distinguish between concepts, and also means that predictions about context cannot be made. Over-generalisation tends to occur, since a concept can be generalised to cover a new instance even if their contexts are very different.

An even more significant limitation is that Labyrinths' classification scheme *relies* on not having context information. If concepts were defined in terms of contextual parent and neighbouring parts, as well as subparts, then it could not classify an instance in its bottom-up fashion, since to classify the primitive 'leaf' parts of an instance, the system would have to classify its parents and neighbouring parts. It seems that the whole strategy used by Labyrinth would have to be modified considerably to cope with this.

Another important limitation of the Labyrinth system is that *every* component of an instance is classified by an independent COBWEB search through the concept hierarchy. Since new concepts are formed for every distinct kind of component of every observed object, the concept hierarchy would be enormous for any real-world system (especially if context were also used, since that would create even more distinctions between instances). Labyrinth does not provide any indexing mechanisms for directly accessing concepts from instance features, and so it is not clear that the scheme would be sufficiently efficient for rapid classification.

An important form of direct indexing, which is the basis of the GRAM matcher, is not employed in Labyrinth at all. This method is to use a hypothesised classification of one instance part to directly suggest likely classifications of its structurally related parts. For example, if a concept C1 is being matched with instance I1, and concept C1 has subparts which must be instances of concepts C2, C3, and C4, then the subparts of I1 can be directly compared with those concepts, without having to search concept memory. Those comparisons can then provide positive or negative confirmation of the C1:I1 correspondence, which may have been originally proposed on the basis of its properties, or relations with other already-matched concepts and instances. This process is discussed in detail later in this thesis. It could perhaps be incorporated into the Labyrinth system, although the manner in which Labyrinth incorporates instances into memory is somewhat dependent on a top-down hierarchical search. In any case, the process would not possible via relations between connected or nearby parts, since context is not representable. For example, if a chair-leg is roughly recognised as a chair-leg (perhaps on the basis of it being a tall thin rod standing on the floor), then Labyrinth could not use this classification to lead directly to the classification of the chair-seat as being a chair-seat. Rather, the chair-seat must be classified independently, by searching down the concept hierarchy.

Also, Labyrinth's strategy of classifying in a post-order manner, beginning with the leaf nodes, means that matcher cannot perform classification using just a coarse level of detail. This could perhaps be achieved by simply ignoring the substructure of the instance parts below some depth in the hierarchy, and treating them as primitive components. However, this may have implications on other aspects of the system.

A related problem is that Labyrinth can only recognise sets of components that have already been partitioned into a distinct instance object. Partitioning cannot be driven by the classification process itself. For example, if a chair and a person on it are treated as a single object, then the classification mechanism cannot discover, on the basis of classification of the subcomponents, that the chair and person can usefully be treated as distinct objects.

Another problem with Labyrinth is that it does not address the level-hopping problem, where instances that should belong in the same concept have been decomposed into different part hierarchies. Labyrinth is not able to find correspondences between parts at different levels of the hierarchy, and therefore must assume canonical descriptions of all instances.

A representation issue that is not addressed in Labyrinth is that of representing explicit *groupings* of similar objects. This means that an object containing several similar subparts must specify all of the subparts individually, rather than in summary form. This has the added consequence that there will be ambiguities when matching two such objects, and this has not been addressed by the system. Also, if the objects have a different number of similar subcomponents, then it is not possible to find a set of one-to-one bindings to enable generalisation to be performed.

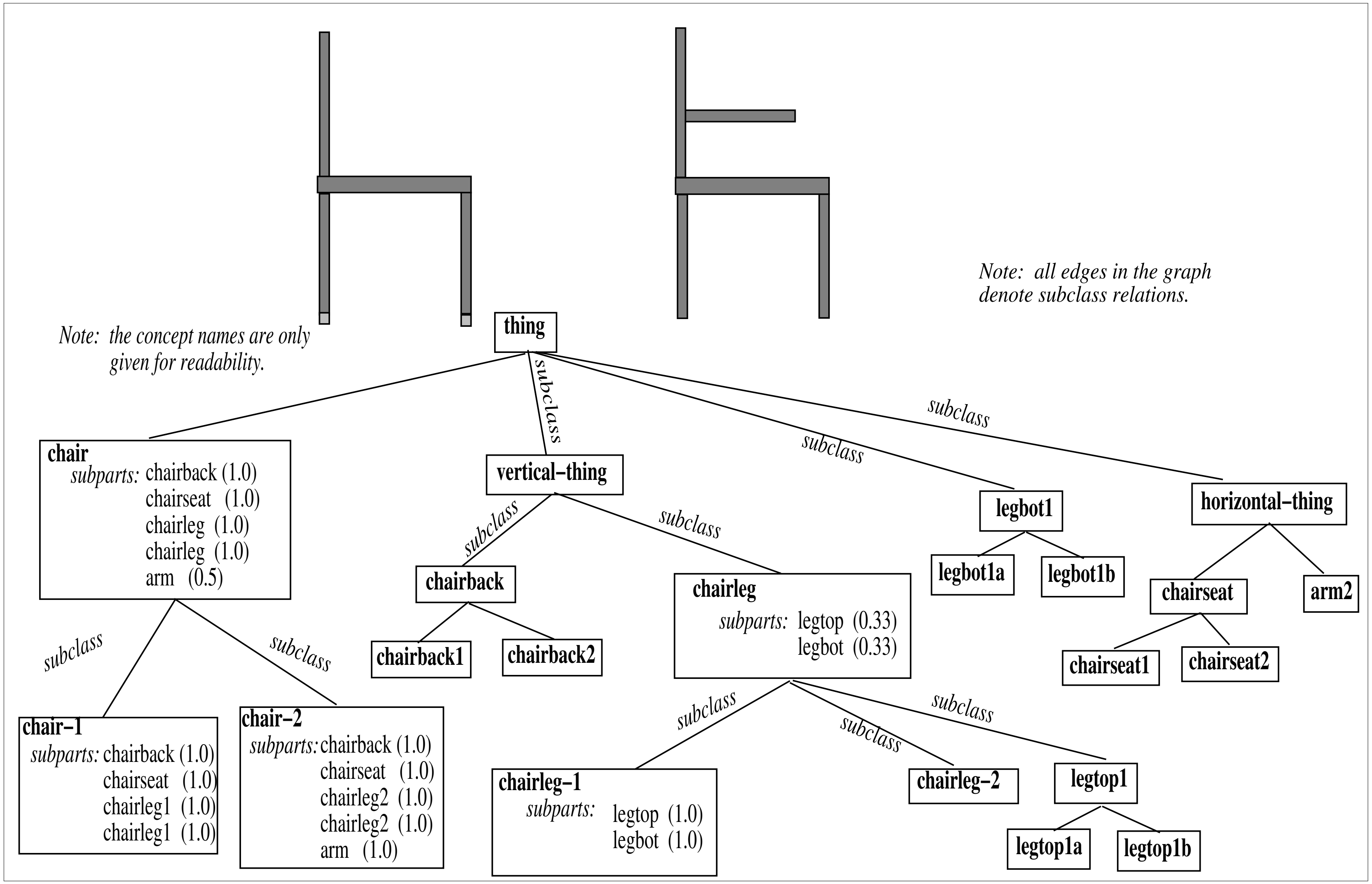


Figure 2.4: Labyrinth's representation scheme.

2.9 PARVO

The PARVO system [Bergevin and Levine, 1993] performs object recognition from two-dimensional line-drawings. Its approach is based on the idea proposed by [Biederman, 1985] that it is usually possible to classify physical objects from a relatively small number of components at a coarse level of detail. PARVO demonstrates this by correctly classifying a number of common man-made objects.

PARVO uses various feature extraction and segmentation techniques to produce a set of parts that characterise an observed object at a coarse level of detail. Each part is associated with a *geon*, which is a particular kind of simple volumetric shape, such as *cone*, *cylinder*, *banana-shape*, *pyramid*, and so on. Parts are represented in a graph, where each edge of the graph denotes a physical connection, of which there are several generic kinds. The system then matches the instance graph against concept graphs (*i.e.* graphs of generic object models), to find the best classification.

In order to reduce the number of concepts considered, PARVO prunes the search space by finding, for each part in the instance graph, the set of concepts that contain a part of the same geon type. The intersection of the sets of concepts obtained for all of the instance parts, is the set of concepts that are then matched more thoroughly. This seems to be a robust and efficient way of indexing directly from instances to concepts. However, the 'intersection' requirement implies that concept descriptions always contain a superset of the components of observed instances. This would not be the case if the system were dealing with incremental concept learning, and an instance contains a feature not present in previously observed instances.

To match an instance graph with a concept graph, an exhaustive search of all sets of 'compatible' part bindings is performed. A compatible part binding is one for which both parts are of the same geon type, and have a similar aspect ratio. This constraint prunes the search considerably. For each set of bindings, PARVO computes a similarity score based on the properties and connection-types between the nodes of the graphs. The similarity score for the best binding set is used as the similarity score for the instance-to-concept comparison. The matcher can cope with missing or extraneous parts and connections, and is thus able to perform object recognition from partial views. This is possible because physical objects tend to be characterised by a very wide variety of combinations of geons, and therefore a subset of geons characterising an observed object is often sufficient to uniquely identify its generic class. This is an important characteristic of the domain of common man-made objects, and is specifically exploited by PARVO.

There are many issues of the domain and task requirements in chapter 1 that the PARVO system does not address. Most significantly, since it deals only with coarse levels of detail, it can be said to be based primarily on *discriminant* descriptions of object categories concepts, rather than *characteristic* descriptions, as distinguished by [Michalski, 1983]. In other words, category descriptions consist of sufficient information to discriminate between categories, but they do not actually provide a full description of the concept as a whole, in detail. This means that tasks such as fault-finding are not possible except at a rough level of detail. Objects in PARVO are not represented using multiple levels of detail, and object graphs are therefore 'flat', and do not capture the decompositional structure of the object.

This also means that PARVO does not have to address issues of matching complex structured objects, since it only deals with the topmost level of the part hierarchy.

The work on PARVO is not intended to deal with the problem of *learning* generic object models, and this is one reason why the representation and matching schemes are sufficient. In many ways, PARVO can be compared with Labyrinth, since it transforms a complex task into a simpler task by labelling the direct subcomponents of an object, so that a simple semi-exhaustive match becomes feasible. PARVO labels each component as being an instance of one of a predefined set of geon types, while Labyrinth labels each component as being an instance of one of its previously acquired concepts. Labyrinth therefore allows concepts to be richly described, in detail, but has also been found to be rather un-robust, due to over-generalisation of concepts, and the lack of context information to help identify part correspondences. PARVO seems to be more robust since geons are predefined, and are not in danger of being overgeneralised by a concept learning mechanism. Also, PARVO's indexing scheme is more robust than Labyrinth's, since the latter relies on the traversal of a concept hierarchy formed by the system itself, which involves comparisons for each node considered. Labyrinth could perhaps alternatively use an indexing scheme like PARVO's, where instead of finding intersections of sets of concepts that share each component's geon-type, it finds intersections of sets of concepts that share each component's *concept* classification. This would simply require that every concept description includes a reference to all other concepts that contained it.

PARVO does not deal with context information, or structural disjuncts, or occurrence-frequency information, and therefore is not able to make predictions about occluded instance features which depend on such information. However, PARVO's central and intended contribution is very important, since it demonstrates Bierderman's claim that effective and efficient recognition is possible on the basis of a small, simple description consisting of predefined generic shape-types. This is encouraging for GRAM since it suggests that the coarser levels of the description are likely to be correctly matched, without *requiring* a match of all of the finer details, and therefore the coarser match will be able to guide the matching of finer details.

The system also demonstrates that partial descriptions, obtained from partial views, are usually sufficient for classification, even without any kind of functional knowledge or reasoning abilities. Since three-dimensional objects are always partially occluded, this is an important property of the domain of common physical objects. Also, the mechanisms for producing volumetric part descriptions from two-dimensional images could be directly applicable to a system such as GRAM, which assumes the availability of such mechanism.

Chapter 3

Representation

The design of a good representation scheme is crucial to most machine learning systems. This is because the quality of the representation largely determines the possible performance of the other components of the system, such as the matcher and generaliser, and also determines which aspects of the domain can be explicitly described and processed. In other words, the representation scheme must support and be consistent with the characteristics of the tasks and domain.

Chapter 1 discussed the overall characteristics of GRAM's domain and tasks, and on the basis of this, a number of requirements of the representation scheme have been identified. Each of these is presented in section 3.1.

Sections 3.2 through 3.5 describe GRAM's representation scheme in detail. Section 3.2 describes how ungeneralised *instance* objects are represented; section 3.3 explains how properties and relationships are represented; section 3.4 describes how generalised *concepts* are represented; and section 3.5 discusses the issue of *group* representation.

As with the other chapters, the boldface section headings provide a summary-overview of the chapter. Section 3.6 gives a reference summary of the representation features discussed.

3.1 Requirements of the Representation.

This section examines various characteristics of the representation scheme that are required to support and be consistent with the characteristics of the domain and tasks discussed in chapter 1.

3.1.1 Structural descriptions should include functionally important information.

Section 1.1.4 explained that this thesis is only concerned with structural (rather than functional or behavioural) knowledge about objects. However, since concepts in the domain of physical objects are formed largely on the basis of common function, the structural description of an object must include information that is likely to be important to its function. For example, a functional description of a chair might state that a chair must be 'stable' and that it is used for people to sit in. A structural description should, therefore, at least be able to explicitly specify that the lengths of the legs of a chair are of the same length, and that its context may include a person in a particular posture.

Another way of stating this requirement is that the functional similarity of objects in the same category should be reflected or embodied in the measures of similarity of their structural 'syntactic' descriptions. If the representation is sufficiently rich, this will often be the case, since the function of an object is 'implemented' in its structure, and therefore objects that have similar function usually have similar structure.

3.1.2 The representation should support the performance of the matcher.

In a system that deals with large numbers of complex real-world objects, it is essential that the matcher be efficient, and this means that the representation scheme should capture the right kind of information to enable the matcher to be guided and constrained by the structure of the descriptions being matched.

The representation scheme should also enable the matcher to produce useful and meaningful comparison descriptions that can be used by the fault-finder and generaliser. In other words, the language should be expressive enough to enable the important features of similarity between two objects to be explicitly noticeable.

3.1.3 Objects must be describable at multiple levels of abstraction and approximation.

Physical objects need to be represented at multiple levels of detail, for a number of reasons, each of which is discussed below.

Coarse descriptions are often sufficient for recognition. Object recognition can often be effectively and efficiently performed by considering only the overall structure and properties of an object, ignoring finer details unless or until a more refined classification is required. For example, a bicycle can often be recognised on the basis of observing two round components with several bars connecting them. Therefore, the representation scheme should allow abstract and approximate descriptions to support this characteristic of the domain and task.

The matcher can be guided by the abstraction/approximation hierarchy. The task of matching two object descriptions can be achieved much more efficiently and effectively with multiple levels of detail because the process of finding correspondences between components can be guided by the decomposition hierarchy. Large coarse components can be matched first, and these correspondences can form the basis for matching finer details.

Abstractions and approximations may reduce storage requirements. The finer details of an object can often be summarised by a single abstract component, thus reducing storage requirements. For example, it may not be necessary to remember the details of the back of an observed chair since it could be summarised as a single rectangular 'block'. Similarly, it is not necessary to record details of every apple in a bowl, since the collection (or *grouping*) of apples can be represented as a single entity whose description refers to a generalised description of the 'typical' apple.

Coarse levels of abstraction and approximation may make similarity explicit, and generalisations possible. Two objects might be similar only at a coarse level of approximation or abstraction, and in order to match and generalise the two objects, abstract and approximate features must therefore be representable. For example, the details of two chair backs may differ considerably, but they may be very similar if they are viewed as single simple 'blocks'.

Coarse features must be representable because fine details may be unavailable. Sometimes the finer details of an object or scene are unavailable, and therefore it is necessary to be able to represent the observation at a coarse level. For example, an autonomous robot may only have time for a brief glance at an object, or the object may be too far away for fine details to be observed.

Both coarse and fine details are necessary for fault-finding. Although coarse levels of description may be sufficient for some tasks, finer details are usually necessary for fault-finding. For example, it is obviously impossible to notice that a stereo turntable is missing a stylus if the *turntable* concept is only described at a coarse level of detail.

Fine details may be necessary for discrimination during classification. An object category may have a number of sub-categories which differ only in their fine details, and therefore the representation must allow fine as well as coarse levels of descriptions.

3.1.4 The representation language should be richly expressive.

A *rich* representation language is one that allows an object to be explicitly described in a wide variety of ways, and allows redundancy. If a description contains a wide variety of information, then it is more likely that functionally significant information will be explicitly present, rather than merely implicit, and hence this information is less likely to be lost during generalisation. Also, redundancy (in the sense of having the same information specified in several ways) is desirable because it means that the common properties of two objects are more likely to be explicit in their descriptions, and so will not be lost during generalisation.

For example, if it is possible to specify that all four legs of a chair have the same length, then the descriptions of two chairs that have different leg lengths, can be produced simply by finding the common features of the descriptions, without losing the “same-length” constraint, and without having to perform more complex constructive induction.

A representation language can be enriched in several ways. Firstly, a wide variety of possibly-redundant descriptive entities and attributes should be provided to represent the many kinds of components, properties, and spatial relationships in the physical world, using values of various kinds, including numerical, symbolic, boolean, categorical, and directional. Both qualitative and quantitative values should be allowed.

Secondly, the representation should support a variety of schemes for describing an object, such as "Generalised Cylinders", "Boundary Descriptions", and "Constructive Solid Geometry" and groupings.[1]

Thirdly, the representation should allow a scene or object to be partitioned in a variety of ways, rather than just as a single decomposition hierarchy in which each object is a subpart of only one composite parent object. Composite objects should be able to overlap other composite objects, since each may capture an important kind of approximation, abstraction, or summary of its subparts.

3.1.5 The context of an object must be explicitly representable.

Chapter 1 discussed how some concepts may be defined largely on the basis of their *context* (or ‘role’) rather than their isolated *structure* (or ‘form’). Their context is directly related to their function, as in the case of the concepts *chair-seat* and *telephone-button*. Other concepts, such as *bicycle*, are primarily defined by their structure (or ‘form’), although their expected context may be useful to aid recognition (such as when recognising a bicycle being ridden on a road in the distance), and to enable the system to predict unobservable surroundings of an object.

3.1.6 Structure and context should be explicitly distinguishable to allow disjunctive concepts and partial matching.

Concepts that have a relatively invariant structure may have a highly variant context, such as *bicycle* or *scissors*. To describe such concepts it is useful to be able to specify a *disjunction* of context descriptions. Conversely, a concept such as *chair-back* has a relatively invariant

[1]The Generalised Cylinder scheme represents objects in terms of a central spine and a cross-section that sweeps along the spine according to some function. The Boundary Description scheme represents an object as a set of surfaces and/or edges. The Constructive Solid Geometry scheme represents an object as a set of subparts or subregions.

context, but a variety of structures, and so the structure should be representable disjunctively. To allow such disjunctive descriptions, the representation scheme must explicitly distinguish between structure and context.

The distinction between structure and context is also necessary to support an important kind of partial matching, where similarity is measured either with respect to structure alone, or context alone. Two objects that have high structural similarity may still be considered generalisable, even if their contexts are dissimilar, as in the case of two bicycles, one on the road, and one in a bicycle shop. Likewise, two objects that have the same contextual role, but with different structural form, could be generalised to create a 'role' concept.

3.1.7 Groups must be explicitly representable.

The physical world is full of repetitions of similar objects, and this is the basis for forming generalised concepts that support prediction-based activity within the world. However, in chapter 1 we also considered the fact that the physical world contains many *groups* of components that are not only similar to each other (as are instances of a concept) but are also structurally *related* to each other, and to other objects, in a similar way. In other words, the structural organisation of the group as a whole is regular. Examples of such groups include books on a bookshelf, chairs in a room, or petals on a flower. Since the primary purpose of a concept-learning system is to notice and make explicit the regularities in the world, it is clearly useful and necessary to explicitly represent groups of similar and similarly related objects.

A description of a group should include the following information: a set of properties characterising the group as a whole, a generalised description of its typical member, and a generalised description of the typical relationships between consecutive or neighbouring members.

There are a number of more specific reasons for representing groups, and these are discussed below.

A group makes an N-ary relationship explicit. An obvious requirement of a representation scheme is to allow relationships between pairs of objects. A group, however, captures the structural relationship between *several* objects. This is especially important for fault-finding in a situation where a multi-part constraint is functionally significant. For example, a grouping of the legs of a chair can capture the requirement that all of the legs be of the same length. This constraint might otherwise be lost through the generalisation process if groups could not be made explicit.

Grouping allows collective properties of several objects to be made explicit. A grouping is a way of forming an abstraction or approximation of several objects, and therefore makes it possible to explicitly specify collective properties which would otherwise be hidden, such as the overall shape and organisation of the set of objects. Thus grouping enriches the representation.

Groups enable more efficient matching. As discussed earlier, the formation of abstract or approximate components from more primitive components supports more efficient matching. Grouping is another form of this, since several objects are combined into a single summary approximation. This allows the matcher to match the group as a whole, simply by matching the overall group properties and the typical-member description, rather than trying to find correspondences between every individual member.

Groups may reduce memory usage. By summarising several similar objects as a single generalised typical-member, the individual members can often be dropped from the description, thus reducing storage requirements.

Groups enable the matcher to match two collections of objects that have different cardinalities. Groups enable two different-sized collections of objects to be matched and generalised, resulting in a description that explicitly allows a variable number of members. This is because the matcher need only compare the typical-member descriptions, rather than trying to match each possible pairing of members. For example, a desk with three drawers can be matched with a desk with five drawers by matching the groups, rather than getting "stuck" with ambiguous correspondences.

The typical-member concept transfers information amongst members. Since the typical-member description is a generalisation of the group members, the formation of a group is effectively generalising each member. The typical-member specifies that *any* member of the group can have *any* of the variations observed amongst the members. For example, in Figure 3.1 the formation of the group in object *A* results in a greater tolerance of variation in each member than if each member was described individually, thus enabling object *B* to be considered similar. The differences between the corresponding components of the *A* and *B* groups are considered less significant due to the transfer of information via the typical-member generalisation.

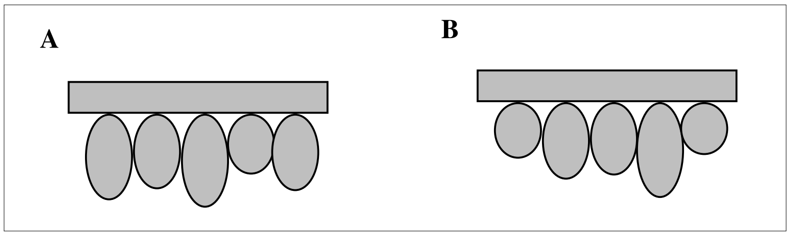


Figure 3.1: Transfer of information by group formation.

This is especially useful during instance-construction since the formation of a typical-member concept from just a few observed members can be used to predict the details of other members without having to look at more than a few details. For example, if we see a room full of chairs that look very similar at a coarse level of detail, we can form a typical-member concept by observing and generalising (in detail) just a few of the chairs. We can then infer that the remaining chairs in the group most likely share the same details, without having to observe them closely. In other words, the typical-member generalisation supports prediction not only of members of groups in future observations, but also within the present observation.

Group formation is concept discovery. The process of finding and forming a group is really a way of discovering a new concept from within a single observed scene (as opposed to forming a concept from several observations at different times). In other words, by generalising the members of a group, the resulting typical-member description is a new concept.

However, there is an important distinction between a group and a concept (or class), since a group specifically characterises a particular set (or collection) of several objects that are spatially related to each other, while a concept is simply a generalisation of several objects that are similar and may have been observed at different times and places. This distinction is discussed further in [Markman, 1979].

3.1.8 The representation should include descriptive entities and relations that humans seem to use.

This thesis does not address cognitive modelling, and therefore the representation scheme is not intended to model a human representation scheme. However, a general purpose robot must interact with humans, and must therefore be able to represent and learn concepts that were originally created by humans. The descriptive features of the representation scheme should correspond to descriptive features that humans seem to consider important.

3.1.9 Concept descriptions must be probabilistic.

Within a generalised description, the variability of the features should be explicitly representable to enable the matcher to make probabilistic predictions of the presence or absence of features. This is especially necessary for the fault-finding process, in which a missing feature of an instance of a concept is significant only if the feature has appeared in a majority of the previously observed instances. Since the system is not dealing with ‘functional’ knowledge, and since the occurrence of particular features (such as an aerial on top of a TV) is not necessarily directly dependent on other features of the object itself, the only way to capture the expectedness of a feature is by recording probabilities based on frequency counts.

3.1.10 Partial descriptions must be representable.

An observed object may be partially occluded and details of an object may not be available from the instance-constructor (perhaps until explicitly requested). Therefore, it must be possible to explicitly state that details are either unavailable, or available only on request, so that the matcher will not merely consider the features to be missing.

3.1.11 The representation must be extendible.

The representation should be designed so that important extensions (such as converting to three-dimensions) are possible without having to completely redesign the existing scheme, or redesign the matcher and generaliser. Some possible extensions to GRAM's current representation are considered during the chapter.

3.1.12 Description construction mechanisms must be available.

The representation scheme must take into account limitations of the robot eye, viewing operations, and the instance-construction process. Mechanisms must be available (or able to be developed) that can construct descriptions in the representation scheme, given the nature of the domain and task.

3.2 Instance representation.

This section describes the representation of an ungeneralised scene or object. This will be referred to as instance representation.

3.2.1 GRAM represents the physical world as an object-decomposition hierarchy.

One of the representation requirements was that physical objects should be represented as multiple levels of detail. GRAM achieves this by representing an observed scene or object as an object-decomposition hierarchy, where an *object* is a single component, piece, or 'chunk' of the observed physical world. It may either be a *composite* object comprising several subparts (each of which may themselves be composite objects), or a *primitive* object that has no further decomposition. Each composite object is an abstraction or approximation of its subparts.

This is illustrated in Figure 3.2, which shows a humanoid on the left, and the rectangular bounding boxes of its composite parts shown on the right. The left foot is a primitive object, while the left arm is a composite object, as is the humanoid as a whole. At the bottom of the figure is the object-decomposition hierarchy. GRAM also allows an object to have more than one parent, and thus the hierarchy is really an acyclic graph. This means that a scene or object can be partitioned in a variety of ways.

A variety of criteria are considered in the process of partitioning an object into a decomposition hierarchy, and these are discussed in detail in the "Instance Construction" chapter. The primary criterion for creating a composite object is that its set of subparts, when treated as a whole, has properties which capture a useful and hopefully 'functionally significant' *abstraction*, *approximation*, or *summary* of the subparts, and which clearly distinguish the object from its surroundings. Such properties may include shape, symmetry, repetition (*i.e.* grouping), topology, and category.

3.2.2 Neighbour relationships are necessary to capture the context of each part.

In the object-decomposition hierarchy for the humanoid in Figure 3.2, each object is only related to its parent(s) and its subparts. One of the requirements of the representation was that the *context* of each object should also be representable. Relationships with parent objects capture the context to some degree, but we also need to be able to specify how 'neighbouring' objects are structurally related to each other. For example, the description of a humanoid should include a description of how the head is related to the torso, and how the left leg is related to the right leg. Therefore, GRAM not only represents relationships with subpart and parent objects, but also with neighbouring objects. Thus, each object is not just a node of a hierarchy, but a node of an object-graph.

A 'neighbour' of an object is any other object which is interestingly related to it according to a variety of *neighbourliness* criteria, which are intended to ensure that 'functionally important' neighbour relationships are made explicit in a description. The criteria are discussed in detail in section 6.3.1 and include factors such as proximity, connectivity, and alignment.

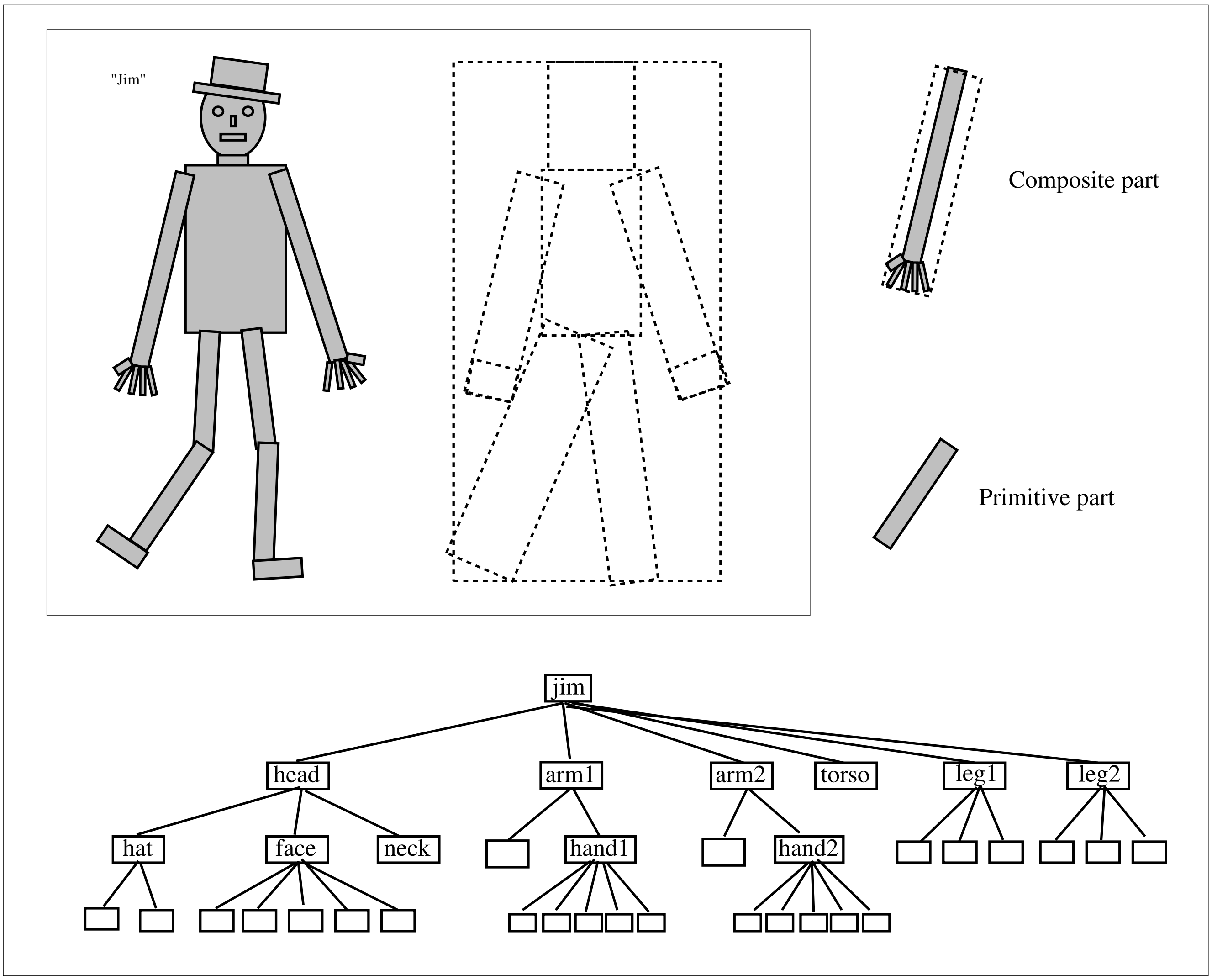


Figure 3.2: Composite and Primitive Parts

Figure 3.3 illustrates the selection of parents, subparts, and neighbours for several component objects of the humanoid “mary”. Each shaded box denotes the bounding box of a particular primitive or composite object. The boxes enclosed in it indicate its subpart objects; the finely-dotted box(es) surrounding it indicate its parent objects; and the coarsely dotted boxes indicate its neighbouring objects. Thus, the description of each object may be highly constrained since it may include relationships with numerous other objects.

3.2.3 Each relationship is a rich descriptive entity.

Most representation schemes for structured objects (such as [Wasserman, 1985],[Winston, 1975], and [Connell and Brady, 1985]) include a variety of relations such as *on-top-of*, *bigger-than*, *etc*, each of which is a single atomic predicate on two objects. GRAM takes a different approach

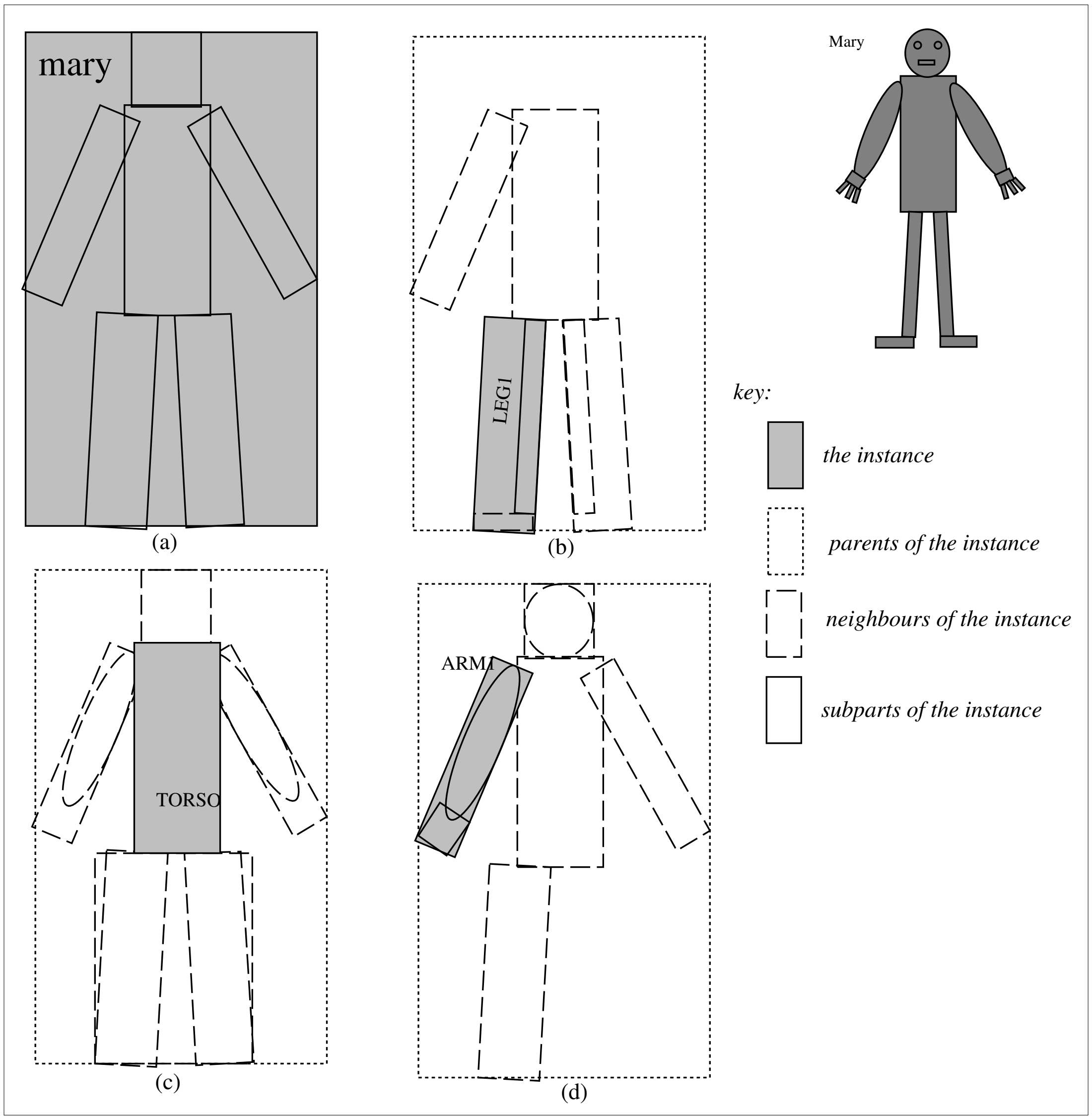


Figure 3.3: Parents, neighbours, and subparts.

by representing all of the information about how two objects are structurally related in a single richly descriptive entity. More specifically, all of the relations between two objects are represented in an attribute vector that includes qualitative and quantitative information characterising the relative *position*, *size*, *orientation*, *proximity*, *alignment*, *etc* of the two objects. The rectangular bounding boxes of the objects are used as frames of reference for defining this information. Each object has a primary axis with respect to which directions and orientations are measured. Individual attribute values can be of a variety of types, such as *numerical*, *nominal*, *angular*, *boolean*, *etc*. Further details of these are given in section 3.3.

Throughout the rest of this thesis, the term *relationship* is used to refer to an attribute vector of this kind. In this scheme, each relationship acts as a direct link from one object to another, and can therefore be exploited by the matcher, since it enables the matcher to traverse the object graph to find correspondences between parent, neighbour, and subpart objects. The exploitability of the representation by the matcher was one of the requirements discussed in section 3.1. Also, the richness of each relationship description means that candidate correspondences between the parents, neighbours, and subparts of two objects being matched, can be quickly rejected on the basis of the comparison of their relationships.

3.2.4 Each object has its own set of parent, neighbour, and subpart relationships.

The above discussion has implied that each pair of objects may have a parent-subpart relationship or a neighbour relationship associated with it. However, GRAM actually associates each relationship with just one object, not a pair of objects. Each object has its own set of *parent relationships*, *neighbour relationships*, and *subpart relationships*, and a duplicate version of each relationship is associated with each relatee. The reason for this is that it keeps each object description independent. This means that an object can be generalised without directly affecting the descriptions of its relatee objects.

In most of the diagrams of object descriptions throughout this thesis, the relationships between two objects are depicted for convenience as just one line on an object graph, but this should be interpreted as two lines. For example, the bicycle in 3.4 is represented as the object-graph in Figure 3.5, where each dotted line denotes two (identical) neighbour relationships, and each solid line denotes parent relationship of the lower object, and a subpart relationship of the higher object. This graph was generated automatically by GRAM. Most of the other diagrams throughout this thesis were created by hand, and in these, the parent, subpart, and neighbour relationships are all shown as solid lines. Neighbour relationships extend from the sides of the rectangular boxes that denote objects; parent relationships extend from the top; and subpart relationships extend from the bottom.

If a line between two objects in an object graph is shown with an arrow in one direction, then this indicates that the relationship is only explicitly included in the description of the object at the origin of the line.

3.2.5 Structure and context are explicitly distinguished, to allow disjunctions and partial matching.

Another requirement of the representation is that structure (or 'form') and context (or 'role') should be explicitly distinguished so that separate similarity scores can be produced by the matcher, and so that disjunctions of them can be represented in a generalised description. Therefore, each instance object description consists of, firstly, a *structure description*, which includes a set of structural properties (such as shape, aspect-ratio, density profile, *etc*) and its set of subpart relationships, and secondly, a *context description*, which includes a set of contextual properties (such as connection profile) and its set of parent and neighbour relationships. Section 3.3 gives more details about properties and relationships.

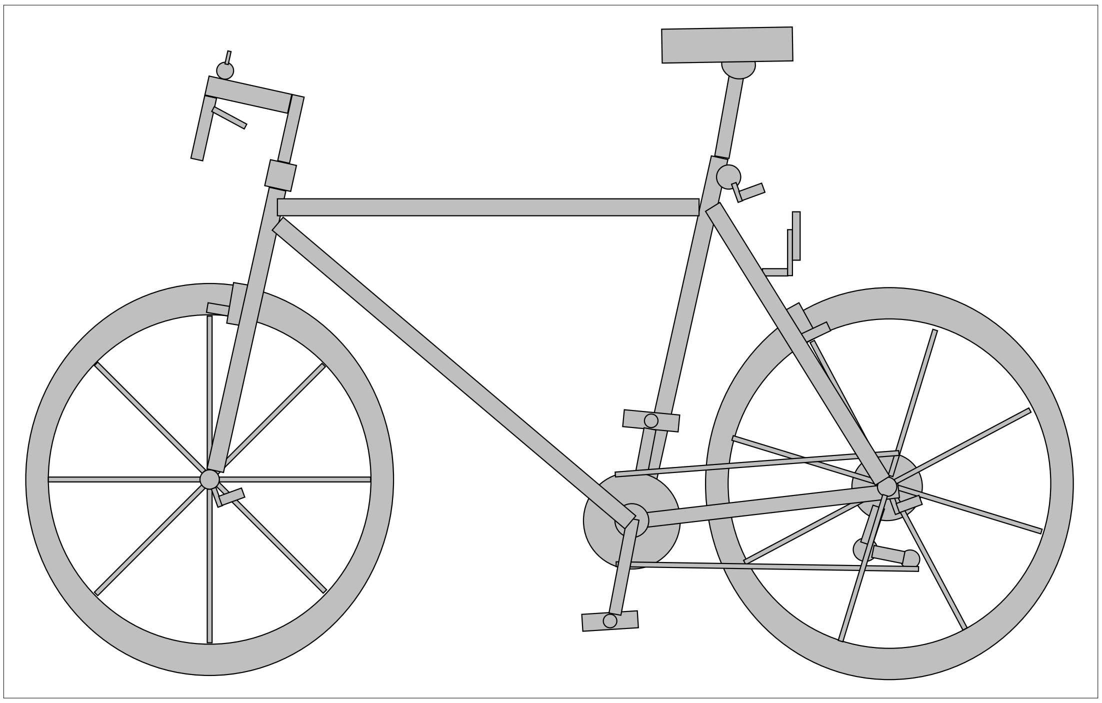


Figure 3.4: A bicycle

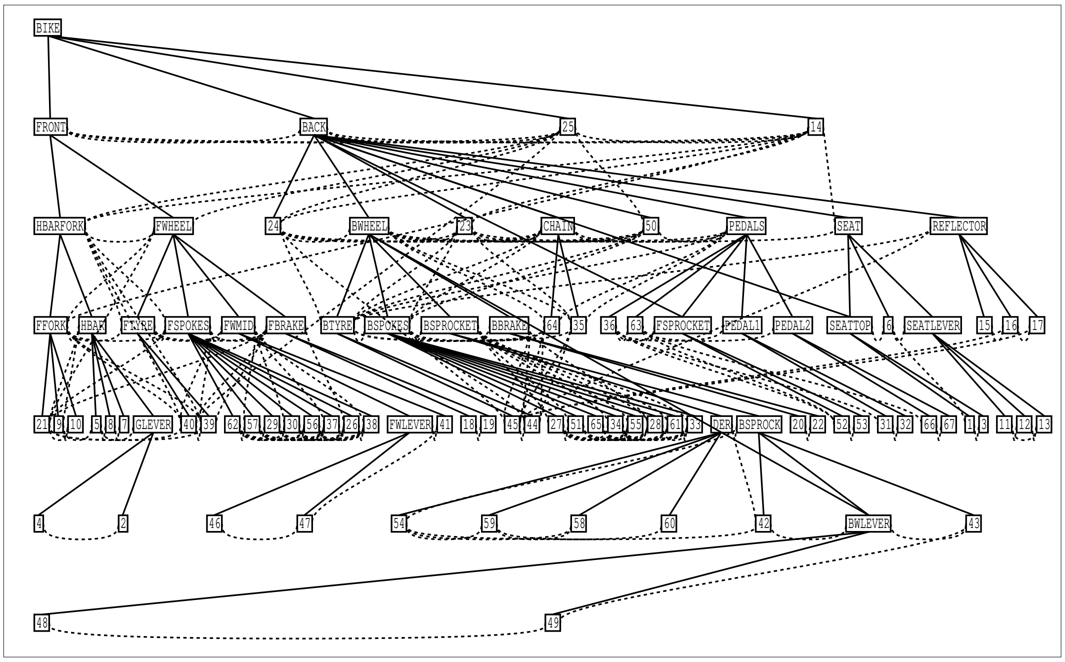


Figure 3.5: The object graph for a bicycle.

3.2.6 A multi-relationship is a generalised relationship to a concept.

A feature of GRAM's representation that is not present in systems such as Labyrinth and MERGE, is the *multi-relationship*. A multi-relationship is a generalised relationship to a particular concept, and specifies that there are n instances of that concept that are related to the source object in a particular way. This is, in fact, a way of representing groupings without actually having an explicit 'group' entity (although explicit groups are also representable). For example, a *shelf* could have a multi-relationship to the concept *book*, with a *howmany* count of 7, meaning that it has 7 books on it.

Multi-relationships are a simple and effective way of reducing the size of a description, and also enabling better generalisation. For example, suppose two shelves are to be matched and generalised, one of which has 7 books on it and the other 15. If multi-relationships were not used, then each of the shelves would have to be described using many distinct relationships, and the matcher could not find unambiguous one-to-one correspondences for generalisation. Furthermore, it would not be possible to represent the fact that the generalised shelf can have a varying number of books on it. The use of a single multi-relationship reduces memory usage and enables the matcher to unambiguously match the two relationships of the shelves, which can then be generalised. The resulting generalised multi-relationship can have a generalised *howmany* count.

A multi-relationship also enables an instance description to be represented in a slightly generalised form, since a multi-relationship is a generalisation of the individual relationships. Forming a multi-relationship is a way of generalising a single instance.

Another example of multi-relationships is given in Figure 3.6. If we suppose that the small attached parts of *C1* have been generalised to form the concept *plinket*, then the description of *C1* has two multi-relationships and two ordinary relationships, all referring to the same generalised *plinket*, as shown at the right of the figure. The two multi-relationships represent the two clusters of relationships to the plinkets at the top and to the right.

In all of the diagrams in this thesis, *howmany* counts are shown as '*n'. Relationships shown without *howmany* counts are ordinary relationships, and have a default *howmany* count of 1. (Note that *instance-counts* shown on diagrams are distinguished by not having a '*'.)

More examples of multi-relationships are given in section 3.5 which discusses groups, since multi-relationships commonly refer to the typical-member of a group.

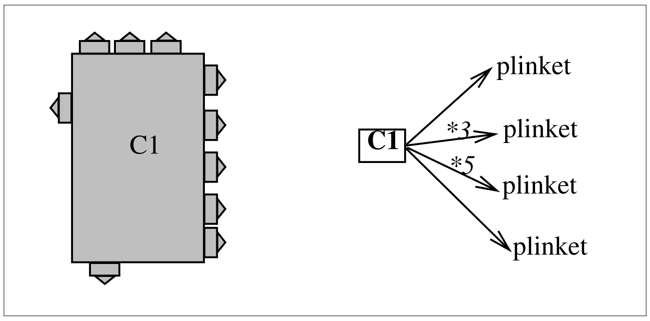


Figure 3.6: Multi-relationships.

3.2.7 An object may be a grouped object defined by a typical-member concept.

A requirement of the representation is that it should support *group* descriptions. GRAM does this by allowing sets of similar objects within an observed scene or object to be represented as a single composite object whose substructure is characterised by a single *subpart multi-relationship* to a concept which is a generalisation of its subparts. This concept is referred to as a *typical-member* concept. The idea of representing a group in terms of a typical-member was introduced by [Winston, 1975], and GRAM extends his representation in a number of ways. However, since concept representation has not been explained yet, the discussion of groups will be delayed until section 3.5.

3.3 Properties and relationships.

This section describes the attributes that are used to characterise the properties and relationships in GRAM's instance and concept representation scheme. There are many other attributes that could also be included, but these capture the most important information, and form a basis for future extensions. The purpose of this section is to show the *kinds* of information that are required, and so the specific details are not particularly important.

3.3.1 Each object has a frame-of-reference for describing properties and relationships.

Most of the attributes characterising properties and relationships of an object or objects require a frame of reference for the objects involved, so that relative positions, orientations, sizes, alignments, *etc* can be described. Therefore, each instance object has a dominant axis (or *x-axis*) and secondary axis (or *y-axis*) which define a coordinate system for specifying the properties and relationships of that object. The dominant x-axis is also referred to as the *spine* of the object.

Axes are defined to be the axes of the minimal rectangular box that bounds each object. It does not matter which axis of this box is chosen to be the x-axis, since the matcher can 'coerce' properties and relationships if two corresponding objects have been defined with different axes. However, to minimise the amount of coercion needed, GRAM maximises the canonicality of descriptions by choosing the x-axis to be the longest dimension pointing rightwards relative to the 'world' in which the object appears. For example, Figure 3.7 shows the primary axes for several primitive and composite objects.

Since a concept is a generalisation of several instance objects, its properties and relationships are defined with respect to a generalised frame of reference.

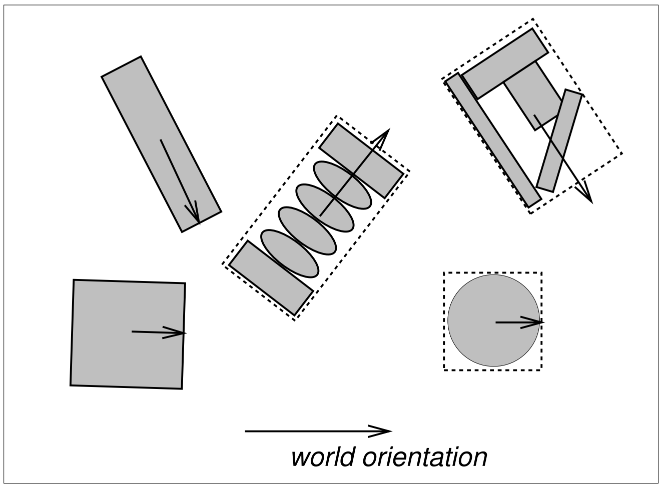


Figure 3.7: Primary axes.

3.3.2 Types of attribute value.

Before considering the specific attributes used to characterise the properties and relationships of a concept, this section first describes the generic *types* of attribute value included in GRAM's representation scheme. A summary of these is given in Figure 3.8, and each is explained below.

Numerical		
	Ungeneralised:	0..1
	Generalised:	• mean • variance • instance–count
Directional		
	Ungeneralised:	−180 .. +180 degrees
	Generalised:	• mean • variance • instance–count
Nominal		
	Ungeneralised:	**symbol:** a single symbol (eg. 'red') **symbol–set:** a set of symbols (eg. {rectangular, square})
	Generalised:	• instance–count • { (symbol count) (symbol count) ...}
Positional		
	Ungeneralised:	(0..1, 0..1)
	Generalised:	(generalised–numerical, generalised–numerical)
Boolean		
	Ungeneralised:	'true' or 'false'
	Generalised:	• instance–count • true–count
Profile		
	Ungeneralised:	(value, value, ... , value) all of the same type.
	Generalised:	(generalised–value, ..., ...) all of the same type.

Figure 3.8: Types of Attribute Value.

A **Numerical** attribute value in GRAM is a real-valued measurement between 0 and 1, which gives quantitative rather than qualitative information. Many attributes, such as *aspect-ratio*, *relative size*, and *object-density*, have numerical values. Numerical attributes values are all normalised to the 0..1 range so that the matcher can efficiently obtain a normalised similarity measure when comparing any numerical attributes.

A generalised numerical attribute value is represented as a mean, variance, and range. The number of instances that contributed to the generalisation is also specified.

A **directional** attribute value is used to describe the orientation and direction of one object with respect to the frame of reference of another object, and is in the range -180 to +180 degrees. A generalised direction is represented as a mean, variance, range, and an instance-count, as for a numerical value. Matching and generalising an direction value is slightly more complex than for an ordinary numerical value because modulo arithmetic must be used.

A **nominal** attribute value specifies qualitative information, such as shape-name, alignment, connectivity, *etc*. A nominal attribute can either be a *symbol* or a *symbol-set*. A *symbol* is a single nominal value such as *connected*, *red*, *etc*. A *symbol-set* is a *set* of symbols, such as *(square rectangle polygon)*.

A generalised nominal attribute value (independent of whether it is a generalisation of *symbols* or*symbol-set*) consists of a set of symbols and their instance-counts, where each count is the number of observed instances having that symbol value. A count of the total number of instances observed is also specified. The ratio of each symbol count to the total instance-count indicates the probability of that value occurring in a future instance. This is the same representation as used in COBWEB [Fisher, 1987a] and CLASSIT [Gennari et al., 1989].[2]

As an example of a generalised nominal value, suppose the system has observed 60 windows, of which 56 are rectangular, 30 are square (and rectangular), and 4 are round. The generalised value for the *shape* attribute of the generalised window will be as follows:

(count=60 (rectangular:56, square:30, round:4))

A **positional** attribute value is a pair of numerical values that specify a cartesian coordinate in some reference frame, and a generalised position is a pair of generalised numerical values.

A **boolean** attribute value is a specialised form of nominal value which has the value *true* or *false*. A generalised boolean value has a count of the total number of observations, and a count indicating how many of these had the value *true*.

A **profile** is a fixed-length vector of attributes of the same type. For example, a profile of the connectivity of an object might consist of a vector of boolean values, each specifying whether the object has a connection along a particular portion of its edge. A generalised profile is of the same form but consists of generalised values.

Now that we have identified the general types of attribute value, we can consider the specific attributes used to characterise properties and relationships.

3.3.3 Structure properties.

The properties characterising the *structure* of an object are represented as an attribute vector consisting of the attributes listed in Figure 3.9. Each of these is explained as follows.

[2]Symbol-sets are based on the idea of *Gray Coding* proposed by [Brady et al., 1984], although the purpose of Gray Coding was to allow the system to compare and generalise two attributes simply by computing their intersection, while GRAM performs a *union* when generalising, and takes into account the instance-counts on each value when matching.

The **shape** is a *symbol-set* nominal value that specifies one or more of {*circle*, *ellipse*, *rectangle*, *etc*}. In the current version of GRAM, all *composite* objects have the shape *rectangle* (and perhaps *square*), since mechanisms have yet to be developed to determine the shape of a composite object.

The **number-of-direct-subparts** indicates the number of objects that are directly below in the object-decomposition hierarchy. If the object is a grouped object, then this is the cardinality of the group. Since this attribute is a numerical value, it is normalised logarithmically to the range 0 and 1, so that 0 means no subparts, and 1 means 'lots'.

The **complexity** of an object is the number of objects in the entire substructure, not just the direct subparts.

The **aspect-ratio** is the width-to-length ratio, and is thus another way of characterising the rough shape of an object. The **dx-dy** is the ratio of the x-axis length to the y-axis length (which may or may not be the width and length, respectively). This attribute is necessary to help to ensure that two objects being matched are given the correct axis correspondences, since the aspect-ratio is independent of whether the x-axis is chosen to be the width or the length.

Density is a measure of how much 'solid stuff' there is within the rectangular box that bounds the object. All primitive objects have density 1. Some composite objects, such as a chair or a bedroom, have relatively low densities, while others, such as a telephone, have high densities.

shape	*one or more of: {circle, ellipse, triangle, square, rectangle, polygon, etc}*
number–of–direct–subparts	*(if grouped, then this is the number of members)*
complexity	*(the total number of descendents in the part hierarchy)*
aspect–ratio	*(ratio of width to length)*
dx–dy–ratio	*(ratio of the x–axis and y–axis dimensions)*
density	*(the ratio of solid material to space within the object's bounding box)*

Figure 3.9: Structure attributes.

There are many other structure properties that could be added to the representation, such as *colour*, *texture*, *etc*, but those described above have been sufficient for giving reasonably good results for the examples considered in this thesis.

3.3.4 Context properties.

Context properties provide a summary of the surroundings of an object, and in the current version of GRAM these are represented as two *profile* attributes. The **connection-profile** characterises the connectivity of each of a set of portions of the boundary of the object. For example, Figure 3.10 shows an object (darkly shaded) with 9 portions of its boundary indicated by the solid lines. The connection-profile has a numerical value associated with each portion, indicating how much of that edge-segment is connected to another object. Similarly, the **distance-profile** indicates the ratio of the perpendicular distance from that edge-segment to the nearest neighbour, to the dimension of the object along that direction. The ratio is logarithmically normalised to the range 0 to 1, where 0 means touching, and 1 means far away. The details of the normalisation formula are not important.

There are other alternative kinds of profiles that could be used, and the above two are just an indication of what is possible. They are very useful for enabling the matcher to estimate the similarity of the contexts of two objects prior to comparing their neighbour relationships.

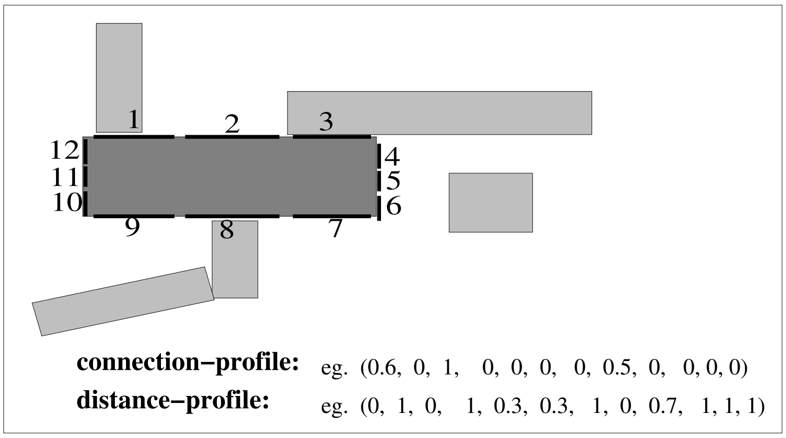


Figure 3.10: Context attributes.

3.3.5 Parent and subpart relationships.

Relationships are represented as an attribute vector that specifies the spatial relationships between two objects. This section discusses the attributes used to characterise parent and subpart relationships.

Parent relationships and subpart relationships are both described in terms of the orientation, position, and size of the subpart relative to the frame of reference of the parent object. Figure 3.11 lists the attributes used in the current version of GRAM.

The **orientation** is an directional value that specifies the rotation of the subpart's x-axis with respect to the parent's x-axis. The x and y **displacements** are the numerical distances from the centre of the parent to the centre of the subpart, along the parent's x and y axes, and relative to the x and y dimensions, respectively. **Coverage** is a profile attribute that specifies the fraction of each region of the parent (evenly subdivided) covered by the subpart. The **size** of the subpart with respect to the parent is given in four values, each being a ratio of one of the subpart dimensions to one of the parent dimensions.

A subcomponent of a object is only included as a direct subpart if it is considered directly 'important' to the description of the object. For example, it is not normally useful to include each drawer-handle as a direct subpart of a desk, since the orientation, position, and size of each drawer-handle are better described with respect to the drawer it belongs to, rather than with respect to the desk as a whole. As a general rule (for which there are numerous exceptions), a subcomponent is not included as a direct subpart if that component is a subpart of one of the object's other components, especially if the subcomponent is small compared with the object. A more elaborate discussion of subpart-selection is given in chapter 6.

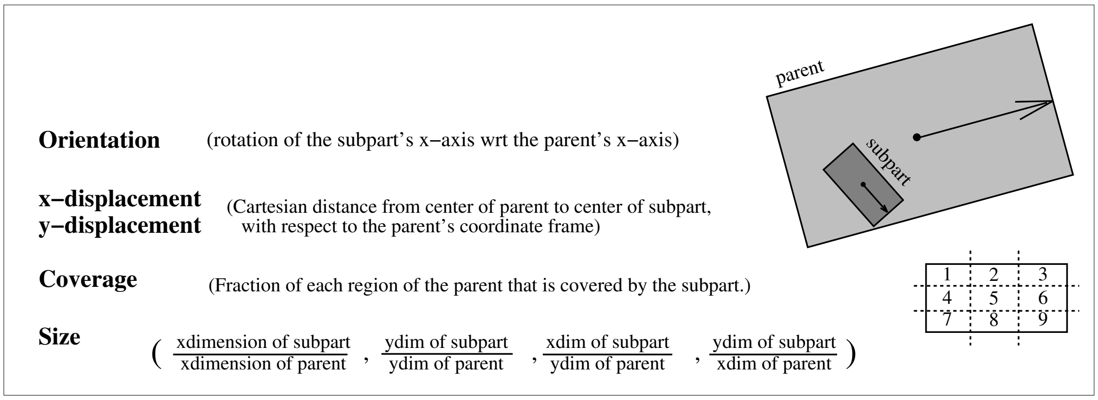


Figure 3.11: Representation of parent/subpart relationships.

3.3.6 Neighbour relationships.

Neighbour relationships are represented in a more complex way than parent and subpart relationships because more frames of reference are involved. Parent and subpart relationships are described solely with respect to the parent's coordinate frame, while a neighbour relationship is characterised by two attribute-vectors, each of which describes the relative size, position, *etc*, of one of the objects with respect to the frame of reference of the other. Thus a description of a chair could include a neighbour relationship to a desk, and the relationship would be described both relative to the chair and relative to the desk. The description of the desk could include an identical relationship. The attributes describing a neighbour-relationship are shown in Figure 3.12.

The **orientation** is the rotation of the x-axis of the neighbour with respect to the frame of reference of the object. **Direction** is the direction of the centre of the neighbour with respect to the object. The x and y displacements are the distances of the neighbour's centre from the centre of the object, along the x and y dimensions of the object. **Size** is a list of ratios of the neighbour dimensions with respect to the object dimensions.

Connectivity and **alignment** are both symbol-sets whose values qualitatively characterise the way the neighbour is connected and aligned with the object. Each of the possible values is an atomic predicate specifying a connection or an alignment between a pair of axes, edges, or vertices. If the objects are composite objects, or non-rectangular primitive objects, then the axes, edges, and vertices of their rectangular bounding boxes are used. The tables below shows the possible connectivity and alignment values used in GRAM:

Connectivity values:

value	meaning
separate	the neighbour is not connected to the object.
connected	the neighbour is connected to the object.
enclosed	the neighbour is completely within the region of the object.
enclosing	the object is completely within the region of the neighbour.
overlap	the neighbour overlaps the object.
ee	an edge of the object is connected to an edge of the neighbour.
ev	an edge of the object is connected to a vertex of the neighbour.
ve	a vertex of the object is connected to an edge of the neighbour.
vv	a vertex of the object is connected to a vertex of the neighbour.
left	the neighbour is connected to the left edge of the object.
right	the neighbour is connected to the right edge of the object.
top	the neighbour is connected to the top edge of the object.
bottom	the neighbour is connected to the bottom edge of the object.

Alignment values:

value	meaning
parallel	the x-axes of the object and neighbour are parallel.
perpendicular	the x-axes of the object and neighbour are perpendicular.
xx	the object's x-axis is collinear with the neighbour's x-axis.
yy	the object's y-axis is collinear with the neighbour's y-axis.
xy	the object's x-axis is collinear with the neighbour's y-axis.
yx	the object's y-axis is collinear with the neighbour's x-axis.
ll	the object's left edge is co-linear with the neighbour's left edge.
lr	the object's left edge is co-linear with the neighbour's right edge.
bl	the object's bottom edge is co-linear with the neighbour's left edge.
etc	

The choice of which pairs of objects are to be explicitly related as neighbours is based on a number of factors, such as proximity, connectedness, and alignment. These and other factors are discussed in section 6.3.1.

Multiple frames of reference are used to minimise loss of information during generalisation.

Section 3.1 talked about the need for a rich representation language to ensure that important information can be made explicit to prevent it from being lost through the generalisation process. The need for two frames of reference for neighbour relationships is another example of this principle, and is illustrated in Figure 3.13 below.

Suppose the matcher is matching the relationship between A1 and A2, and between B1 and B2. If the two relationships were described only with respect to A2 and B2 respectively, then there would be a mismatch, since the directions and displacements of A1 relative to A2, and of B1 relative to B2, are significantly different.

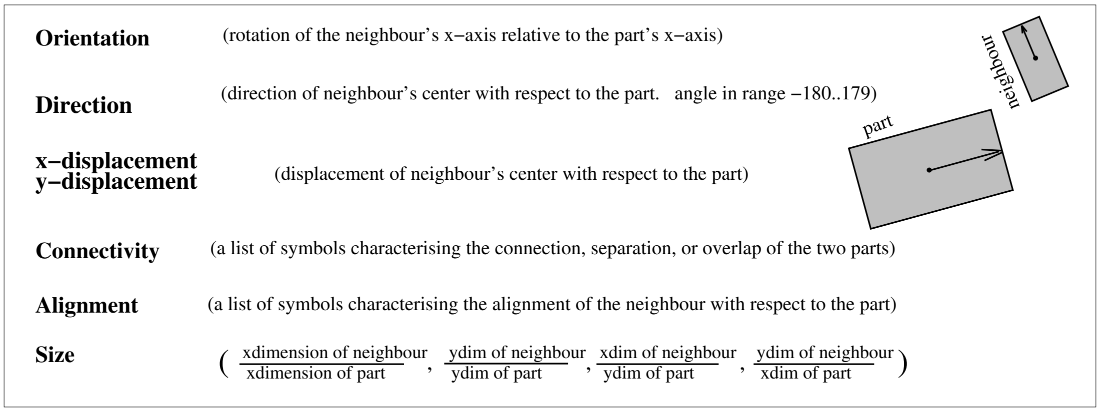


Figure 3.12: Representation of neighbour-relationships.

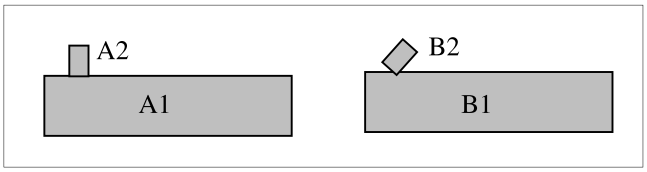


Figure 3.13: The need for multiple frames of references for neighbour relationships.

The frame of reference of a common parent could be used.

An extension to GRAM's neighbour relationship representation, which has only partially been implemented, is to allow the relationship between two neighbouring objects to be also described with respect to the frame of reference of a common *parent* object. The benefit of this is illustrated in Figure 3.14: When matching the relationships between C2 and C3, and between D2 and D3, both descriptions match poorly, whether using the frame of reference of the rectangle, or the frame of reference of the oval. However, the direction and distance between the two objects relative to their respective enclosing composite objects (C1 and D1) are very similar. Therefore, if this information is able to be included in the representation, then better matching and generalisation performance is possible.

Figure 3.15 illustrates the minimum information that would be needed. This includes the direction between the two objects, and the distances between their centres, with respect to the x and y axes of the parent. This information could either be included in each neighbour relationship, or it could be stored in the description of the parent object.

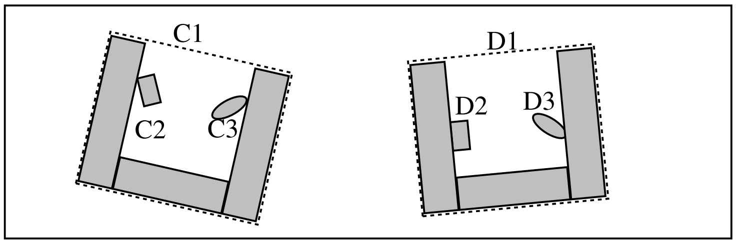


Figure 3.14: The need for the frame of reference of a common parent.

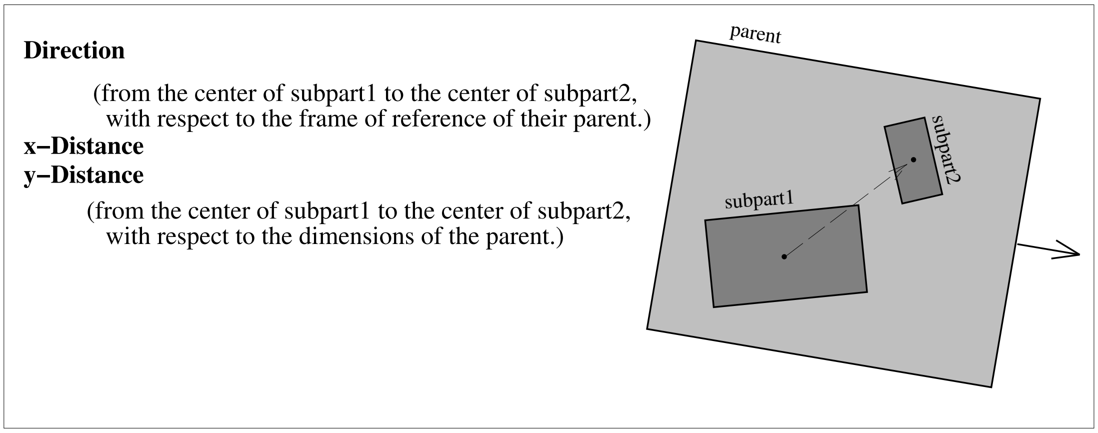


Figure 3.15: Using the frame of reference of a common parent.

3.4 Concept Representation.

A *concept* in the GRAM system is a descriptive entity in concept-memory that characterises a class of physical objects. It is either a generalisation of several instance objects, or an ungeneralised description of a single instance object that has been stored in concept-memory as a concept in its own right. Concepts are organised in memory as an AKO hierarchy, where each concept may have several subconcepts, and one or more superconcepts.

This section begins by giving a brief overview of the concept representation scheme, and then sections 3.4.1 to 3.4.3 address specific aspects in more detail.

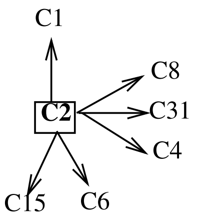
A concept is represented in much the same manner as an instance: It includes a structure description, which specifies a set of structure properties and a set of subpart relationships, and a context description, which specifies a set of context properties and a set of parent and neighbour relationships. It may also have multi-relationships to other concepts, and may be represented as a generalised group. The main difference between concepts and instances is that the attribute values characterising properties and relationship of a concept may be generalised. Also, the structure and/or context of a concept may be represented disjunctively, or by referring to another concept (which may or may not be a superconcept) by an *import-from* relationship.

Another difference is that a relationship of an instance refers to a particular instance object, while a relationship of a concept usually refers to another concept, and is interpreted to mean "one of those". The interpretation of a concept, from the point of view of the matcher, is roughly as follows: for an observed object to be considered a valid instance of a concept, the relationships of that instance object must be similar to the relationships of the concept, and the relatees of the instance should be valid instances of the corresponding concept relatees. Chapter 4, on matching, explores this in more detail.

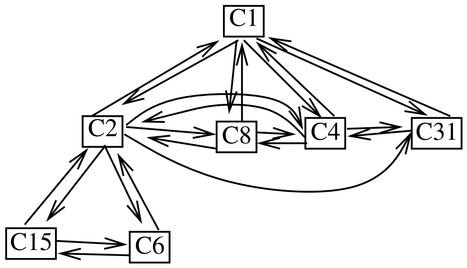
Throughout the thesis, concepts are depicted graphically. For example, Figure 3.16 (a) shows the definition of a single concept, where each link denotes a parent, neighbour, and subpart relationship to some other concept. Figure 3.16 (b) shows multiple concepts depicted in the same graph. Sometimes, for clarity, the diagrams only show one line between concepts, and (unless the line has a single arrow-head) this is to be interpreted as two distinct relationships.

The graphical depictions of concepts (or instances) do not convey the richness of the descriptions, since the properties associated with each node, and the attributes characterising each relationship, are not shown. It is hard to illustrate a concept graphically because attributes are usually generalised. However, the graphical illustrations are sufficient for most of the discussions in the thesis.

Unlike concept descriptions in systems such as Labyrinth [Thompson and Langley, 1991], a concept description in GRAM does not include explicit relationships between its subparts. This is because the concepts that the subpart relationships refer to, have neighbour relationships with other concepts, and this provides sufficient constraint. Figure (b) illustrates this, where the required relationships between the subparts of an instance of concept C1 are represented as the neighbour relationships of the concepts C2, C8, C4, and C31. In Labyrinth, concepts do not have context information, and so the inter-subpart relationships have to be explicitly included in the concept description.



(a) A single concept depicted graphically.



(b) Multiple concepts depicted graphically

Figure 3.16: Concepts depicted graphically.

In GRAM's scheme, each concept description is small and simple, since there is no need to include a set of local variables representing its subparts, or a set of relationships between them. However, since each concept is defined in terms of richly described relationships with other concepts, in the manner of a semantic net, a concept may still be implicitly very complex. For example, the description of the concept *letter-box* could implicitly include the constraint that it has a door-handle nearby, if it is defined in terms of the concept *house* which is defined in terms of the concept *front-door*.

A consequence of GRAM's representation scheme is that when a new concept is formed from an observed instance, *every* component of that instance must also be added to concept-memory as a new concept (unless it is an instance of a previously learned concept). It is not possible to create just one concept defined by a local part-graph, since every concept is only definable by relationships with other concepts (or instances interpreted as concepts). Therefore, concept-memory could include a concept for *left-chair-legs*, *bottom-screws-of-wall-sockets*, *top-halves-of-Bic-pens*, and countless other classes of 'object' that we do not normally think of as being objects, and that do not have common names in the English language. The Labyrinth system and the MERGE system [Wasserman, 1985] also share this characteristic, although Labyrinth does not represent context and so it can learn a smaller variety of concepts, distinguished solely by substructure.

3.4.1 Context or structure may be 'imported' from other concepts.

It is useful to be able to define the structure or context of a concept (or even an instance) by referring to the structure or context of an existing concept, rather than including details explicitly. This helps to reduce memory usage and enables a greater transfer of information amongst descriptions. Suppose the system observes *bedroom1* in Figure 3.17 and creates an ungeneralised concept for each of its components. Each of *chair1*, *desk1*, *clock1*, and *bed1* have complete structure and concept descriptions. However, if the system has already learned the generic concepts *chair*, *desk*, *clock*, and *bed*, then the description of *bedroom-1* should not have to include complete descriptions of its subparts. Instead it should be able to make use of descriptions of the existing concepts.

One way of doing this is for *bedroom1* to refer directly to the previously learned concepts *chair*, *bed*, *etc*, as shown in Figure 3.17 (a). However, it would then lose the information about how these components are related to each other. The relationships of each component with the bedroom as a whole would be explicit in the subpart relationships, but the neighbour relationships *between* the subcomponents would not be, since they would be defined only in terms of generalised concepts. In some cases this is acceptable if only a rough description is required, but in other cases the neighbour relationships between subparts are important.

Another way to deal with this is to use GRAM's *import-from* relationship. This allows the structure or context of a concept (or an instance) to be specified by referring to another concept. This is shown in Figure 3.17 (b), where the structure of concept *chair1* is defined by an *import-from* relationship to the more general *chair*. It is not necessary for *chair* to be an explicit superconcept of *chair1*, and later we will see examples where a concept's *import-from* relationship refers to a concept that is not even more general. Therefore, the *import-from* relationship subsumes the usual form of inheritance.

If a concept is defined using *import-from*, it may also include explicit local relationships which override or specialise the information imported from the other concept. For example, if *chair1* had an unusual kind of back, but was otherwise just a standard chair, then it could include a single subpart relationship to *chairback1*, as shown in the figure. To be an instance of *chair1*, the structure of an instance must not only match the structure of chair (via *import-from*) but must also include a matching subpart relationship to a back that matches *chairback1*. The description of *chair1* does not include explicit bindings to indicate that the subpart relationship corresponds to a particular subpart relationship of the imported *chair*. Such bindings are not necessary, since, when matching a new object with *chair1*, the appropriate correspondence can be found by matching. This does require additional effort in that the matcher must compare the local subpart relationship with instance subpart relationships, rather than being able to make use of explicit bindings with the already-matched subpart relationships of *chair* (or vice versa). However, it also avoids the need for dealing with bindings, and this is considered beneficial because it simplifies the representation scheme, the matcher, and the generaliser. This is an approach taken throughout the GRAM system.

An *import-from* specification may also refer disjunctively to several concepts. We will see examples of this in section 3.4.2.

Import-from specifications could actually be added to an instance description, not just to a concept description, so that details of classifiable components can be removed. This would be done by the instance-constructor, although GRAM does not currently support this. Such a process is actually generalising the instance, since the relationships of the modified instance refer to generalised relatees.

3.4.2 Concept variability is expressed by attribute distributions, instance-counts, and disjunction.

Concepts are generalisations of instances, and must therefore capture the variability of permissible instance features. This is achieved in several ways:

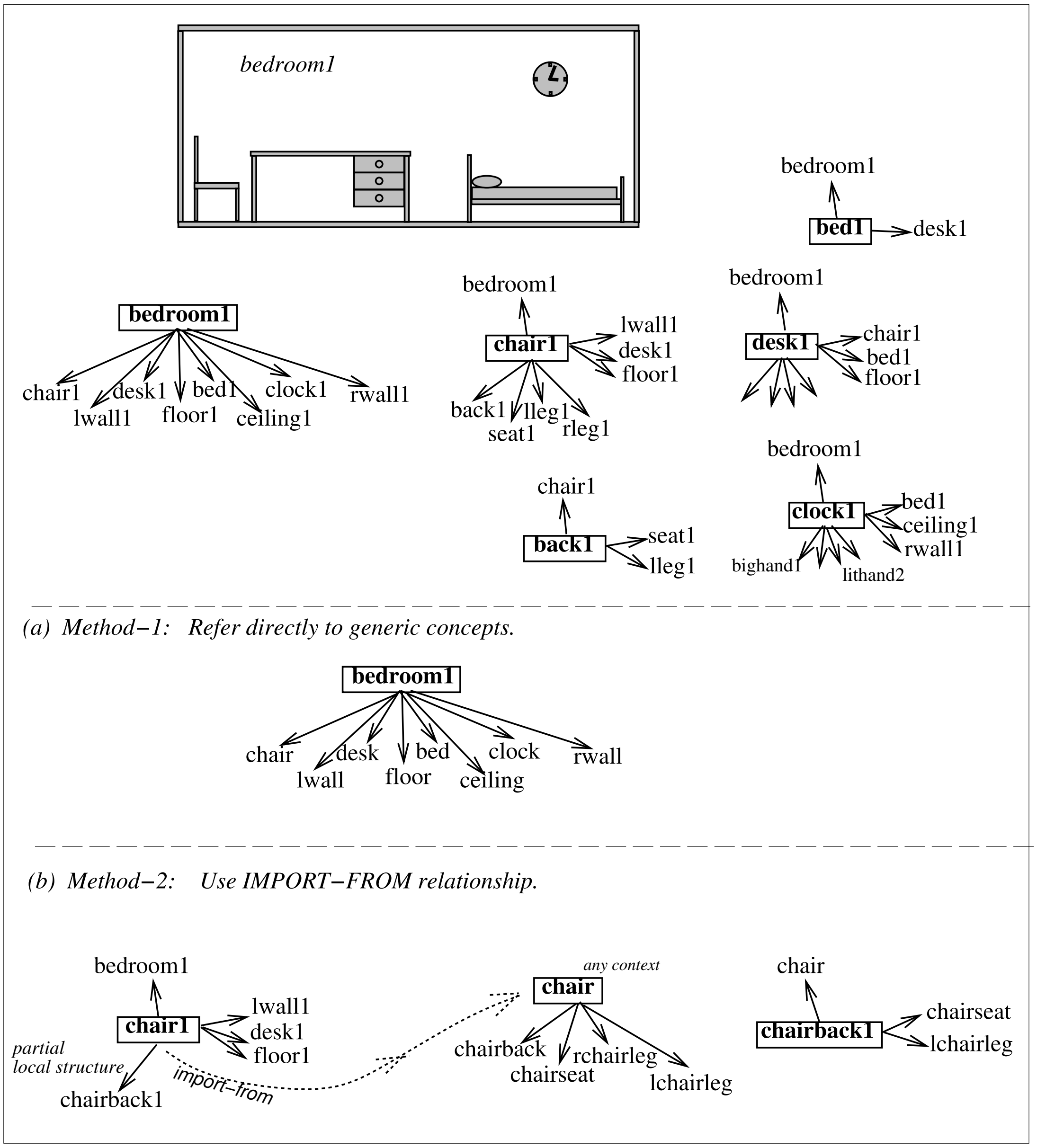


Figure 3.17: Imported structure.

Attribute distributions

One kind of variability is characterised by representing attributes as value distributions. More specifically, a generalised numerical attribute value is represented as a mean, standard deviation, and range. A generalised nominal attribute value is represented by the frequencies of each observed value. This was explained in section 3.3.2.

Instance-counts

The second way of representing concept variability is by giving each parent, neighbour, and subpart relationship an *instance-count* indicating how many observed instances of the concept included this relationship. The concept as a whole also has an instance-count, so that the ratio of each relationship's instance-count to the concept's instance-count indicates the degree of optionality of that relationship, and can therefore be used to make predictions about a partially unobservable object, or for determining the significance of a missing feature. This ratio is called the *instance-count-ratio*.

Through the process of generalisation, the sets of parent, neighbour, and subpart relationships of a concept may gradually increase in length as new instances are observed that have relationships that were not present in previous instances. For example, new models of telephone might be observed, each with some different subparts and different relationships, although retaining the basic telephone structure. The most common subpart relationships will have high instance-counts, and these will be given most significance by the matcher.

As another example, consider the four objects in Figure 3.18. After generalising the instance-concepts A, B, C, and D to form a new concept E, the subpart relationships of E have various instance-counts as shown on the diagram. The ratio of the instance-count of each of these, to the instance-count of E, is its instance-count-ratio, and can be treated as a probability for the presence of that subpart relationship in a future instance. A larger number of observations will result in probabilities that have greater predictive accuracy.

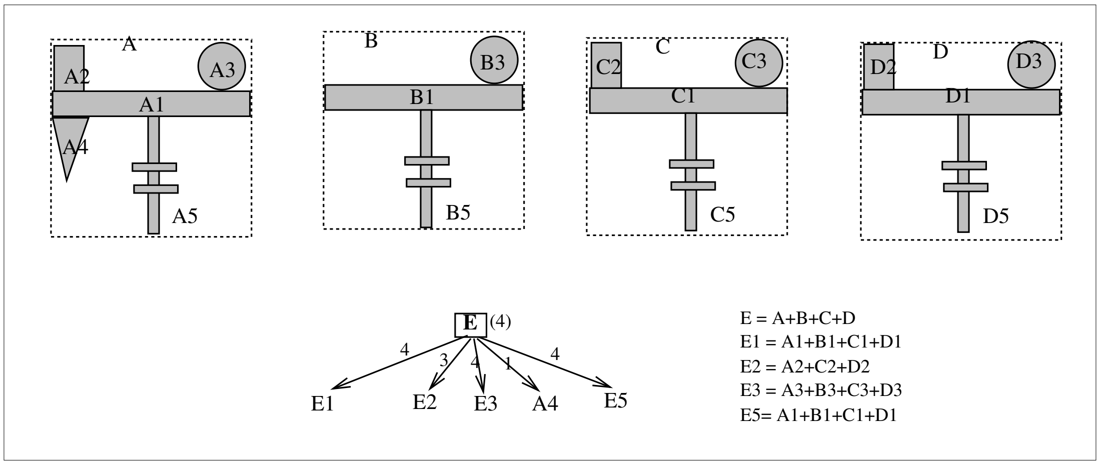


Figure 3.18: instance-counts.

One apparent limitation of GRAM's representation is that it does not allow a concept description to explicitly include the relationships *between* its parents, neighbours, and subparts. This also means that the co-existence of optional relationships/relatees is not explicit. However, the dependencies between the relatees are usually captured in the relatee descriptions themselves. Consider the example in Figure 3.19. Object A and object B have been generalised to produce the concept AB shown at the bottom of the figure. Concepts AB6 and AB7, which were each constructed from the single instances A6 and A7 respectively, are also shown. AB has subpart relationships to AB6 and AB7, each with an instance-count of 1. The problem is that there is no indication of whether these must be *both* present or *both* absent in an instance of AB, or whether they can be independently present or absent. However, the description of concept AB6 has a non-optional relationship to AB7, and AB7 has a non-optional relationship to AB6. Thus if the matcher compares AB with an object that contains an AB6-like part but no AB7-like part, then a mismatch would be detected, since the AB6-like part would not match the AB6 concept perfectly.

Therefore, the neighbour relationships of optional relatees of a concept provide co-existence constraints without having to represent them explicitly within their parent concept description. Of course, this is only true if the optional relatees have explicit relationships between them. When optional components are *not* explicitly related via a neighbour relationship, then co-existence is much less explicit, such as if A7 had been on the other side of object A, rather than next to A6. However, this limitation is not a problem, since if two components are not structurally connected or close to each other, then in general (in the absence of functional knowledge) it is more likely that their presence or absence is independent.

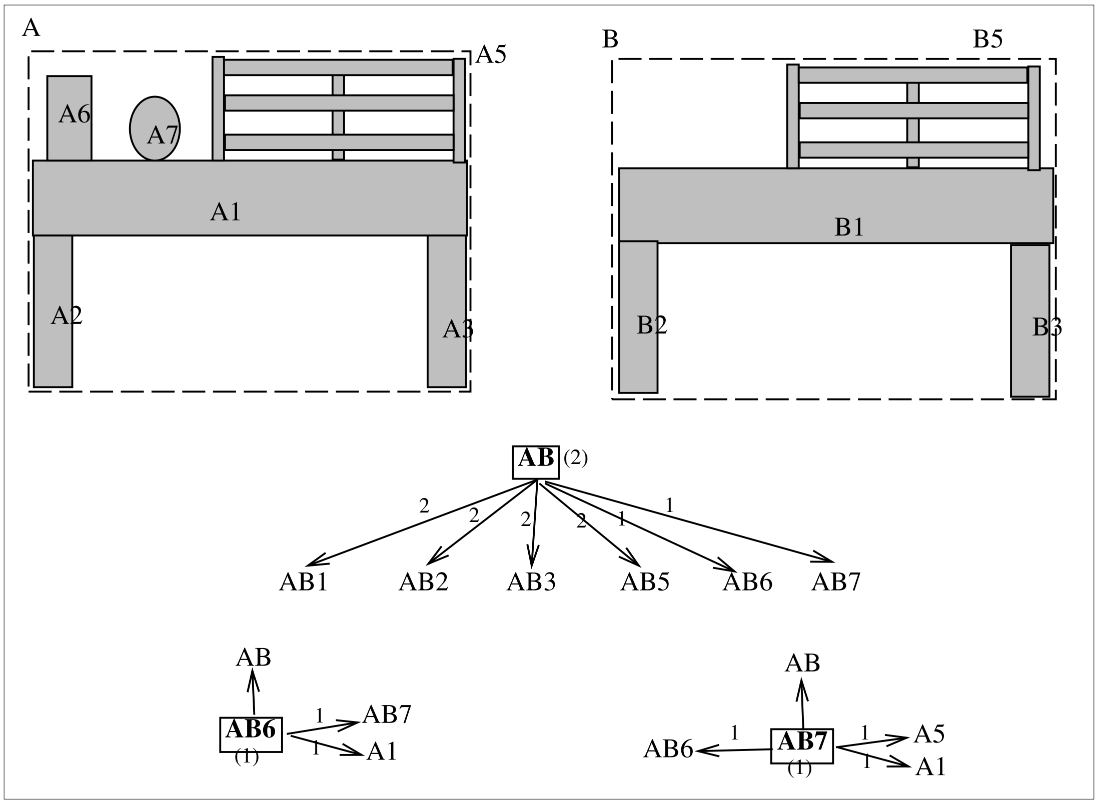


Figure 3.19: Co-existence dependencies between optional components.

Structure and Context Disjunction

Sometimes it is undesirable for the generaliser to simply merge two descriptions into single lists of components with instance-counts indicating the degree of optionality. If two structures or two contexts being generalised are significantly different, then it is better to represent the structure or context as a set of structure disjuncts or context disjuncts, which means that co-existence dependencies are explicit. This is the third way of representing concept variability.

For example, Figure 3.20 shows two doors with handles that are sufficiently different that they cannot be generalised to form a single generalised structure without resulting in a large list of optional subpart relationships, with coexistence dependencies specified only implicitly via the subpart concept descriptions. To deal with this, the generalised *handle* concept should be represented disjunctively.

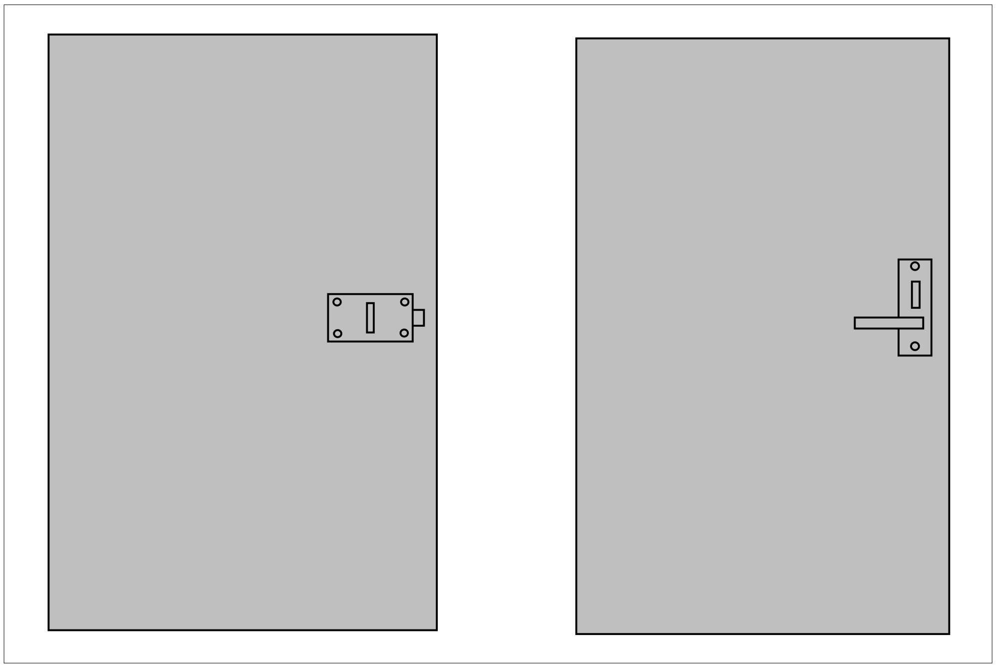


Figure 3.20: A generalised door requires a disjunctive door-handle description.

Several ways of representing disjunctions have been considered for the GRAM system. The first way is illustrated in Figure 3.21 (a), which shows a concept A1 whose structure is disjunctively defined. The three small grey squares denote three alternative structure descriptions. Each is represented by its own set of structure properties and subpart relationships. In addition to the disjuncts, a disjunctive concept in this scheme can also include a set of properties that is a generalisation of all of the disjunct properties, and also a set of any subpart relationships that are common to all or most of the structure disjuncts.

Likewise, figure (b) shows a concept B1 whose *context* is defined disjunctively, where each of the context disjuncts has its own set of context properties, neighbour relationships, and parent relationships. A concept can also include a set of parent and neighbour relationships common that are to all or most of the context disjuncts.

A concept may also have disjunctions of structure *and* context, such as for a concept *chair*, which could be defined in terms of a variety of structures and a variety of contexts, with a few structure and context features common to most chairs, such as being upright on a floor, and having a vertical back and horizontal seat and some kind of support structure beneath.

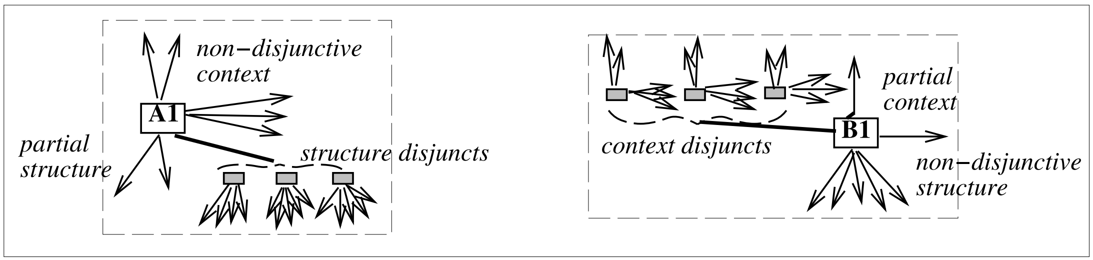


Figure 3.21: Method-1: Include a set of structure disjuncts and/or context disjuncts.

One limitation of this scheme is that it assumes that structure and context are always independent. The generaliser creates disjuncts whenever there is a significant mismatch with the existing structure or context descriptions of a concept and an instance that are to be generalised. If successive generalisations of a concept result in it having both structure *and* context disjuncts, then these must be interpreted to mean that *any* pairing of structure disjunct and context disjunct is permitted in an instance, since there is no information specifying co-existence dependencies. Therefore, over-generalisation will tend to occur. In the case of a *chair* concept, the system is not able to represent the fact that swivel chairs tend to appear in offices rather than living rooms.

Another limitation of this scheme is that it cannot represent a multi-level taxonomy, since a disjunction consists of just one level of disjuncts which are not further subdivided into more specialised variants. However, concepts are represented in an AKO hierarchy anyway (although this thesis does not address the details of this), and so the set of disjuncts is redundantly specifying what is already specified as *subconcept* descriptions. For example, if a *chair* concept has several structure disjuncts, one for each variety of chair it has seen, then each of these varieties will also be represented as subconcepts of the chair. For example, Figure 3.22 illustrates how the description of A1 from Figure 3.21 has its structure disjuncts redundantly duplicated in the subconcept descriptions.

This suggests that perhaps it is sufficient to allow the AKO hierarchy to capture the disjuncts of a disjunctive concept. Common features could be retained explicitly in the top-level concept description, while its details would be defined as being "any of its subconcepts". Figure 3.23 shows the concept A1 represented in this manner. This avoids the problems of duplicated information, and also simplifies the representation scheme since there is no need to deal with additional disjunct descriptions.

Figure 3.24 shows a more complex example where the structure and context of the concept *chair* are *both* defined disjunctively. A few features common to all or most of the subconcepts (such as relationships to the concept *chair-back*, *chair-seat*, *floor*, and *room*) are included in

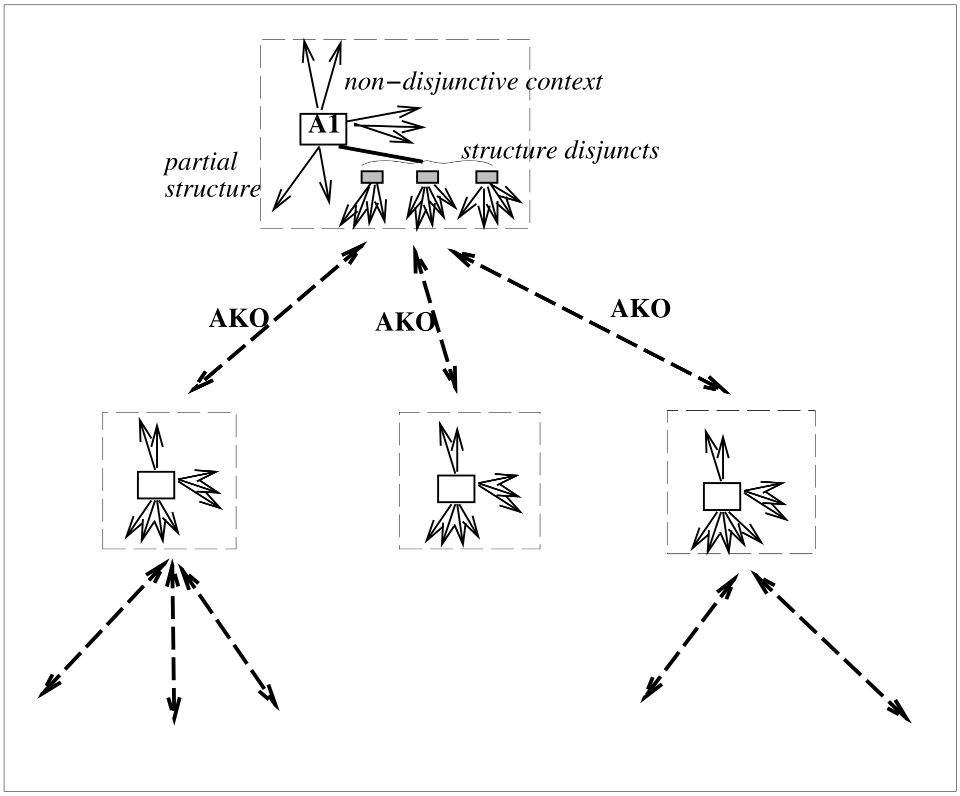


Figure 3.22: Method-1 problem: Disjuncts are duplicated as subconcepts.

the *chair* concept itself, as a partial description. (The distinction between partial and complete descriptions is discussed in section 3.4.3).

For an instance to match the *chair* concept, its structure must match one of the subconcepts, and its context must also match one of the subconcepts. The percentages on the AKO links indicate the proportion of concept instances that were instances of each subconcept, and these can be used for making predictions about the structure and context of a partially observed instance.

There is no explicit distinction between subconcepts that define structure disjuncts, and subconcepts that define context disjuncts, since that would require two different kinds of AKO link, and a more complex memory organisation system.

This may seem to be a limitation of this scheme, since the matcher must match the structure of a candidate instance with the structure of all of the subconcepts, and must also match its context with all of the subconcepts. However, the use of the *import-from* relationship prevents unnecessary work, and also reduces memory usage, since it can be used to define the structure or context of a concept in terms of the structure or context of another context, or a disjunction of other concepts. For example, the *armchair* concept has its own local structure description, but its context can be imported from either *chair-in-living-room* or *chair-in-office*. The percentages shown in the figure indicate the proportion of observed instances of the concept that had each

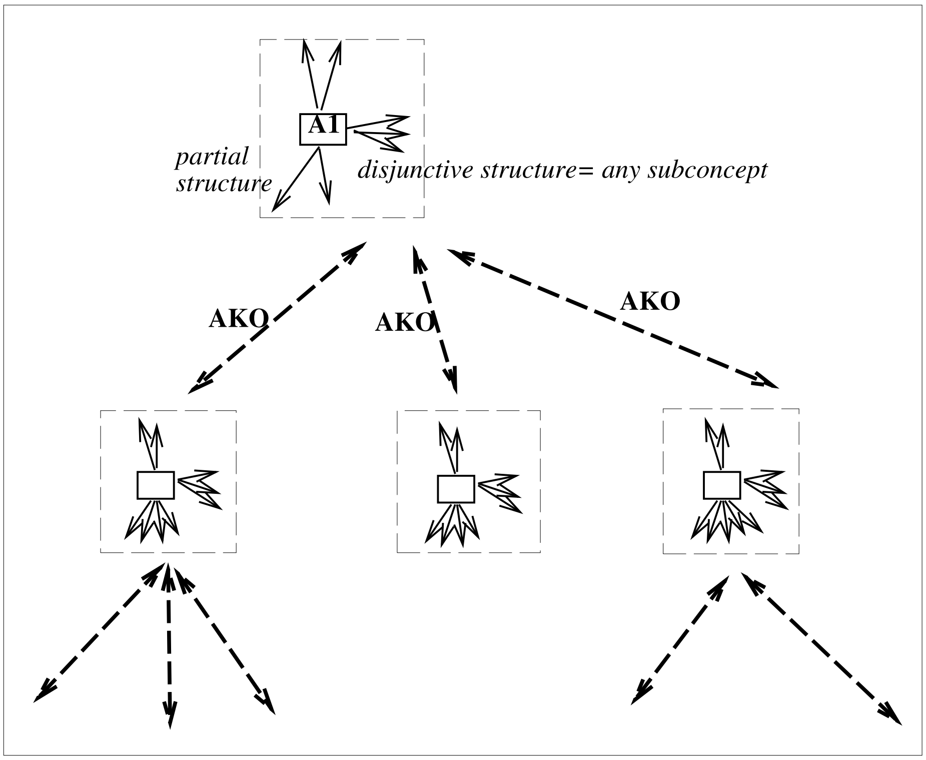


Figure 3.23: Method-2: Disjuncts are represented only by subconcepts.

of the contexts. Similarly, the concept *chair-in-office* has its own local context description, but its structure is imported from *swivel-chair* or *standard-chair*.

As another example from Figure 3.24, the subconcept *standard-chair-in-office* of *standard-chair* has a context that is the same as *chair-in-office*, and a structure that is imported from *standard-chair*. The latter case is like the usual form of inheritance, since it refers to a superconcept.

The *import-from* relationship allows a concept to have overlapping subconcepts, some of which define distinct kinds of structure, some of which define distinct kinds of context, and others (such as the *dentists-chair-in-dentist-office*) which define a subconcept that is distinct in both structure and context. Each structure-defining subconcept can refer to the context of a context-defining subconcept, and vice-versa, and thus the *import-from* enables a transfer of information amongst concepts. If a concept is generalised, then any concepts that are defined in terms of it, via *import-from* (or any other relationship, in fact) are also implicitly generalised. Of course, this has the danger of over-generalisation occurring, and the concept-learning and memory-organisation systems must deal with this. This is a difficult problem which is not addressed in this thesis. In fact, GRAM is not currently able to create *import-from* relationships, since this is part of the larger learning and memory-organisation system, rather than the generaliser. However, the matcher is able to deal with them.

It is important to note that the matcher is only *required* to match a new candidate instance

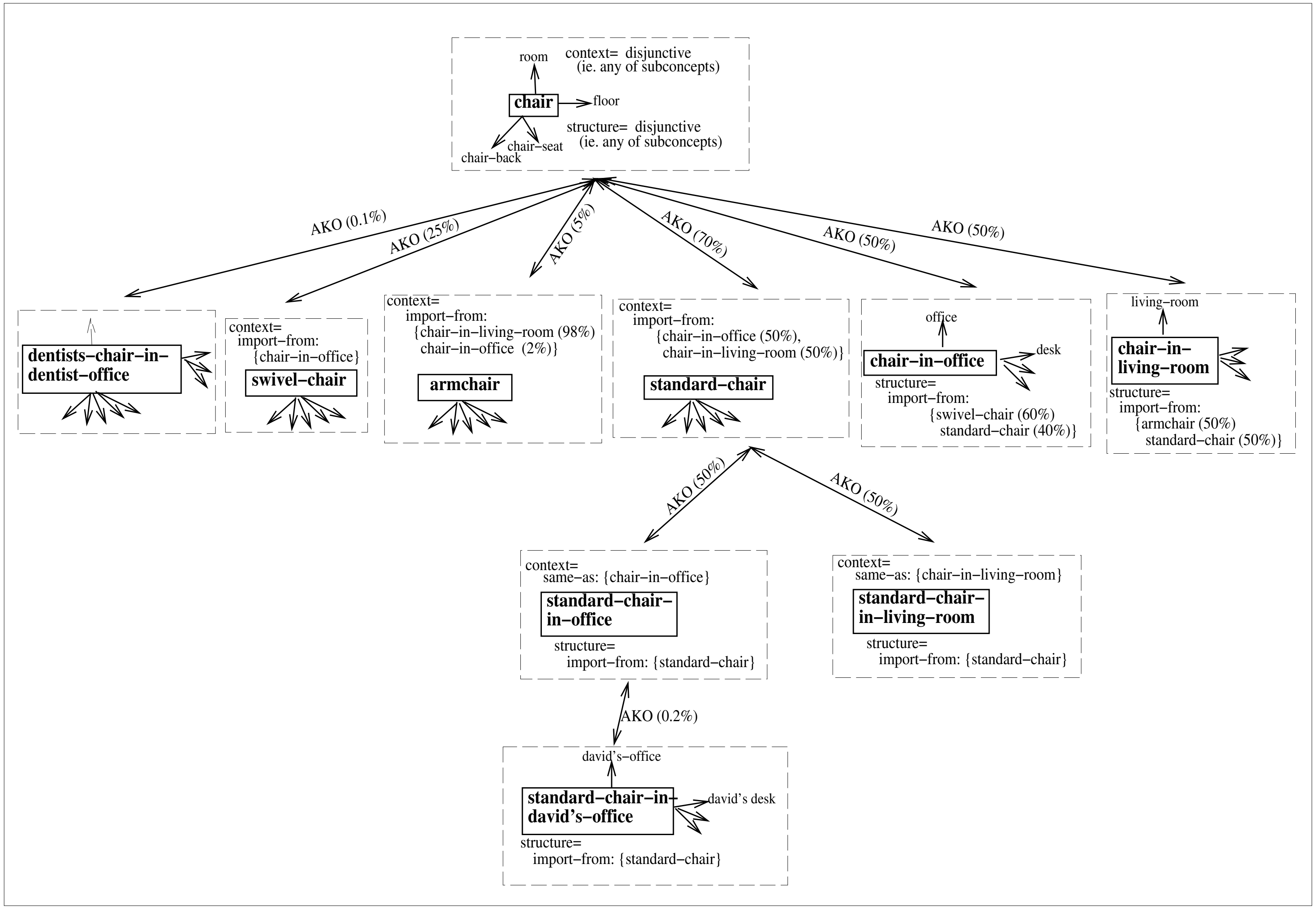


Figure 3.24: Method-2: A more complex example.

with the subconcepts of a concept if the concept is defined disjunctively. In the case of the *standard-chair* in the figure, whose structure is not defined disjunctively, the matcher does not need to compare the structure of an instance with *standard-chair-in-office* or *standard-chair-in-living-room*, since all the required information is captured in the local structure description. However, the classification system would need to match the instance with the subconcepts if a more specialised classification is required.

A more elaborate scheme for representing disjunction is to allow *arbitrary* disjunctions of parent, neighbour, and subpart relationships. However, dealing with arbitrary disjunctions is a difficult problem because, when performing generalisation, there may be innumerable ways of creating disjunctions, and it is difficult to backtrack later if a wrong decision was made. Therefore, GRAM only allows disjunction of the larger chunks of information that characterise structure and context.

Overall concept variance

A fourth way of representing concept variability is to include, in each concept, measures of variance (or 'fuzziness') of the concept as a whole, of structure and context, and of each parent, neighbour, and subpart relationship. It is useful to know how 'fuzzy' a generalised concept is, since this indicates how useful it is for making detailed predictions. If detailed predictions are required, and the concept has a high variance, then this indicates that it may be necessary to consider its subconcepts. These measures are also used by the matcher as importance-weightings when combining similarity scores of the features being matched. Measures of concept variance could also be used by the larger concept-learning system when reorganising concept-memory, such as when determining whether a concept has become too general to be usefully retained in concept-memory.

3.4.3 Concepts can have a variety of interpretations.

Stepp [Stepp, 1987b] made the distinction between two kinds of concept interpretation – "contains" or "is". If a concept has an "is" interpretation then it requires that an instance has all of the specified features, and no additional features. A "contains" interpretation is much weaker: an instance is considered to be a valid instance of the concept if it has all of the specified features, even if it also has additional features. For example, a "contains" interpretation of a *chair* concept might allow a 'chair+person' object to be classified as a chair, since it contains the required subcomponents and relationships between them.

It may seem that an "is" interpretation is obviously the most desirable. However, if an "is" interpretation is required, then this means that when two or more instance descriptions are generalised to form a new concept, the generaliser cannot employ the "drop feature" operation [Michalski, 1980] which simply excludes features that are not common to all or most of the instances. This is because after dropping features, the concept would need to have a "contains" interpretation to allow the dropped features to be present in new instances without them being treated as mismatches.

If the "drop feature" operation is not used, then concept descriptions can become quite cluttered with low-occurrence relationships. Therefore, GRAM *does* allow the generaliser to drop features, and does allow concepts to have a 'contains' interpretation in that situation. This is acceptable because a concept has other features, such as its structure and context properties, and subpart-concept relationships, which usually constrain what instances can match the concept. For example, suppose a *chair* concept is represented as a "contains" description that specifies a relationship to a *back* and a relationship to a *seat*. Although additional parent, neighbour, and subpart relationships in an instance are permitted without being considered a mismatch, the structure and context properties, such as overall shape, aspect-ratio, density-profile, *etc*, would prevent the chair concept from matching an observed 'chair+person' object. Furthermore, the details of the subpart relationships of the chair to its seat and back would not closely match the subpart relationships of the 'chair+person' to the chair and seat, since the 'chair+person' has a different bounding-box with respect to which the relationships are defined.

In GRAM, the "is" interpretation is referred to as **complete**, and the "contains" interpretation is referred to as **partial**. Each concept description specifies the interpretation of its structure description and the interpretation of its context description.

The idea of allowing a description to have either a *partial* or a *complete* interpretation, each of which affects how the matcher and generaliser operate on it, has not been used in the other systems reviewed in this thesis, although it is similar to the distinction between *characteristic* and *discriminant* descriptions proposed by [Michalski, 1980]. A characteristic description is a 'full' description of the details of a concept, while a discriminant description include sufficient details to distinguish instances of it from instances of other concepts. However, a *partial* interpretation is not specifically intended for discriminant descriptions.

It is also possible for an *instance* description to have a *partial* interpretation, which means that information is incomplete or unavailable. For example, if a bicycle is observed with a brief glance, then its structure could be described partially. This indicates to the matcher that, if possible, further observation of the instance should be performed if a more thorough classification of the instance is required.

Although the above discussion has suggested that a *partial* interpretation is used when features have been dropped, this is not always true: A *partial* interpretation is normally used in GRAM when relationships have not just been dropped, but when they have been *replaced* by other information, such as a disjunction, a multi-relationship to a typical-member concept, or an *import-from* specification. Consequently there are several different types of *partial* interpretation.

The first kind of *partial* interpretation is **partial+disjunctive**. This indicates, firstly, that the structure (or context) is disjunctive, and secondly, that the structure (or context) features included explicitly in the description (non-disjunctively), are only partial. This means that when the matcher compares the non-disjunctive description with a new instance, it allows any additional subpart relationships (or parent and neighbour relationships for a context) without treating them as a mismatch. However, the instance must also match at least one of the disjuncts.

For example, the context of a *chair* concept might also be described with a *partial+disjunctive* interpretation, as illustrated in Figure 3.25, where a concept *chair* has been formed from the instances *chair1* and *chair2*, and likewise for the concepts *back* and *seat*. The contexts of *chair*, *back*, and *seat* are *partial+disjunctive*, and their structures are **complete**. In the case of *chair*, no neighbour or parent relationships have been explicitly included in the example, but in the case of the *back* and *seat*, both have include parent and neighbour relationships to the chair as a whole and to its other parts. Since the description is partial, the matcher would allow a candidate *chair*, *back*, or *seat* instance to have other relationships as well (such as to a table, to a person, or to an elephant). However, because their descriptions are also disjunctive, the context of the instance must match one of the disjuncts, all of which have *complete* interpretations, and which effectively override the partial interpretation.

Another kind of interpretation is **partial+typical**. This applies to structure descriptions (and not context descriptions) for groups, and where most or all of the individual subpart relationships have been removed, leaving only a multi-relationship to the typical-member concept and perhaps a few subpart relationships to atypical members. This interpretation allows the matcher and generaliser to ignore unmatched subparts of an observed instance, so long as they match the typical-member concept. The discussion of groups in section 3.5 and the discussion of group matching in section 4.3.3 explore this in more detail.

If a grouped structure has *not* had its individual subpart relationships removed, then it has an ordinary *complete* interpretation, although the matcher and generaliser still treat these relationships with less importance, since the multi-relationship to the typical-member characterises them in summary form.

If a concept structure (or context) is imported from another concept, then it has a **partial+imported** interpretation. From the point of view of the matcher, importing is very similar to disjunction, since in the former case the structure or context of an observed instance must match one of the specified concepts, and in the latter case the structure or context of an observed instance must match one of the concept's subconcepts. In both cases, if the concept also includes a *partial* local structure or context description, this must also match the instance.

Another kind of structure interpretation that I considered including in the GRAM representation is *contents-only*. This was to be used for concept structures that are primarily defined by their contents rather than the arrangement of their contents. Section 3.1 discussed a bedroom as an example of this, and it is illustrated in Figure 3.26 which shows descriptions of the components of *bedroom2*. At the bottom of the figure is a description of *bedroom12* which is a generalisation of *bedroom1* (from Figure 3.17 on page 90) and *bedroom2*. The arrangement of these generalised components is very variable and so the generalised bedroom is defined primarily by what it contains, rather than how its contents are arranged. The advantage of allowing an explicit *contents-only* interpretation is that it immediately indicates to the matcher that the subpart relationships themselves are not useful for determining the best correspondences amongst the subcomponents of such a concept and a new instance. The matcher needs to compare the subpart object descriptions themselves, without the guidance and constraint of subpart relationship similarity.

However, a *contents-only* interpretation in not included in GRAM's representation because

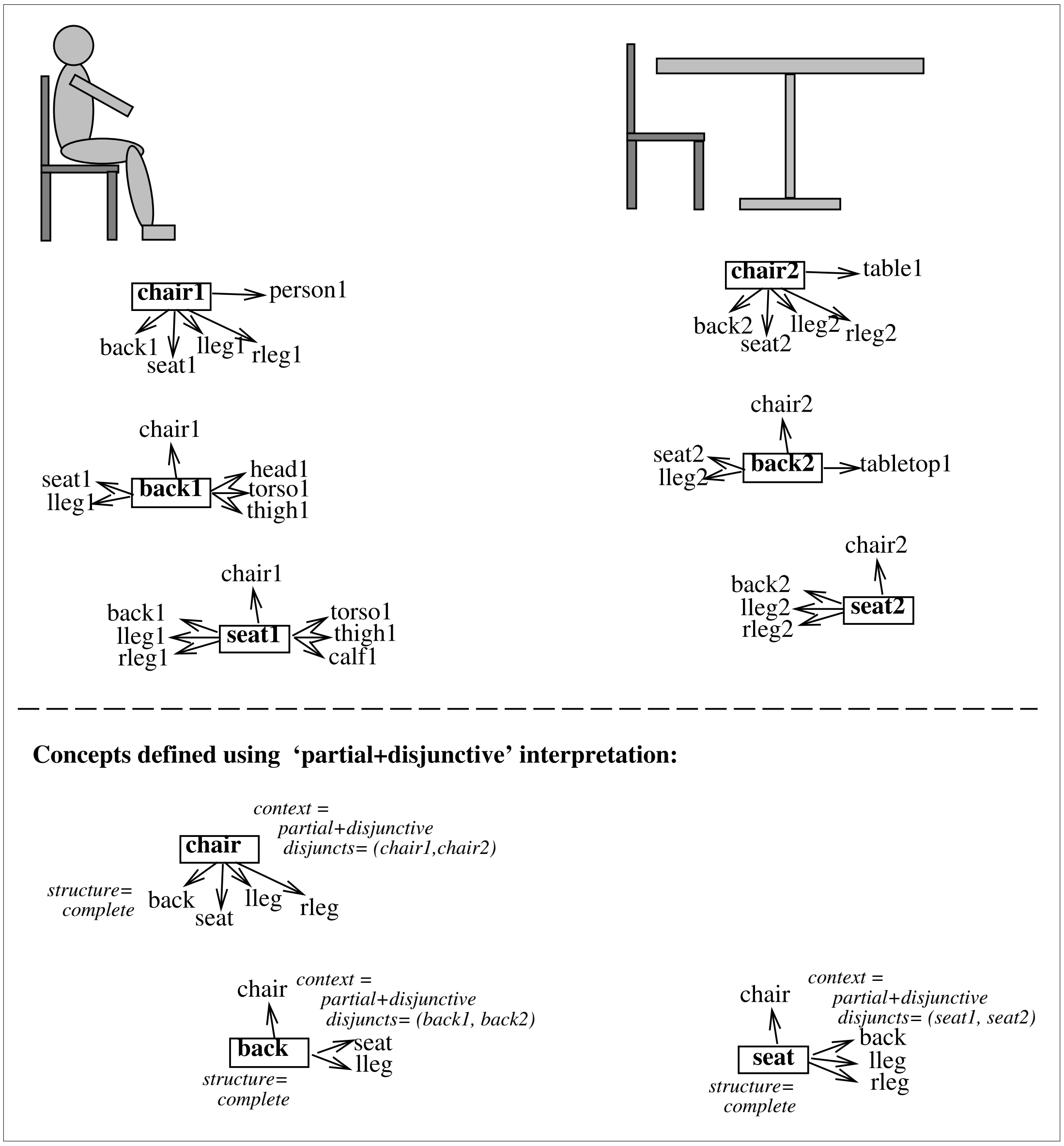


Figure 3.25: "partial+disjunctive" interpretation.

it does not specify the *degree* of arrangement-independence. This is better indicated by the variances of the subpart relationships. High-variance relationships contribute less to the similarity score computed by the matcher, and so the arrangement-independence is accounted for more accurately than by using a *contents-only* interpretation which would cause the matcher to either use all of the subpart relationships or none of them.

One more kind of structure or context interpretation is *any*, which is almost equivalent to a *partial* interpretation with no relationships specified, except that properties are also ignored.

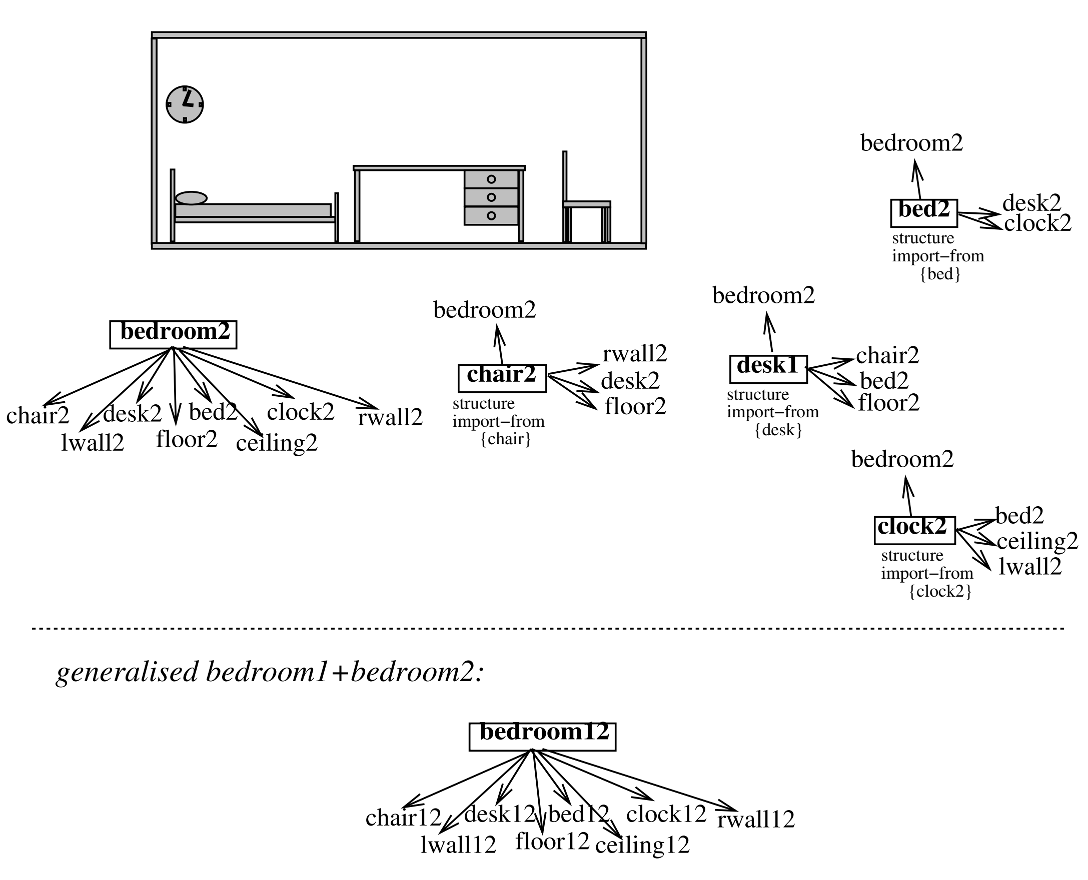


Figure 3.26: Structure may be characterised by *contents* not arrangement.

To summarise the structure and context interpretations discussed above, each is listed and briefly explained below.

Complete: An instance must match all of the concept's relationships, with no additional relationships.

Partial: The concept's relationships must match the instance's properties and relationships, but the instance may have additional relationships.

Disjunctive: An instance must match the structure [or context] of one of the disjuncts (*i.e.* the subconcepts).

Any: An instance can have any properties and relationships.

Partial+Disjunctive: An instance must contain the specified structure (or context) relationships, and must also match the structure (or context) of one of the disjuncts (*i.e.* the subconcepts).

Partial+Typical: [For grouped structures.] An instance must match the partial list of subpart relationships, and all of its subpart relationships must match the multi-relationship to the typical-member.

Partial+Imported: An instance must contain the specified structure (or context) relationships, and must also match the structure (or context) of one of the *import-from* concepts.

3.5 Groups

A requirement of the representation is to be able to explicitly represent groups of similar and similarly related components of an object. This section explains how this is achieved in the GRAM system.

A group object in GRAM is an ordinary concept or instance, with the usual structure and concept properties and relationships, but with the addition of a multi-relationship to a *typical-member* concept, and a few special properties that characterise the group as a whole. The typical-member concept is also an ordinary concept, except that it can have neighbour relationships *to itself*, and these capture the typical inter-member relationships between neighbouring members of the group, such as between neighbours in a row of bicycles. Other concepts (or instances, if the group has been formed within an instance-graph) may also have multi-relationships to the typical-member concept.

This scheme is very similar to the scheme initially proposed by [Winston, 1975], except that his system defined typical inter-member relations by relations from the typical-member node to an "another-member" node. Also, his system did not include multi-relationships, and dealt only with simple qualitatively-defined, non-disjunctive objects.

Before considering group representation in more detail, the following section describes the different types of group that commonly occur in the kinds of domain that GRAM is intended for, and should therefore be representable.

3.5.1 There are several types of group, distinguished by their inter-member relationships.

There are several different types of group, each distinguished primarily by the nature of their inter-member relationships. These are illustrated in Figure 3.27, and are explained below. Other examples of these group types can be found in the bookshelf in Figure 3.28. Figure 3.29 illustrates the typical-inter-member relationships for each type of group, depicted as self-referring relationships of the typical-member concept.

A *chain* is a linear sequence of similar parts.

The most common and important type of group is the *chain*, consisting of a linearly ordered series of similar parts for which the relationships between consecutive parts are also similar. For example, each row of books on the bookshelf in Figure 3.28 can be represented as a chain. Many examples of chains can be observed in the world, such as stacks of plates, rows of chairs, queues of people (not necessarily in a straight line), and chests of drawers.

In GRAM's representation, the typical-member concept of a chain group has two neighbour relationships to itself, one for each direction along the chain, as shown in Figure 3.29 (a).

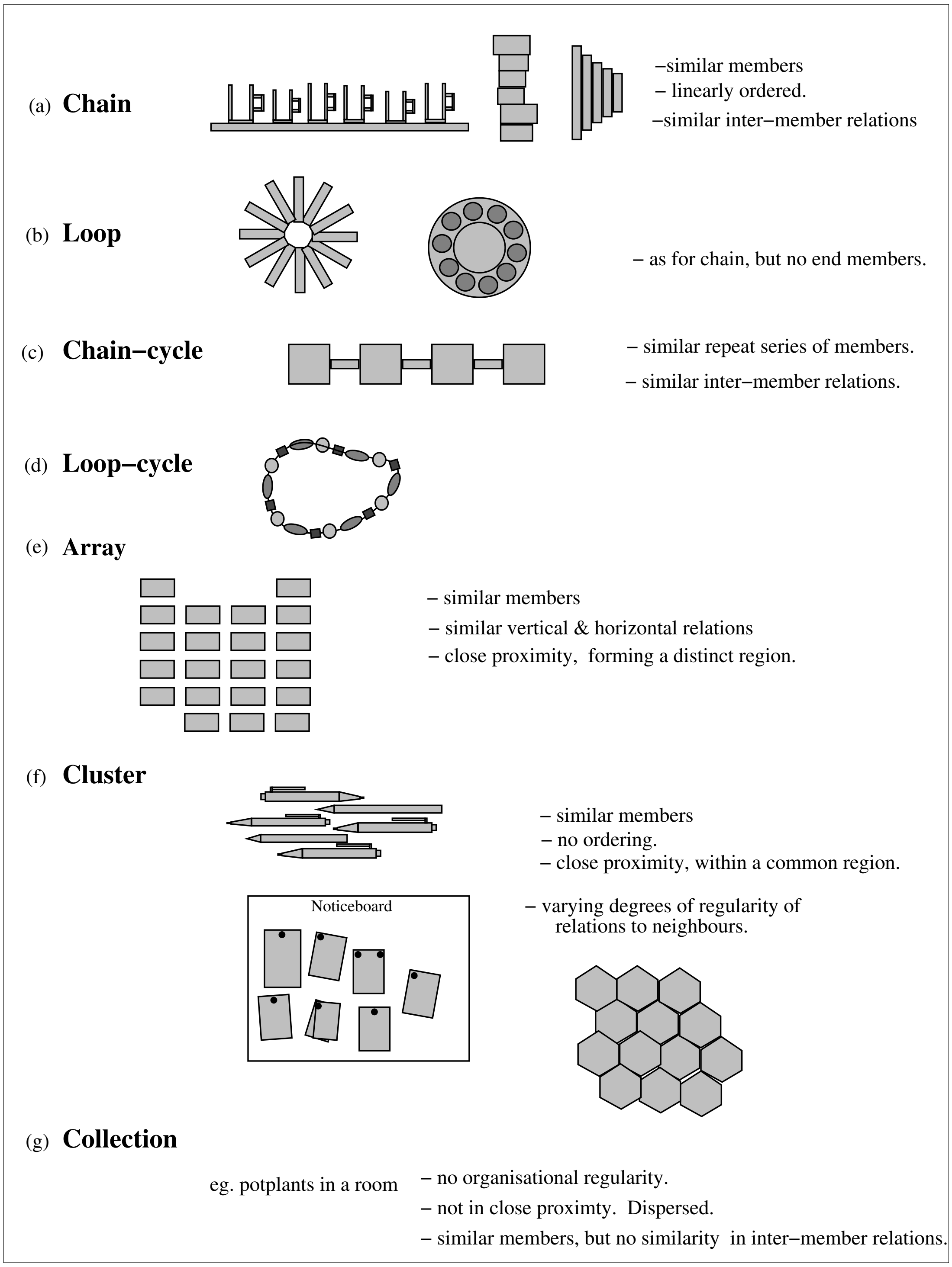


Figure 3.27: Types of Group.

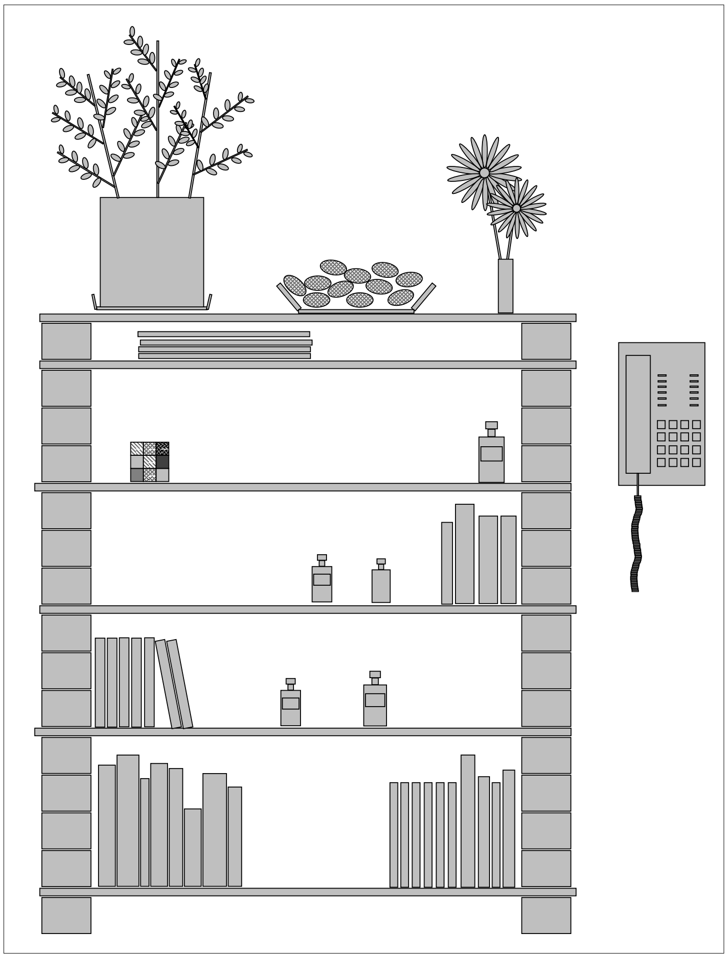


Figure 3.28: Examples of Groups.

A *loop* is a chain that is joined at the ends.

Loops are very similar to chains except that there are no end-most parts. For example, the petals on a flower and the hour-marks on a watch are loop groups. A loop is represented in exactly the same way as a chain, except that the grouped object has the value 'loop' for its *group-type* property. Also, *all* of the members contributing to the typical-member concept have inter-member neighbour relationships in both directions, while in the case of a chain all but the two end-members have, and so the instance-counts on the self-referring relationships will be one less than the instance-count of the typical-member concept itself.

An *array* is a vertically and horizontally aligned group of similar parts.

Another type of group is the *array*, whose members are organised in regular vertical and horizontal rows and columns, as in the case of the buttons on a telephone, or windows on a building. An array can be viewed as a chain of chains, in either of two alternative dimensions. However, GRAM can also represent an array explicitly by its typical member.

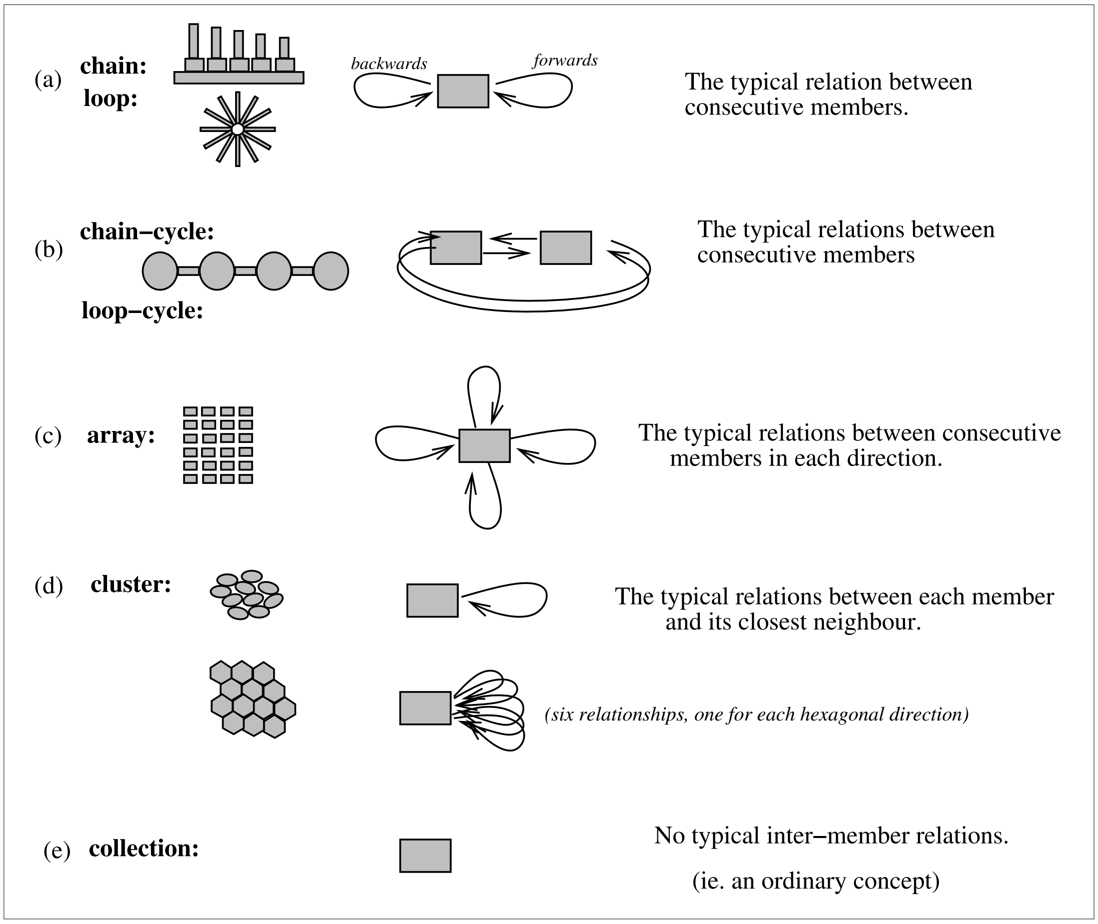


Figure 3.29: Typical inter-member relationships.

The typical-member of an array has at least four self-referring neighbour relationships, each for one vertical or horizontal direction, as illustrated in Figure 3.29 (c). Relationships on the diagonal are excluded by GRAM's group-constructor because they are significantly weaker neighbour relationships than the vertical and horizontal relationships.

A *cluster* is an unorganised group of similar parts clumped together in the same region.

A *cluster* is a collection of similar objects that are located in close proximity to each other within a region, but are otherwise unordered and unorganised structurally. The cookies on the bookshelf are an example of this. A pile of shirt buttons, a stack of bricks, a crowd of people, and the notices on a notice-board, are some other examples. Various degrees of regularity of inter-member relationships are possible, such as the highly regular honeycomb pattern in Figure 3.27 (d), or the irregular plate of cookies on the bookshelf, in which the only regularity is the distance between each cookie and its closest neighbour. These different forms of regularity are implicitly represented by the self-referring neighbour relationships of the typical-member concept, but are otherwise not made explicit as distinct group types, with the exception of the *array* type which can be considered to be a cluster that is explicitly distinguished.

The typical inter-member relationship for a cluster has a high variance, at least with respect to direction and position. For example, the neighbour relationships of the cookies on the bookshelf would be generalised into a single neighbour relationship for the typical-member that characterises the cluster of cookies, as shown in Figure 3.30 below. Each cookie has a significantly different set of neighbour relationships to its neighbouring cookies, some having two neighbours, others having four or five, in a variety of arrangements. Therefore, when the group-constructor merges the cookie descriptions into a single generalised typical cookie concept, the neighbour relationships are also merged into a single multi-relationship that has a generalised *howmany* count in the range 2 to 5 (with mean of 3), meaning that there are typically between 2 and 5 cookies that are explicit neighbours of a cookie in the bowl.

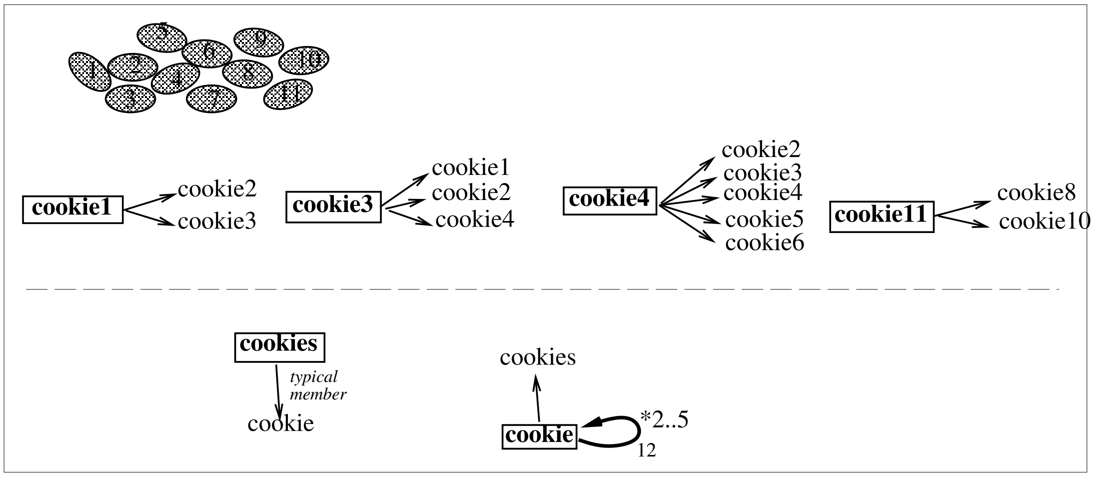


Figure 3.30: Inter-member relationships for a cluster.

A *collection* is an unorganised and dispersed group of similar parts.

A *collection* is a group of similar items that are not structurally related at all, other than being contained within the same object. This is an unusual sort of group because it cannot really be considered to be an 'object' in the sense that other groups can. It does not have any structural identity. Rather, the presence of a collection in the description of an object, means that "there exist several parts like this". For example, a description of room may include a collection-group of potplants, meaning that there are several potplants in the room.

Although 'collection' has been included as a distinct group type, GRAM does not actually construct group-objects for collections. Instead, it just creates a typical-member concept (which is really just an ordinary concept formed from instances within the same scene) and other objects can have multi-relationships to it. So a *room* could have a subpart multi-relationship to the *pot-plant* concept, with a *howmany* count of 6, while one shelf of the room may have a multi-relationship to the same pot-plant concept with a *howmany* count of 2. All the required information is available without requiring a group object. The only reason for having a group object is when the group as a whole has collective properties that are worth making explicit, such as the topology, overall shape, or inter-relatedness of the members.

Chain-cycles **and** *loop-cycles* **have two or more alternating typical members.**

A less common form of group is the *cycle*, which has a linear structure like a chain or loop but is characterised by *several* typical members that alternate in a cyclical manner along the chain or loop, as shown in Figure 3.27 (c) and (d). Cycles observed in everyday objects are most commonly characterised by just two, rather than more than two, alternating typical members. One of the typical members is often a connector between instances of the other, as in the case of a line of horses joined by ropes. The leaves on a (two-dimensional) stem can also be described as a chain-cycle consisting of a 'left' leaf and a 'right' leaf, as in the bookshelf in Figure 3.28. An example of a *loop*-cycle is a bead necklace with a cyclical pattern of different shaped or coloured beads.

It may seem that a cycle could just as easily be represented as an ordinary chain or loop. For example, the bead necklace in Figure 3.27 (d) could be described as a loop in which each typical member consists of a fixed sequence of distinct beads. Similarly, the bookshelf in Figure 3.28 could be represented as a chain with a typical member consisting of a shelf and a pair of brick-stacks. However, this is not possible in the case of a line of horses joined by ropes since there may be one fewer ropes than horses. It is more meaningful and correct to represent such a group as a cycle of several distinct typical members, where each has neighbour relationships to the other, as shown in Figure 3.29 (b).

The current version of GRAM does not support cycles because it would require group representation to be non-homogeneous, since the group must specify two typical-members, and the matcher must determine which corresponds with which when matching two cycles. Since cycles do not occur as commonly as the other types of group, this aspect has been left for future work.

3.5.2 Group properties.

This section and the following sections explain in more detail how a group is represented. A group is not actually an additional descriptive entity in the GRAM representation scheme, but is an ordinary instance or concept whose structure description includes some additional information – in particular, a multi-relationship to a typical-member concept, and several properties that characterise the group as a whole

The first group-property is the *group-type*, which can be any one of the types that were discussed earlier, such as *chain*, *loop*, *cluster*, *etc*.

The second group property is the *cardinality* of the group, which may be a generalised numerical value if the group is generalised. However, it is not actually necessary to include cardinality as an additional structure property, since it is already captured by the *number-of-subparts* property of an ordinary structure description.

The third property, or rather set of properties, are several measures of the *regularity* of the group, which is the inverse of *variance*. These indicate how regular the members are, in terms of (a) their structure, (b) their context, and (c) their inter-member relationships. These measures are used to indicate the strength or 'groupness' of the group, and to indicate which aspects of the group are most crucial during matching. In the case of a *collection* group, structure-regularity is usually high and context-regularity is much lower. In the case of a chain, context and inter-member relationship regularity is high, and structure regularity is also normally high but may be lower than in a cluster or collection, as in the case of a row of different shaped cups and glasses along a shelf.

3.5.3 The typical-member concept.

The typical-member concept of a group characterises not only the typical substructure of the members, but also the *context* of the members. The generalised typical context includes self-referring neighbour relationships that characterise the inter-member relatedness of the group, as illustrated for the various kinds of group topology in the previous section.

The typical-member concept usually also contains generalised typical relationships to non-member objects or concepts. For example, a *book* typical-member concept may have a generalised relationship to a particular bookshelf. In addition, a typical-member concept also includes a generalised relationship to the group-object as a whole. In the case of the book, this relationship indicates that each book is at roughly the same vertical position within, and roughly the same height as, the group of books as a whole.

A more complex example of a group is given in Figure 3.31, which shows the typical-member concept (*tm1*) for a sequence of chairs. The concept *tm1* has a parent relationship to the *chairs1* concept, neighbour relationships to the non-member *wall1* and *floor1* concepts, and neighbour relationships to itself indicating the 'next' and 'previous' chair in the chain. The number on each relationship is its instance-count.

The group constructor only forms typical inter-member relationships for the root part of the typical member concept, not for its subparts, and these are explicitly distinguished in the representation, as indicated by the heavy lines in the figure. Therefore, the neighbour relationships for the generalised *seat* and *rleg* in figure 3.31 do not include generalised self-referring relationships to represent the relationships with subcomponents of the neighbouring chairs. Originally GRAM was designed to allow this, but it lead to rather confused typical-member descriptions for which the matcher could not successfully disambiguate between inter-member and intra-member relationships.

3.5.4 A non-member object may have a multi-relationship to a typical-member concept.

Objects that are not members of a group, can also have multi-relationships to a typical-member concept. For example, the wall and floor in Figure 3.31 are both close to or connected to at

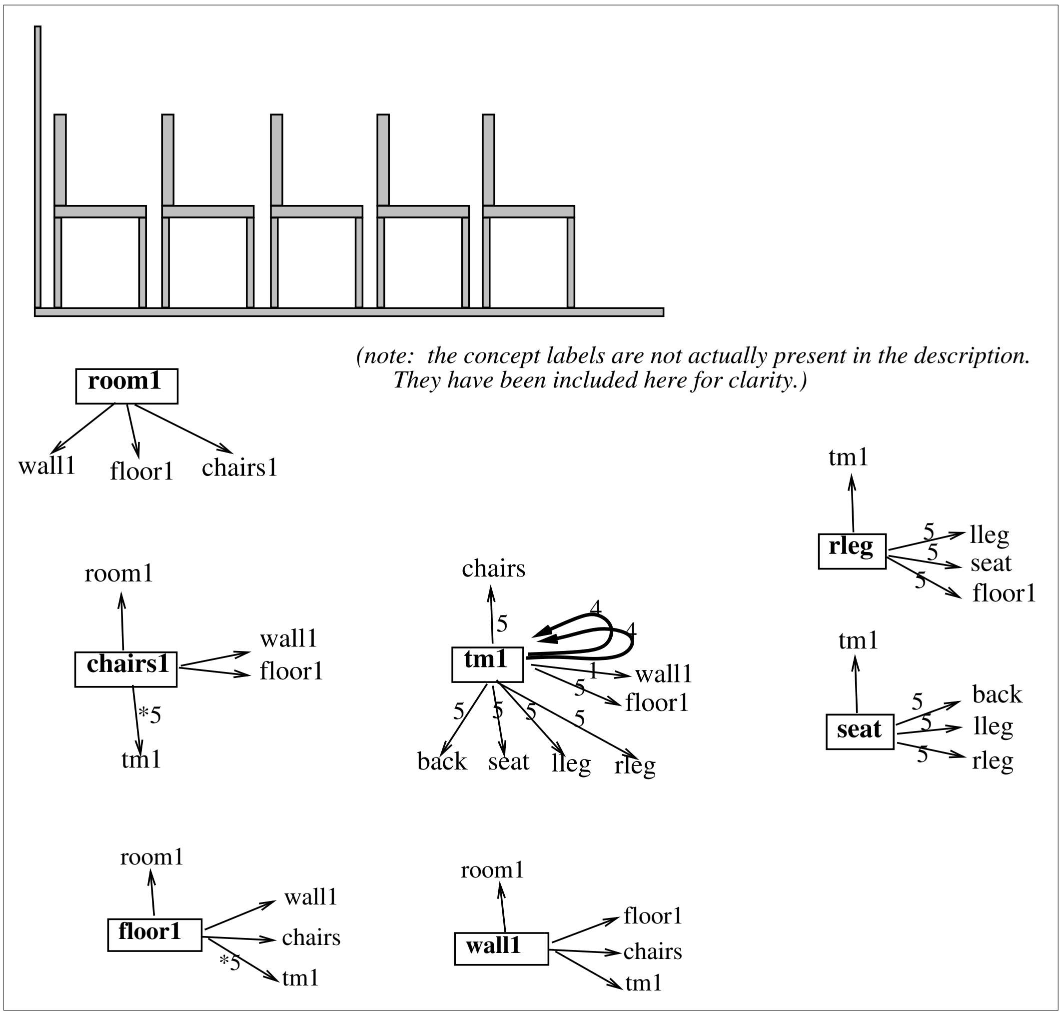


Figure 3.31: A typical-member concept of a chain

least one of the chairs, and so their context descriptions should include neighbour relationships to *tm1*, as well as neighbour relationships to the *chairs* object as a whole.

More specifically, *floor1* has a neighbour multi-relationship to *tm1* with a *howmany* count of 5, indicating that it is related to five chairs in the manner specified in the generalised relationship. It also has a relationship with the *chairs* group as a whole. *Wall1* is described in a slightly different way, since it is only directly related to the leftmost chair, and therefore has an ordinary (not multi) relationship to *tm1*. Notice that although this relationship represents only the relationship to the leftmost chair, the relatee of this relationship is the *generalised* chair, and thus the wall description has also been implicitly generalised.

3.5.5 Individual subparts of a grouped object may or may not be included in the description.

One of the advantages of representing a set of objects as a group is that the descriptions of the individual objects can be dropped from the description altogether. This not only reduces memory use, but it also makes it unnecessary for the matcher to find correspondences between members of two groups, which requires significantly more computational time than for just comparing two typical-member concepts, especially for very large groups. If the two groups have different cardinalities, then one-to-one correspondences cannot be found anyway, and a generalisation of the two groups must necessarily exclude individual members.

The *chairs* concept in Figure 3.31 above is an example of a concept which has no subpart relationships to any individual chairs, and is thus a more compact description. The description is also more generalised than if individual members were retained, since there is a transfer or sharing of information amongst the now-implicit members.

However, it may be important to retain individual member descriptions if they are sufficiently distinct from other members that to remove them would be an over-generalisation. This situation can take two forms:

Firstly, the group might be quite a weak group, with all of the members being quite distinct from many or most of the other members, such as a pencil case containing a variety of different pens and pencils. A grouping may be justified, but the removal of the descriptions of each of the members may not be. GRAM's group-constructor removes all individual members only if a group has more than a certain number of members and if all the members are sufficiently similar. Currently GRAM has fixed cutoff levels for this, but in a complete robot system the decision would depend on the task being performed: If space and speed are most important, individual members can be dropped even if the generalised typical member loses quite a lot of the details of the individual members. If accuracy and detail are important, individual members are only removed if the typical-member concept is a very strong (*i.e.* low-variance) generalisation. The chair in Figure 3.32 is an example where the individual members are *not* removed, since there are only a few of them, and their contexts differ sufficiently that member removal would lose too much information.

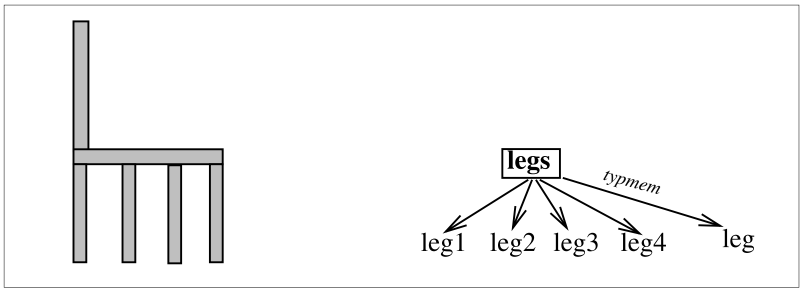


Figure 3.32: Individual members may be retained.

The second situation for which members could be retained is when there are a few *atypical* *members*, where atypicality is measured simply by the similarity of the member and the typical-member concept. The most common occurrence of an atypical member is the end-most member of a chain. For example, the end-most chairs of the row of chairs in Figure 3.31 are atypical because they do not have chairs on both sides of them. They could therefore be retained in the *chairs* description as explicit subparts (although for the purposes of the earlier discussion they were not included).

As another example, the leftmost and rightmost balls in Figure 3.33 are distinct because they have a side of the box next to them and only have one neighbouring ball. Therefore they have been retained as subparts of the *balls* concept, and the structure is given a *partial+typical* interpretation.

This achieves the same effect as Michalski's [Michalski, 1980] group representation which allowed a 'LST' and a 'MST' member, meaning the first (LeaST) and the last (MoST) members of the group respectively. However, GRAM's method is more general because atypical members are not restricted to being at the ends. For example, the distributor cap in Figure 3.34 shows a group that has a single atypical member in the middle, rather than at the ends. Other examples of groups with atypical members include a desk whose top-drawer has a keyhole, a bad apple in a box of apples, and the space-bar on a keyboard.

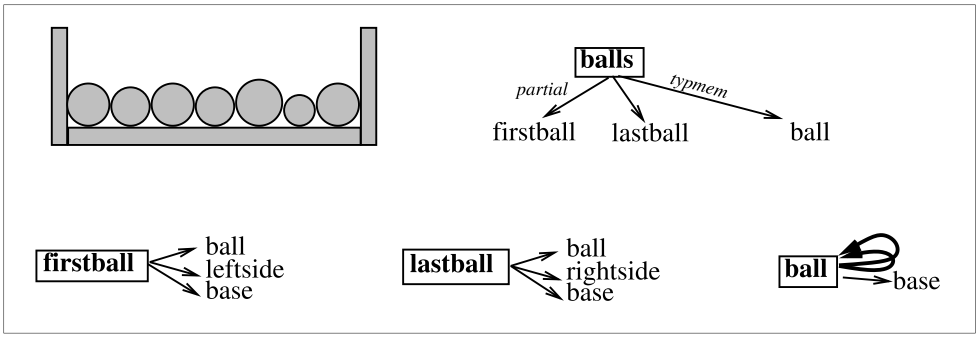


Figure 3.33: Atypical end-most members.

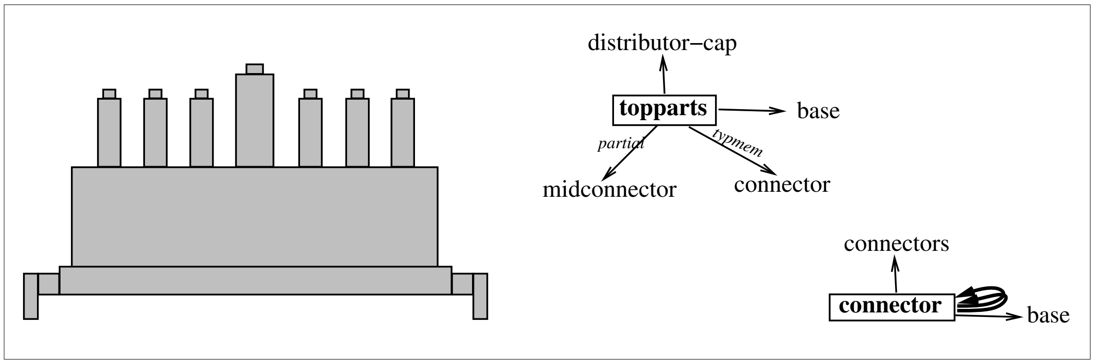


Figure 3.34: Atypical members.

In the above examples, the generalised typical-member was shown with only the *typical* relationships. For example, the *ball* concept in Figure 3.33 has a neighbour relationship with the *base* but not with the left or right sides of the box. This is because GRAM's group-construction mechanism removes atypical relationships from the generalisation so that the typical-member concept is not cluttered with low-frequency relationships.

One situation that forces the removal of individual members is when matching and generalising two grouped concepts that both contain complete sets of subparts which have different cardinalities. In such a situation it is not possible to establish unambiguous one-to-one correspondences between the subparts, and so the generalised grouped concept cannot include a complete set of individual generalised subparts, unless some are marked as optional. A partial set of members could be included, however. Each of these would be a generalisation of the distinct, unambiguously matched members of the two contributing groups. This is illustrated in Figure 3.35, where groupA has 4 subparts, groupB has 6 subparts, and the generalisation of them is described only in terms of a single multi-relationship to the generalised typical-member concept, and the two relationships to the generalisation of the pairs of end-members.

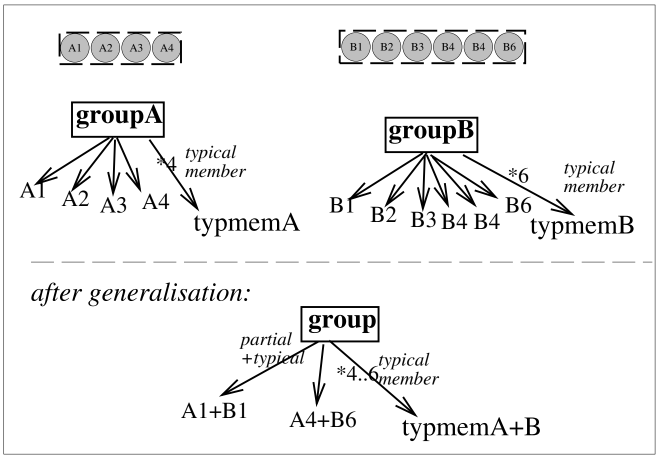


Figure 3.35: Members may be removed during generalisation.

3.5.6 The structure of a typical-member concept may be imported from another concept.

In many situations the members of a group are instances of some existing concept, as for chairs in a room, shoes in a cupboard, or books on a bookcase. Therefore it may be unnecessary to specify the entire substructure of the typical-member concept, but rather to import from an existing concept, as illustrated in Figure 3.36. The *context* description must be specified locally, so that the organisation of the group is explicit. Therefore, a typical-member concept normally only has its structure imported, not its context, since otherwise the organisation of the group as a whole, specified by its typical inter-member relationships, would be lost. This is also why the group-object cannot be described simply by a multi-relationship directly to the *chair* concept itself.

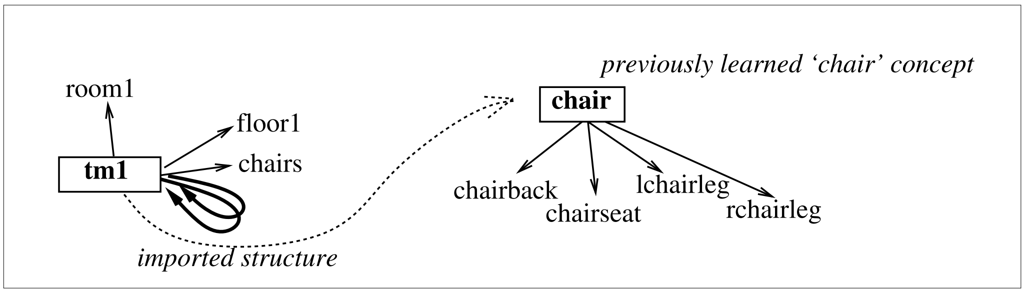


Figure 3.36: Typical-member structure may be imported.

3.5.7 The structure of a typical-member concept may be disjunctive.

Section 3.4.2 showed how the structure (or context) of a concept may be described *disjunctively*. Since a typical-member concept is just like any other concept, it can also be described disjunctively. For example, in Figure 3.37 the grouped concept *things* refers to the typical-member concept *thing* whose structure is described disjunctively by referring to its subconcepts *blidget* and *plidget*.

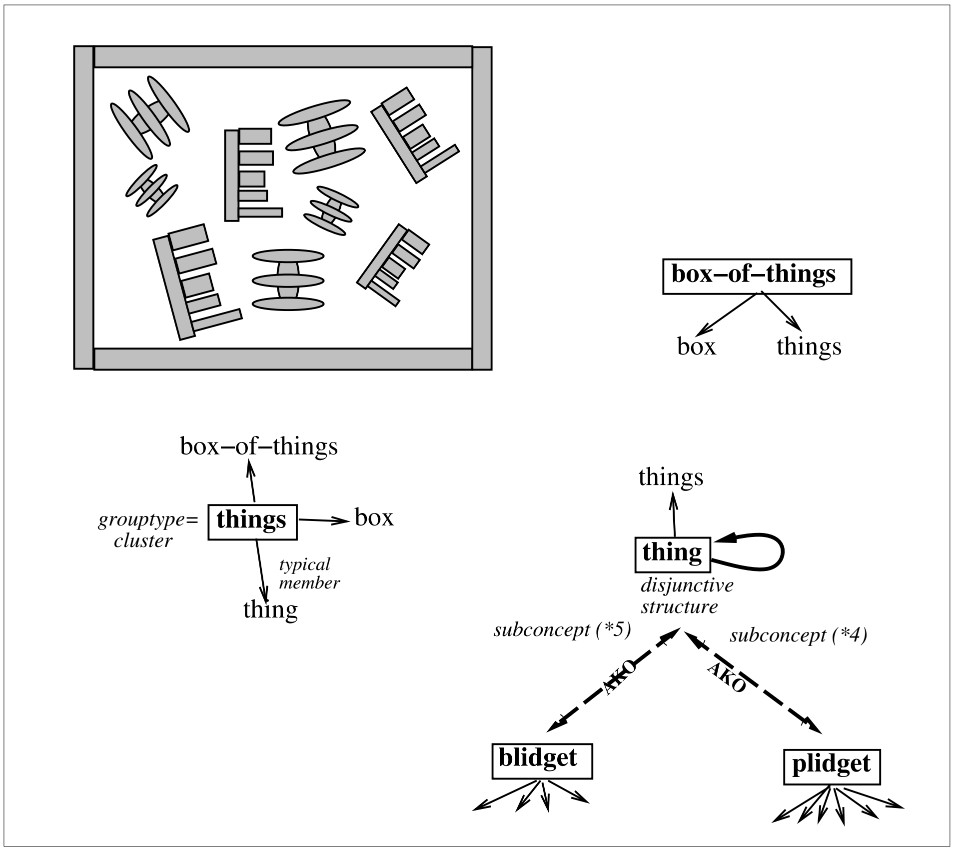


Figure 3.37: A disjunctive typical-member concept

A typical-member may also be defined disjunctively by a disjunctive *import-from* specification, as shown in Figure 3.38 where the structure of the typical-member of the items on the shelf is imported from either the *wineglass* concept or the *mug* concept.

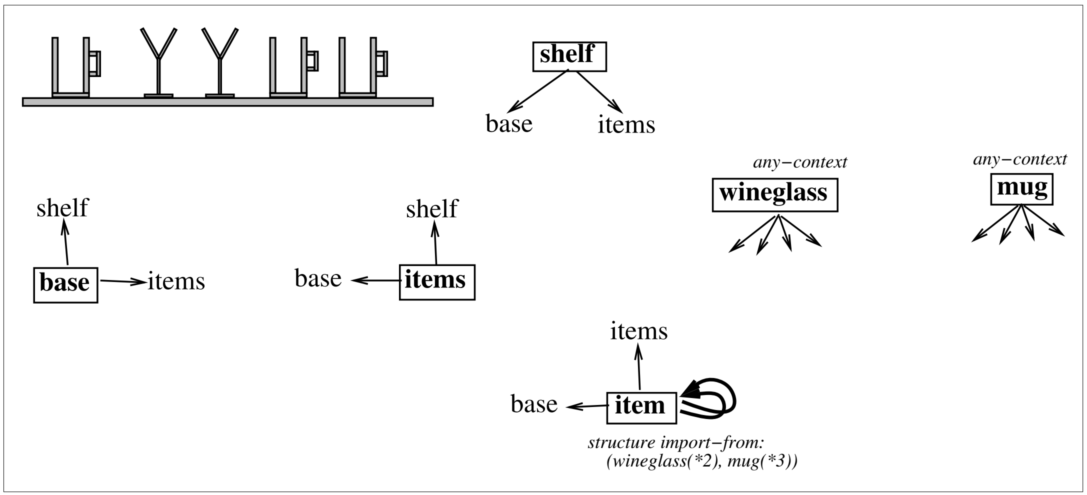


Figure 3.38: Disjunctive importing.

Instance-counts associated with disjuncts (*i.e.* subconcepts) or concepts in an *import-from* specification, indicate how many members are of each type. For example, a typical-member concept representing the pens and pencils in a pencil-case may have a count of 8 for the pen disjunct, and 2 for the pencil disjunct, indicating that 80 percent of the group members are expected to be pens.

Unfortunately, when two pencil cases are generalised, the instance-counts of the pen and pencil disjuncts might no longer capture the expected frequency in an instance. For example, if the second pencil case had 8 pencils and 2 pens (*i.e.* the inverse of the first pencil case) then the generalised typical-member concept would have an instance-count of 10 for the pen disjunct and 10 for the pencil disjunct, indicating that a pencil case is expected to have 50 percent of each. For an ordinary concept this is not a problem, since we *do* want the concept to state that fifty percent of the instances of the concept have been pens, and fifty percent have been pencils. But groups are a little different. The current GRAM representation has no way of explicitly representing the proportion of each disjunct for the typical-member of a group, although extending the representation to include this would not be difficult.

3.5.8 Groups of groups

The typical-member of a group may itself be grouped, as illustrated in Figure 3.39 which shows a row of stacks of blocks. The typical-member concept, *stack*, is a generalisation of the concepts *stack1*, *stack2*, *stack3*, and *stack4*. Thus the typical-member of *stack*, called *item*, is a generalisation of the typical-member concepts *item1*, *item2*, *item3*, and *item4*, and is therefore a generalisation of all 19 rectangular blocks.

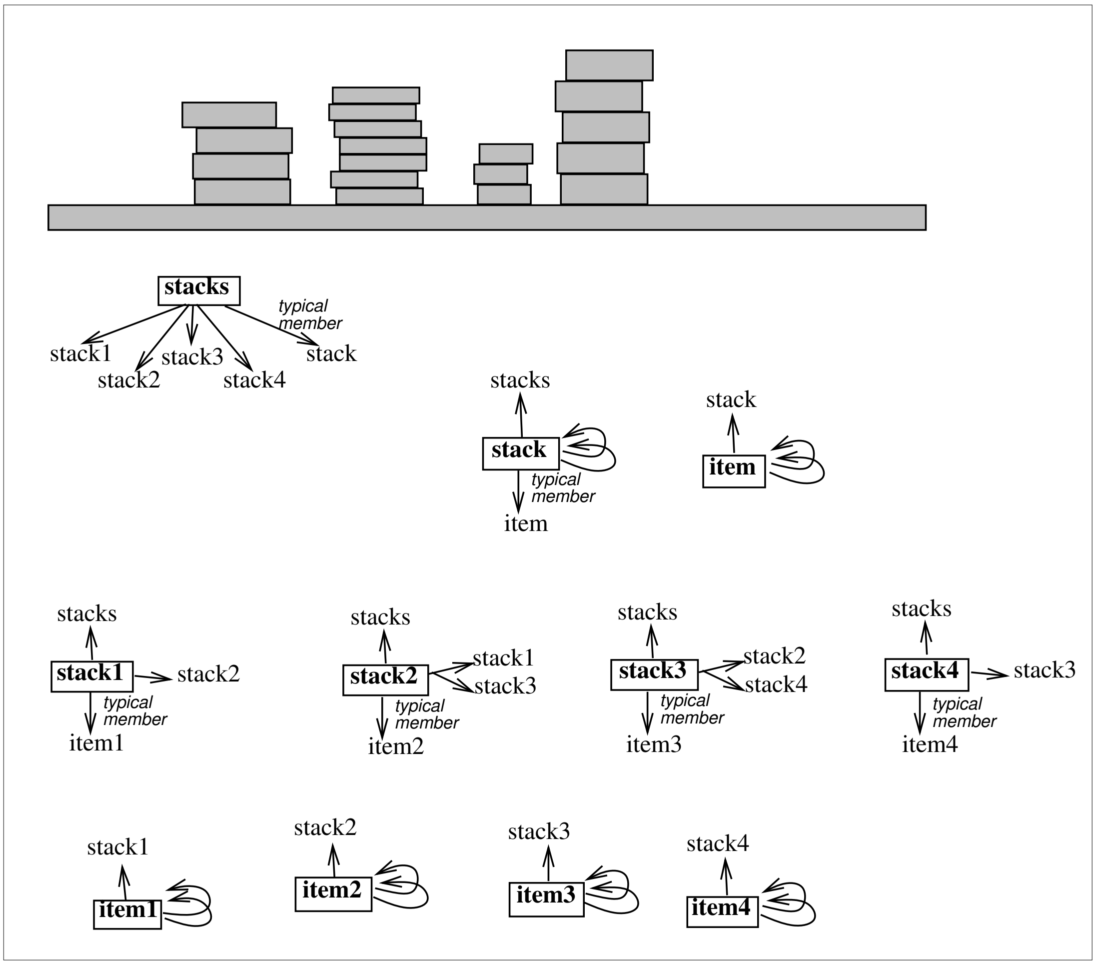


Figure 3.39: Groups of groups.

Another more complex example of groups containing groups is given in Figure 3.40. This also shows how the representation allows parts to be grouped in a number of different and overlapping ways, although the current group-constructor is not (quite) able to build such a description.

The *shelves* concept is a grouping in which the typical shelf contains a *shelfbase* and a grouped-concept called *shelfcontents*. The typical member of shelfcontents is a generalisation of all 18 glasses and mugs. Each *shelf-item* is described using *import-from* specification consisting of a disjunction of the *wineglass* and *mug* concepts.

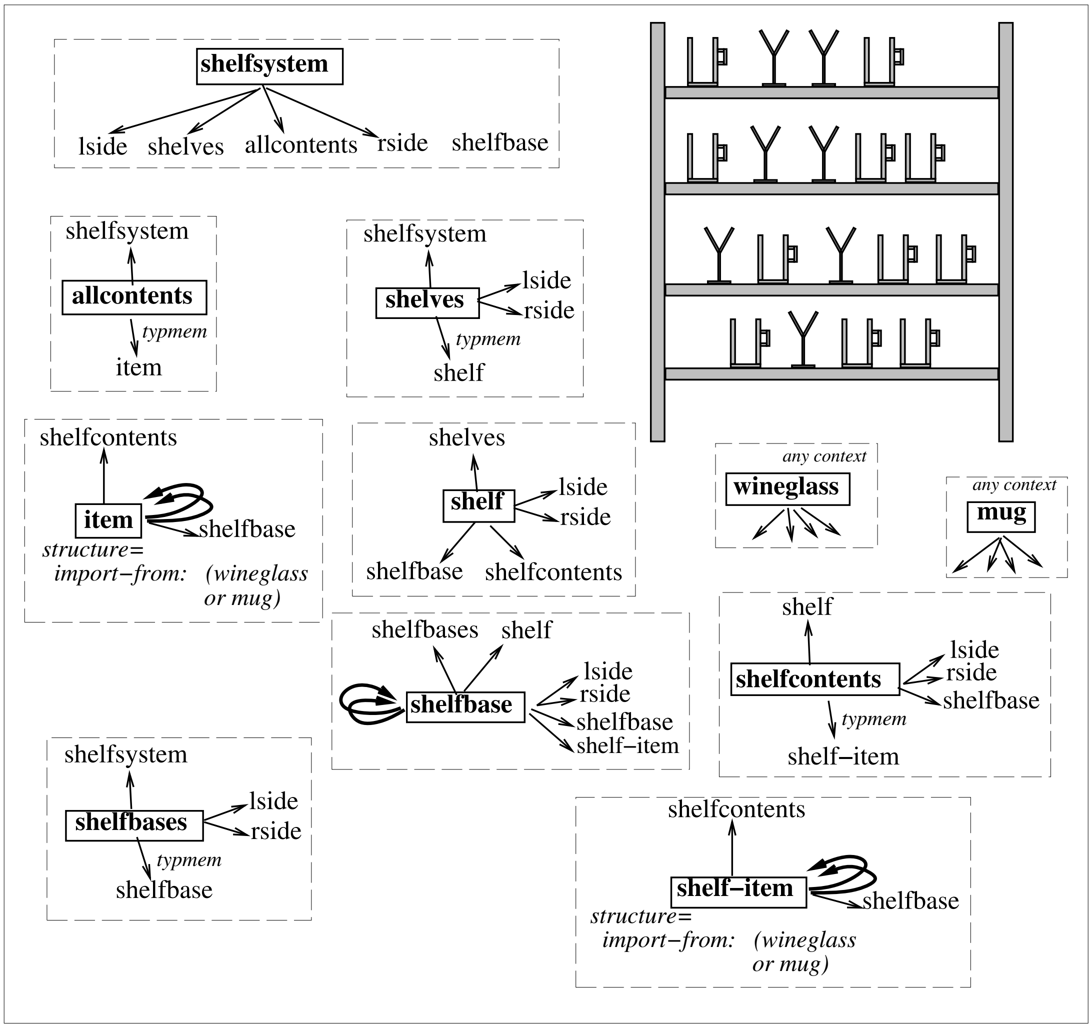


Figure 3.40: Groups of groups, and relationships between groups.

The *shelfsystem* concept also contains another grouped concept, called *allcontents*, which represents all of the glasses and mugs as a grouping (independent of the shelves). This grouping, could also refer to the *shelf-item* concept, but in fact refers to a separate concept called *item*. This is because *shelf-item* is created by the generaliser when it generalises the four shelves, while *item* is created from the grouping of all 18 mugs and glasses earlier in the instance-construction process.

Another grouping called *shelfbases* consists only of the *bases* of the four shelves.

Thus the three grouped concepts, *shelves*, *allcontents*, and *shelfbases* are alternative ways of grouping the components of the shelf-system.

Typical-member concepts may have neighbour relationships with other typical-member concepts.

Figure 3.31 on page 111 showed a neighbour relationship from the *wall1* concept to the typical-member concept *tm1*, and also showed a multi-relationship from the *floor1* concept to *tm1*. It is also possible for a typical-member concept to have a relationship to another typical-member concept. For example, the typical-member of the *shelfbases* group (*i.e.* *shelfbase*) has a neighbour multi-relationship with the typical-member of the shelf-items concept (*i.e.* shelf-item), with a generalised *howmany* count of 4..5. This indicates that a typical *shelfbase* has 4 or 5 *shelf-items* on it.

3.5.9 Generalised groups.

Most of the examples given so far have been of ungeneralised groups, as found in instance graphs. We now consider groups that are generalisations of two or more groups from within instance graphs.

Firstly, the properties characterising a generalised group as a whole have generalised values. For example, the cardinality (or rather, the number-of-subparts property) specifies a mean, standard deviation, and range, to indicate the permissible sizes of instances of the group. Likewise, the measures of regularity are generalised numerical values, and the *group-type* property is a generalised nominal value which could specify multiple group types.

Secondly, the generalised group refers to a generalisation of the typical members of the instance groups. Thus the formation of a generalised group from two instance groups is one situation where the matcher and generaliser have to deal with two concepts, rather than a concept and an instance, or two instances, since both typical-members are concepts. The generalised group is always as 'weak' or weaker than the strongest original group because the self-referring relationships of the two typical-members are combined, and may lose some of the structure of the original groups. For example, if an array group is generalised to cover a cluster group, the result will be a cluster, since the four self-referring relationships characterising the array will be merged into the self-referring relationship(s) characterising the cluster. This is discussed in chapter 5.

A generalised group should retain the significant constraints and regularities common to the contributing instance groups. To explore this issue, consider the two grouped concepts A1 and B1 on the left of Figure 3.41 whose typical-members have aspect-ratios [3] of 1:4 and 1:2 respectively.

[3] the width-to-length ratio

The generalisation of A1 and B1 results in a grouped concept AB1 with a typical-member concept AB1t, as shown on the right of the figure. The aspect-ratio of this generalisation is variable, in the range 1/4..1/2. It may seem that the generalised description no longer captures the constraint that all of the blocks must be of the same height. In other words, the matcher might consider object C (in the Figure below) to be a valid member of the concept.

However, the inter-member relationships for both A1t and B1t specify that consecutive blocks must be of the same height, and this constraint is *not* lost through the generalisation process, and hence object C would be considered a mismatch. This is an example of the principle that redundancy helps prevent loss of important constraints through generalisation.

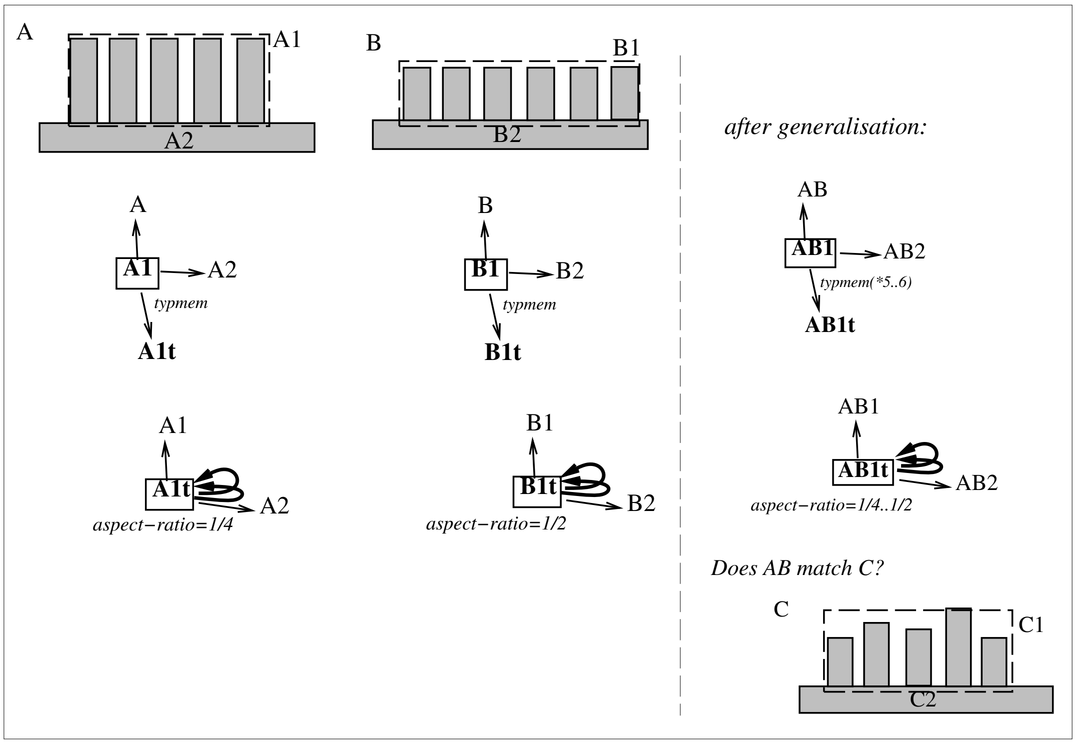


Figure 3.41: Maintaining regularity constraints in generalised groups.

Now consider the set of shelves, D, in Figure 3.42. When the instance-constructor is creating the typical *shelf* description, it has to generalise each of the four shelves. In doing so it will have to generalise the four groups of *shelf-items*, and this will result in a typical item whose structure is defined disjunctively in terms of *wineglass* and *mug*. Thus it may seem that the generalised shelf no longer captures the constraint that all of the items on each shelf should be of the same type. Thus the set of shelves, E, would perhaps be considered a near-perfect match with D, since the mismatch in terms of irregular shelf contents could not be noticed.

However, as in the previous example, other information in the description prevents this constraint from being lost, this time in the form of the *measures of regularity* of the group itself. In other words, each of the generalisations of the shelf-contents has a high measure of *structure-regularity* since all items are identical. When the four shelves are generalised, this

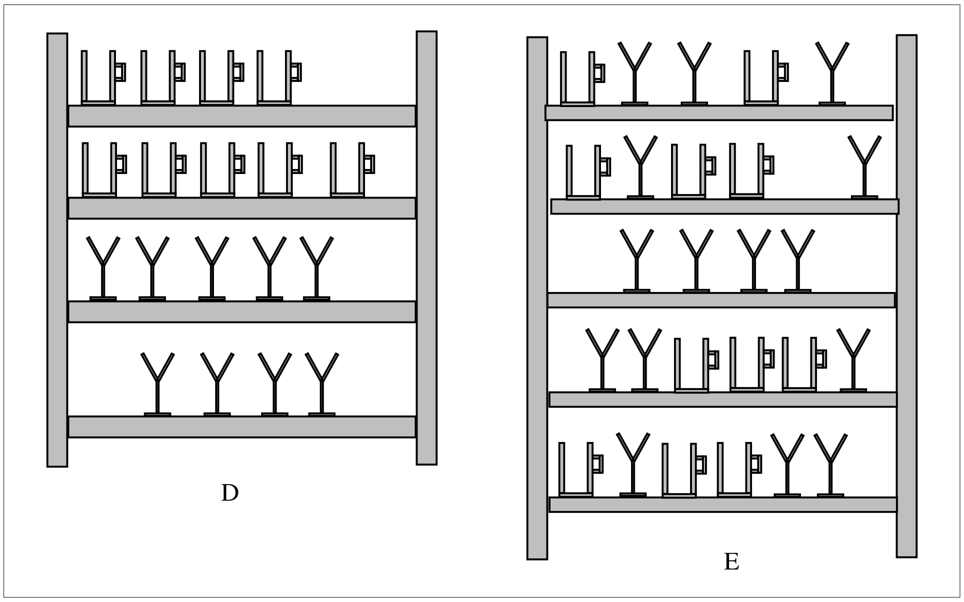


Figure 3.42: Maintaining regularity constraints in generalised groups.

constraint is common to all of them, and is therefore not lost. When matching against the set of shelves E, which does not have a high structure-regularity for the shelf-contents, the mismatch can be noticed.

In addition, if GRAM was to include a *similarity* attribute in the description of a neighbour relationship, then this information would also be present in the generalised typical inter-member relationships, and would reduce the information loss due to the generalisation. This again illustrates the importance of including redundant information that specifies the same characteristics of an object in different ways, each of which makes some aspects more explicit than in the other ways.

3.6 Reference summary of the representation scheme.

This section gives a reference summary of the components of the representation scheme described in this chapter.

Object: *(instance or concept)*
- Super-concepts.
- Sub-concepts.
- Instance-count. *(1 if ungeneralised)*
- Instance-count. *(1 if ungeneralised)*

Context:
- Properties. *(connectivity-profile, etc). (See page 82)*
- Parent-relationships.
- Neighbour-relationships.
- Interpretation. *(Complete, Partial, Disjunctive, Any, Imported, Partial+disjunctive, Partial+imported)*
- Import-from specification. *(a disjunctive list of concepts)*
- Variance.

Structure:
- Subpart-relationships.
- Interpretation. *(Complete, Partial, Disjunctive, Any, Partial+disjunctive, Partial+typical, Partial+imported)*
- Properties. *(shape, aspect-ratio, etc.) (See page 81)*
- Typical-member concept. *(if grouped)*
- Group properties. *(type, regularity measures)*
- Import-from specification. *(a disjunctive list of concepts)*
- Variance.

Relationship: *(parent, neighbour, or subpart)*
- Instance-count. *(1 if ungeneralised)*
- Attributes. *(relative orientation, direction, etc). (See pages 83 and 85)*
- Relatee. *(an object)*
- *Howmany*-count. *(1 if not a multi-relationship.)*
- Variance.

Chapter 4

The Matcher

The matcher is the central component of the GRAM system. This is because the generaliser, classifier, fault-finder, and group-finder all rely on its results to determine their behaviour. This chapter considers the issues of matching complex structured objects and describes GRAM's matching algorithm.

Section 4.1 identifies the requirements of the matcher, based on the characteristics of the domain and task discussed in chapter 1. Section 4.2 then gives an overview of the issues of the matching problem, and outlines the main contributions of the GRAM matcher.

Section 4.3 explores what it means for two structured objects to be 'similar', and defines the scoring scheme used in GRAM.

Section 4.4 considers the issues of how to *search* for the best correspondences between the parents, neighbours, and subparts of two objects being compared, and describes GRAM's "incremental-spread" matching algorithm.

4.1 Requirements.

4.1.1 Input and output requirements.

The input to the matcher is two generalised or ungeneralised object descriptions[1], and the output is a description of their similarity. This description must specify (a) the overall similarity score, (b) structure and context similarity scores, and (c) the similarity scores for the best (and "almost best") correspondences between their parent, neighbour, and subpart relationships and relatees.

The distinction between structure and context similarity is necessary because generalisation may be justifiable on the basis of a high score of just one or the other. The distinction is also necessary to enable the generaliser to determine when to create a disjunctive structure or context description. The fault-finder also needs both scores to be able to make reports such as "that X is in an unusual location".

Similarity scores for relationship and relatee correspondences are required by the generaliser to determine whether individual relationships should be generalised, or whether a relationship should be considered unmatched. Likewise, the fault-finder can use these scores to identify the specific differences between a concept and an instance, since each low score indicates a particular fault or unusual feature. If there is ambiguity in selecting the best correspondences between parents, neighbours, and subparts of the objects, the "almost best" correspondences should also be included in the output so that the generaliser can create multi-relationships or groups if necessary.

An additional kind of similarity score that has not yet been implemented is *contents-similarity* which measures the similarity of the subparts of two objects, independent of their arrangement. This could be used by the generaliser to justify generalisation. For example, two bedrooms may have a high contents-similarity score, and this could justify creating of a new concept defined by a set of subpart concepts with a highly variable arrangement.

4.1.2 An 'any-time' matcher with effort-control and scope-restriction parameters is required.

Since the GRAM matcher is to be eventually used in a real-time robot system interacting in a complex physical environment, it must provide means for controlling how much effort is applied to a match, so that rapid and approximate comparisons can be performed when efficient classification is crucial (such as when walking through a building), and slower detailed comparisons can be performed when accuracy is crucial (such as when performing quality control on a product). Thus the matcher must have some kind of *effort-control* parameters.

Also, the matching process should not have to be continued until 'completion' before a similarity score is available. Rather, the matcher should employ a robust 'any-time' algorithm that can be unexpectedly interrupted and still provide a usable estimate of similarity.

Sometimes it is necessary to focus the attention of the matcher towards a particular portion of an observed scene or object. For example, when comparing an observed item with a concept in

[1] Throughout this chapter, the term 'object' is used to refer to either a concept or an instance.

memory, the task might only require the *structure* of the item to be considered, not its context. Thus the matcher should provide a means for specifying what portion of the instance graph is to be matched. This can be called *scope-restriction*.

4.1.3 The matcher should not assume canonical descriptions.

The GRAM matcher should not assume canonical object descriptions. In particular, it should not assume a canonical subpart decomposition hierarchy, and it should not assume that the same relationships are made explicit in both descriptions. The instance-constructor justifies creating a relationship or a composite object on the basis of a *feature-utility* score defined in terms of various criteria. If, for a potential feature of an object-graph, this score is just below the required threshold, but the score for a corresponding feature in the other object-graph is just above the threshold, then the two resulting descriptions will differ, even if the objects are very similar.

The matcher should also not assume that a description is complete, since an object may be partially obscured or only partially observed. Therefore it should be able to cope with partial information, perhaps requesting the instance-constructor for more information when required, if it is available.

4.1.4 The objects being matched may be generalised or ungeneralised.

The matcher is primarily used to compare an observed instance with an existing concept in memory, where the concept may either be generalised or (if only one instance of it has been observed) ungeneralised. However, the matcher must also be able to compare two instances or two generalised concepts. Instance–instance comparison is required when matching two objects within the same scene, such as when forming groups. Concept–concept comparison is necessary when comparing the generalised relatees of two multi-relationships, or when the larger concept-learning system needs to compare two concepts in memory to reorganise the AKO hierarchy.

Many of the examples in this chapter deal with comparing an instance (or ungeneralised concept) with another instance, since it is difficult to depict generalised object descriptions, and it is only necessary to address the concept–instance situation for particular issues, such as disjunction matching.

4.1.5 Two types of scoring are required: *fit-scoring* and *proximity-scoring*.

The matcher should be able to produce two types of similarity scores: *fit-scores* and *proximity-scores*. A proximity-score should measure how close an instance is to a concept, based on an absolute measure of what ‘close’ means. A fit-score, on the other hand, should measure how close an instance is to a concept, based on the variance of the concept. For example, the proximity-score for an observed swivel-chair with respect to the concept *standard-chair* should be high, but its fit-score should be low (assuming that all instances of *standard-chairs* have had four legs), since it is very atypical.

This distinction is necessary for the generaliser to determine how to incorporate a new instance into concept memory. If an instance has a high fit-score with respect to an existing concept, then that concept can be modified to cover the instance. Otherwise, if the fit-score is low but the proximity-score is sufficiently high, then a new concept can be created that is a generalisation of the original concept and the instance, without affecting the original concept.

Fit-scores are also necessary for fault-finding, since they indicate when a feature is atypical. Proximity-scores are required by the matcher for finding correspondences between objects.

4.2 Issues and Contributions.

This section discusses various issues in the design of a matcher that satisfies the requirements given in section 4.1, and outlines the main contributions of the GRAM matcher.

4.2.1 The two primary issues are *similarity* and *search*.

The design of the matcher can be characterised by two issues: firstly, the meaning of 'similarity' must be defined, and secondly, an algorithm for evaluating similarity must be developed. The first issue is declarative, since it involves defining a formula for measuring the similarity of two objects, on the assumption that the best correspondences between parents, neighbours, and subparts are known. The second issue is procedural, since it deals with how to search for the best correspondences between parents, neighbours, and subparts, and how to actually compute similarity scores.

These two issues (addressed in section 4.3 and section 4.4 of this chapter) are inter-dependent, since the problem of finding the best correspondences is based on measures of similarity, and measures of similarity are based on a chosen set of correspondences.

4.2.2 Object similarity evaluation is complex and recursive.

Since an object is defined in terms of other objects, within a potentially vast object graph, similarity evaluation is recursive and complex. In order to determine the best correspondences of parents, neighbours, and subparts of two objects, many other pairs of objects may have to be compared. The process of matching two objects could involve an expensive and complex search through the object graphs, spreading up, out and down through parent, neighbour, and subpart relationships. Therefore, to ensure efficiency, the matcher's search strategy must avoid unnecessary comparisons whenever possible.

The GRAM matcher does this by employing a breadth-first beam search using iterative deepening: The matcher initially compares two objects using minimal information (just their properties and relationships, ignoring their relatees), and then applies more effort to the comparison by comparing the properties and relationships of corresponding relatees, and then comparing the relatees of those relatees, and so on, incrementally extending the 'horizon' of the match, abandoning poor correspondences whenever possible. Thus, the object-graphs guide and constrain the search using an "incremental spread" approach. [Connell and Brady, 1985] employed a similar scheme, although it only operated top-down from coarse details to fine details, while GRAM operates in any direction through the object graphs, and uses a more elaborate similarity scoring scheme.

Since object graphs usually contain circularities, the matcher must avoid re-evaluating comparisons that have already been performed, or are currently being performed. GRAM does this by keeping track of the level of spread effort that has already been applied to a comparison, and only re-invokes the matcher if a greater level of spread effort is required.

An additional complication is that since the similarity of two objects is defined in terms of the similarity of other relatee objects, which are defined in terms of the first two objects, a similarity score is recursively defined in terms of itself. Since there is no base-case for this recursive definition, GRAM must rely on computing estimates. Its "incremental-spread" algorithm enables recursively-defined similarity scores to converge on a reasonable estimate.

The incremental-spread approach also satisfies the requirement that the matcher be an *any-time* algorithm, since it can be interrupted at any point and still provide a reasonable estimate of similarity. In fact, [Bergevin and Levine, 1993] have demonstrated that many objects are recognisable on the basis of coarse features alone, and thus even if the GRAM matcher spreads by just one step via parent, neighbour, and subpart relationships, this will often be sufficient for obtaining a good estimate of similarity. Further spreading will just refine the score, since the distance through the relationship network is inversely proportional to the importance of those features to the measure of similarity.

4.2.3 Requiring a globally consistent set of correspondence is expensive and unnecessary.

In a usual graph match, a comparison is performed by finding a globally consistent set of one-to-one correspondence bindings, meaning that the evaluation of each correspondence is dependent on a particular selection of other correspondences. This is an expensive process: If the two graphs each contain n objects, then $n!$ sets of correspondences need to be evaluated. If the requirement for finding the best set of consistent correspondences is relaxed, then a "greedy algorithm" could be used, which would reduce the expense considerably. A compromise solution would be to employ some kind of backtracking to give better but still non-optimal performance, as in the system developed by [Connell and Brady, 1985].

However, this thesis claims that good matching performance can be achieved by relaxing altogether the requirement for a globally consistent set of one-to-one correspondences. The matching algorithm can therefore be simpler because consistency-checking and backtracking need not be performed. More importantly, it is potentially more efficient because parallel computation can be exploited more fully, since a comparison between two objects can be performed independently from other comparisons (except for making use of their similarity scores). The justification for this approach is that the richness of the property and relationship descriptions and the nature of physical objects tends to enforce consistency.

4.2.4 The "Level Hopping" problem.

Since the matcher cannot assume canonical descriptions, and in particular cannot assume a canonical part decomposition hierarchy, it must cope with the "level-hopping" problem [Wasserman, 1985]. For example, objects *A0* and *B0* in Figure 4.1 should be considered similar, even though object *B0* includes an additional level in the decomposition hierarchy because parts *B3* and *B4* are combined into a single composite object. If the matcher takes a top-down approach, traversing down from the root objects, *A0* and *B0*, and only considers correspondences between subparts of objects that match at each level, then it cannot find the correspondences between *A2* and *B3*, or between *A3* and *B4*. Therefore, although it is desirable for the matcher to be guided by the decomposition hierarchy, it should not be overly constrained by it, since it should be able to find correspondences between components at different levels.

Wasserman's MERGE coped with the level-hopping problem by inserting 'null nodes' into the part hierarchy to account for all possible alternative decomposition hierarchies, and then matching each alternative. However, this strategy only deals with hops of one level, and it is not clear whether the mechanism could be easily extended to efficiently cope with hops of multiple levels. In GRAM, correspondences between components at different levels can be automatically found via the traversal of neighbour relationships. For example, in Figure 4.1, when comparing objects *A1* and *B1*, their neighbour relationships lead to the discovery of the correspondence between *A2* and *B3*, and between *A3* and *B4*, even though these correspondences cross levels in the hierarchy.

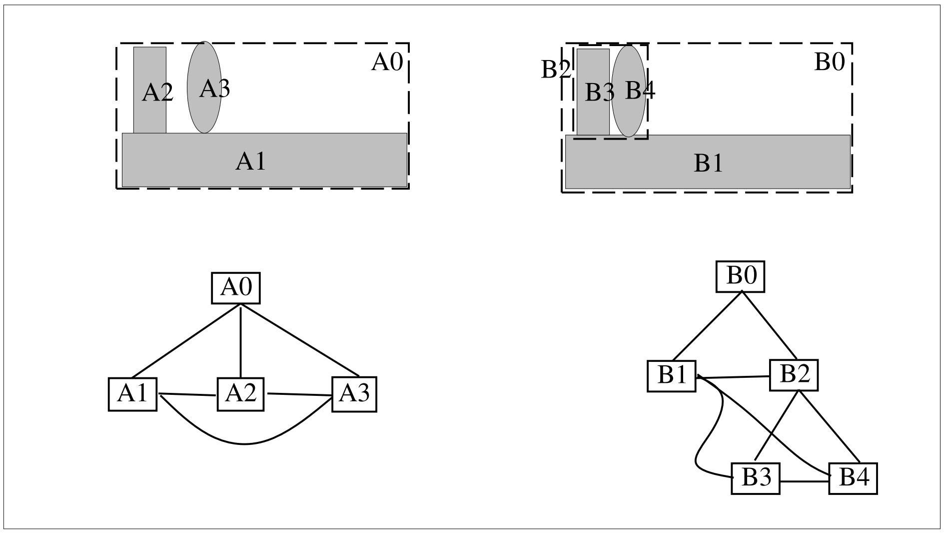


Figure 4.1: The "level-hopping" problem.

4.2.5 A description may need to be augmented.

Since the matcher cannot assume that descriptions are canonical and complete, it may need to invoke the instance constructor to augment a description with more information so that it can obtain a more accurate measure of similarity. This requires making a new relationship explicit, or creating a new composite object.

In the example in Figure 4.1, a relationship between *B0* and *B3* could be created, thus enabling *A0* to be compared more accurately with *B0*, and likewise for *B2* and *B3*. This relationship would be created on the basis of finding the *A2:B3* correspondence via neighbour relationships from the *A1:B1* correspondence. Alternatively, or additionally, a new composite object could be created, consisting of objects *A2* and *A3*, so that this new object could be matched with *B2*.

New neighbour relationships could also be created to resolve the "sideways level hopping" problem: if *A2* did not have an explicit neighbour relationship with *A3*, then one would need to be created in order to compare it with the *B3–B4* relationship.

4.2.6 Estimates of similarity should be obtainable from superconcept or subconcept similarity scores.

An object is usually an instance of several concepts, ranging in degree of specificity (eg. *phillips-screw-driver* and *hand-tool*), and the classification system may require more than just one classification. Therefore, when comparing an observed object with a concept in memory, the matcher should be able to produce an estimated similarity score on the basis of previous comparisons between the object and the superconcepts or subconcepts of the concept, if they are available, since the estimate may be sufficient for the task without having to perform a complete comparison.

Sections 4.3.10 and 4.4.8 discuss how this can be done by treating a superconcept similarity score as an upperbound, and a subconcept score as a lowerbound, adjusted according to the *typicality* of the concept within the superconcept, or of the subconcept within the concept.

4.2.7 Instance-counts and feature variances affect similarity.

The matcher needs to take into account the *instance-counts* and *variances* of features when evaluating similarity. This is because an instance-count indicates the degree of optionality of feature, and variance indicates the range of acceptable values. Therefore, these measures are incorporated into GRAM's similarity evaluation scheme.

4.2.8 Object similarity depends on axis correspondences, and may require attribute coercion.

The matching task is made more expensive by the fact that the measure of similarity of two objects depends on how their axes are put into correspondence. For example, suppose we are matching objects *A2* and *B2* in Figure 4.2, whose primary axes are shown by the arrows. If the two primary axes are assumed to be in correspondence, then the overall similarity score is not high, since their contexts are significantly different. If, on the other hand, the primary axis of *A2* is put in correspondence with the negative-secondary axis of *B2*, then the structures and contexts are both reasonably similar.

Whenever two objects are compared in such a way that their primary axes are *not* in correspondence, as in the example above, then it is necessary to *coerce* the attribute values of one of the descriptions so that similarity can be evaluated correctly. For example, the *direction* attribute of the relationship from *B2* to *B1* is 90 degrees relative to *B2*'s primary axis. To compare the *direction* attributes of the *A2–A1* relationship and the *B2–B1* relationship, where *A2*'s primary axis corresponds to *B2*'s negative-secondary axis, the *B2–B1* direction must be coerced to be relative to *B2*'s negative-secondary axis, giving a direction of 180 degrees, which can now be meaningfully compared with the *A2–A1* direction.

GRAM assumes that two two-dimensional objects can be corresponded in four ways, one for each 90 degree rotation. Currently correspondence by reflection is not accounted for. In a three-dimensional domain there are a minimum of 24 possible axis correspondences. This is one reason why the matcher should be amenable to a parallel architecture, since it should be able to evaluate many alternatives simultaneously. On the other hand, the evaluation of the alternatives is usually computationally inexpensive, since a comparison using a low spread effort is sufficient to reject most of the alternatives.

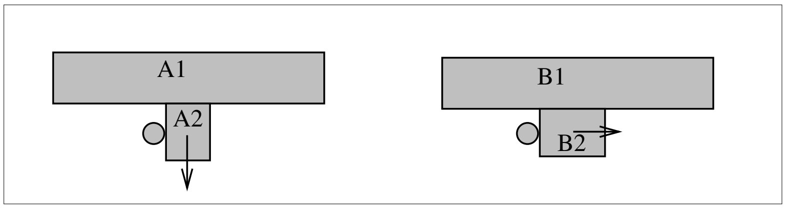


Figure 4.2: Axis Correspondences.

4.3 Similarity.

This section considers GRAM's definition of 'similarity'. It begins by explaining the basic definition of similarity of two objects, and then explores various aspects of the definition in more detail. This section is not concerned with how the evaluation is actually performed.

4.3.1 The basic definition of similarity.

An object is defined by its structure and context, which are in turn defined by its properties, relationships, and relatees. Therefore, the similarity of two objects is defined in terms of the similarities of these features.

The definition of similarity is given more precisely by the formula in Figure 4.3. The *overall-similarity* score is defined to be a weighted average of the structure similarity and context similarity, where the weights are based on the variances of the structure and context, such that high variance means a low contribution to the overall score.

The *context-similarity* score is defined as a weighted average of the similarity of the context properties and the similarities of the parent and neighbour relationships and relatee objects. The weights are based on various factors such as variance, relationship 'importance', and instance-counts, as will be discussed in section 4.3.6. The choice of which relationship similarities contribute to the score (since all possible combinations of pairings could potentially contribute) is explained in section 4.3.3.

Structure-similarity is defined in much the same way as context-similarity: it is a weighted average of the similarity of the structure properties and the similarities of the subpart relationships and relatees.

The definition of similarity must also account for groups, disjunctions, multi-relationships, *import-from* relationships, and the various *interpretations* of structure and context descriptions. These issues are discussed in later sections.

$$\textbf{overall–similarity} = \begin{array}{c}\textit{weighted}\\ \textit{average of}\end{array} \begin{cases} \textbf{context–similarity} = \begin{array}{c}\textit{weighted}\\ \textit{average of}\end{array} \begin{cases} \text{similarity of context–properties} \\ \text{similarities of parent relationships and relatees} \\ \text{similarities of neighbour relationships and relatees} \end{cases} \\ \\ \textbf{structure–similarity} = \begin{array}{c}\textit{weighted}\\ \textit{average of}\end{array} \begin{cases} \text{similarity of structure–properties} \\ \text{similarities of subpart relationships and relatees} \end{cases} \end{cases}$$

Figure 4.3: The basic definition of similarity.

4.3.2 Attribute similarity

Since the properties and relationships of an object are represented as attribute-vectors, their similarity is defined by a weighted average of the similarities of the individual attribute values, where the weights are based on various factors discussed in section 4.3.6. The definition of attribute value similarity is different for each type of attribute, whether *numerical*, *nominal*, *directional*, *boolean*, *etc*. This section explains the definitions of similarity for each of these attribute types.

There are many possible scoring schemes that could be used for comparing attributes, and the detailed formulas are not particularly important. Therefore, the purpose of this section is to convey the *kinds* of scoring that can be done, and the basic requirements of the scoring scheme.

Attribute similarity scores must be normalised so that a measure of similarity of two attribute values of one kind has the same meaning as a measure of similarity of two attribute values of another kind. If, for example, the similarity score for two nominal-valued *shape* attribute values is 0.8, and the similarity score for two numerical-valued *aspect-ratio* attribute values is also 0.8, then these two scores should have the same meaning, and should therefore be able to be combined sensibly when computing the overall 'structure-similarity' score. In GRAM, this is achieved by normalising all scores into the range 0 and 1, with the following interpretations:

- 0 means "very different"
- 0.5 means "bordering between similar and dissimilar"
- 1.0 means "identical"

There are four factors that need to be considered when comparing two values: Firstly, the absolute *difference* between the values; secondly, a measure of what a "very different" difference is, so that the result can be normalised to the interpretation given above; thirdly, the *variance* of the values if they are generalised, as this can be used as a normalisation factor; and fourthly, whether fit-scoring or proximity-scoring is required, since this determines how tolerant of differences the matcher is to be.

Given these factors, we can now consider how particular kinds of attribute values are compared.

Figure 4.4 illustrates the definition of similarity of **numerical** values. Diagram (a) shows the situation for two ungeneralised values, where the horizontal axis of the graph represents the magnitude of the difference between the two values, and the vertical axis is the similarity score. The score slopes down quite gradually for some distance, where this distance is a "default tolerance" for that attribute, such that any value difference in this range is given a high similarity score. Beyond this point the score drops down more sharply, where the slope is based on a globally pre-defined measure of a "very different" difference for that attribute, so that a zero score indicates that the difference between two values (minus the tolerance) is least that amount.

Diagram (b) defines the similarity of a generalised (concept) value and an ungeneralised (instance) value, using *proximity-scoring*. This is the same (a) except that the tolerance factor is based on the *variance* of the generalised value, which may be smaller or larger than the default tolerance.

Diagram (c) is the same as (b) in that the tolerance is based on the variance. However, *fit-scoring* is required, and this means that the slope of the sharp drop is also based on the variance, rather than being the default "very different" difference, and thus it may potentially be a very steep drop if the variance of the generalised value is very small, thus giving very bad scores for all instance values that are even a small difference from the mean. (The more general distinction between fit-scoring and proximity-scoring will be discussed in section 4.3.8.)

If the generalised value has been obtained from only a few instances, then the tolerance factor in (b) and (c), and the "very different" difference in (c), are based partly on the variance and partly on the default values. The fewer the instances, the less the variance contributes. In the case of a 'generalised' concept value formed from just one instance, the variance is not defined and situations (b) and (c) reduce to the situation in (a).

Directional (*i.e.* angular) values are compared in the same way as ordinary numerical values, except that modulo arithmetic is used to compute differences.

The similarity of **nominal** values is defined in various ways, depending on whether the values are single *symbols*, sets of symbols (*symbolsets*), or generalised symbols (*gsymbols*) for which each symbol has an instance-count. The top of Figure 4.5 gives the definitions of each of these more precisely, as from section 3.3 in chapter 3. The term 'frequency' is used in this discussion to mean the ratio of the instance-count of a particular symbol in a *gsymbol*, to the total instance-count of the *gsymbol*. The frequency of a symbol in a *gsymbol* indicates the probability of that symbol occurring in a future instance.

The similarity of two ungeneralised *symbol* values is defined in Figure 4.5 (a). It is simply 1 ('identical') if the two symbols are the same, and 0 ('very different') otherwise.

Figure 4.5 (b) gives the definition of similarity of two *symbolsets*. It is defined by the cardinality of the intersection of the sets divided by the cardinality of their union, thus measuring the proportion of symbols that are common to both *symbolsets*. This is the same as the formula used by [Winston, 1975] to measure membership in a group. Currently GRAM uses the same formula for fit-scoring and proximity-scoring, although a future implementation should be less tolerant of differences in the case of fit-scoring.

The definition of similarity of a *gsymbol* and a *symbol* is in shown in Figure 4.5 (c). It is defined to be 0 if the *symbol* is not present in the *gsymbol*. Otherwise the score is 0.5 (meaning 'poor-but-acceptable') plus half the ratio of the frequency of the *symbol* in the *gsymbol* to the largest frequency of *any* symbol in the *gsymbol*. Thus if the *gsymbol* consists of a large number of low-frequency symbols (such as car colours), and if the instance *symbol* is one of these, then the similarity score is high. If, on the other hand, the *gsymbol* includes a symbol that was observed in most instances (such as the 'rectangular' *shape* attribute value of the concept *table*), and several other low-frequency symbols (such as for a few round or triangular tables), and if the instance *symbol* is one of the low-frequency symbols, then the score will be

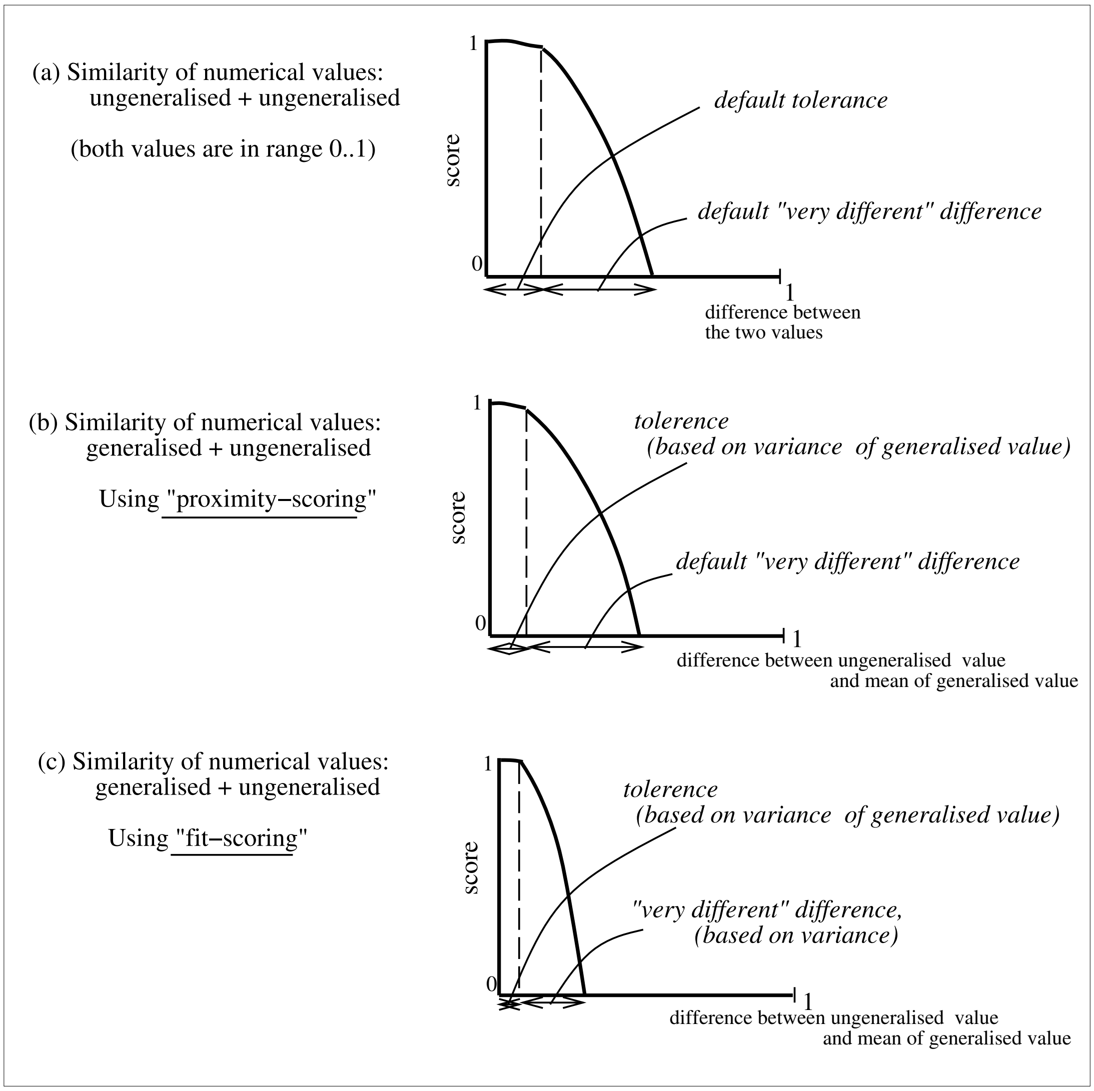


Figure 4.4: Similarity of *numerical* attribute values.

not much higher than 0.5. In other words, the maximum symbol-frequency for a *gsymbol* is used as a *tolerance* factor. As for *symbolset* similarity, the current version of GRAM does not distinguish between fit-scoring and proximity-scoring. The above formula was not based on other research, and needs to be more thoroughly evaluated.

The similarity of a *gsymbol* and a *symbolset* is defined by averaging the similarities of the *gsymbol* and each of the individual symbols in the *symbolset*, using the definition in (c). The formula for this is shown in (d).

The definition of similarity of two *gsymbols* is more complex, and is not shown in the figure. It is similar to the definition of similarity of two *symbolsets* except that the differences between the frequencies of each symbol is taken into account.

Boolean values are a special case of nominal values where only two symbols are allowed,

"symbol" : a single nominal value. *eg. touching*

"symbolset" : a set of nominal values. *eg. (rectangle, square)*

"gsymbol" : a generalied symbol or symbolset, defined by
- a set of symbols
- a 'symbol–count' for each symbol. (ie. the frequency of observation)
- an instance–count (ie. the total number of observed instances)

eg. count=6 (touching:5, separate:1)
eg. count=7 (rectangle:6, square:4, circle:1)

(a)

$$\text{proximity-score} \;(\text{symbol1}, \text{symbol2}) = \text{fit-score} = \begin{cases} 0 \;(\text{"very different"}) & \text{if different} \\ 1 \;(\text{"identical"}) & \text{if same} \end{cases}$$

(b)

$$\text{proximity-score} \;(\text{symbolset1}, \text{symbolset2}) = \text{fit-score} = \frac{\left|\text{intersection}(\text{symbolset1}, \text{symbolset2})\right|}{\left|\text{union}(\text{symbolset1}, \text{symbolset2})\right|}$$

(c)

$$\text{proximity-score} \;(\text{gsymbol1}, \text{symbol2}) = \text{fit-score} = \begin{cases} 0 \;(\text{"bad"}) \text{ if symbol2 is not present in Gsymbol1} \\ \text{otherwise:} \quad 0.5 + \dfrac{\text{symbol-count}}{2 * \max(\text{symbolcount}_i)} \end{cases}$$

frequency of symbol2 in gsymbol1..

maximum frequency of a symbol in gsymbol1.

(d)

$$\text{proximity-score} \;(\text{gsymbol1}, \text{symbolset2}) = \text{fit-score} = \frac{\sum \text{similarity}(\text{gsymbol1}, \text{symbol}_i)}{\left|\text{symbolset2}\right|}$$

Figure 4.5: Similarity of *nominal* attribute values.

namely "true" and "false". The definition of an ungeneralised *boolean* value, and the definition of a generalised *gboolean*, are given at the top of Figure 4.6.

If both values are single ungeneralised *boolean* values, then similarity is 1 if they are the same, and 0 otherwise, as shown in Figure 4.6 (a). If one value is a *gboolean*, then the score is the frequency of the single *boolean* value within the *gboolean*, as shown in (b). If both values are *gbooleans*, then the score is based on the difference in the *true-count* frequencies, as shown in (c).

The similarity of two **profile** values (*i.e.* vectors of values of the same type) is the average of the similarities of the individual values. This assumes that the two profiles are of the same type: they must have the same length, and each position of one *profile* vector has the same meaning as the corresponding position of the other profile vector.

"boolean" : either *true* or *false*

"gboolean" : a generalied boolean, defined by
- true–count. (the count of *true* instances.
- instance–count
  (ie. the total number of observed instances)

eg. *((true–count = 5)*
(instance–count = 7))

(a) proximity–score (boolean1, boolean2) =
= fit–score

$$\begin{cases} 0 \text{ ("bad") if different} \\ 1 \text{ ("perfect") if same} \end{cases}$$

(b) proximity–score (gboolean1, boolean2) =
= fit–score

$$\begin{cases} \text{if boolean2 is true: truecount / instancecount} \\ \text{if boolean2 is false: } 1 - \text{(truecount / instancecount)} \end{cases}$$

(c) proximity–score (gboolean1, gboolean2) =
= fit–score

$$1 - \left| \text{(truecount1 / instancecount1)} - \text{(truecount2 / instancecount2)} \right|$$

Figure 4.6: Similarity of *boolean* attribute values.

4.3.3 Relationship and relatee similarities.

The definition of similarity given in section 4.3.1 referred to measures of similarity for pairs of relationships and relatees, but did not state which pairs contribute. This section explains this in detail. Throughout this section, and the rest of the chapter, the term *relationship/relatee* will be used to refer to a relationship and its associated relatee object.

First we must consider the similarity score for two relationship/relatees. This is defined to be a weighted average of the similarity of the two relationships (defined in terms of their attribute similarities) and the similarity of the two relatees (defined by the *overall-similarity* formula given earlier in Figure 4.3). The weights are predefined parameters.

The similarity of two structures or two contexts is defined in terms of *property* similarity scores and the scores of the set of *winning correspondences* between their parent, neighbour, and subpart relationship/relatees. A *winning correspondence* is a pairing of two relationship/relatees for which the similarity score is higher than the score of any other correspondence involving one or both of the relationship/relatees.

The best way to understand this is by referring to the example in Figure 4.7, which shows two objects *A0* and *B0.* Suppose we are concerned with the similarity of objects *A1* and *B1* (the large central blocks). Below the pictures are lists of the relationship/relatees for *A1* and *B1*, and between these are dotted lines that indicate the winning correspondences, each labelled with a similarity score. Lines that have arrows at both ends indicate that the correspondence is higher scoring than any other correspondence involving *either* of the relationship/relatees. Lines that have arrows at just one end indicate that the correspondence is higher scoring than any other correspondence involving the relationship/relatee at the *source* of the line. For example, *A1*–

A5 is most similar to *B1–B4*, but not *vice versa* since *B1–B4* is much more similar to *A1–A6*. Each of the scores shown in the diagram contribute to the context-similarity score for *A1* and *B1*. (In this particular case there are no subparts, and so the structure-similarity score is defined only in terms of properties similarity.) The scores for all the possible correspondences that do not contribute are not shown.

Thus, the set of winning correspondences includes the best correspondence for each relationship/relatee for each of the two objects, even if these conflict with each other. This scheme has various beneficial consequences that are discussed in section 4.3.4. Before this, however, we will consider how *multi-relationships* are incorporated into the above scheme.

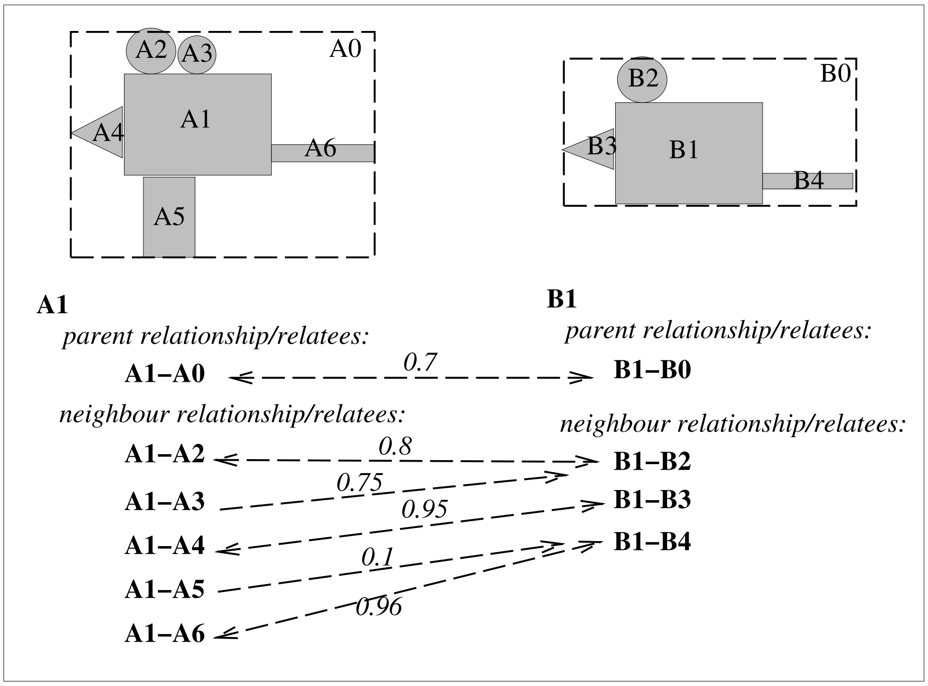


Figure 4.7: Winning relationship/relatee correspondences.

Multi-Relationships.

A multi-relationship is treated in exactly the same way as an ordinary relationship, except that its winning correspondence may contribute more to the overall similarity score. The weighting scheme is discussed in section 4.3.6.

Figure 4.8 shows the simplest situation, where two objects *A1* and *B1* both have a multi-relationship to a generalisation of several neighbouring circles. (It is assumed that individual relationships to the circles have been dropped from the description, either during instance construction or during generalisation.) The correspondences are straightforward and unambiguous, as shown at the bottom of the figure. The difference between the *howmany* counts contributes minimally to the similarity score of the two multi-relationships, since it is desirable to be quite tolerant of such differences. Distinct individual relationship/relatees should contribute more to the similarity score than similar relationships that can be clustered into a multi-relationship.

In Figure 4.9, object *A1* only has ordinary neighbour relationships with its neighbouring circles, rather than a multi-relationship. The winning correspondence for each of these is with *B1*'s multi-relationship. However, the winning correspondence for *B1*'s multi-relationship is with the *A1–A2* relationship because only *A2* has a circle on both sides of it, as have *B3* and *B4*. The fact that A and B have a different number of circles is not explicitly accounted for by the similarity scheme, except by properties such as *number-of-subpart*. Dissimilarities are reflected by the fact that the *A1–A2* and *A1–A4* relationship/relatees do not match the *B1* multi-relationship perfectly, since the generalised multi-relationship relatee expects 75% of instances to have a circle on the left, and 75% of instances to have a circle on the right, and *A2* and *A4* only satisfy one of these conditions.

Figure 4.10 shows a situation in which the difference in the number of circles is much more significant. Again, the scoring scheme does not explicitly take this into account, but the difference is adequately captured in the similarity score because *A2* does not match the generalised relatee of *B1*'s multi-relationship particularly well, since it does not have circles on either side.

Figure 4.11 shows a comparison for which object *B1* includes not only a multi-relationship to the generalised circle, but also has two ordinary relationships to the circles, *B2* and *B6*, which are *atypical* (since neither of them have circles on both sides, and *B6* has a rectangular block immediately to its right). Similarity is evaluated in the same way as above, with each relationship contributing its winning correspondence. In this case, the *A1–A2* relationship matches *B1–B2*, the *A1–A3* relationship matches *B1*'s multi-relationship, and the *A1–A4* relationship matches *B1–B6*. The correspondence involving the multi-relationship contributes the most, which ensures that a mismatch for this relationship will have a greater negative effect on the overall score than a mismatch on the ordinary (atypical) relationships.

Two objects may be related to the same relatee.

A minor aspect of comparing relationship/relatees is that sometimes two objects being matched might both have a relationship to the *same* relatee object. This occurs most frequently when comparing instances within the same scene, as when the group-finder is looking for groups of similar objects. For example, the similarity of *A2* and *A3* in Figure 4.12 both have a neighbour relation to the same object, *A1*, and both have a parent relationship to the same object, *A0*. The two relationships differ, but the relatee similarity must obviously have a perfect similarity score, and does not require any evaluation.

This situation may also occur when comparing two concepts in concept-memory, both of which have a relationship to the same relatee concept. For example, two existing generalised concepts, *swivel-chair* and *four-legged-office-chair*, may both have a parent relationship to

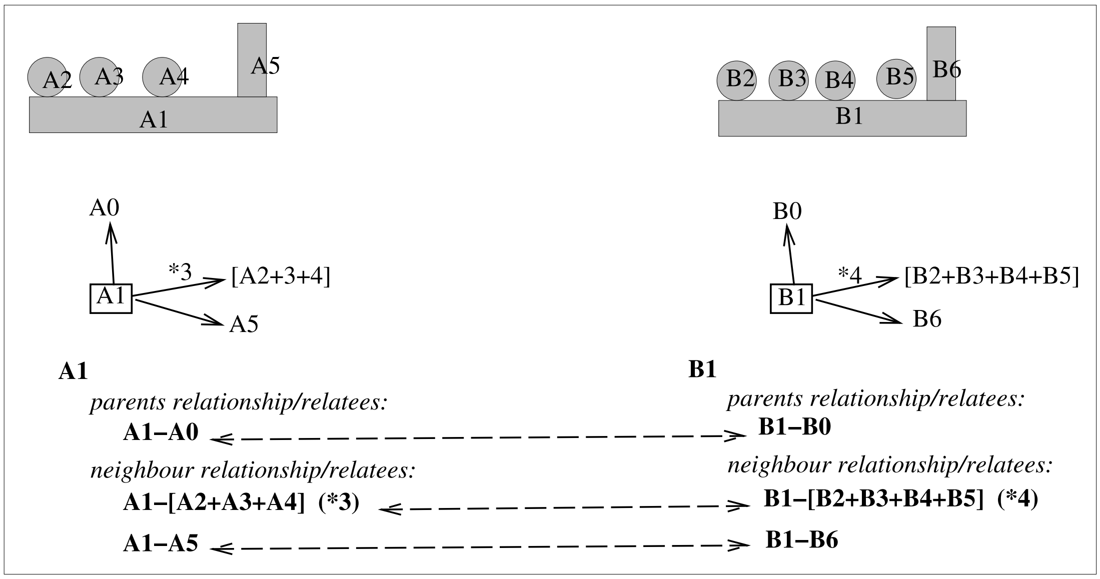


Figure 4.8: Multi-relationship similarity.

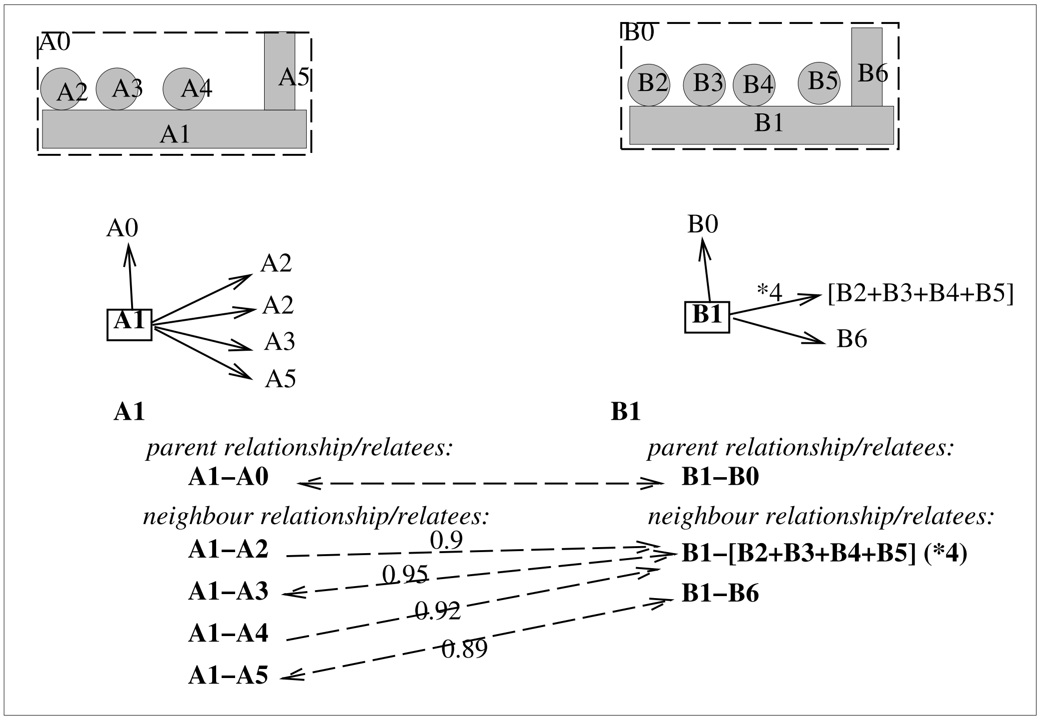


Figure 4.9: Multi-relationship similarity.

the same concept *office* and a neighbour relationship to the same concept *desk*. Therefore, although the relationships with the *office* and *desk* concepts may differ, the scores of the relatee similarities must be perfect, without requiring evaluation.

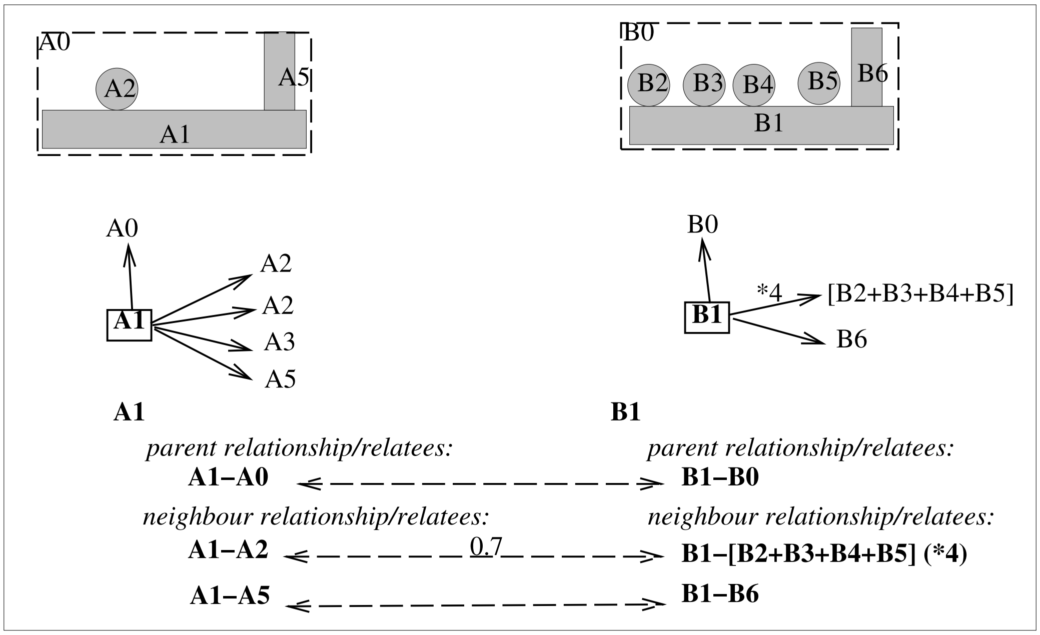


Figure 4.10: Multi-relationship similarity.

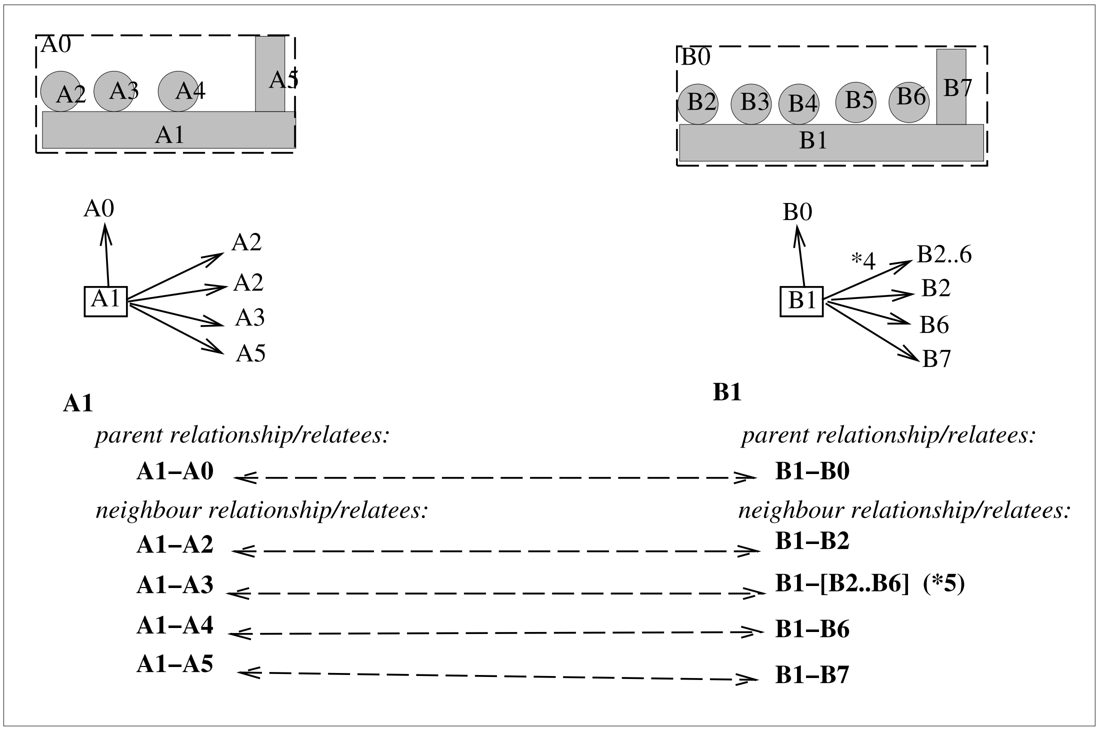


Figure 4.11: Multi-relationship similarity.

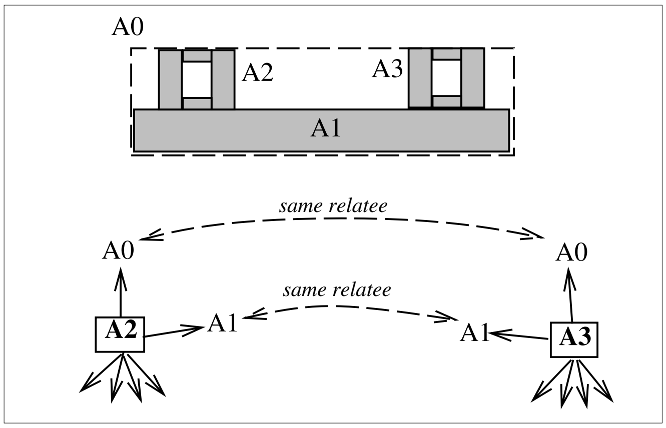


Figure 4.12: Two objects with the same relatees.

4.3.4 Local consistency between correspondences is not enforced.

A distinctive characteristic of the similarity definition is that it does not enforce local consistency between the winning relationship/relatee correspondences that contribute to the object similarity score. This contrasts with a scheme that requires a consistent set of one-to-one bindings. This section discusses some of the beneficial consequences of this approach. (The similarity definition also does not require *global* consistency, and this will be discussed later.)

Local ambiguities are accounted for by 'implicit grouping'.

A consequence of allowing locally conflicting correspondences, and a reason why it is appropriate, is that it implicitly accounts for ambiguities (local to the comparison) that could be resolved by the creation of explicit multi-relationships or groups. To illustrate this, consider the example in Figure 4.7 given earlier. Object *A1* has an additional circular neighbour, *A3*, that *B1* does not have. However, this does not lower the similarity score significantly, since the *A1–A3* relationship still matches the *B1–B2* relationship quite well, and its similarity score contributes to the overall similarity score, rather than contributing a zero score because it conflicts with the higher-scoring (*A1–A2*):(*B1–B2*) correspondence. Thus, the scoring scheme implicitly assumes that *A2* and *A3* could be generalised to form a concept (or a typical-member concept of a group) and referred to via a single multi-relationship, which could then be matched unambiguously with the *B1–B2* relationship.

On the other hand, the *A1–A5* relationship has no high-scoring correspondences with the relationships of *B1*, and so it *does* contribute a poor score, since the highest-scoring correspondence is with *B1–B4*.

Therefore, a missing parent, neighbour, and subpart only contributes a poor similarity score if it doesn't match *any* of the parents, neighbours, and subparts of the other object.

Another more obvious example of the usefulness of this scheme is given in Figure 4.13, in which it is assumed (for the sake of this example) that *C2..C5* and *D2..D7* have not already been explicitly grouped. The overall similarity score of *C1* and *D1* is high, as it should be, even though there are two extra D circles.

The winning correspondences of the two extra circles (considered to be *D4* and *D5*) are higher-scoring than the *A1–A3* correspondence in the previous example, since in that situation the extra part, *A3*, differed more significantly from *B2* because it had a circle on one side, while both of the extra D circles match very well with one or more of the C circles. Thus the similarity scheme has the desirable consequence that it is implicitly more tolerant of extra 'unmatched' relatees that are within a large group of similar relatees (whether explicitly grouped or not).

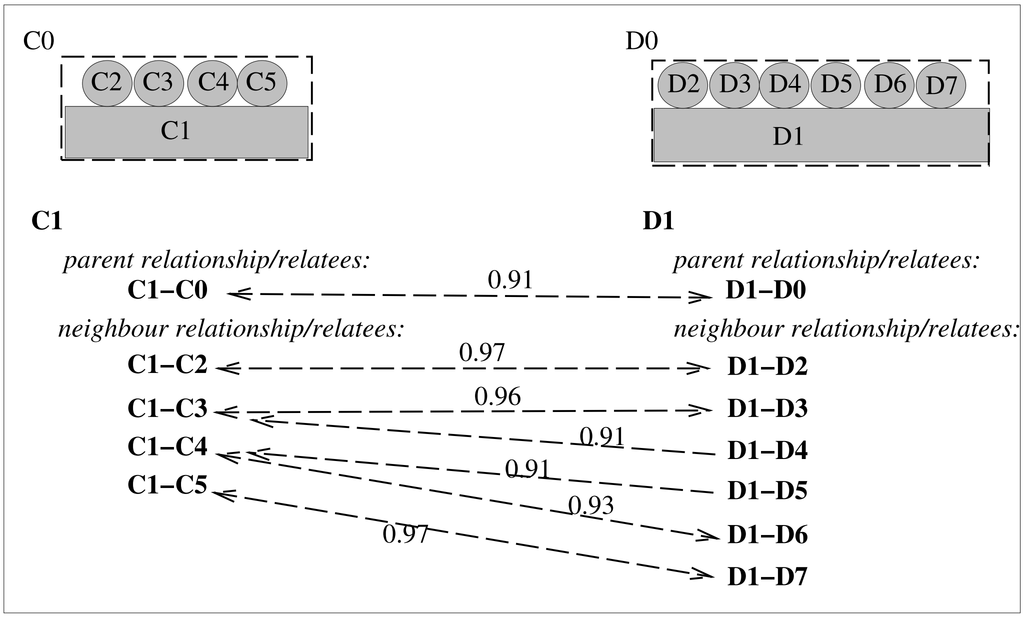


Figure 4.13: Locally ambiguous correspondences.

Local ambiguities are accounted for by implicitly allowing 'multiple roles'.

Sometimes the locally conflicting winning correspondences may not be due to several relatees of one or both objects being similar to each other, but due to several differing relatees in one object ambiguously matching one or several relatees of the other objects in different ways. The similarity scores may be roughly the same, but as a consequence of different forms of similarity. For example, Figure 4.14 shows two objects *A0* and *B0*, and the winning correspondences that contribute to the similarity score of the *A1:B1* correspondence. From the point of view of the *A1:B1* comparison, the highest similarity score for both the *A1–A3* and *A1–A4* relationships is with the *B1–B3* relationship. The former has a high context similarity and a low structure similarity, while the latter has a high structure and a low context similarity. The highest score for the *B1–B3* relationship is with the *A1–A3* relationship, although this is only marginally better, on the basis of the relationship similarity and the context similarity of *A3* and *B3*, since *B3* is structurally more similar to *A4*.

Thus, although object *B1* could be considered to be missing a neighbour, the overall similarity score for *A1* and *B1* is not significantly lowered by the fact that *B1* has fewer neighbours (although the difference will be reflected to a smaller extent in other ways, such as the dissimilarity of the context profiles of *A1* and *B1*, and the dissimilarity of the aspect-ratios and density-profiles of *A0* and *B0*.) Thus GRAM's similarity definition implicitly accounts for the fact that object *B3* partially matches both *A3* and *A4* (in different ways), and could therefore be generalised in two alternative ways by the generaliser (as is discussed in chapter 5).

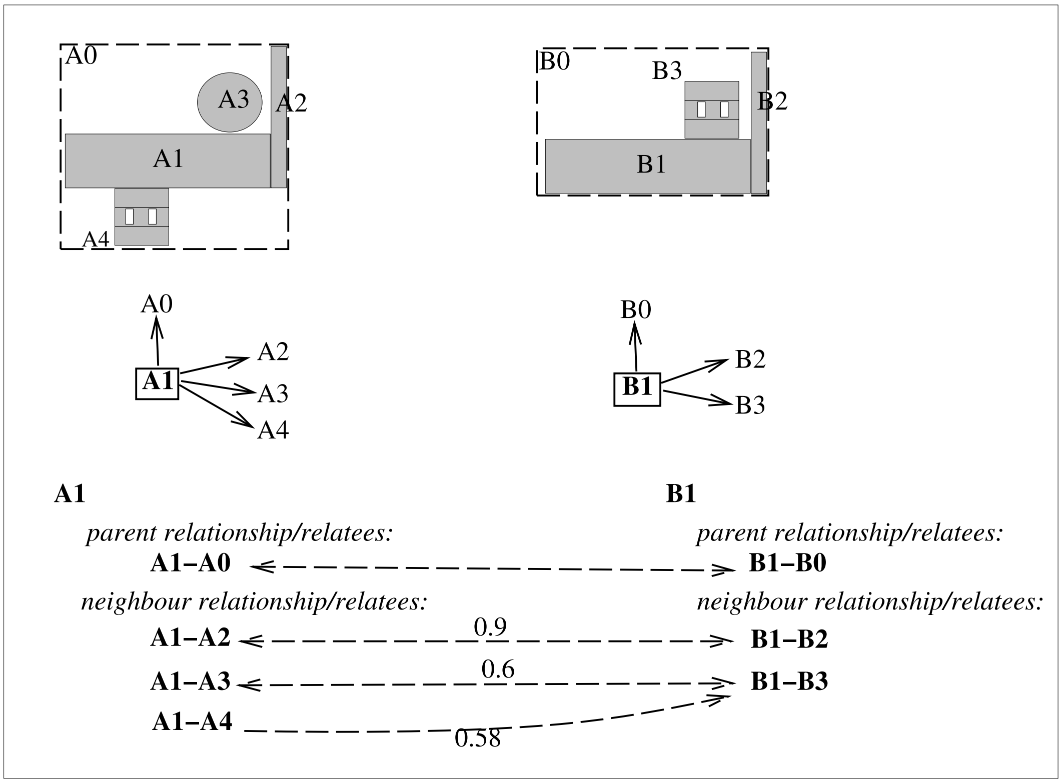


Figure 4.14: Local ambiguities.

Similarity can be more efficiently evaluated.

The absence of a requirement for local consistency has beneficial consequences not only for the effectiveness of the matcher, but also for its efficiency. This is because the search for the winning correspondences only involves finding the best correspondence for each relationship/relatee *independently*, rather than having to find the best consistent *set* of correspondences. Thus the comparisons of relationships for a particular correspondence can be evaluated in $O(n^2)$ rather than $O(n!)$ time, where n is the number of relationships.

4.3.5 Global consistency between correspondences is not enforced.

Another characteristic of the scoring scheme is that the set of winning correspondences that define the similarity of two objects do not have to be consistent with the set of winning correspondences of any other object comparisons. In other words, in addition to not explicitly enforcing local consistency, the matcher also does not enforce *global* consistency. This simplifies the definition of similarity, and makes evaluation simpler and more efficient, since each comparison can be performed independently (except for *using* similarity scores produced by other comparisons) without having to search for or maintain a globally consistent set of correspondences between objects. It keeps multiple competing hypotheses active simultaneously, thus avoiding the need for a backtracking mechanism, and potentially permits a much greater degree of parallel computation. It also enables components of an object to play multiple roles when matched with another object.

For example, in Figure 4.7, the winning relationship/relatee correspondences that contribute to the similarity score of *A4* and *B3* do not have to be consistent with winning relationship/relatee correspondences that contribute to the similarity score of *A1* and *B1*, or *vice versa*. However, in this particular example they are consistent, with both "points of view" having the same winning correspondences.

An example of an inconsistency is illustrated in Figure 4.15. The similarity of *A1* and *B1* is defined in terms of two winning neighbour correspondences, (*A1*–*A2*):(*B1*–*B2*) and (*A1*–*A3*):(*B1*–*B3*), as shown in (a) of the figure. During the process of finding these winning correspondences, the matcher may try to evaluate the *A2:B3* correspondence, shown in (b), whose similarity is defined in terms of the winning correspondence (*A2*–*A1*):(*B3*–*B1*). This is inconsistent because it is based on the similarity score for *A1* and *B1*, which in turn is based on the assumption that *A2* is matched with *B2*, not *B3*. This inconsistency is ignored. The similarity of *A2* and *B3* simply requires that the neighbours *A1* and *B1* are similar, and is not concerned with the selection of winning correspondences on which the *A1:B1* score is based.

The inconsistency in this particular example does not indicate a problem with the similarity scheme, since the *A2:B3* correspondence is not a globally-best correspondence anyway. *A2* matches *B2* better, and *B3* matches *A3* better. In general, the local winning relationship/relatee correspondences of a globally-best object correspondence will be consistent with those of other globally-best object correspondences, unless there are ambiguities. In the case of ambiguities we *want* the matcher to produce multiple alternative correspondences, so that the generaliser can deal with them appropriately.

Global ambiguities may be accounted for by 'implicit groupings'.

In the case of local ambiguities discussed earlier, the matcher assumes that the competing relationship/relatee correspondences could be combined into a single multi-relationship, and this justifies allowing inconsistencies between the correspondences. In much the same way, the matcher assumes that the competing *object* correspondences could be combined into a single *group*, and this justifies allowing inconsistencies between the correspondences.

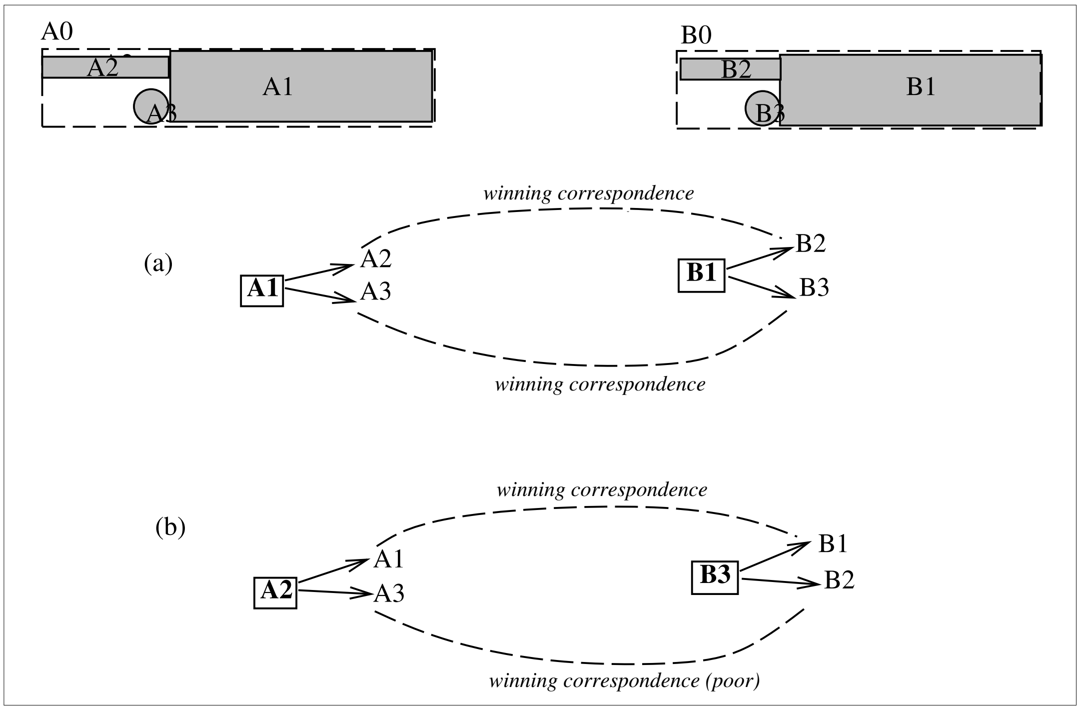


Figure 4.15: Global inconsistencies.

Global ambiguities may be accounted for by ‘implicit multiple generalisations’.

Inconsistencies are also permitted because objects sometimes play multiple roles (structural or contextual), depending on the “point of view” from which they are considered. Section 4.3.4 discussed how objects can play multiple roles *locally* within a single comparison. This section considers it at the global level.

For example, Figure 4.16 shows two objects *A0* and *B0*, and the winning correspondences for the comparisons of *A3* and *B3*, and of *A4* and *B3*. The similarity of *A3* and *B3* is defined in terms of a winning correspondence (*A3*–*A2*):(*B3*–*B2*), while the similarity of *A4* and *B3* is defined in terms of a winning correspondence (*A4*–*A5*):(*B3*–*B2*). Locally there is no ambiguity or inconsistency, but globally there is an inconsistency because *B2* is matched with both *A2* and *A5*. However, it is desirable that such inconsistency is accepted within the similarity scheme since otherwise it would not be possible for *A3* and *A4* to both be considered similar to *B3* on the basis of having a tall rectangular block immediately to the right. From the point of view of the *A4:B3* comparison, object *A5* satisfies this role, while from the point of view of the *A3:B3* comparison, object *A2* satisfies it.

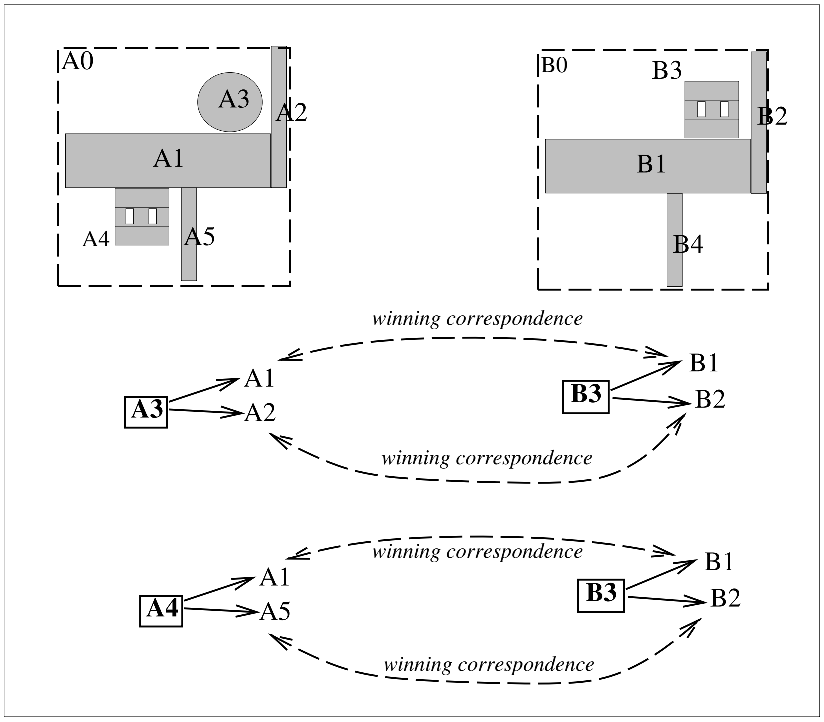


Figure 4.16: Global ambiguities.

4.3.6 Weightings.

Section 4.3.1 stated that overall similarity, structure similarity, and context similarity, are all defined by weighted averages of feature similarities, where the weights determine how much that feature contributes to the score. This section explains the different kinds of weights that are used. For clarity it is assumed that a (generalised) concept is being compared with an (ungeneralised) instance.

Attributes are weighted by importance and variance.

The similarity of attribute vectors is defined by a weighted average of the individual attribute similarities, and each weight is based on two factors. The first factor is a globally defined domain-specific measure of importance of an attribute within a particular kind of attribute vector. For example, in the structure-properties attribute vector, *colour* might be given a lower weight than *shape*, especially in a domain where objects in the same categories frequently have a variety of different colours.

The second factor is the *variance* of the concept's attribute, which can be considered to be a learned measure of attribute importance. For example, if the *colour* attribute of the concept *book* has a high variance as a result of seeing many books of different colours, then that attribute should not contribute much when trying to classify an observed object as a *book*. This has been discussed by [Fisher, 1987a], who referred to a high-variance feature as having low predictive utility.

Relationship/relatees are weighted by importance, variance, and instance.

In the definition of structure and context similarity, the contribution of the similarity score for each winning relationship/relatee correspondence is weighted on the basis of several factors. One factor is *relationship-importance* which is computed by the instance construction mechanism. In the case of neighbour relationships this is the *neighbourliness* of the two objects involved, which (as explained in section 6.3.1 of chapter 6) is based on factors such as distance, relative size, connectivity, *etc*. Likewise, subpart and parent relationships are weighted primarily on the basis of relative size. A subpart relationship/relatee involving a very small subpart is assumed (in the absence of other knowledge) to be less important than a relationship/relatee involving a large subpart.

The weighting of the similarity for a winning relationship/relatee correspondence is also based on *instance-counts*. If the relationship of the concept has a low instance-count (relative to the instance-count of the concept), and if the similarity score is low, then that score is given a low contribution weight. If, on the other hand, the similarity score is high, then the score contributes fully, even though the instance-count is low.

For example, if a concept *television-set* has a low-occurrence neighbour relationship with the concept *aerial*, then if we observe a television without an aerial, the similarity score should not be reduced by this 'mismatch'. But if we see a television that *does* have an aerial that matches the *aerial* concept, then its similarity score should contribute fully to the overall similarity score, thus providing predictive evidence that the object is a television. To take this into account, the relationship/relatee can be weighted by the maximum of the relationship frequency and the similarity score.

The contribution weighting for a relationship/relatee is defined as the *minimum* of the two factors discussed above, as shown in the following formula. If both relationships are generalised, then a slightly more complex formula must be used, which takes into account the variances and instance-counts of both descriptions.

$$\begin{aligned}&\textit{weighting [relationship1/relatee1, relationship2/relatee2]} = \\ &\quad \textit{minimum} \;(\quad \textit{maximum} \;(\; \textit{relationship1-importance}, \quad \textit{relationship2-importance} \;), \\ &\qquad\qquad \textit{maximum} \quad \left\{ \frac{\textit{instance-count[relationship1]}}{\textit{instance-count[concept1]}} \;, \quad \textit{similarity-score} \right\} \\ &\qquad\qquad)\end{aligned}$$

A multi-relationship weighting is based on its *howmany* count.

The contribution weight of a multi-relationship is based on its *howmany* count, as shown by the graph in Figure 4.17, where the horizontal axis is the range of *howmany* values from 1 upwards, and the vertical axis is the weight. The weights on ordinary relationships, as considered in the previous section, are all assumed to be in the value 0 to 1, but since a multi-relationship is a summary of several ordinary relationships, it can contribute more. The weights shown in the graph may, however, be lowered by relationship unimportance, high variance, and low instance-count.

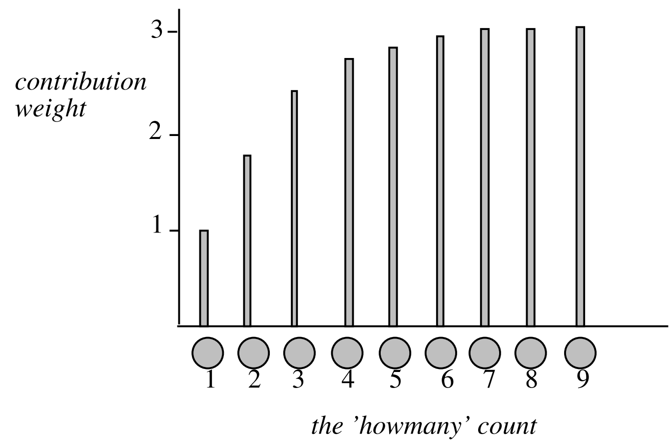


Figure 4.17: Multi-relationship weightings.

4.3.7 Scope restriction is used to measure structure-only or context-only similarity.

Sometimes it is necessary to measure the similarity of only the structure of two objects, ignoring context; at other times it is necessary to measure the similarity of only the context, ignoring structure. For example, if an operator who is holding onto an object instructs a robot to "find one of these in that pile of objects", the robot should not search for another object which also has a hand wrapped around it. The robot should focus its attention only on the structure of candidate objects, not their context. Conversely, if the operator instructs the robot to "collect whatever is in the letterbox", it should identify "whatever" on the basis of it having a letter-box context, and should ignore its structure. Thus, measures of similarity may require *scope-restriction*.

The simplest way to measure structure-only similarity is defined by the *structure-similarity* formula given earlier in Figure 4.3. Likewise, context-only similarity can be defined by the *context-similarity* formula.

Unfortunately this is not sufficient to give an accurate measure, because the structure-similarity score actually measures some context similarity as well, and the context-similarity score measures some structure similarity. This is because the subpart objects of an object *A* may have neighbour relationships to other objects that are *not* within the substructure of A. Likewise, the neighbours of A might have neighbour relationships (or even subpart relationships) to objects that *are* within the substructure of A.

For example, in Figure 4.18, the structure-similarity formula applied to *potplant1* and *potplant2* would ignore the neighbour relationships from *potplant1* to *desk1*, and from *potplant2* to *shelf2*, but would not ignore the neighbour relationships from *pot1* to *desk1*, and from *pot2* to *shelf2*. Likewise, the context-similarity formula would ignore the subpart relationships, but would not ignore the neighbour relationships from *desk1* to *pot1* and from *shelf2* to *pot2*.

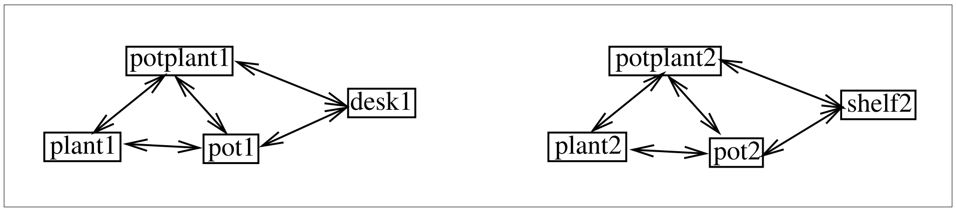


Figure 4.18: Structure-only similarity.

The structure and context similarity formulas as given earlier are adequate for giving a rough structure-only or context-only similarity score for basic matching and generalising, since the relationships that are mistakenly included (such as the *pot1-desk1* relationship) are never direct relationships of the two root objects being matched, and hence they do not contribute as much to the score as the direct relationships. However, if accurate scope restriction is required, the entire substructure, or entire context, must be excluded from contributing to the measure of similarity.

For example, a measure of structure-only similarity for *chair1* and *chair2* in Figure 4.19 would need to ignore all of the relationships from the subcomponent objects that refer to other objects in the room that are not part of the chair substructure. Likewise, a context-only similarity must be based on the parent and neighbour relationships of the two chairs, but ignoring any relationships from contextual objects to subcomponents of the chairs. It could be argued that, to be even more accurate, the similarities of the neighbour relationships *from* the subcomponents of the chairs to non-subcomponents should be included in the context-only score, but this would require a more elaborate evaluation mechanism.

Scope restriction need not be used just for obtaining object context-only or structure-only scores, since any selection of objects could be treated as *in-scope* for a particular similarity measure. For example, when matching the potplants in Figure 4.18 it might be desirable to match the relationships to the external context, such as to the desk and shelf, but without actually matching the relatees themselves. Thus the matcher would find that the pots of both potplants are on top of something horizontal and much larger than the pot, but would not be concerned with the difference between those two somethings.

Contents-similarity ignores the arrangement of subparts.

A special kind of scope-restriction would be needed to obtain a *contents-similarity* score. This score is defined to be high if the two objects have similar *contents* (*i.e.* subpart relatees), even if the arrangement of the subparts differs considerably. This is the case when comparing the

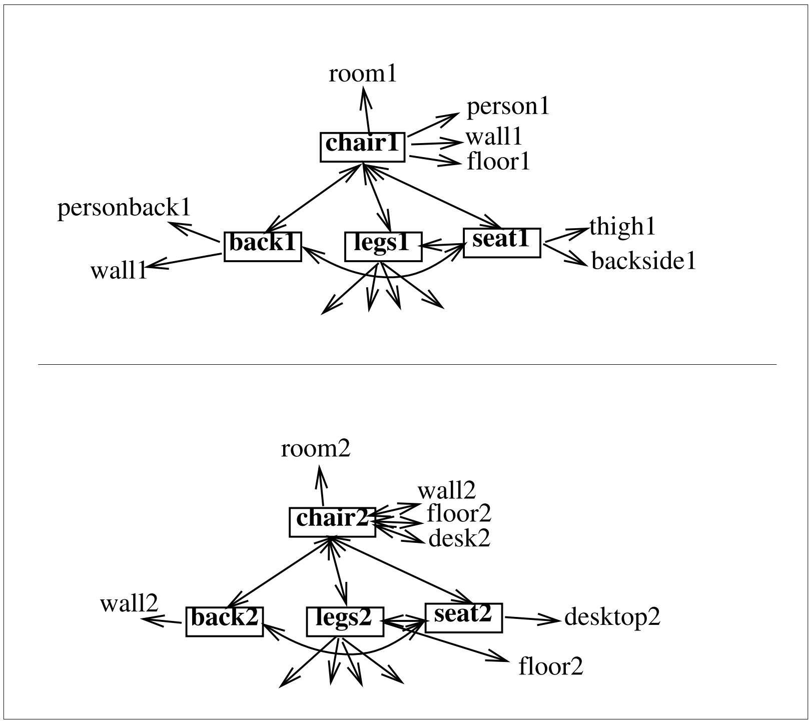


Figure 4.19: Structure-only

two bedrooms in Figure 4.20. The contents-similarity score could be used by the generaliser to justify the creation of a new generalised concept from two instances, even if the ordinary structure-similarity score is poor.

An estimated measure of contents-similarity can be defined in terms of the similarity scores of the subpart relatees, ignoring the similarity scores of the subpart relationships. This is simple to evaluate, but it suffers from the same limitations as for the simple method of computing structure-only and context-only similarity described earlier: The subpart similarities are defined in terms of neighbour relationships to other subparts, and thus their arrangement is not actually being ignored. Therefore, an accurate measure must be defined in terms of the *structure-only* similarities of the subparts.

4.3.8 Proximity-scoring versus Fit-scoring.

Section 4.1 stated that two kinds of scoring are required by the matcher: proximity-scoring and fit-scoring. This section explains this distinction in more detail.

Both forms of scoring are defined by dividing the measure of absolute difference between an instance and a concept by some factor, where this factor indicates what difference is considered to mean "very different". The result is then normalised to give a score between 0 and 1, so that a pair of "very different" objects will score 0, and identical objects will score 1.

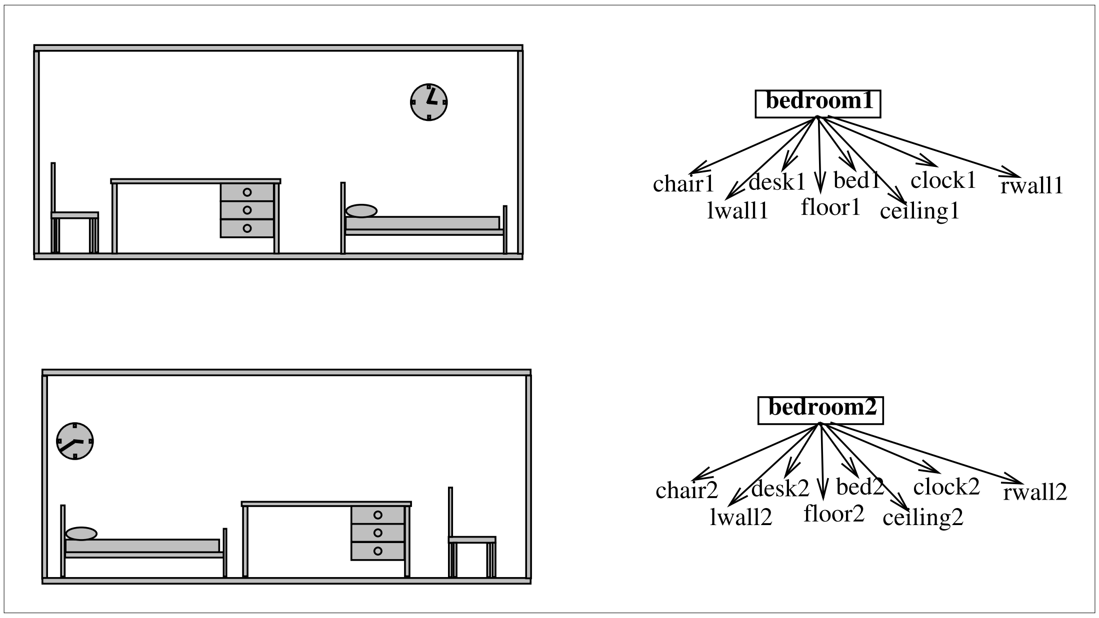


Figure 4.20: Two objects with high contents-similarity.

The absolute difference measure is defined by the 'distance' in object-space between the instance and the boundary of the concept, and this is inversely proportional to the measure of similarity. The 'boundary' of a concept is defined in terms of the variance of the concept, such that the larger the variance, the wider the boundary. This is illustrated abstractly in Figure 4.21 which shows two regions within object-space, each indicating the boundary of a concept. (Atypical instances may be outside the boundary.) The dot on the diagram denotes a particular observed object.

The factor defining "very different" depends on which kind of scoring is required. In proximity-scoring, it is a globally-defined value that defines what "very different" means within object-space as a whole. In fit-scoring, it is based on the variance of the concept itself. (This was discussed in section 4.3.6 for the specific case of numerical attribute values.) Thus, for the example in Figure 4.21, the instance has the same proximity-score with respect to both concepts, but has a much lower fit-score with respect to *concept2* than *concept1*, since the variance of *concept2* is much smaller.

A proximity-score indicates how close an instance is to a concept, regardless of its variance (except to define the concept boundary), and is used by the generaliser to determine whether an instance is close enough to a concept to justify generalisation. A fit-score, on the other hand, indicates how typical an instance is of a concept, and indicates to the generaliser whether the existing concept could be generalised to cover the observed instance, without causing too large a drop in specificity, or whether a new concept should be created.

Proximity-scoring, rather than fit-scoring, is used within the matching process itself to find winning correspondences between relationship/relatees. This is necessary to prevent it from producing a very low score for an obviously correct relationship/relatee correspondence just because the generalised relationship or relatee has a very low variance. For example, consider Figure 4.22 which shows two chairs, one of which has a number of 'faults', or at least unusual features. Suppose the system has already observed 100 chairs identical to *chair1*, and has created a concept *chair* from these. If the matcher is given a description of *chair2*, then the fit-score (and the fit-scores for the correspondences between its subcomponents and the subcomponents of *chair*) will be very low, due to the very low variances of *chair* features. However, *chair2* clearly matches *chair* well (within the space of all possible objects, which includes elephants and paper-clips) and the correspondences between its components are strong. The proximity-scores will be high, and are therefore more appropriate for evaluating and selecting relatee/relationship correspondences.

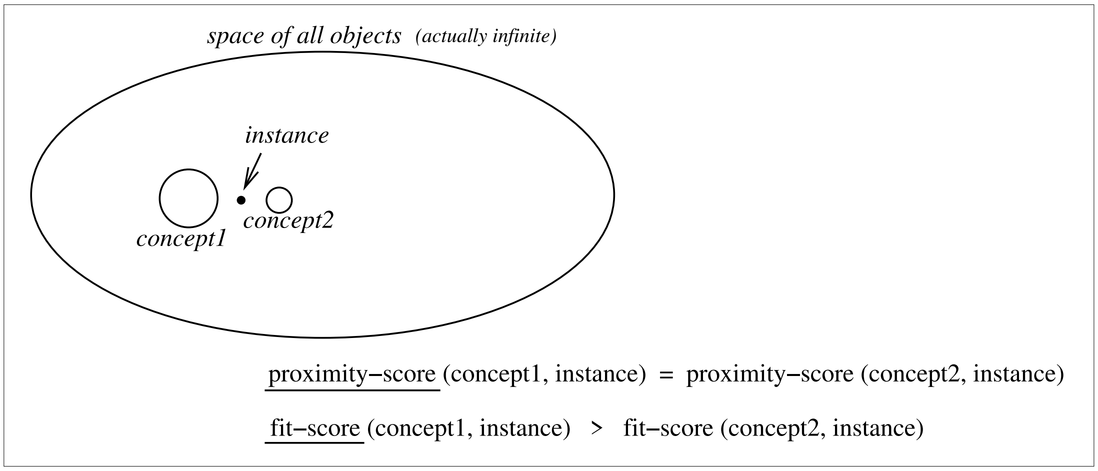


Figure 4.21: Proximity-scoring *versus* Fit-scoring

4.3.9 Structure and context interpretations affect similarity.

Section 3.4.3 of chapter 3 explained the various types of *interpretation* that each structure description and context description can have, including *complete, partial, disjunctive, imported, any, partial+disjunctive, partial+imported,* and *partial+typical.* Each of these affects the definition of similarity. To simplify the discussion, most of the examples will refer only to structure similarity, but the same points apply to context similarity. Also, the examples deal with the situation of matching a concept with an instance, except where indicated otherwise.

The previous examples of structure similarity in this chapter have involved structures with *complete* interpretation, in which case the best correspondences of *all* relationship/relatees of both structures contribute to the similarity score. If, at the other extreme, the concept structure has an *any* interpretation, then it matches perfectly with everything. However, the weighting of such a structure will normally be zero so that the "perfect score" does not contribute to the overall object comparison score anyway.

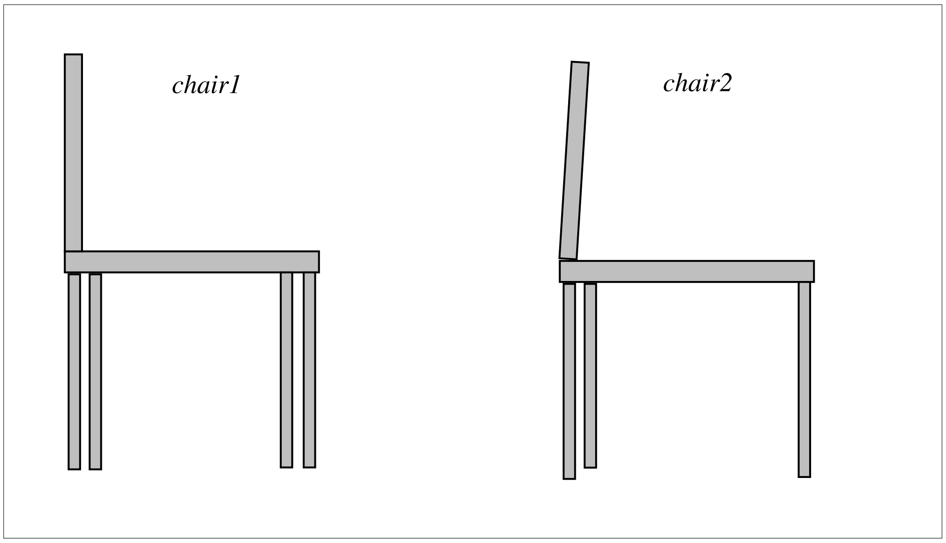


Figure 4.22: Proximity-scoring *versus* Fit-scoring

If the concept structure is *partial* and the instance structure is *complete*, then only the winning correspondences for the *concept's* relationship/relatees contribute to the score. This is illustrated in Figure 4.23 where the subparts of the *complete* structure of *B1* that 'miss out' do not affect the similarity score, and therefore *A1* and *B1* have quite a high measure of similarity. However, since a structure description that has *partial* interpretation must also have a high *variance*, this prevents the structure similarity from contributing significantly in the overall similarity score, unless the context similarity score also has a low weighting.

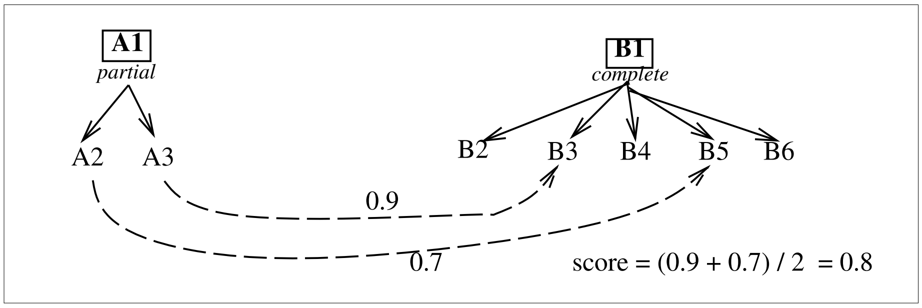


Figure 4.23: *partial* compared with *complete*

If both objects being matched are generalised concepts, and if both structure descriptions are *partial*, then there are two ways of measuring similarity, depending on whether we require fit-scoring or proximity-scoring: In the case of fit-scoring, the first concept is assumed to be more generalised than the second, and therefore the subparts of the first concept should be a subset of the second concept's subparts, since the former can be assumed to be more partial, as a consequence of the generalisation process. Therefore, only the winning correspondences for the first concept's relationship/relatees contribute to the similarity score, and any of the second concept's subparts that miss out do not affect the score, just as for the *partial:complete* situation above. An example of this is given in Figure 4.24, where only the best correspondences involving *A2*, *A3*, and *A4* contribute to the similarity score, using equation (a), thus allowing any additional *B1* subparts to be present without affecting the score.

In the case of proximity-scoring it is less clear how to measure similarity. GRAM simplifies the problem by assuming that both concepts are equally general, and should have the same partial set of relationship/relatees to be considered similar. Therefore, the similarity score is measured in the same way as for *complete:complete* similarity, where all winning correspondences of both concepts contribute to the score. This is shown by equation (b) in Figure 4.24.

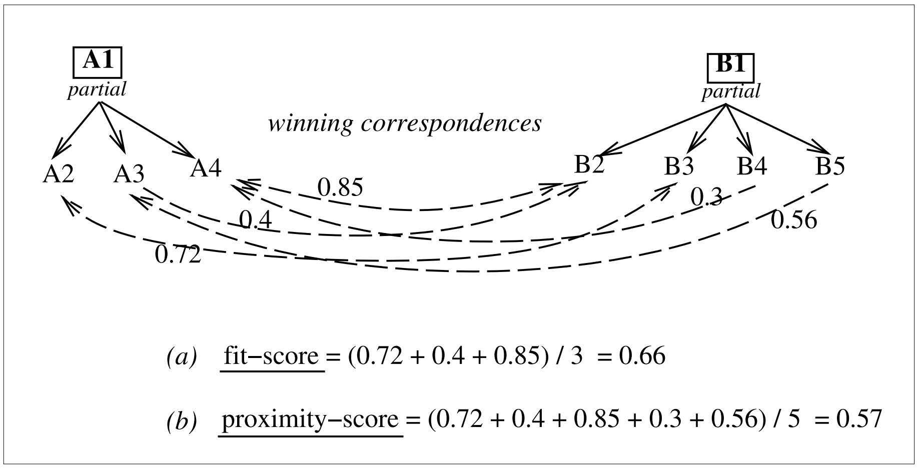


Figure 4.24: *partial compared with partial*

The similarity of disjunctive concepts is the similarity of the best disjunct pairing.

When the structure (or context) of the concept being matched has a *disjunctive* interpretation, then similarity is defined by the highest disjunct (*i.e.* subconcept) similarity, using *structure-only* scope restriction. For example, Figure 4.25 shows a *chair* concept whose structure is defined disjunctively by the subconcepts *kneelerchair*, *armchair*, and *swivelchair*. The structure-similarity of *chair* and *chair1* is defined as the highest score of the structure-only similarity of *chair1* and the disjunct subconcepts.

If the *chair* concept also includes some *partial* structure, then this partial structure is matched with the *chair1* structure in the manner defined earlier for *partial:complete* comparisons, and the similarity score is combined with the best disjunct similarity to give the overall structure similarity.

If the two objects being matched are both generalised concepts (such as when matching the relatees of two multi-relationships, or when reorganising concept memory), and both have disjunctive structures, then the comparison is more complex. For example, consider the three shelves in Figure 4.26, for which each shelf and each shelf-base are described in terms of a multi-relationship to a disjunctively-defined concept, such as *shelf1-item*. The disjuncts for

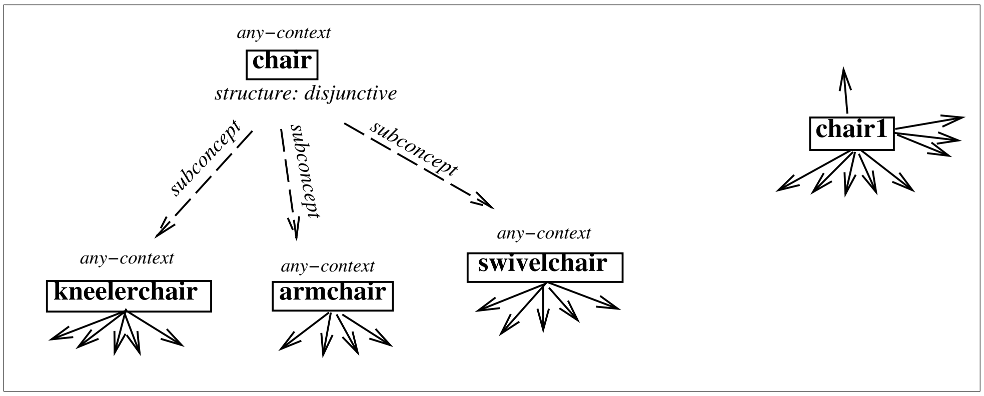


Figure 4.25: Disjunctive structure comparison

each shelf-item concept are indicated with frequency ratios indicating how many instances of each disjunct contributed to the generalisation.

If we are matching the disjunctive concept *shelf1-item* with the disjunctive concept *shelf2-item*, then the best correspondences between the disjuncts are identified in the same manner as for finding winning correspondences for parent, neighbour, and subpart relationship/relatees, as illustrated in Figure 4.27. In other words, the best correspondence for each disjunct contributes to the score, even if it conflicts with other winning correspondences. A single similarity score is defined as an average of these winning scores, as shown by the equations (a) and (c) in the figure. Although not shown on the figure (for simplicity) these scores are weighted by the instance-frequencies (*i.e.* the ratio of the instance-counts to the concept's instance-count).

In addition to the disjunct similarities, the similarities between the instance frequencies are also averaged, and both of these scores are combined to form the overall disjunction similarity score. For example, when matching *shelf1-item* with *shelf3-item*, the differences in instance-frequencies for the pots, jugs, and frypans lower the similarity score, as shown in equation (d) of Figure 4.27, where instance-frequency similarity is only 0.74 rather than 0.89 (from equation (b)).

In the current version of GRAM, disjunction similarity for two generalised concepts is defined in the same way for both fit-scoring and proximity-scoring. It has not yet been determined how to meaningfully define and distinguish the two types of scoring when dealing with two generalised disjunctive descriptions.

If the structure or context of one concept is defined by an *'import-from'* specification that refers to a disjunction of other concepts, then similarity is defined in the same way as for ordinary disjunction, since both specify a list of concepts, with associated instance-counts.

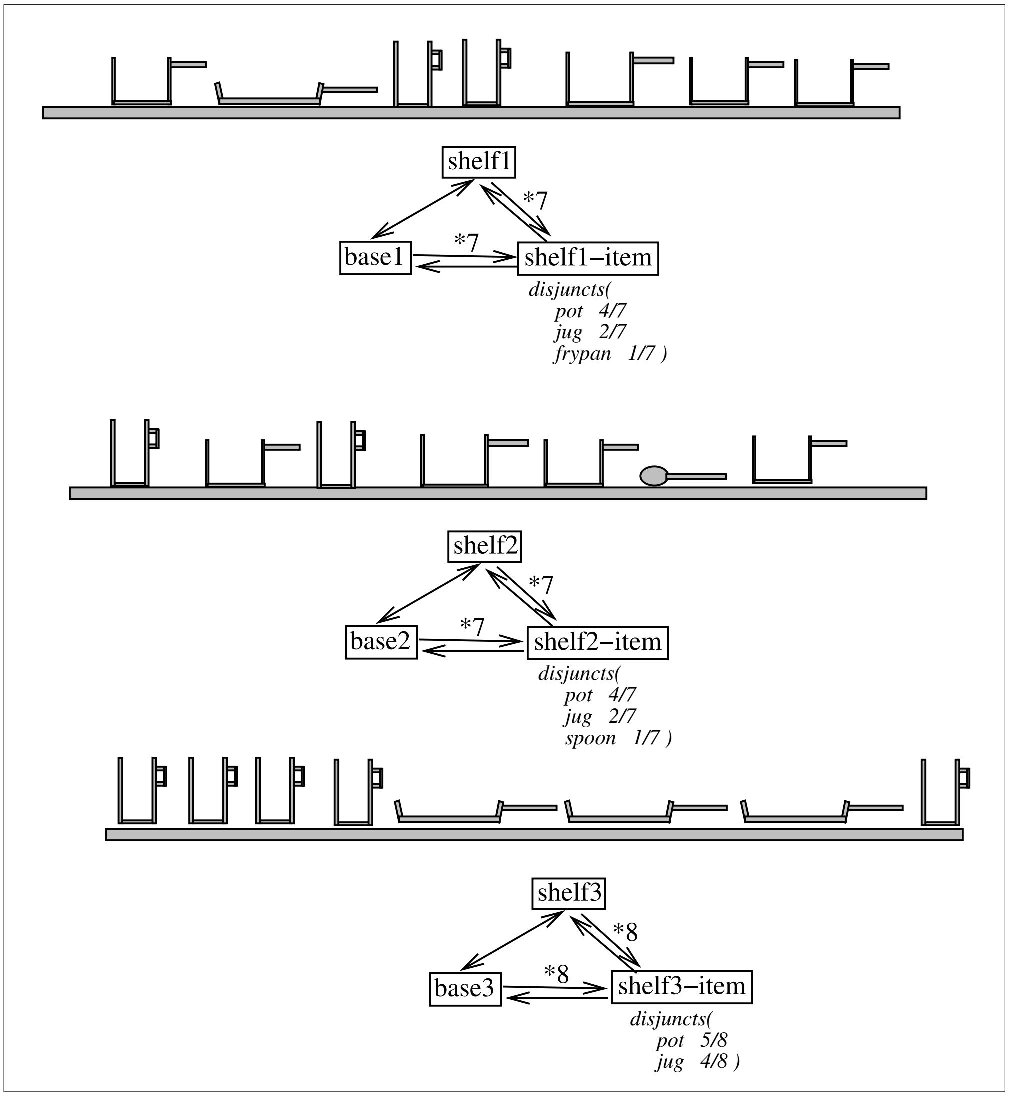


Figure 4.26: Similarity of two disjunctive concepts.

4.3.10 Superconcept and subconcept similarity can be used to estimate the score.

Section 4.2.6 stated that the matcher may sometimes be asked to match two objects, one of which one has already been matched with a superconcept or a subconcept of the other. In this situation, an estimate of similarity should be definable in terms of the subconcept or superconcept similarity. This section explains how this can be done, although it has not yet been implemented in the GRAM system. It is only applicable to proximity-scores.

Figure 4.30 (a) shows an example where a measure of similarity for *object23* and *chair* is required, given that the similarity score between *object23* and *swivelchair* has already been computed. This score can be used as a *lower-bound* on the required score, as illustrated abstractly in (b) of the figure: This shows the boundaries of a concept and a subconcept, and an object (the dot) to be compared with the concept. The length of the line from the object to

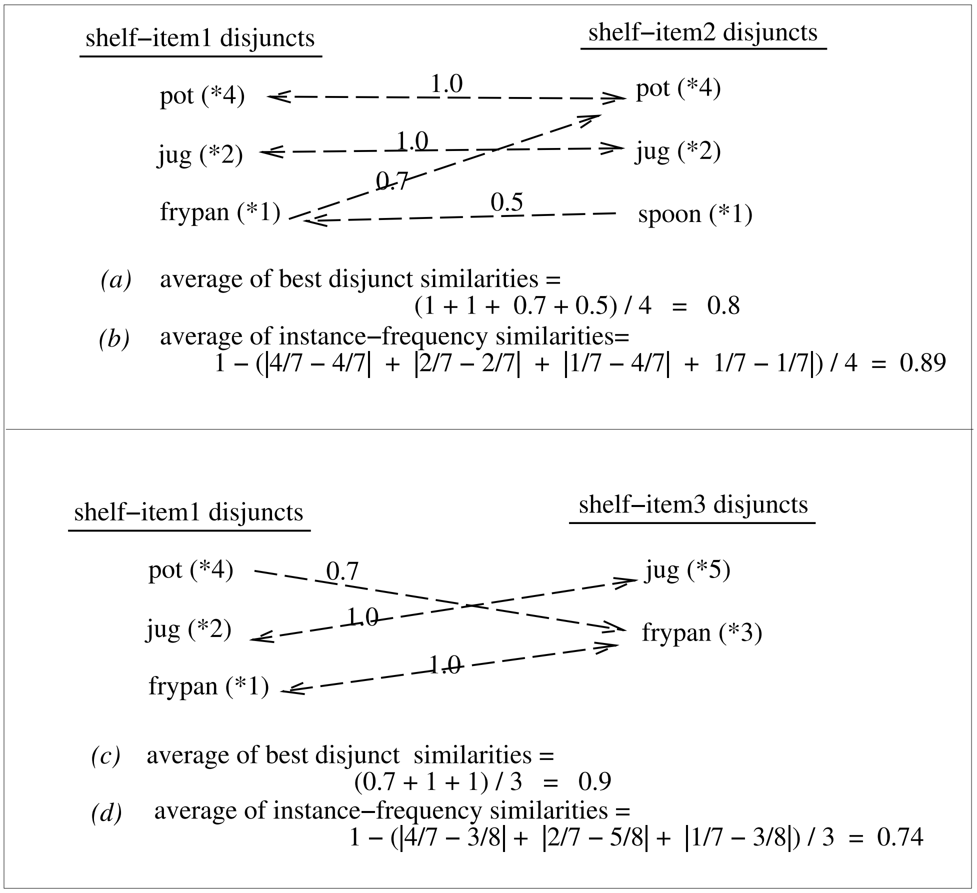


Figure 4.27: Similarity two disjunctive concepts.

the boundary of the subconcept is inversely proportional to the similarity score, and it can be seen that this is a lower-bound on the proximity-score for the object and the concept

However, this assumes that the subconcept is sufficiently typical of the concept that it lies within the concept boundaries relative to which proximity-cores are measured. Figure (c) shows why the object:subconcept score is not a true lower-bound if this is not the case. Therefore, the estimated similarity score must be based not only on the available object:subconcept score, but also on the *typicality* of the subconcept with the concept, assuming that this is recorded in each subconcept description.

A lower-bound score obtained in this way is useful because it may enable the matcher to immediately abandon further evaluation of other object comparisons if their upper-bound scores are lower than this lower-bound.

Proximity-scores can be estimated in a similar manner when an object has already been matched with a *superconcept.* This situation is shown in Figure 4.29 (a), where a measure of similarity between *object23* and *swivelchair* is to be defined in terms of the already-available similarity score between *object23* and the superconcept *chair.*

In this situation, the superconcept similarity score is an *upper-bound* on the required score.

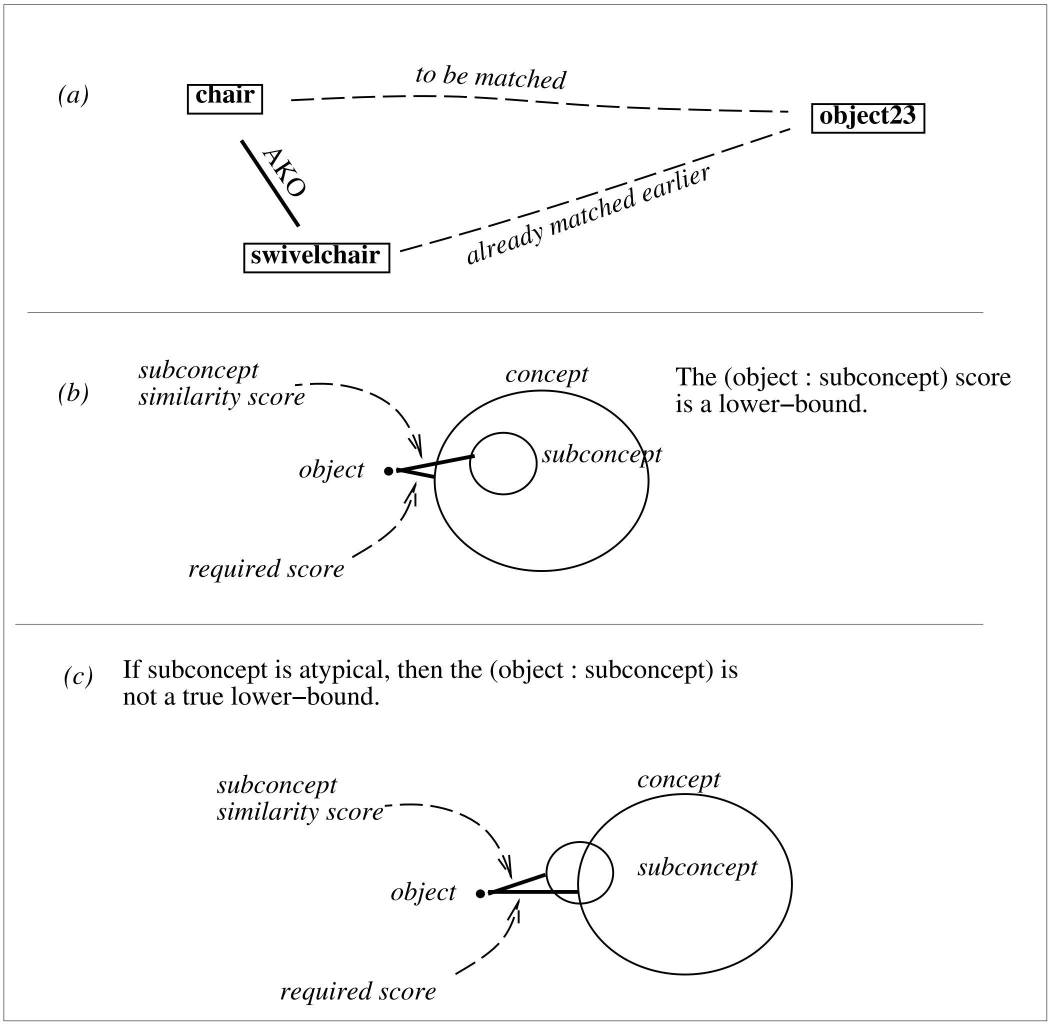


Figure 4.28: Similarity using subconcept and superconcept similarities.

The reason for this can be seen in (b), where the distance from the object to the superconcept (which is inversely proportional to the measure of similarity) is less than the distance to the concept. However, as before, this assumes typicality of the concept within the superconcept, and so the upper-bound must be modified according to the measure of typicality specified in the concept description.

An upper-bound score obtained in this way is useful because it may enable the matcher to immediately abandon further evaluation of the comparison if that upper-bound is lower than the minimum score required by the larger system that invoked the matcher, such as when the classification system has already found one classification for *object23*, and is trying to find a better classification.

Sometimes the matcher might be asked to match two objects which have both been previously classified as belonging to different subconcepts of the same superconcept. Figure 4.31 (a) gives an example of this situation. Suppose that *object1* in *office1* has already been matched well

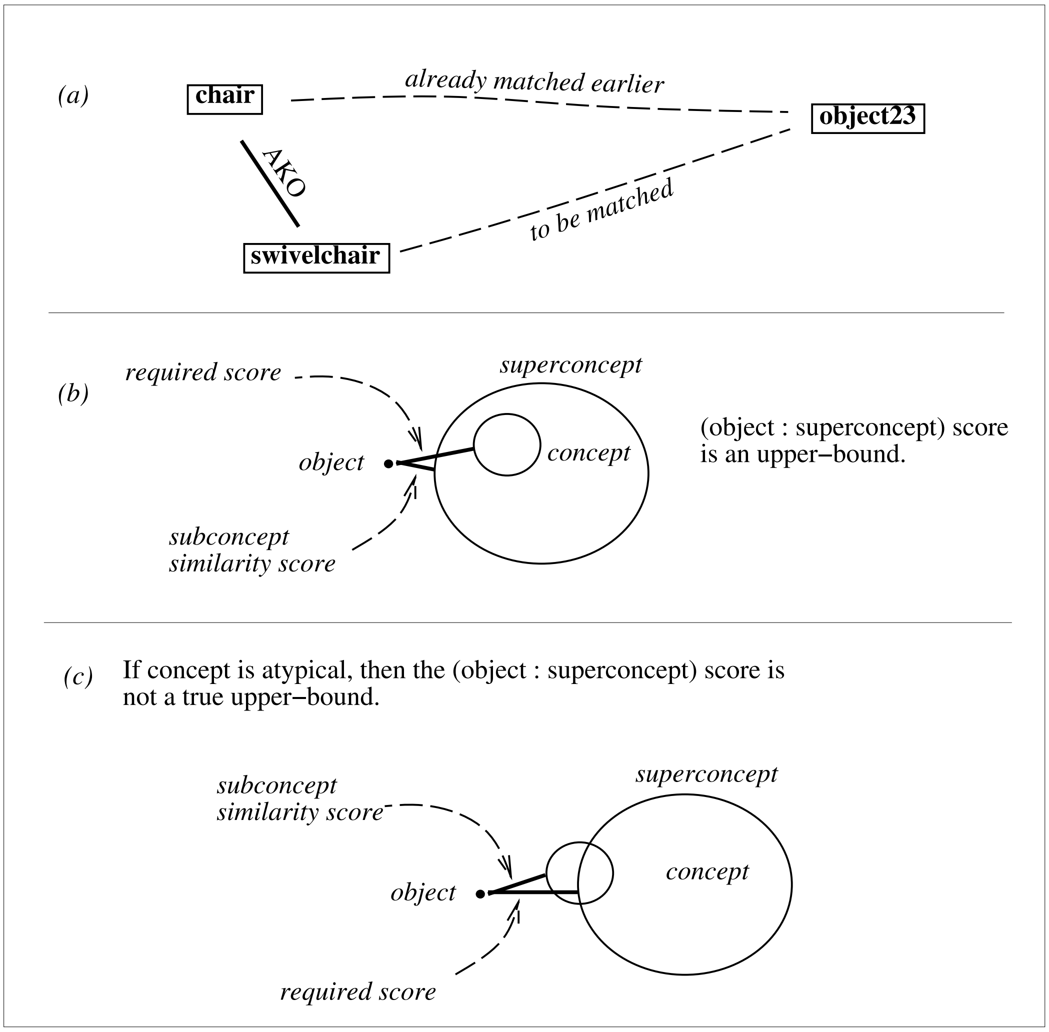


Figure 4.29: Similarity using subconcept and superconcept similarities.

with the concept *swivel-chair*, perhaps during instance-construction, but has not been explicitly matched with the more general concept *chair*. Suppose also that *object2* in *office2* has already been matched well with the concept *standard-chair*. If the system is now required to compare *office1* and *office2*, then in doing so it will need to compare *object1* and *object2*. Given their previous classifications, an estimated measure of similarity can be defined in terms of the similarity of *swivel-chair* and *standard-chair*, and this can be estimated in terms of the variance of their common superconcept, *chair*. More specifically, an *upper-bound* on the *object1-object2* similarity can be defined in terms of the variance of the *chair* concept.

This is illustrated abstractly in (b) of the figure, where the large circle denotes the boundary of *chair*, and the smaller enclosed circles denote *swivel-chair* and *standard-chair*, with *object1* and *object2* lying within these. The maximum width of the *chair* boundary must be larger than the distance between *object1* and *object2*, assuming that the two subconcepts are sufficiently typical. If they are atypical, then the upper-bound must be modified accordingly.

The upper-bound score defined in this manner is most useful if the common superconcept is not much more general than the subconcepts. In other words, the lower its variance, the closer the upper-bound score will be to the actual score.

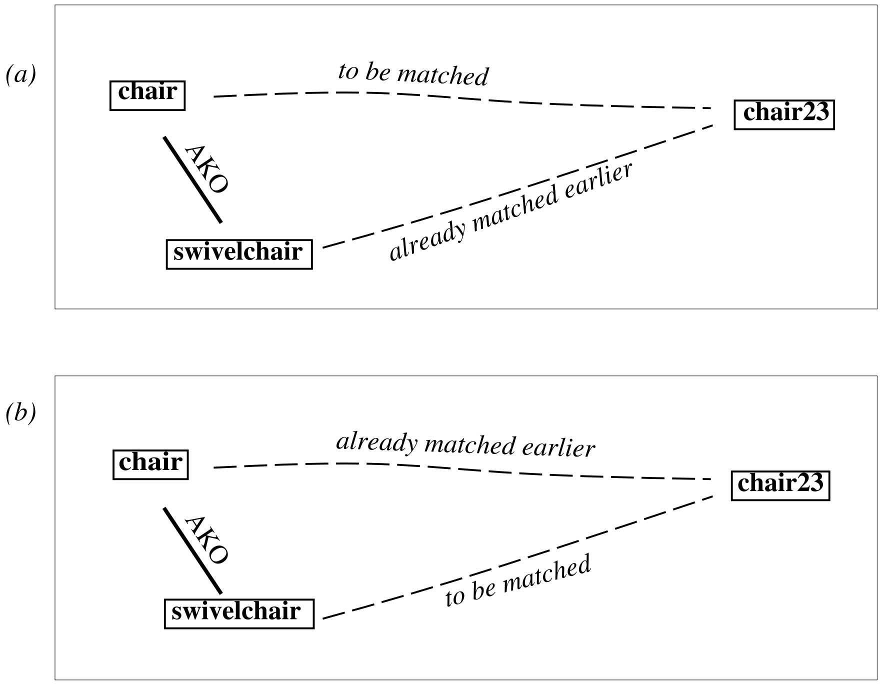


Figure 4.30: Similarity using subconcept and superconcept similarities.

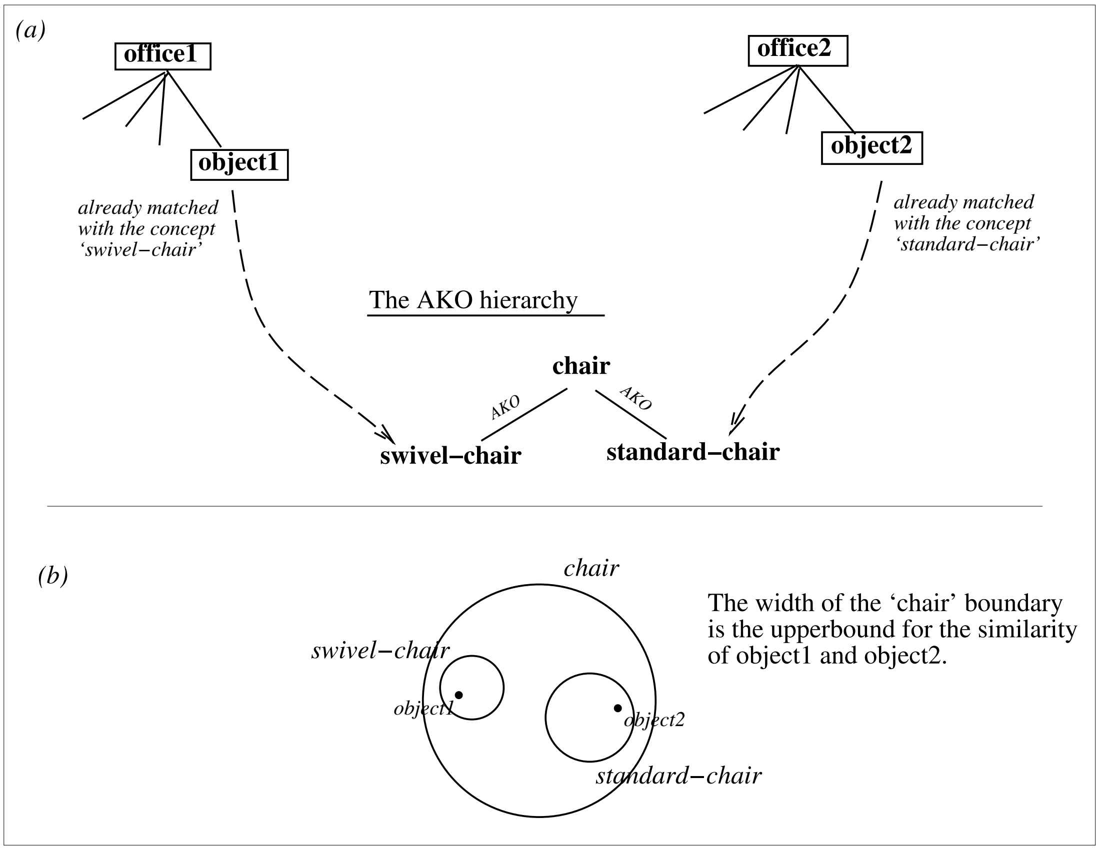


Figure 4.31: Similarity using superconcept similarities.

4.4 The Matching Algorithm

The previous section focussed on defining the meaning of 'similarity' in GRAM's matcher. This section considers the problem of how to evaluate a similarity score and describes GRAM's matching algorithm.

Section 4.1 considered the requirements of the matcher, and these included the requirement that descriptions of similarity (or dissimilarity) are produced as output, so that they can be used by other components of the system). These descriptions must specify scores for overall object similarity, structure similarity, context similarity, and relationship/relatee correspondences. The matcher must therefore have some way to represent this information, and this is discussed in section 4.4.1.

Central to the definition of similarity is the notion of 'winning correspondences' between the relationship/relatees of the two objects. Central to the design of the matcher is, therefore, the problem of how to search for these correspondences. The main difficulty is that a similarity score for a pair of objects is defined directly and indirectly in terms of similarity scores for a potentially vast number of other pairs of objects, including itself. Thus the matcher must employ techniques for pruning and controlling the search, and for enabling converging estimates of recursively-defined scores to be obtained. This is achieved by GRAM's "incremental-spread" algorithm which is discussed in section 4.4.2.

Various aspects of the process, such as level-hopping, disjunct comparison, and scope-restriction, are considered in sections 4.4.4 through 4.4.6. Section 4.4.7 discusses the problem of improving the accuracy of a similarity score by augmenting an instance graph with additional relationships and composite objects. Section 4.4.8 considers a situation where it is useful to be able to make use of subconcept and superconcept similarity scores.

A detailed description of the algorithm is given in section 4.4.10.

4.4.1 Match results are represented in *cnotes*.

The first issue to be addressed is how to represent the information produced by the matcher, both for its own use during the search, and for use by other systems such as the generaliser and fault-finder. The information in these descriptions must specify scores for overall object similarity, structure similarity, context similarity, and relationship/relatee correspondences.

GRAM represents match information in *cnotes*, which is short for 'comparison-note', and was borrowed from [Winston, 1975]. Each cnote specifies comparison information about two descriptive entities, such as objects, structures, contexts, or relationship/relatees. The output of the matcher is a single *object-cnote*, which consists of a *structure-cnote*, a *context-cnote*, an overall similarity score, and an *effort* value which indicates how much effort was applied to produce the similarity score. If the two objects have been generalised, then a pointer to the concept is also included.

A *context-cnote* specifies the comparison between two contexts. It includes a context-similarity score, a cnote that specifies the similarity of the context properties, and the winning (and losing) correspondences between the parent and neighbour relationship/relatees of the two contexts. Each of these correspondences is represented in a *correspondence-cnote* which describes the comparison between the two relationships and the comparison between the two relatee objects. If either or both of the concepts have disjunctive contexts, then *object-cnotes* describing the comparisons of each disjunct (*i.e.* subconcept) pairing are also included.

A *structure-cnote* specifies the similarity between two structures in much the same way as for a *context-cnote*, except that it consists of *subpart correspondence-cnotes* rather than parent and neighbour *correspondence-cnotes*. Also, a *structure-cnote* may include a *contents-similarity* score.

GRAM stores each *object-cnote* so that it is directly accessible from both of the objects it involves. This is important because it allows the matcher to immediately find all of the comparisons produced for any object during the search for winning relationship/relatee correspondences, since each object may have been matched with many other objects.

Although the matcher only produces one *object-cnote* as direct output, it may also generate many other *object-cnotes*, since during the search for winning relationship/relatee correspondences, each object may be matched with many other objects. These *object-cnotes* can be considered to form a *cnote-graph*, as illustrated in Figure 4.32, where the similarity score for each cnote is defined in terms of the similarity scores of its related cnotes.

object-cnotes could be retained in concept memory, long term, to provide information about the similarity or difference between concepts. Such cnotes could be considered to be 'difference links' [Bareiss and Porter, 1987] which provide another means for accessing items in memory.

For reference, the following gives a summary definition of each of the main kinds of cnote.

object-cnote =

- Score (of overall-similarity). (including a lower and upper bound)
- A *structure-cnote.*
- A *context-cnote.*
- Effort-applied.
- A concept. *(if the objects have been generalised)*

context-cnote =

- Score (of context-similarity).
- A *context-properties-cnote*
- *Correspondence-cnotes* (for the relationship/relatees of the two objects.)
- Disjunct *object-cnotes* *(if necessary)*

structure-cnote =

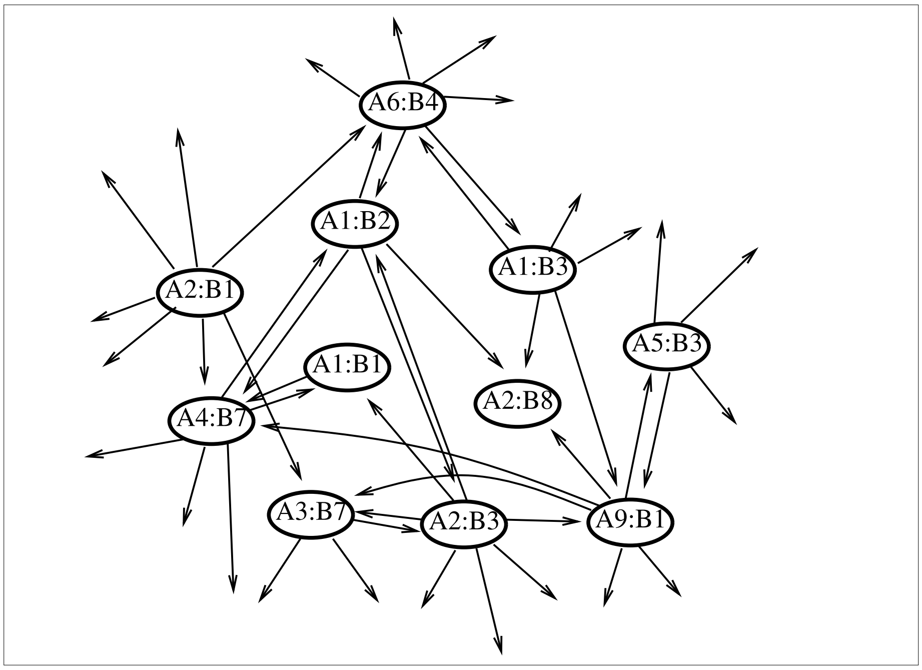


Figure 4.32: An *object-cnote* graph.

- Score (of structure-similarity)
- A *structure-properties-cnote.*
- *Correspondence-cnotes* (for the relationship/relatees of the two objects.)
- Disjunct *object-cnotes* (if necessary)
- Contents-similarity score (optional)

correspondence-cnote =

- Score. (including a lower and upper bound)
- A *relationship-cnote* (for the two relationships)
- An *object-cnote* (for the two relatees)

4.4.2 The "Incremental-Spread" search strategy.

This section considers how to search for the winning correspondences of relationship/relatees of two objects being matched, and thereby obtain a similarity score. We begin by considering the simplest, most obvious algorithm, and then consider various refinements to improve efficiency and to account for circularities, recursive similarity, disjunction, effort control, scope restriction, and augmentation.

Figure 4.34 shows two television sets, of which the component objects *tvmain1* and *tvmain2* (*i.e.* the television set excluding the aerials and legs) are to be compared. (This particular comparison is considered, rather than the comparison of the whole televisions, so that parents and neighbours need to be matched.) The simplest algorithm is a depth first search which recursively invokes the matcher for every pairing of parent relationship/relatees, every pairing of neighbour relationship/relatees, and every pairing of subpart relationship/relatees. These pairings are indicated by the dotted lines between the relationship/relatees at the bottom of Figure 4.34. The similarity scores produced for these correspondences are then used to determine the winning correspondences to contribute to the similarity score. This algorithm is shown below:

```
MATCH-OBJECTS ( object1, object2)

  For each parent of object1:
    For each parent of object2:
      Match the relationships.
      MATCH-OBJECTS (parent1, parent2).
      Compute the correspondence score.

  Repeat for neighbours and subparts.

  Select winning correspondences.

  Compute similarity score.
```

Figure 4.33: A simple (and ineffective) algorithm.

There are two major problems with this algorithm. Firstly, it is highly inefficient because it performs a complete depth first search for every relatee correspondence. For each object comparison, the number of invocations of the matcher is $O(n^2)$, and each of these invocations leads to a further $O(n^2)$ invocations, extending on through the object graph. The second problem with the algorithm is that it does not work: it rapidly gets stuck in infinite cycles, reevaluating comparisons that are currently already being evaluated. This is because objects are defined in terms of other objects, and vice versa, and therefore the measures of object similarity are recursively defined in terms of themselves.

The first problem requires the matcher to prune the search by abandoning a comparison as soon as it is clearly not a winning relationship/relatee correspondence. This means that a depth first search is inappropriate, since a depth-first search completely evaluates a correspondence before evaluating another, even though a partial evaluation might be sufficient to reject it if partial evaluations of other correspondences are available.

The second problem requires that the matcher should not recursively invoke itself to perform a comparison that is already being processed. Rather it must be able to make use of estimates of similarity scores for comparisons that are currently active.

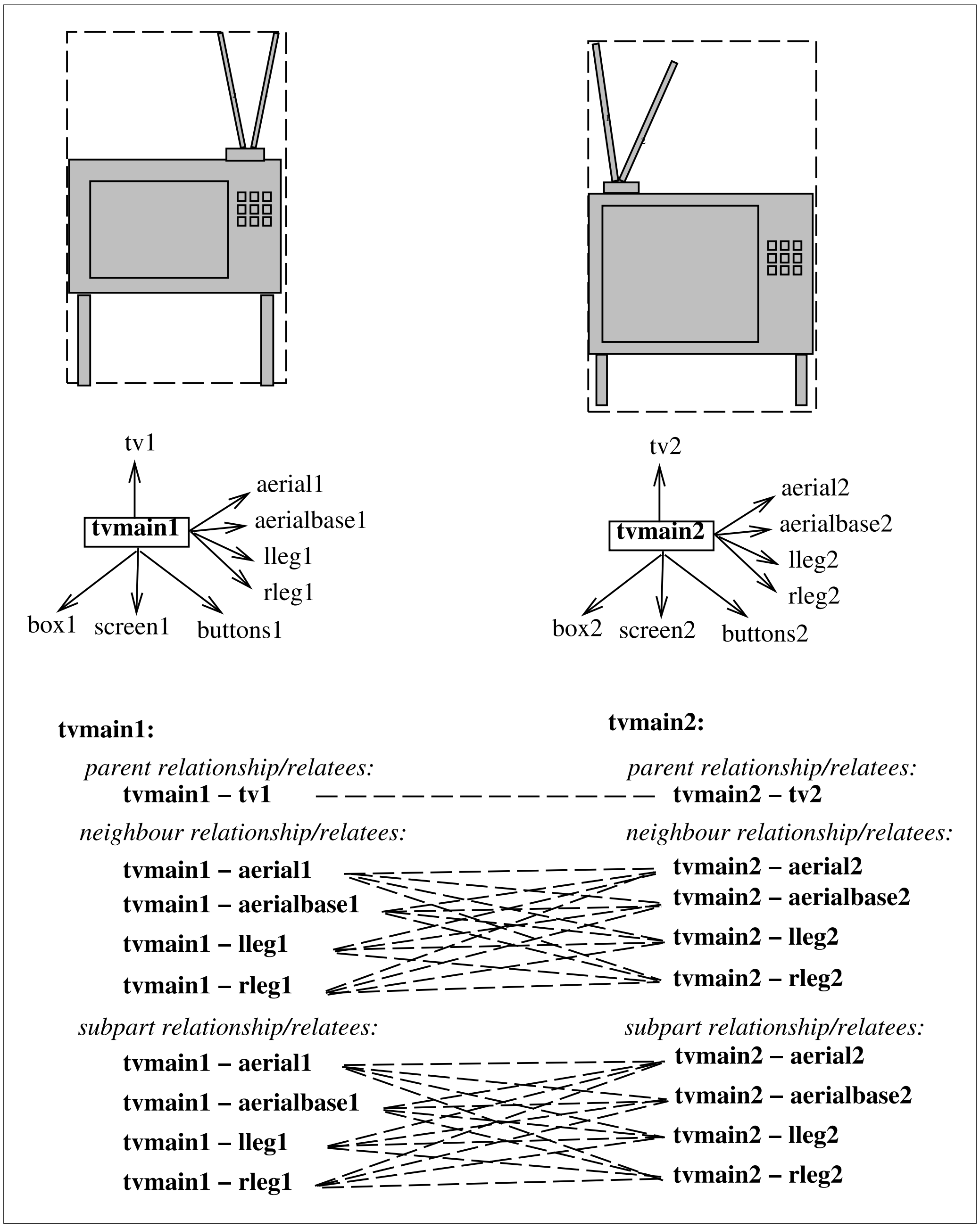


Figure 4.34: The search problem.

GRAM's "incremental spread" algorithm is a breadth-first beam search using iterative deepening.

GRAM's matching algorithm deals with the above problems by using a breadth-first beam search with iterative deepening, as explained below:

The matcher begins by performing a "1-spread" comparison of the two objects. This means that the structure and context properties are compared, and all pairings of relationship/relatee correspondences are evaluated comparing only the relationship descriptions, ignoring the relatees. The winning correspondences are then selected from these, and a similarity score computed. An approximate lower-bound and an upper-bound on the score is also computed, based on a predefined measure of inaccuracy of a 1-spread comparison. Thus, a 1-spread comparison gives a rough and inexpensive estimate of object similarity, and also provides estimates of the relationship/relatee correspondences scores. In fact, GRAM actually first performs a 0-spread comparison, which only considers properties, and the only continues with the 1-spread comparison if the score is sufficiently high.

The matcher then performs a "2-spread" comparison. This involves recursively invoking the matcher to perform a 1-spread comparison on each pair of relatees. Thus the matcher is extending or deepening its 'horizon' or 'fringe' incrementally. However, it only invokes the matcher on a relationship/relatee correspondence that could potentially become a winning correspondence for either or both of the two relationship/relatees. This is determined on the basis of lower and upper bounds that are computed and stored with each *correspondence-cnote*: If the upper-bound for a relationship/relatee correspondence is higher than the lower-bound of a currently winning relationship/relatee correspondence involving either of the two relatees, then the relatees are matched using a 1-spread comparison. Otherwise that correspondence is ignored. Later it might become a potential winner again, if the score of the winner drops sufficiently as a result of more accurate comparison.

After reevaluating the potentially winning (and already winning) correspondences, a new set of winning correspondences is identified, and a more accurate similarity score is thereby computed.

The matcher next performs a 3-spread comparison in exactly the same way as for a 2-spread comparison, except that the potentially-winning relationship/relatee correspondences are reevaluated by recursively invoking the matcher using a *2-spread* comparison on the pairs of relatees. Each of these comparisons will cause (some) pairs of relatees of those relatees to be compared using a *1-spread* match.

A 4-spread comparison is then applied, and so on, up to an n-spread, where n is the required effort for the comparison. Thus, the matcher is incrementally spreading outwards (using a kind of breadth-first beam search) through the object graphs via good correspondences between parent, neighbour, and subpart relationships. Figure 4.35 shows a rough hand-generated illustration of the object comparisons that are considered in a 0-spread, 1-spread, 2-spread, and 3-spread comparison, with *A1* and *B1* being the root objects of comparison. At each increased effort, the fringe of the comparison extends outwards. The different shadings on the object-boxes indicate the spread-level at which an object is considered by the matcher.

Effort is controlled by a *required-spread* parameter.

Earlier we discussed the requirement that the matcher must have an effort-control parameter, and this in fact defines how much spread is applied to a comparison. In the rest of this chapter

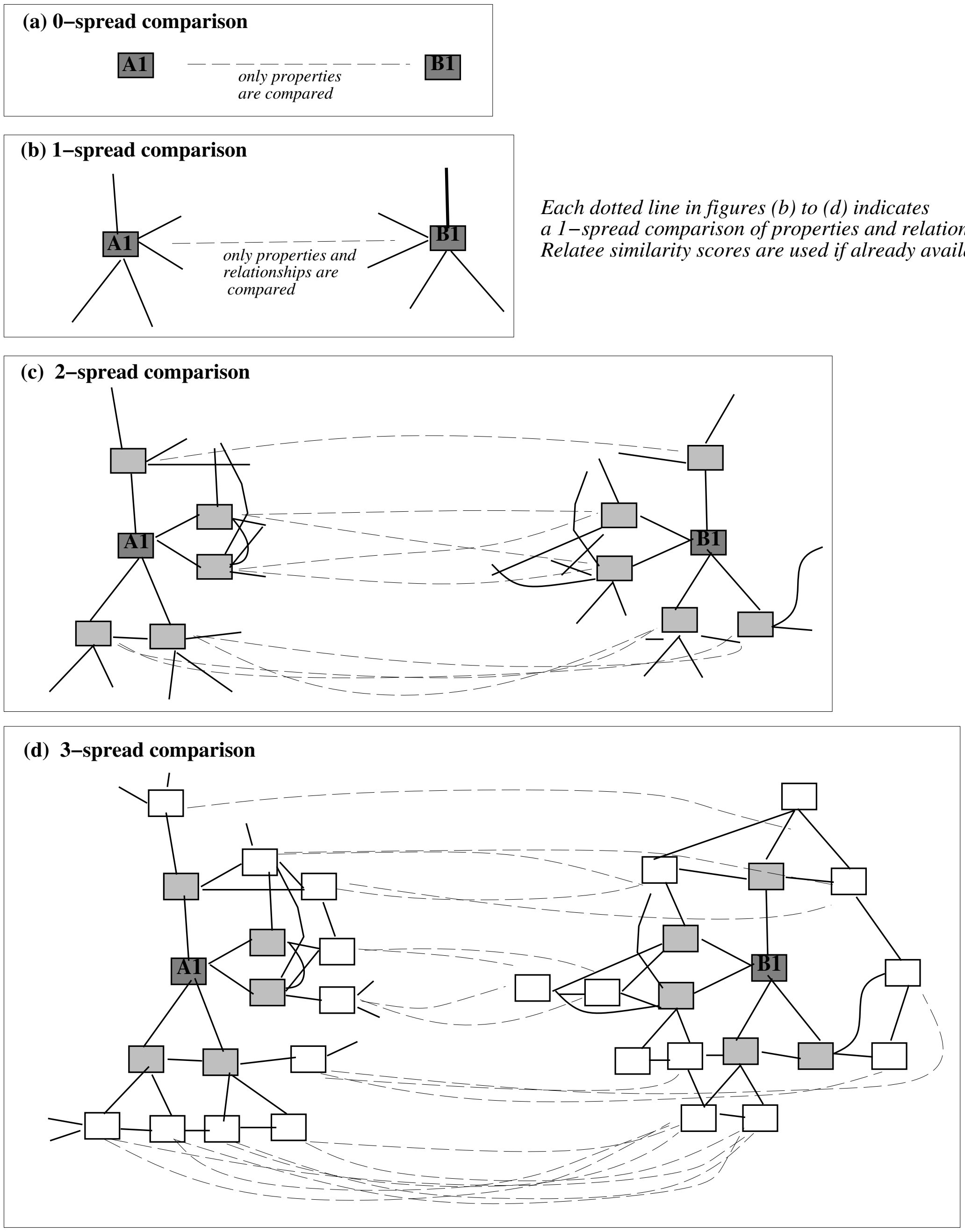


Figure 4.35: 0-spread, 1-spread, 2-spread, and 3-spread comparisons.

it is called the *required-spread* parameter.

Each time the matcher recursively invokes itself to compare pairs of relatees, it passes some value of *required-spread*. Likewise, the larger system that originally invokes the matcher must also specify *required-spread*.

A standard top-down comparison could achieve this kind of effort-control by restricting the depth of traversal down the subpart hierarchy, but in GRAM the effort control is also applied to *context* matching via parent and neighbour relationships.

Effort is also controlled by the *acceptability-cutoff* and *rejection-cutoff* parameters.

The effort applied by the matcher is also controlled by two optional parameters which can force the matcher to prematurely abandon the comparison.

The *rejection-cutoff* parameter prevents unpromising matches from being explored further. For example, when performing rapid classification of large numbers of objects in a scene, we do not want the matcher to spend much time matching an object with a concept when it is already clear from a rough comparison that the similarity is worse than the object's similarity to some other concept previously considered. Thus the *rejection-cutoff* parameter indicates the minimum similarity score required. As soon as the upper-bound of the *object-cnote* score is worse than the rejection-cutoff, then it is not worth evaluating the comparison more thoroughly, and the match is abandoned.

The *acceptability-cutoff* parameter prevents good comparisons from being evaluated further. Sometimes when performing rapid classification it is desirable to accept a classification as soon as it is clearly better than any other classification. Thus it is only concerned with finding a comparison with a sufficiently good score, rather than obtaining an exact measure of similarity. If this is the requirement, then whenever the lower-bound of the *object-cnote* score is higher than the *acceptability-cutoff* score, then no further evaluation of the comparison is necessary.

Cycles are avoided by recording how much spread has been applied.

We have not yet addressed the problem of getting stuck in infinite cycles. This is easily solved by the "incremental-spread" algorithm: Firstly, each *object-cnote* must record the effort (or more precisely, the *spread*) that has already been applied to a comparison. Secondly, if the matcher is invoked on a pair of objects, A and B, which have already been matched, and whose current spread is already at least as high as the required-spread, then nothing needs to be done.

The algorithm in its most basic form.

Having discussed the main features of the algorithm in its most basic form, it is now possible to presents it more formally, as given in Figure 4.36. The first step of MATCH is to create a cnote if the two objects have not already been matched. It then matches relationship/relatees with an incrementally increasing spread, until the *required-spread* has been completed.

Refinements to the algorithm are considered in the remainder of the chapter, and a more complete description of the algorithm is given section 4.4.10.

```
MATCH ( object1,          {the concept (or perhaps an instance)}
        object2,          {the instance  (or perhaps a concept)}
        required-spread     {the spread-effort to be applied}
      )

  IF object1 and object2 have already been matched THEN
     cnote ← the recorded cnote.
  ELSE
     cnote ← create a cnote using a 0-spread comparison.
          (i.e. compare properties)
     IF the score is sufficiently high,
        Evaluate the cnote using a 1-spread comparison.
          (i.e. compare pairs of relationships)

  WHILE  (cnote-spread < required-spread)
    AND (cnote-score is sufficiently high)

     Increment cnote-spread.
     FOR each potentially-winning relationship/relatee correspondence:
        MATCH (relatee1, relatee2, ( cnote-spread – 1))
     Reselect winning correspondences.
     Recompute structure, context, and overall similarity scores.

  RETURN the cnote.
```

Figure 4.36: The “incremental spread” algorithm (in its most basic form)

A 1-spread comparison can make use of available relatee similarity scores.

Although a 1-spread comparison does not recursively invoke the matcher to compare relatees, it can make use of similarity scores for any pairs of relatees that have been already compared. Thus a 1-spread comparison may actually be more accurate than its spread-effort suggests. In fact this is true at any spread level, since every relationship/relatee reevaluation can make use of existing relatee similarity scores even if they are more accurate than the reevaluation actually requires.

Scores converge iteratively.

This algorithm also accounts for the fact that similarity scores are defined recursively. For example, suppose two objects *A1* and *B1* are compared, and one of their winning neighbour relationship/relatee correspondences is between *A2* and *B2*, the similarity of which is defined in terms of the *A1:B1* correspondence. The *A1:B1* correspondence is first evaluated using a 1-spread search and then a 2-spread search, neither of which require the *A2:B2* comparison to refer back to the *A1:B1* score. When a 3-spread comparison for *A1* and *B1* is performed, it will require the *A2:B2* comparison to obtain a 1-spread score for the *A1:B1* comparison. In fact, a 2-spread estimate of this is already available, and so it can be used immediately in the *A2:B2* comparison. The resulting *A2:B2* score is used to compute the 3-spread score of *A1:B1*, which will then have been computed in terms of itself (or rather, a previous estimate of itself). The same process continues as the spread increases, and the recursively-defined similarity score is iteratively improved in accuracy.

An informal proof of convergence is based on the fact that each best relationship/relatee correspondence for an object comparison only contributes roughly $1/n$th of the score, where n is the number of relationships of the object. Therefore, in the example above, $1/n$th of the *A1:B1* score comes from the *A2:B2* score, which in turn comes from $1/n$th of the *A1:B1* score. Thus, when the *A1:B1* score changes, the effect of that change on itself has a contribution of at most $1/(n^2)$. This change will therefore be small, and so the change resulting from *that* change will be an order of magnitude smaller again.

The winning relationship/relatee correspondences for a cnote, as defined by GRAM's similarity measures, is guaranteed to be found if, firstly, sufficient effort has been applied so that an unambiguous clear winner has been identified for each relationship/relatee, and secondly, if the computed lower and upper bounds on the scores are correct. Since these bounds are only estimated by GRAM, occasional errors may occur.

Fewer relationship/relatee correspondences are evaluated for high-spread comparisons.

It may seem that a high-spread comparison of two objects will be very expensive to evaluate, since all potentially-winning relationship/relatee correspondences must be evaluated using a spread comparison of only 1 less than the required spread of the objects. However, the higher the spread that has already been applied to a comparison, the fewer potentially-winning relationship/relatee correspondences there will be. This is because the lower and upper bounds of the correspondence scores will define a much smaller error range, and so there will be fewer potential winners that need to be evaluated. Conversely, the score for a low-spread comparison is less reliable, and so there will be more relationship/relatee correspondences that need to be evaluated, but these can be evaluated at significantly less cost.

The algorithm is 'any-time'.

The incremental-spread algorithm satisfies the requirement for an 'any time' matcher, since if a comparison is interrupted at any point, the best estimate based on the comparisons performed so far is available. It also enables the amount of effort applied to a comparison to be explicitly controlled, thus allowing rough and rapid matching if necessary. It is also guided completely by the structure of the objects themselves, via relationships.

Since each increase in spread-effort has less effect on the score than the previous increase (because distant object correspondences contribute less to the similarity score than nearby object correspondences), an adequate measure of similarity for basic classification is often obtained from just a 2 or 3 spread comparison. This characteristic of the algorithm is based on the assumption that objects are defined primarily by their 'closest' details in the object graph.

Global consistency is not enforced.

Section 4.3 discussed how the definition of object similarity in GRAM does not require global (or even local) consistency between object correspondences. Consequently, the incremental-spread algorithm does not enforce consistency. Each comparison is performed independently from other comparisons, apart from making use of their results (in the manner of a backward-chaining rule-evaluation system). This makes the algorithm amenable to a parallel implementation to give significantly greater efficiency. Future versions of GRAM might explore this.

4.4.3 An example.

This section presents a simple example of the matching process for the comparison of the objects *tvmain1* and *tvmain2* in Figure 4.37. Object graphs for the two televisions are also shown. This comparison as chosen, rather than the *tv1:tv2* comparison, because it involves context matching. Although the graphs are of two instances, they can also be interpreted as graphs of concepts in concept-memory. It is assumed for simplicity that the *buttons* objects are defined only in terms of a single multi-relationship to a *button* object, without atypical subparts being included.

The trace of the search below does not show the actual similarity scores, since these are not important here. To avoid clutter, the trace only includes invocations of the matcher that require additional work to be performed. In other words, if a comparison requires a similarity score for a pair of relatees that have already been compared to the required spread effort, then that invocation of the matcher is not shown in the trace. The indentation indicates which invocations were called from within which comparison.

It begins by doing a 1-spread comparison of *tvmain1* and *tvmain2*, and then a 2-spread comparison, which invokes a number of 1-spread comparisons between its parents, neighbours, and subparts. Some of these 1-spread comparisons can make use of the 1-spread similarity score for *tvmain1* and *tvmain2*, and can also make use of the 1-spread scores for the comparisons above them in the trace, hence the scores will actually be more accurate than a 1-spread suggests.

A 3-spread comparison of *tvmain1* and *tvmain2* then applies a 2-spread comparison to the parent, neighbour, and subpart relationship/relatee correspondences that are potential winners. Most of these do not require much effort, since 1-spread scores for their relationship/relatee correspondences are already available. The only new correspondences identified are between the aerial subcomponents.

In the 4-spread comparison of *tvmain1* and *tvmain2*, even fewer correspondences need to be considered, since the others can be confidently considered non-potential winners. Most of those that do need to be considered do not have to spread far, since scores are already available at the required spread level.

Applying 5-spread, 6-spread, *etc* would not change the score significantly, since no new correspondences are identified, and most of the comparisons performed already have a higher accuracy that their spread-level indicates, since they have been able to make use of existing relatee similarity scores. This would not be the case if there was more context surrounding the televisions, unless scope-restriction was being employed.

1-spread (*tvmain1*, *tvmain2*)
2-spread (*tvmain1*, *tvmain2*)
    1-spread (*tv1*, *tv2*) *(uses the 1-spread tvmain1-tvmain2 score)*
    1-spread (*box1*, *box2*) *(uses the 1-spread tv1-tv2 score)*
    1-spread (*screen1*, *screen2*)
    1-spread (*box1*, *screen2*)
    1-spread (*screen1*, *box2*)
    1-spread (*buttons1*, *buttons2*)
    1-spread (*screen1*, *buttons2*)
    1-spread (*buttons1*, *screen2*)
    1-spread (*aerial1*, *aerial2*)
    1-spread (*aerialbase1*, *aerialbase2*)
    1-spread (*lleg1*, *lleg2*)
    1-spread (*rleg1*, *rleg2*)
    1-spread (*lleg1*, *rleg2*)
    1-spread (*rleg1*, *lleg2*)
    (many of these can now be rejected as potential winning correspondences)
3-spread (*tvmain1*, *tvmain2*)
    2-spread (*tv1*, *tv2*) *(uses the 2-spread tvmain1-tvmain2 score)*
    2-spread (*box1*, *box2*) *(uses the 2-spread tv1-tv2 score)*
    2-spread (*screen1*, *screen2*)
    2-spread (*box1*, *screen2*)
    2-spread (*screen1*, *box2*)
    2-spread (*buttons1*, *buttons2*)
    2-spread (*aerial1*, *aerial2*)
        1-spread (*aerialleft1*, *aerialleft2*)
        1-spread (*aerialleft1*, *aerialright2*)
        1-spread (*aerialright2*, *aerialleft1*)
        1-spread (*aerialright2*, *aerialright2*)
    2-spread (*aerialbase1*, *aerialbase2*)
    2-spread (*lleg1*, *lleg2*)
    2-spread (*rleg1*, *rleg2*)
    2-spread (*lleg1*, *rleg2*)

2-spread (*rleg1*, *lleg2*)
4-spread (*tvmain1*, *tvmain2*)
3-spread (*tv1*, *tv2*)
3-spread (*box1*, *box2*)
3-spread (*screen1*, *screen2*)
3-spread (*buttons1*, *buttons2*)
3-spread (*aerial1*, *aerial2*)
2-spread (*aerialleft1*, *aerialleft2*)
2-spread (*aerialleft1*, *aerialright2*)
3-spread (*aerialbase1*, *aerialbase2*)
3-spread (*lleg1*, *lleg2*)
3-spread (*rleg1*, *rleg2*)

4.4.4 Level-hopping is implicitly performed.

Section 4.2.4 discussed the level-hopping problem that occurs when two similar objects have been represented as different decomposition hierarchies such that corresponding components are not on the same level. For example, the two televisions in Figure 4.38 have been decomposed differently, so that in *tv1* the *main* object is a subpart of the *mainplus* object which includes the main part of the tv and the left and right legs. In *tv2*, the legs are considered to be subparts of the root object *tv2*, and there is no *mainplus* object. If a top-down search was employed, by finding correspondences at each level of the hierarchy, the correspondences between the legs and the *main* objects could not be found. However, GRAM's matching algorithm allows these correspondences to be found via the traversal of neighbour relationships, since it spreads in all directions through the object graphs.

This is shown by the abbreviated trace below. When spreading from the *tv1* and *tv2* comparison, the correspondence between *main1* and *main2* is not found. However, when performing the comparison between *aerial1* and *aerial2*, the correspondence between *main1* and *main2* is found via neighbour relationships. This comparison then also leads to the legs being compared.

However, the *tv1* and *tv2* comparison will still have a poor similarity score, since the *main1*:*main2* comparison does not contribute to it directly (only indirectly, via the aerial similarity score. This is resolved by *augmentation* techniques described in detail later. In brief, when reevaluating the *tv1:tv2* comparison to obtain its 3-spread score, the matcher notices that the subpart *main2* has been matched well with *main1*. This causes it to try and create a subpart relationship between *tv1* and *main1*, which consequently can be used to improve the score.

1-spread (tv1, tv2)
2-spread (tv1, tv2)
1-spread (mainplus1, main2)
1-spread (aerial1, aerial2)
1-spread (aerial1, llegs2)
1-spread (aerial1, rlegs2)
3-spread (tv1, tv2)

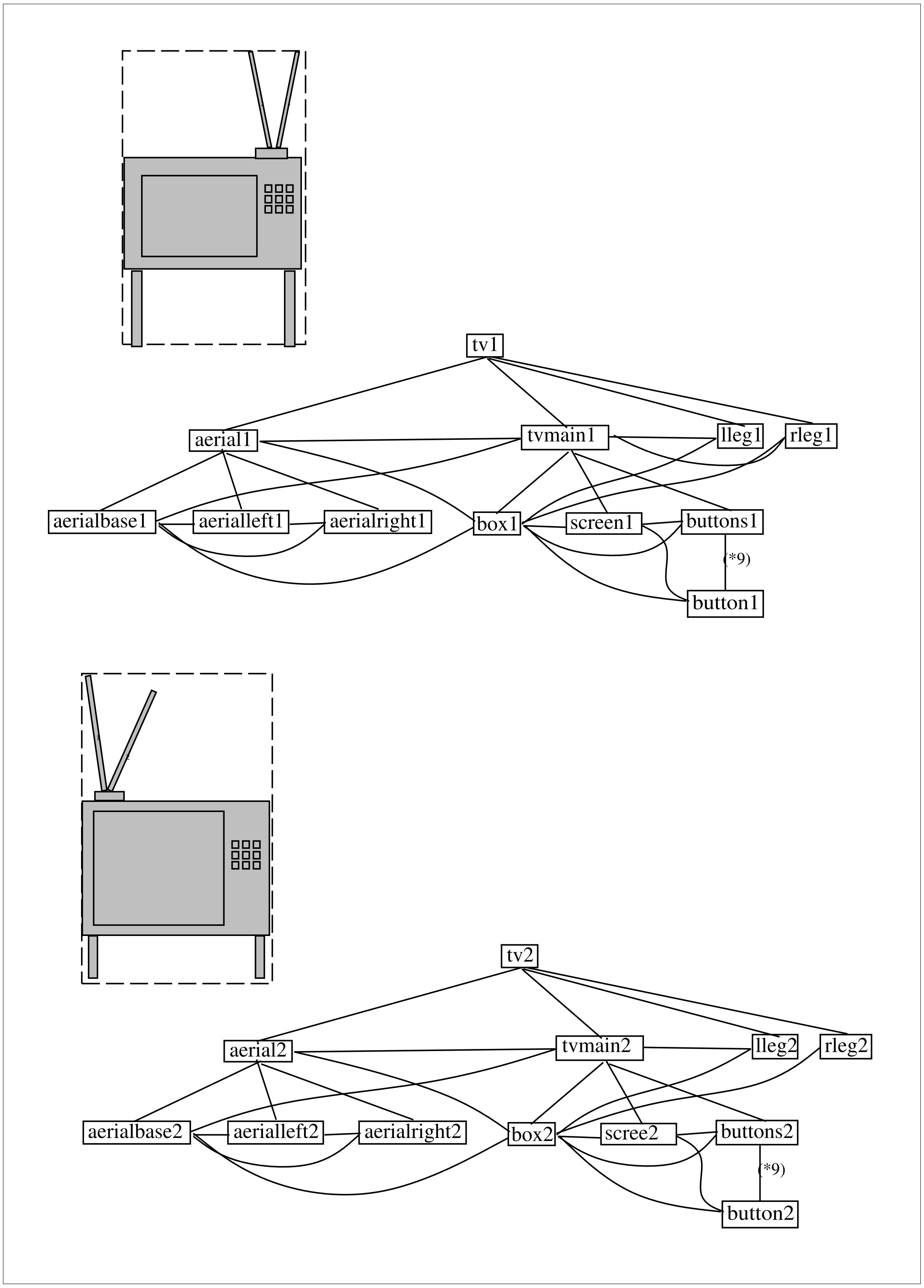


Figure 4.37: Object graphs for two televisions.

2-spread (mainplus1, main2)
    ...
2-spread (aerial1, aerial2)
    1-spread (main1, main2)
    ...
...

4.4.5 Disjunctive structures and contexts are also evaluated using incremental-spread.

This section describes the method by which the matcher compares disjunctively defined concepts, based on the definition of disjunction similarity given in section 4.3.9.

If the matcher is comparing a concept and an instance for which the structure of the concept is described disjunctively, then each disjunct (*i.e.* subconcept) is matched against the structure of the instance, using structure-only scope restriction. The score of the best pairing is used as the structure similarity score (perhaps combined with the winning correspondence scores for any *partial* set of subpart relationships also included in the structure description). If both objects are disjunctively defined concepts, then all pairings of disjuncts are compared, and the winners chosen.

The disjunct pairings are evaluated using whatever *required-spread* is currently being applied to the object comparison, and thus the winning pairing may change as the spread increases.

The method as explained above suggests that *all* disjunct pairings are reevaluated for every increased spread-effort. However, efficiency is improved by employing the same pruning strategy as for evaluating relationship/relatee correspondences: Since each disjunct comparison is an ordinary object comparison, with lower and upper bounds on its similarity score stored in its *object-cnote*, these bounds can be used to determine whether a disjunct pairing is potentially a winning pairing, and if not, it need not be reevaluated. If, later, the score of the winning pairing drops sufficiently, then it may be reevaluated.

The above method (which is based on the definition of disjunction similarity given in section 4.3.9) is also applied to concepts that have disjunctive *contexts*.

4.4.6 Scope Restriction.

A requirement of the matcher is that it should be possible to restrict the scope of the match by indicating which objects in the object graphs are to be compared or ignored. This is most

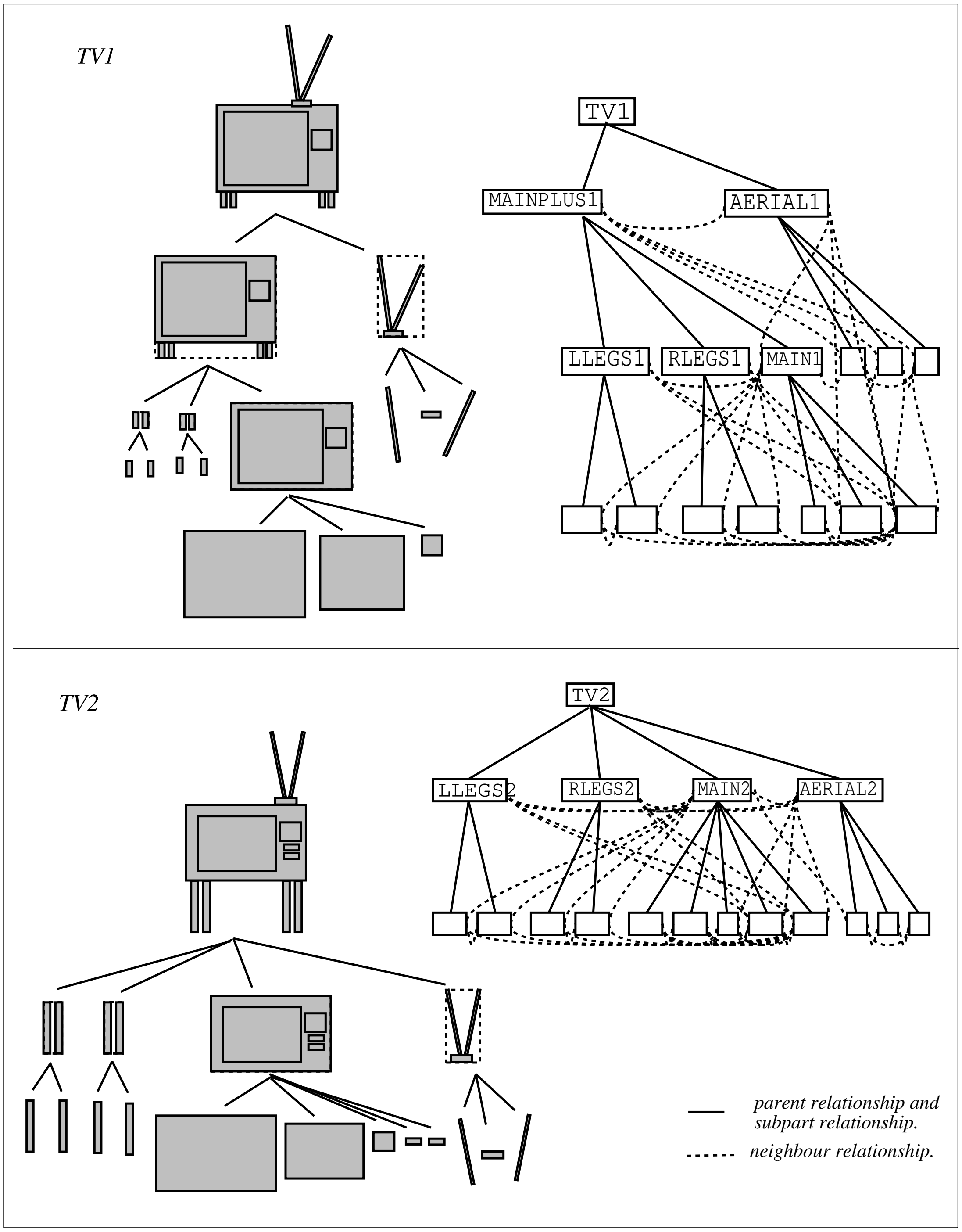


Figure 4.38: The level-hopping problem when matching two televisions.

commonly required for performing a *structure-only* or *context-only* comparison, although other possibilities (such as a partial-context or partial-structure comparison) must also be supported.

Section 4.3.7 discussed how similarity can be measured using such scope-restriction. To measure structure-only similarity (or context-only similarity) the simplest method is to just ignore the context (or structure) description of the two objects being compared. However, this is inaccurate because the subparts (or neighbours) may have relationships to objects that are not within the substructure (or context) of the two objects. Therefore, as discussed in section 4.3.7, it is necessary to identify all of the objects that are "in scope", so that the matcher can ignore all relationships to objects that are out of scope. We now consider how this is done, and some of the problems with this method.

In the case of a structure-only comparison, the matcher traverses all subpart relationships of the two objects being matched, marking each object reached as being *in-scope*. For a context-only comparison, it does the same, except that it marks the objects as being *out-of-scope*, and the search process ignores relationships to objects that are marked as such. In other words, there are two alternative ways of marking and of ignoring objects.

One limitation of this scheme is that it does not account for the structure and context *properties* of the objects being matched. In particular, context properties include *profile* attribute vectors that describe the context of an object in summary form, and therefore, when performing a structure-only comparison of two objects, some of the values within the profiles of their subpart objects should be ignored (or the profiles recomputed). Currently this is not done by GRAM.

Another problem with the method occurs in the case of a structure-only comparison of a concept that has direct or indirect subpart concepts that also define the *context* of the concept. For example, the concept *lecture-room-chair* and its subcomponent concepts might have neighbour relationships to themselves, defining the typical relationships between *lecture-room-chairs*. This is shown by the concept graph at the top of Figure 4.39, which is formed from the row of chairs. Suppose it is to be matched with the *chair1* object using structure-only similarity. The relationships from *lecture-room-chair* to itself can easily be ignored by ignoring the entire context description of the root objects of the match, but this is not possible for the subcomponent concepts (such as *seat* and *seat1*). GRAM does not explicitly distinguish between relationships that are 'typical-inter-member' relationships and ordinary relationships. Therefore, using the scoping method described above, the typical-inter-member relationships will not be ignored by the matcher, since they refer to objects that are *in-scope*, and so the measure of structure similarity for the chair descriptions will be slightly inaccurate.

One solution to the problem is to only mark *instance* objects as in-scope, and perform the structure-only comparison by assuming that concept objects have a *partial* interpretation. This means that the similarity score will not be negatively affected by additional out-of-scope concept relationships (such as typical inter-member relationships) that do not correspond with the in-scope instance relationships. However, this leads to further inaccuracies, since it will not detect *missing* instance relationships.

A better solution would be to extend the representation slightly so that it *does* distinguish between the two kinds of relationships. Each typical-inter-member relationship would have to specify which root concept it is associated with. For example, the neighbour relationship from *rleg* to *lleg* (of the typical 'next' chair in the row) would specify that it is a relationship to a *lleg* of a *different lecture-room-chair*. ([Winston, 1975] seemed to represent typical-member descriptions of a group in a manner similar to this.) Thus, scope-restriction in this case could ignore such relationships since all context of *chair* is to be ignored. This extension to the representation could also help to resolve ambiguities when matching such relationships, and to prevent relationships of different kinds being generalised. Future work on GRAM may address this.

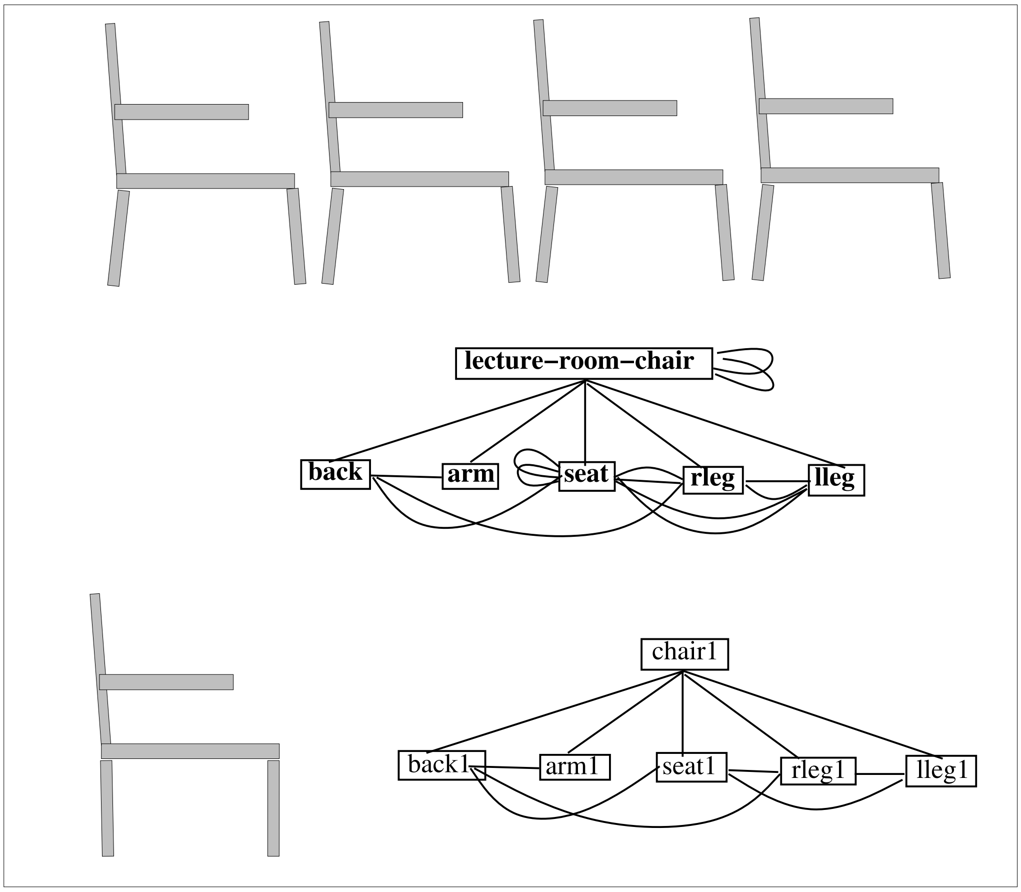


Figure 4.39: A scope-restriction problem.

4.4.7 Augmentation: Dealing with missing relationships and relatees.

Section 4.1.3 stated that the matcher should not assume that two objects being compared are described canonically. In particular, if the objects are similar, but have been partitioned into different decomposition hierarchies, or have descriptions with different relationships made explicit, then the matcher should still be able to determine that the objects are in fact similar.

Section 4.4.4 has addressed this problem in part by showing how correspondences can be found between objects that are on different levels of the decomposition hierarchy. We now consider when and how mismatches due to non-canonicality can be resolved by creating new relationships and relatees to improve the accuracy of a similarity score. Such augmentation could be called *expectation-driven perception*, since it causes GRAM to 'see' new relationships or composite objects that it expects on the basis of an existing concept or another object.

Figure 4.40 illustrates a number such situations requiring augmentation. For example, *A0* has a subpart relationship with *A9*, which is a composite object consisting of *A2* and *A3*. Object *B0* does not include such a composite object, even though *B2* and *B3* clearly correspond with *A2* and *A3*. This situation can be viewed in two ways. On the one hand, *A0* is missing the direct subpart relationships to *A2* and *A3*. On the other hand, *B0* is missing a composite subpart. Thus, improved similarity evaluation requires either a new composite part to be created, or two new subpart relationships to be created, or preferably both.

A similar situation occurs for neighbour relationships. For example, *A1* and *A5* have a direct neighbour relationship, but *B1* and *B5* do not, since *B5* is so small that it is not considered sufficiently 'neighbourly' for an explicit relationship to be included. Thus the matcher needs to create a new neighbour relationship so that the similarity score can be more accurate.

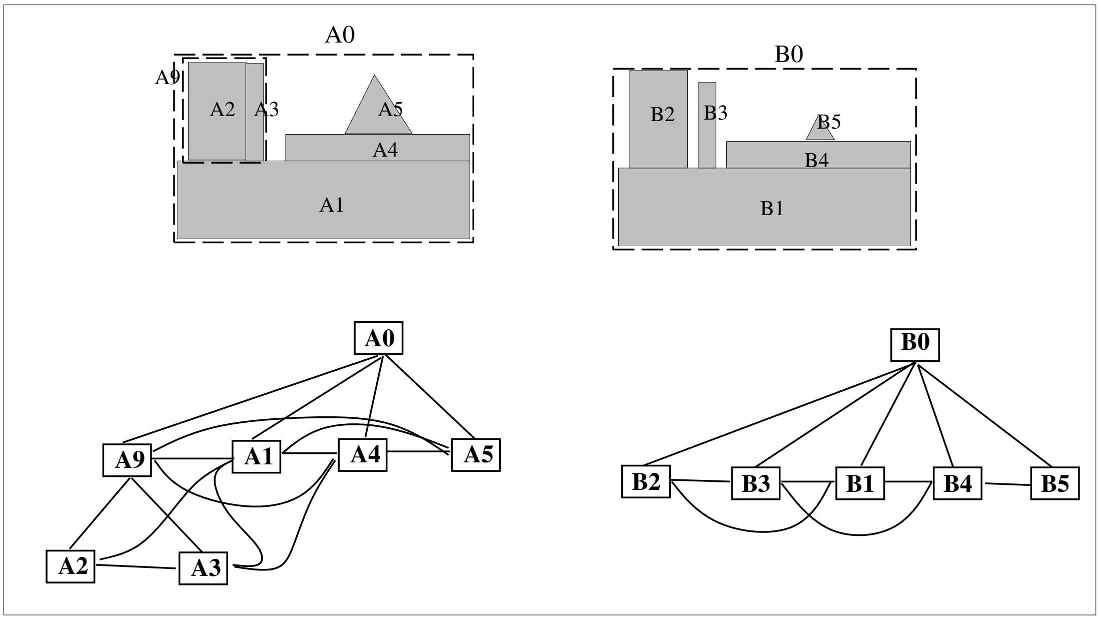
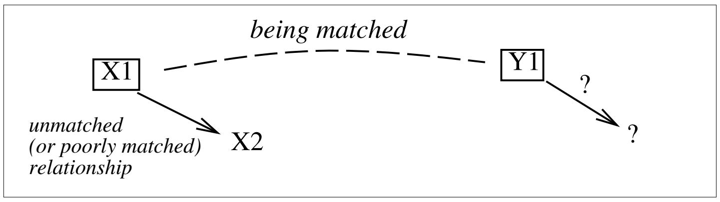


Figure 4.40: Matching may require augmentation.

The main problem to be addressed in this section is how the matcher should decide when and in what way to augment a description with a new relationship or composite objects. More specifically, when matching two objects *X1* and *Y1*, if a relationship of *X1* to a relatee *X2* does not have a good correspondence with any relationship in object *Y1*, how does it know which relatee of *Y1* to create a relationship with in order to establish a correspondence, and when this should be done? Likewise, how does it know which objects could be combined into a composite object?



New relationships can be created on the basis of existing relatee comparisons, or an explicit search.

One solution to relationship augmentation is that if *X1*'s relatee *X2* (in the above example) has been previously matched with some object *Y2*, and with a reasonably high score, then a relationship between *Y1* and *Y2* can be created and compared with the *X1–X2* relationship. An example of this strategy was briefly discussed in section 4.4.4 with regard to the level-hopping problem when matching *tv1* and *tv2*. Other examples will be given later.

Another strategy is to actively search for an object that is implicitly related to *Y1* in the same way that *X2* is explicitly related to *X1*, and which is similar to *Y2*. However this should only be done if *X1* and *Y1* are already known to be quite similar, otherwise every weak comparison could lead to many new relationships being formed. Examples of this strategy will be given shortly.

The creation of new composite objects is a more complex problem for which a solution has not been fully implemented.[2] The decision to create a new object involves noticing that a set of objects in one description matches a set of several objects in the other description, and that only one set has been combined into a single composite object. This signals the matcher to request the creation of a new object in order to establish a better match.

Spurious augmentation should be avoided.

The matcher should not arbitrarily create new relationships or composite parts in the object graphs, in the hope that augmentation might improve similarity, since spurious augmentation can clutter or distort the original description, as well as requiring additional computation. If many additional relationships were added to an object description, then the matcher's incremental-spread algorithm would have to perform considerable more work since it would have to find relationship/relatee correspondences for all of the new relationships, even if they are obscure or 'weak' relationships (unless it employed a strategy of explicitly ignoring 'weak' relationships.)

However, new relationships can be created for purposes of similarity evaluation without actually modifying the object graph. In fact, the current version of GRAM does not change the object graphs at all. Rather, new relationships are stored with the *object-cnotes* as required, and

[2] A basic mechanism for creating new objects has been implemented, but I have not yet implemented the mechanism for incorporating a new object into the current match process, since this requires modifying existing *object-cnotes*. This is not a difficult problem, and will be implemented in future versions.

it is up to the generaliser to decide what to do with them. Thus the incremental-spread search only traverses relationships that were present in the original descriptions, and only uses new relationships to obtain more accurate similarity scores. Augmentations are therefore 'invisible' to the search process, although they are used to evaluate similarities.

Only instance descriptions are augmented.

In addition to the above constraint, GRAM only produces new relationships between instances, not concepts. This is because all instances of an instance graph are defined within a single visual coordinate system, such that relationship details can be computed easily from absolute coordinates, orientations, and dimensions, or can be requested directly from the vision system.

This is not the case for concepts, for which creating new relationships is computationally more complex. Concept relationships can only be computed on the basis of other (perhaps generalised) relationships. Future versions of GRAM may include this capability. For example, suppose the system observes *chair1* next to *filing-cabinet1*, and then compares the chair with the known concept *chair*. *Chair1* has a neighbour relationship to *filing-cabinet* which has no corresponding relationship in the *chair* concept, as shown in Figure 4.41. However, if the *chair* concept has a relationship with the concept *desk*, which has a relationship with the concept *filing-cabinet*, then a generalised relationship between *chair* and *filing-cabinet* could be computed on the basis of those relationships, and then compared with the *chair1–filing-cabinet2* relationship.

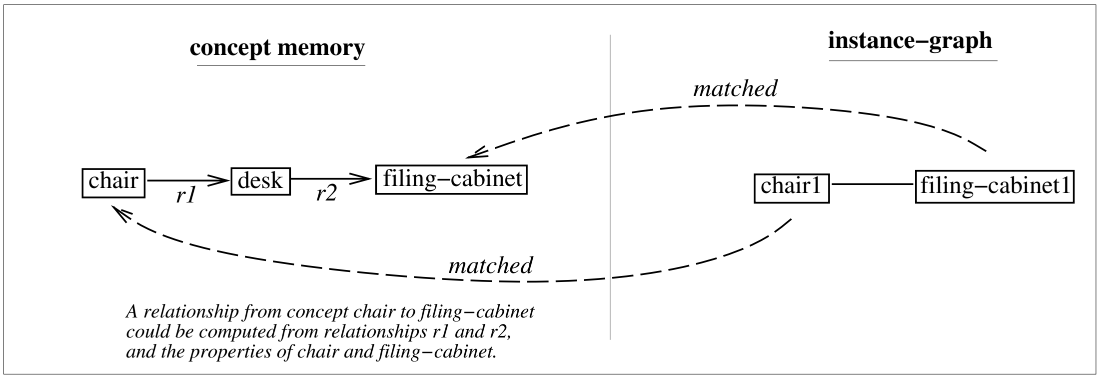


Figure 4.41: Adding a relationship between concepts.

Augmenting to match a parent relationship.

Although the general strategy for relationship augmentation is the same for parent, neighbour, and subpart relationships, the specifics of the process differ for each type. This section considers parent relationship augmentation in more detail.

Figure 4.42 shows two chairs that have different subpart hierarchies and, as a consequence, different relationships. *Chair1* has an additional composite object, *back+leg1*, and *chair2* has an additional composite object, *wholeseat2*.

Suppose the matcher is comparing *seat1* and *seat2*, the former of which has a direct parent relationship to *chair1*, and the latter only has a parent relationship to *wholeseat2*. Based on the existing relationships, the similarity score is weaker than it could be if an additional parent relationship is added which corresponds with the *seat1–chair1* relationship. This requires finding an indirect parent of *seat2* which matches *chair1*.

The first strategy for finding a matching indirect parent is to make use of existing comparisons with *chair1*, by looking at the *object-cnotes* that are associated with it. Each object that *chair1* has already been found to be similar to is checked to see whether it is an indirect parent of *seat2*. If so, then a parent relationship can be created between *seat2* and that object. It is straightforward to determine whether an object is an indirect parent of another object, simply by climbing the path of parent-relationships of the latter object. This method of augmentation is therefore very inexpensive, but does rely on already having matched the direct parent (*chair1*) with the indirect parent (*chair2*).

The second strategy for finding the required indirect parent is to simply compare *chair1* with *all* indirect parents of *seat2*. Since objects usually have only one or perhaps two parents, this strategy is not overly expensive, especially if heuristics are used to constrain the search, perhaps by terminating the climb up the hierarchy if the size of the indirect parent is obviously too large for a good correspondence to be possible.

This second method is only applied if the current similarity score (of *seat1* and *seat2*) is good enough to justify a more thorough comparison using augmentation. More specifically, the score should be better or almost as good as any other competing correspondence (such as the *seat1:wholeseat2* correspondence).

Both of the above situations are illustrated in Figure 4.43, where *chair2* is found either on the basis of existing *object-cnotes* or by an explicit search via parent relationships. The new relationship created is shown as a heavy line.

A different kind of mismatch situation is where there is no corresponding indirect parent. For example, when matching *lleg1* with *lleg2*, the *lleg1*'s parent relationship to *back+leg1* is not matchable with any of the parent relationships of *lleg2*, and there is no indirect parent of *lleg2* which matches *back+leg1*. This mismatch is only resolvable by creating a new composite object consisting of *lleg2* and *back2*.

As stated earlier, GRAM does not yet have the capability to do this fully. It requires noticing that all of the subparts of *back+leg1* (*i.e.* *lleg1* and *back1*) have been matched well with objects of the other chair (*i.e.* *lleg2* and *back2* (assuming that the leg match is sufficiently strong even without the parent relationship correspondence)), and creating a new composite object consisting of those corresponding objects (*lleg2* and *back2*). This also requires the addition of new relationships in the object graph. Furthermore, it requires that the new composite object be incorporated into the *object-cnotes* of all of its related objects, and this will result in the *leg1-leg2* comparison having an improved score.

One more kind of parent mismatch needs to be considered, and that is when a parent relationship of an *instance* object does not have any acceptable correspondence with relationships of a *concept*. If we again assume for the sake of this example that *chair1*, *back1*, *etc* are concepts

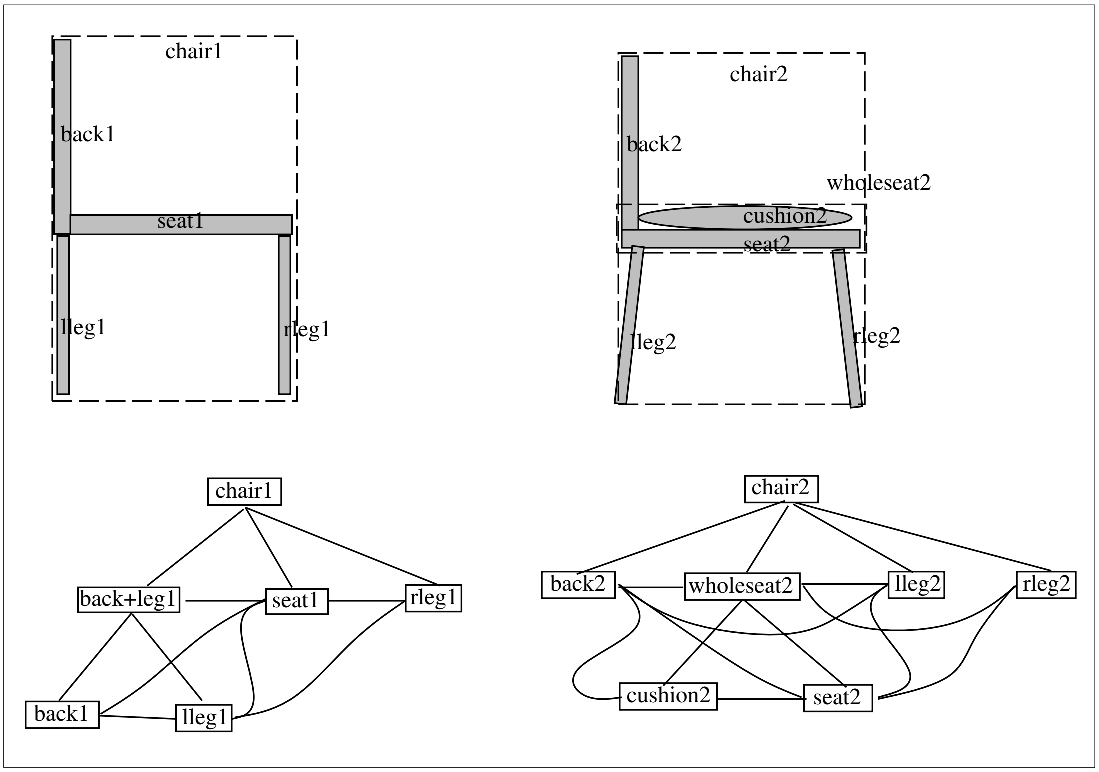


Figure 4.42: Two chairs with unmatched relationships.

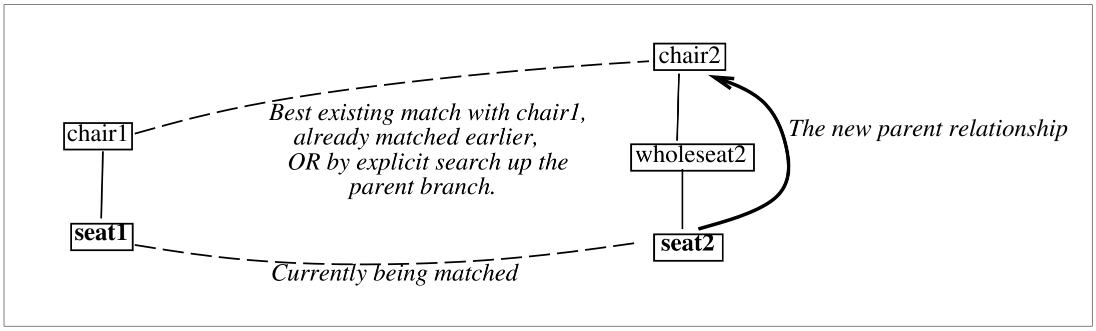


Figure 4.43: Creating a new parent relationship.

rather than instances, then this mismatch situation occurs when comparing *seat1* and *seat2*, for which the *seat2*-*wholeseat2* relationship does not match any of the relationships of *seat1*. Resolving this would involve creating a new concept in concept-memory, consisting of the concepts *seat2* and *cushion2* as subparts, and this is not permitted. It would only be meaningful if these two concepts had neighbour relationships between them, and if so, the properties and relationships of the composite concept would have to be computed on the basis the properties and relationships of the *seat2* and *cushion2*. The current version of GRAM does not support this.

However, it is worth noting that when creating a generalisation of *seat1* and *seat2* (or generalising *seat1* to cover *seat2*), new objects might be added as optional features, so that *wholeseat2* would become an optional parent of the new concept, and this would allow a future observation of a chair containing a ‘wholeseat’ composite object to be matched reasonably successfully, without having to perform augmentation.

Augmenting to match a subpart relationship.

Most of the above discussion is also applicable when there are unmatched *subpart* relationships. This is illustrated in Figure 4.44 where the subpart relationship between *chair1* and *seat1* requires augmenting the *chair2* description with a direct relationship to *seat2*.

The main difference between subpart augmentation and parent augmentation is that GRAM does not perform any explicit search for a corresponding indirect subpart because such a search would require traversing down the entire subpart hierarchy, at least to some level, rather than merely up a single non-splitting (or minimally-splitting) parent branch. For example, when comparing *chair1* and *chair2*, finding a correspondence with the unmatched subpart *lleg2* would require matching it with all subparts of *chair1*.

Therefore, augmentation is only done when the unmatched subpart has already been matched well with an indirect subpart of the other object. It is inexpensive to test whether an object is an indirect subpart of another object, since this only involves a tree traversal without any object or relationship comparisons required.

The example in Figure 4.44 actually has an important difference from the parent-relationship example in Figure 4.43. The difference is that the subpart relationship from *chair1* to *seat1* *does* have a well-matching correspondence with a subpart relationship of *chair2*, namely with *wholeseat2*. Therefore it would seem that augmentation would not be performed. However, GRAM notices that *seat1* matches better or almost as well with *seat2* than with *wholeseat2*, and therefore will still create a new relationship in order to try improving the similarity score. In other words, it creates new relationships to *any* objects that have already been matched better (or almost as well) with any of the subparts of *chair1*.

One characteristic of GRAM is that augmentation can either work from the bottom up, as when creating new parent relationships, or top-down as when creating new subpart relationships. It may also work ‘sideways’ via neighbour relationships, as discussed in the next section. The behaviour of the incremental-spread process determines which augmentations are done, and when. If two objects low in the part hierarchy are processed for augmentation before the higher objects, but after the higher objects have at least been partially matched, then parent relationships will be created first. Conversely, if the higher objects are processed for augmentation first, but after the lower objects have been matched sufficiently to justify a correspondence, then new subpart relationships are created first.

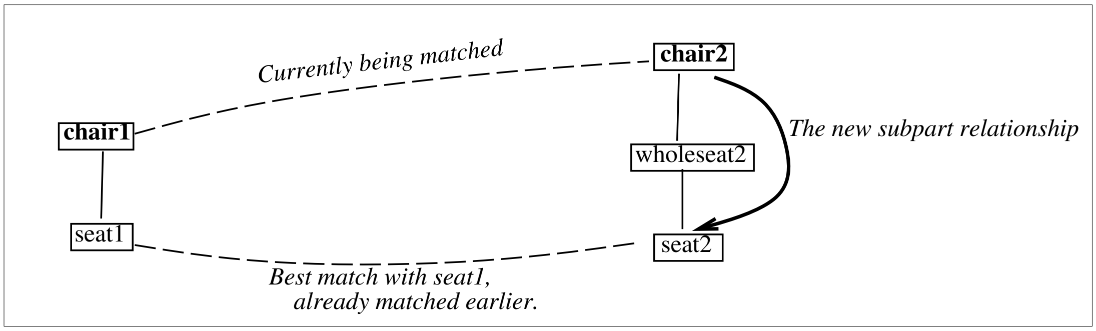


Figure 4.44: Creating a new subpart relationship.

Augmenting to match a neighbour relationship.

An object can be augmented with neighbour relationships in much the same manner as for subpart relationships. For example, in Figure 4.42 *lleg1* has an explicit neighbour relationship with *rleg1*, but *lleg2* does not have a neighbour relationship with *rleg2* (perhaps because they are not parallel). When matching *lleg1* and *lleg2*, if we assume that *rleg1* has already been matched (at least partially) with *rleg2*, then this justifies creating a relationship between *lleg2* and *rleg2* to improve the accuracy of the similarity score, as shown in Figure 4.45. However, this is only done after making sure that *rleg2* is not a subpart (direct or indirect) of *lleg2*, or vice versa, since this would indicate that a neighbour relationship between them is not meaningful.

Neighbour relationship augmentation is only done on the basis of already-matched indirect neighbours, and not by doing an explicit search, since too many comparisons would be required (unless complex search-pruning heuristics were applied). Also, in most situations, the incremental-spread process tends to find all reasonable correspondences anyway. In fact, each time more spread effort is applied to a comparison, more correspondences with unmatched relatees will become available to justify augmentation where it is necessary.

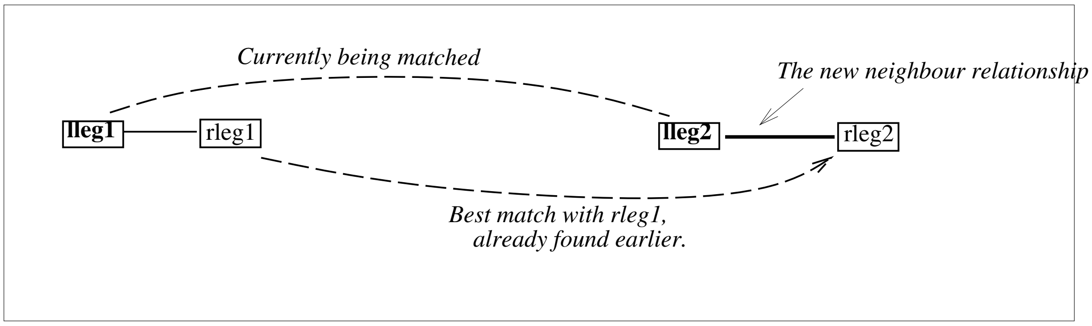


Figure 4.45: Creating a new neighbour relationship.

4.4.8 Using the AKO hierarchy.

Section 4.3.10 discussed how an estimate (or rather, a lower or upper bound) of the similarity of two objects can be defined in terms of previously computed similarity scores for their superconcepts or subconcepts. This section looks at a particular situation for which this can be applied.

Suppose the system has observed the four chairs at the top of Figure 4.46 (*i.e.* *chair1*, *chair4*, *chair4*, and *chair4*), only the last of which has an arm. To form a generalised concept (*chair*) from these, one of the techniques used by GRAM's generaliser when dealing with a new unmatched object (such as *arm4*) is to create a relationship from the generalised object to the unmatched and ungeneralised instance object (such as from *chair* to *arm4*), and *vice versa*, as shown in the figure. This avoids the need to create a new copy of the unmatched instance object (*arm4*) to be a relatee of the generalised concepts. (This process is discussed in chapter 5). Although this strategy is convenient for the generaliser, and requires less memory for representing the concept, it creates difficulties for the matcher:

Suppose the system has to compare the concept *chair* with a newly observed object *chair5*, shown at the bottom of the figure. Two possible situations are considered below, each of which illustrates a different way in which the AKO hierarchy can be used.

Firstly, suppose the matcher compares *chair* with *chair5* using a 1-spread and then a 2-spread. This will involve matching *arm4* with *arm5*, *back* and *back5*, and other pairs of their subparts. The problem is that to match *arm4* with *arm5*, the matcher will spread throughout the graph of *chair4* subcomponents, comparing them with *chair5* components, while simultaneously spreading throughout the generalised *chair* subcomponent concepts. It is comparing *chair5* with the entire *chair4* instance as well as with the generalised *chair*, all for the purpose of evaluating the *arm4*-*arm5* similarity.

In some sense this is appropriate, since it could be the case that arms can only be on chairs that are identical to *chair4* (with its particular back and leg lengths), and thus the matcher should only give a perfect score to the *arm4*-*arm5* comparison if the context of *arm5* perfectly matched the context of *arm4*. This context similarity score would not contribute significantly to the *chair*–*chair5* match, since the direct subpart correspondences contribute much more strongly, and this is desirable because we want the *chair* concept to capture a transfer of information amongst the instances, otherwise the *chair* concept may as well be represented as a disjunction of four instances. However, the problem of the search effort being doubled as a consequence of the *chair*-*arm4* relationship still remains.

A solution to this is to make use of superconcept and subconcept comparisons to obtain similarity estimates in the manner described in section 4.3.10. For example, the steps below shows how *chair* could be matched with *chair5*. The 2-spread comparison initially computes scores for a variety of relationship/relatee correspondences, including *arm4*:*arm5*. The 3-spread then produces a more accurate score for the *back*:*back5* comparison. Later, the *arm4*:*arm5* is evaluated more thoroughly, and this requires a 1-spread comparison of *back4* and *back5*. However, since *back4* is a subconcept of *back* (assuming this AKO link is created when the *chair* concept is formed), the matcher should be able to produce a more accurate similarity

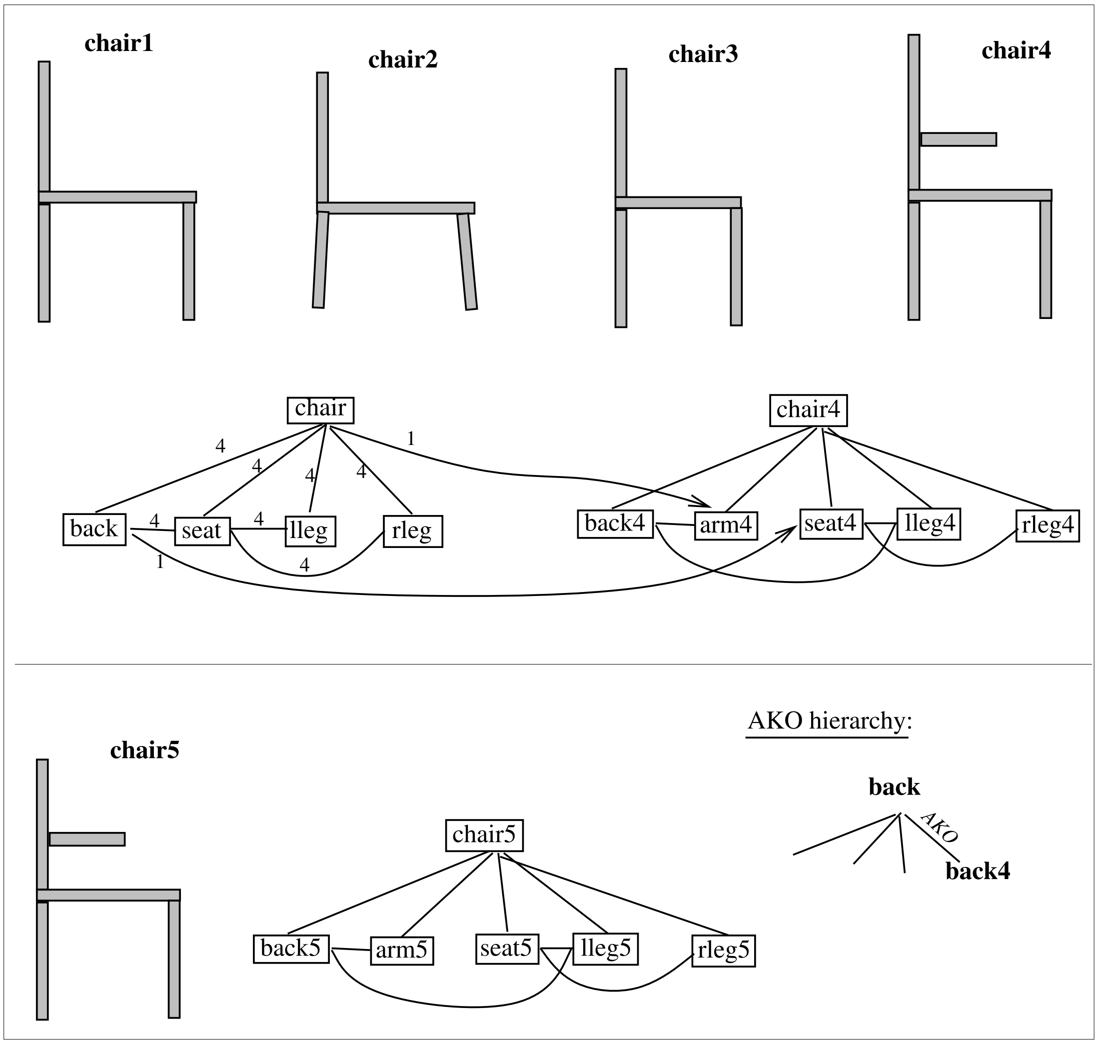


Figure 4.46: Using subconcept or superconcept similarity scores.

score on the basis of the previous *back*:*back5* comparison. This may enable the matcher to immediately reject other otherwise potentially-winning correspondences, such as *back4:lleg5*. In other words, more accurate scores are obtained sooner than would be possible without making use of the subconcept to superconcept similarity scores.

```
1-spread ( chair,  chair5)
2-spread ( chair,  chair5)
  1-spread ( back4,  back5)
  1-spread ( back4,  lleg5)
  ...
  1-spread ( arm4,  arm5)
3-spread ( chair,  chair5)
  2-spread ( back,  back5)
    ...
  2-spread ( arm4,  arm5)
    1-spread ( back4,  back5)      (should make use of (back, back5) score.)
    1-spread ( back4,  lleg5)      (should be immediately rejectable.)
  ...
...
```

The inverse process could also occur if the search was performed in a different order: The *back:back5* comparison could make use of the score of the *back4:back5* score, since back4 is a subconcept of back.

The process discussed in this section is interesting from a cognitive science point of view because it still allows an instance to be matched simultaneously with general and specific descriptions in concept-memory. This seems (from personal introspection) to be the kind of process that humans perform: when we observe some object, we not only recognise its generic category, but we also simultaneously notice correspondences between more specialised concepts or particular previously-observed instances.

This thesis has not significantly explored the implications of this aspect of the matching strategy since it requires a more elaborate system for creating and maintaining AKO hierarchies. Currently GRAM only creates concept hierarchies in a very simplistic manner.

4.4.9 Fit-scores are obtained by traversing winning correspondences in the cnote graph.

During the match search, the only similarity scores that are computed are proximity-scores, since these are required for finding correspondences. However, once the match has completed, the generaliser needs fit-scores for determining whether existing concepts can be modified to cover a new instance, or whether a new concept should be created. To obtain a fit-score for a cnote, the matcher first computes a fit-score based on property similarity and winning relationship/relatee correspondences scores. This score is stored in the cnote to prevent cycles. Then the fit-scores for pairs of relatees (for the winning relationship/relatee correspondences) are obtained by recursively invoking the fit-scoring mechanism. These scores can then be used to compute a more accurate (and recursively defined) fit-score. If the objects are defined disjunctively, then this must also be taken into account.

This is similar to the way the matcher obtains proximity-scores, except that it is simpler, and most importantly it only evaluates scores for the already-found winning relationship/relatee correspondences, and so the process is just an inexpensive traversal of the cnote graph.

4.4.10 Details of the algorithm.

The previous sections have discussed the basic match algorithm and various specific components of it. This section now presents the algorithm as whole, incorporating all of the details that have been described.

Figure 4.47 shows the main steps of the **MATCH** procedure. Its arguments include the two objects to be matched (which are normally a concept and an instance, but could be two concepts or two instances), and several control parameters. The first control parameter, *required-spread*, indicates the spread-effort to be applied. The *scope* and *scope-mark* parameters specify what kind of scope-restriction (if any) is required.

The *rejection-cutoff* parameter indicates that if the upperbound score of the comparison drops below this level, then no further evaluation should be performed. The *acceptability-cutoff* parameter indicates that if the lowerbound score of the comparison rises above this level, then the correspondence can be considered acceptable, and no further evaluation needs to be performed.

The algorithm begins with a couple of preliminary tests to decide whether a match is in fact necessary:

First it tests whether *object1* and *object2* are the same object. If they are, then they obviously match perfectly, and so a 'dummy' *object-cnote* with a score of 1 is returned. This may occur if we are matching two components within the same scene, both of which have relationships to same object.

Second, the matcher tests whether *object2* is a subconcept (directly or indirectly) of *object1*, and if so, it returns a score of 1. This is because GRAM currently considers all subconcepts as having a perfect match with their superconcepts. A more accurate score would be based on the *typicality* of *object2* within *object1*.

Third, the matcher tests whether *object1* and *object2* have already been matched. If so, there is no need to create a new *object-cnote*. However, the recorded *object-cnote* might not have been evaluated to the level of *required-spread*, in which case further matching using greater spread-effort must be performed.

If an *object-cnote* does *not* already exist, then one is created by **CREATE-CNOTE**, which is shown in Figure 4.49. This compares the two objects using a 1-spread effort, by comparing their properties and *in-scope* relationships (for each possible axis correspondence). Relationships may need to be coerced to account for the axis correspondence before comparing the attributes. The procedure does not compare relatees, although it makes use of scores for any pairs of relatees that have already been compared. If the objects are defined disjunctively, then pairs of disjunct subconcepts are matched using a structure-only or context-only 1-spread comparison, as discussed in section 4.4.5. Upper and lower bounds on similarity scores are also computed.

Two more tests are then applied. The first test considers whether *object2* has already been matched with a direct or indirect superconcept of *object1* (assuming *object1* is a concept). If so, the recorded similarity score, combined with the typicality of *object1* within the superconcept, can be used as an upperbound for the *object1*:*object2* comparison, as discussed in section 4.3.10. However, it only changes the upperbound of the *object-cnote* if it is lower than the current upperbound. The second test checks whether *object2* has already been matched with a direct or indirect *subconcept* of *object1*, and if so, the recorded similarity score can be used as a lowerbound for the *object1*:*object2* comparison, if it is higher than the current lowerbound.

The main loop of the algorithm then begins, which incrementally applies more effort to comparison, by applying SPREAD-CNOTE, and abandons the loop when the required spread has been completed, or when the scores satisfy the rejection-cutoff or acceptability-cutoff requirements.

The **SPREAD-CNOTE** procedure, shown in Figure 4.48, reevaluates parent, neighbour, and subpart relationship/relatee correspondences, recomputes similarity scores, and augments descriptions if necessary.

First it attempts to augment the instance with new relationships if necessary and if possible, as follows: For each in-scope relationship of each object, it considers each of the existing *object-cnotes* of the relatee. If that relatee is matched sufficiently well with an *indirect* relatee of the other object, then a relationship between the other object and that relatee can be created and matched, since it might be a potentially winning relationship/relatee correspondence. This process was discussed in section 4.4.7. Relationships may also be created on the basis of an explicit search for matching relatees, but this is only done for parent relationships.

The next step is to reevaluate each potentially winning relationship/relatee correspondence using the required spread (in *new-spread*). This is done by recursively invoking the MATCH procedure to compare the relatees. This may immediately return if the relatees have already been matched to the required spread level.

If the structure (or context) of one of both objects is defined disjunctively, then the pairings of disjunct correspondences (*i.e.* subconcept correspondences) are reevaluated using the required spread and using structure-only or context-only scope.

When all potentially winning relationship/relatee correspondences and disjunct correspondences for the structure and/or context of the two objects have been reevaluated, new structure and/or context similarity scores are computed, based on the set of winning correspondences. An overall similarity score is computed from these. The *interpretation* of the structures or contexts must be taken into account when evaluating these scores, as explained in section 4.3.9.

If the structure and/or context of one or both objects is disjunctive, then disjunct pairings are also evaluated by applying CREATE-CNOTE to each pairing.

```
MATCH ( object1,            {the concept (or perhaps an instance)}
        object2,            {the instance  (or perhaps a concept)}
        required-spread,    {the spread-effort to be applied}
        scope               { complete, structure-only, context-only, as-marked}
        scope-mark,         {don't process objects with/without this mark}
        rejection-cutoff    {immediately abandon the match if the
                            upperbound score drops below this. Default=-99}
        acceptability-cutoff  {immediately quit the match if the
                            lowerbound score rises above this. Default=+99}
  )

 IF  object1 and  object2 are the same object THEN
   RETURN a cnote with score of 1.

 IF  object1 is a superconcept of  object2 (direct or indirect) THEN
   RETURN a cnote with score of 1.

 IF  object1 and  object2 have already been matched THEN
    cnote ← the recorded cnote.
 ELSE
    cnote ← CREATE-CNOTE ( object1,  object2, scope-mark)

 IF  object2 has already been matched with a superconcept of  object1 THEN
     upperbound ← fn ( similarity score (of superconcept and  object2),
                       typicality (of  object1 within superconcept) )
     cnote-upperbound ← min(cnote-upperbound, upperbound)

 IF  object2 has already been matched with a subconcept of  object1 THEN
     lowerbound ← fn ( similarity score (of subconcept and  object2),
                       typicality (of subconcept within  object1))
     cnote-lowerbound ← max(cnote-lowerbound, lowerbound)

 WHILE  (cnote-upperbound > rejection-cutoff)
   and (cnote-lowerbound < acceptability-cutoff)
   and (cnote-spread < required-spread)

      SPREAD-CNOTE (cnote,  (cnote-spread + 1),  scope, scope-mark
                    rejection-cutoff, acceptability-cutoff)

 RETURN the cnote.
```

Figure 4.47: The MATCH Algorithm

```
SPREAD-CNOTE (cnote, new-spread, scope, scope-mark,
                rejection-cutoff, acceptability-cutoff)

 IF scope is not structure-only:                { reevaluate context similarity }
   FOR each in-scope parent relationship of object1:

          { create new relationships if necessary, based on existing comparisons. }
     IF object2 is an instance {i.e. is augmentable} THEN
       FOR all cnotes involving parent1:
         IF parent2 (of the cnote) is an indirect parent of object2:
           IF potentially a winning correspondence
             Create object2– parent2 relationship (but don't add to the description)
             Compute correspondence-score.
             IF potentially a winning correspondence THEN
               Add to the list of potential winning correspondence-cnotes.

          { create new relationships if necessary, based on explicit search. }
     IF the object1– parent1 relationship has no good correspondence THEN
       Explicitly search up parent relationships of object2.
       If a good parent1: parent2 match is found THEN
         Create object2– parent2 relationship.
         Compute a correspondence-score.
         IF potentially a winning correspondence THEN
           Add to the list of potential winning correspondence-cnotes.

          { reevaluate relatee comparisons using increased spread. }
     For each potentially-winning correspondence-cnotes for object1– parent1:
       MATCH (parent1, parent2, (new-spread - 1), scope, scope-mark)
       Recompute the correspondence-cnote score.

   Repeat the above for in-scope parent relationships of object2
   Repeat the above for in-scope neighbour relationships of object1 and object2
       (but without the explicit search)

   IF context1 and/or context2 is defined disjunctively THEN:
     For each potentially-best disjunct pairing (ob1,ob2):
       Mark context using a new scope-mark.
       MATCH (ob1, ob2, new-spread, context-only, new-scope-mark,
             lowerbound of best disjunct pairing (as rejection-cutoff),
             1 (as acceptability-cutoff))

   Compute a new context-score.

 IF scope is not context-only                { reevaluate structure similarity }

   Repeat the above for in-scope subpart relationships of object1 and object2.
   IF structure1 and/or structure2 is defined disjunctively THEN:
     Reevaluate disjunct pairs, as for context disjunction.
   Compute a new structure-score.

 Compute a new overall score for the cnote.
 cnote-spread ← required-spread
```

Figure 4.48: SPREAD-CNOTE

```
CREATE-CNOTE ( object1, object2, scope, scope-mark)

  Match the contexts
  If scope is not structure-only

    Compute similarity of context properties.

    FOR each in-scope parent relationship of object1:
      FOR each in-scope parent relationship of object2:
        FOR each possible axis-correspondence:
          relationship-score ← similarity of the relationships
                (coerce relationship2 if necessary)
          Create a correspondence-cnote and compute similarity score.
            (if relatees have been matched, use the similarity score)

    Repeat the above for in-scope neighbour relationships.

    IF one or both contexts are disjunctive:
      Invoke CREATE-CNOTE for each subconcept pairing.

    Compute the context-similarity score.
       (using winning relationship/relatee correspondences)

  Match the structures

  If scope is not context-only

    Compute similarity of structure properties.

    Repeat the above in-scope subpart relationships.

    IF one or both structures are disjunctive:
      Invoke CREATE-CNOTE for each subconcept pairing.

    Compute the structure-similarity score.
      (using winning relationship/relatee correspondences)

  Compute the overall object similarity score.

  Store the results in a new object-cnote.
```

Figure 4.49: CREATE-CNOTE

Chapter 5

The Generaliser

This chapter addresses the problem of producing a generalisation from two concept or instance descriptions, given a description of their comparison produced by the matcher. Various issues are explored, and GRAM's generaliser is described.

As discussed in chapter 1, the generaliser is to be part of a larger concept learning system which updates concept-memory in response to observed objects. The learning system is responsible for deciding which existing concepts are to be modified, or new concepts to be created, and for performing appropriate generalisations by invoking the generaliser. The learning system is also responsible for maintaining and reorganising concept-memory to ensure optimal classification and learning performance. This may involve removing concepts that are no longer useful; merging two or more concepts; splitting concepts into several subconcepts; removing spurious low-frequency features from concept descriptions; and various other maintenance and optimisation operations.

In a domain of complex structured objects, concept learning is by no means trivial, and has not been significantly addressed in machine-learning research. Systems such as MERGE and Labyrinth have addressed the problem to some degree, but the effectiveness of these systems is limited by the limitations in their representation schemes, matching strategies, and generalisation mechanisms, as discussed in chapter 2. Some of these limitations have been addressed in the previous two chapters, and others (relating to generalisation) are discussed in this chapter. The methods employed by GRAM's generaliser are expected to provide a good foundation for future development of a full learning system.

Organisation of this chapter.

Section 5.1 begins this chapter by defining the input and output requirements of the generaliser. Section 5.2 then outlines the issues that have been addressed and the contributions of the generaliser.

Section 5.3 presents an overview of the generalisation algorithm. The remaining sections discuss particular components of the algorithm in more detail.

Section 5.4 explains how properties and relationships are generalised, or more specifically, how attribute values are generalised, since all properties and relationships are represented by attribute vectors.

Section 5.5 considers how the scope, or focus of attention, of the generaliser can be restricted, if necessary. It also discusses how the generaliser decides whether to generalise a description (of a concept or instance), and if so, whether the original description should be modified, or a new concept description created.

Section 5.6 explains how unmatched relationship/relatees of a concept or instance are dealt with by the generaliser.

Section 5.7 explores the problem of partial similarity and disjunct formation. More specifically, it considers when and how context or structure disjuncts are formed if two objects are similar in structure but not context, or *vice versa*.

Section 5.8 addresses the problem of ambiguities. It describes several different kinds of ambiguity and the mechanisms that can be used to resolve them.

Finally, section 5.9 explains how the *interpretation* of a structure or context description affects the way generalisation is performed.

5.1 Input and Output.

The input to the generaliser is a single *object-cnote* produced by the matcher, and the output is a generalisation of the two objects in the cnote. This generalisation might either be a new concept or one of the original objects, modified.

The input cnote specifies various similarity scores and the winning (and any marginally losing) parent, neighbour, and subpart relationship/relatee correspondences. Figure 5.1 shows an example of a cnote for the comparison between *tvmain1* and *tvmain2*. The dotted lines between the relationship/relatees indicate the winning correspondences. In this example there are no marginally losing correspondences.

5.1.1 The input is explicitly a single cnote, but implicitly an entire cnote graph.

Since each relationship/relatee correspondence in a cnote refers to another cnote that describes the comparison of the two relatees, the input is, in effect, an entire cnote graph, rather than a single cnote. The cnote graph for the *tvmain* example is given in Figure 5.2 in which each node is a cnote, and each edge denotes a parent, neighbour, or subpart relationship/relatee correspondence.

5.1.2 A "side effect" of a generalisation is many other generalisations.

The inter-dependency between cnotes means that a generalisation of two objects requires generalisations of other pairs of objects. For example, to produce a generalisation of *tvmain1* and *tvmain2*, the generaliser would have to obtain a generalisation of *tv1* and *tv2*, of *lleg1* and *lleg2*, and of the other pairs of corresponding parent, neighbour, and subpart relatees. Therefore, although the output of the generaliser is a single cnote, it also produces other generalisations as a "side effect". Thus GRAM's generaliser is required to deal with multiple concept learning, although not to the extent that is required of a full concept learning system, since the generaliser focuses its attention on a particular correspondence, while the full learning system must deal with concepts and instances at a more global level.

5.1.3 Scope restriction parameters are required.

In addition to the input cnote, the generaliser also requires scope-restriction parameters that specify how much of the cnote graph surrounding the input cnote should be generalised. For example, the matcher may have been applied with a large spread, perhaps for the task of classifying a complete scene and all of its components, but the generaliser may be required to focus its attention on, say, the substructure and a small amount of the context of the two objects of the input cnote. Conversely, the matcher may have been applied with a low spread, and the generaliser is required to generalise with a higher spread, in which case it will have to re-invoke the matcher on some or all of the cnotes in the cnote graph.

5.1.4 Parameters for determining generalisability and modifiability are needed.

An input parameter is also required that specifies the minimum similarity needed to justify producing a generalisation of two objects. Likewise, another input parameter is needed to specify the minimum similarity needed to justify modifying one of the original objects to cover the other object, as opposed to creating a new concept. These parameters must be able to be passed to the generaliser, rather than being predefined global parameters, since different learning tasks will require different behaviour of the generaliser.

5.1.5 The input objects may be concepts or instances.

In most cases the generaliser will be required to generalise a concept to cover an instance, or to create a new concept from two instances. However, it must also be able to produce a generalisation from two generalised concepts. This is required when generalising two multi-relationships (either of concepts or instances) since the relatee of a multi-relationship is always a generalised concept. Concept–concept generalisation would also be required by the larger learning system when reorganising concept-memory.

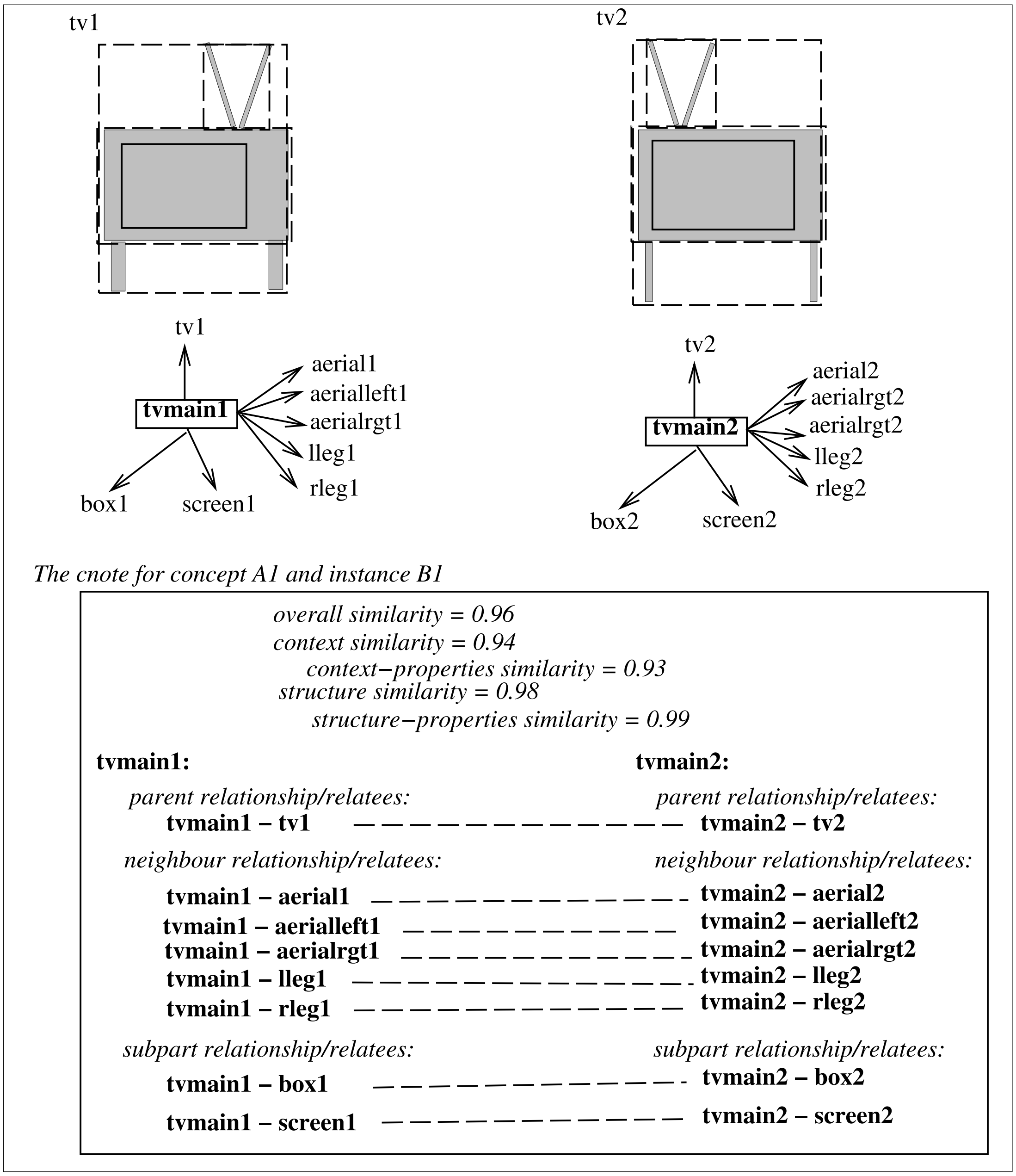


Figure 5.1: A cnote graph.

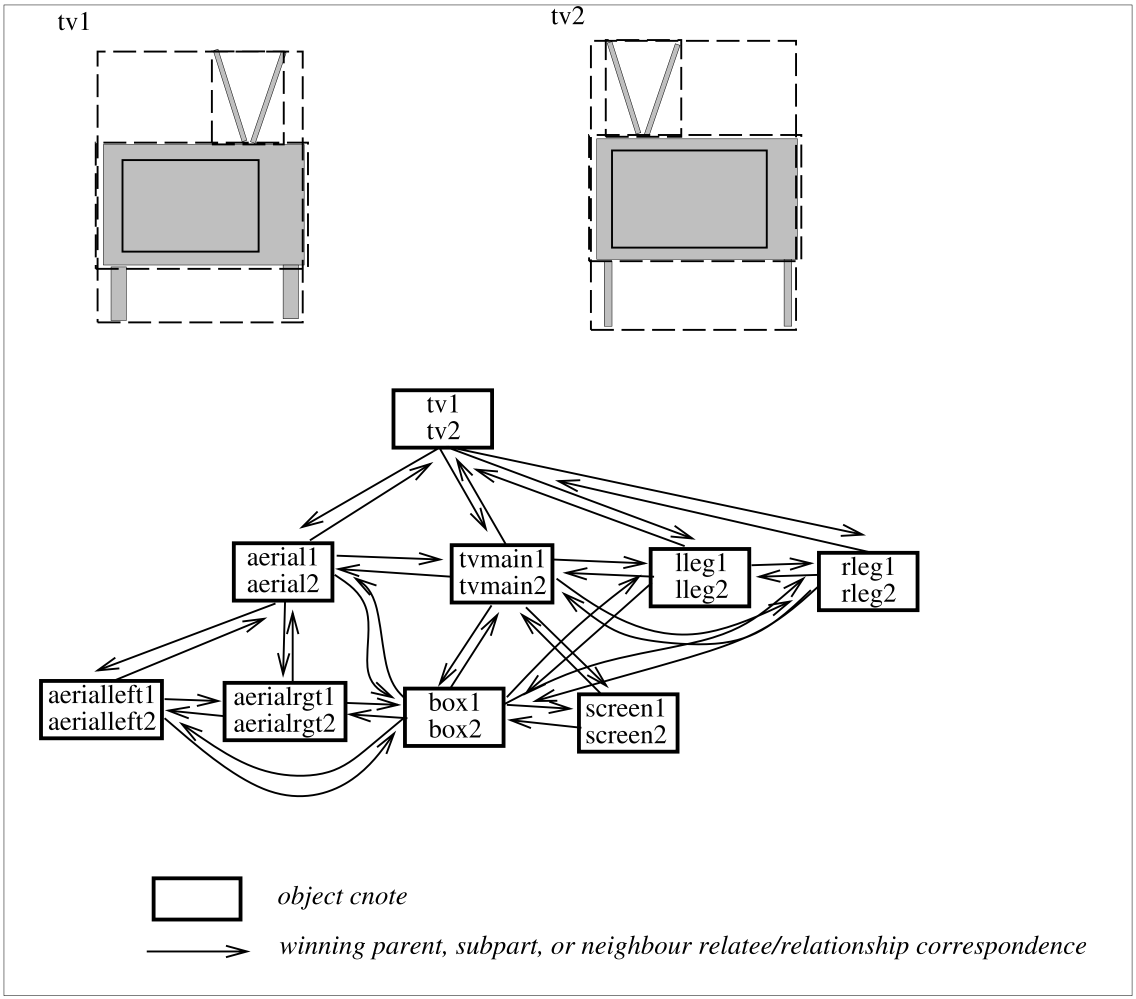


Figure 5.2: A cnote graph.

5.2 Issues and Contributions.

This section discusses the various issues of generalisation that are addressed in this chapter, and outlines the main contributions of GRAM's generaliser.

5.2.1 Over-generalisation and under-generalisation should be avoided.

The problem of over-generalisation is common to most machine learning systems. The problem is accentuated in a structured domain such as GRAM because each concept is defined in terms of other concepts, and therefore over-generalising one concept results in over-generalising all concepts that are defined in terms of it. Even if the teacher is specifically requesting a generalisation, the problem still remains, since the teacher cannot specify the generalisability of every component that the two objects are defined in terms of.

Apparently over-generalisation was one of the problems encountered in the Labyrinth system [personal communication]. One reason for this could be that it does not include *context* information in its concepts, since instances that should be kept distinct on the basis of their context (such as a *chair-leg* and a *coffee-table-leg*) would be generalised. GRAM's use of context helps to reduce this cause of over-generalisation. In a full learning system, functional knowledge, reasoning abilities, and task requirements would reduce the problem even more significantly.

Under-generalisation, by creating new concepts rather than generalising existing ones, is also a problem, since it can lead to a large and cluttered concept-memory.

In GRAM, these problems are addressed to some degree by the use of the two parameters discussed in section 5.1.4 for determining generalisability and modifiability. This is discussed in section 5.5.3, which explains how GRAM bases generalisability on the *proximity-score* of a comparison, and bases modifiability on the *fit-score* of a comparison, since the former measures absolute similarity, while the latter measures how well the instance fits a concept.

5.2.2 A relationship/relatee may be unmatched.

In the example of the two televisions in Figures 5.1, each relationship/relatee has a clear unambiguous high-scoring winning correspondence with a relationship/relatee of the object object. However, if the aerial of *tv2* had a base (*aerialbase2*, say), then the neighbour relationship from *tvmain2* to *aerialbase2* would not have any high-scoring correspondence with a relationship of *tvmain1*. *Aerialbase* can therefore be considered unmatched, hence the generaliser must be able to deal with unmatched relationship/relatees appropriately.

Since there is only one example of an *aerialbase*, it is not clear how to generalise it. In particular, it is not clear whether the generalised *tv* should refer to a new copy of *aerialbase2* which is altered to relate to other new generalisations of the *tv1* and *tv2* components, or whether *tv* should refer to the original *aerialbase2* which is defined by relationships to other components of *tvmain2*. This issue is discussed in section 5.6. Currently GRAM takes the former choice, because it prevents the matcher from having to spread through the part-graphs of instances and subconcepts from which a concept that was formed. is simpler and leads to fewer spurious concepts being created, although it has other consequences for the matcher, since the new concept becomes partially defined in terms of a specific instance.

5.2.3 Partial similarity may require disjunct formation.

Two objects may be considered generalisable by GRAM on the basis of a particular kind of partial similarity, where either the structures of the two objects are very similar, or the contexts are very similar, but not both. For example, two telephones, one on the wall and one on a desk, would be justifiably generalised on the basis of structure similarity alone. There are many classes of objects whose instances have a relatively invariant structure and a variant context, which may be initially formed from two instances that are partially similar in terms of structure alone. Likewise, two chair-backs may have very similar context, but dissimilar substructure, and this partial similarity should also be sufficient to justify generalisation.

This requirement leads to the problem of creating a concept whose structure or context is defined disjunctively, rather than merging the relationship/relatees of the contributing objects into a single non-disjunctive description. The issues and techniques used by GRAM to achieve this are discussed in section 5.7.

GRAM is able to perform this limited but useful kind of disjunction formation as a consequence of distinguishing structure and context in its representation scheme. Without this distinction, it would have to deal with the very difficult problem of creating arbitrary disjunctions of properties and relationships, which can lead to undesirable under-generalisation. This is presumably why the other learning systems discussed in chapter 2 avoided disjunction altogether.

5.2.4 Several kinds of ambiguity must be resolved.

Sometimes one or more relationship/relatees of an object may match ambiguously with relationship/relatees of another object, in which case the cnote will include winning and marginally-losing correspondences. For example, if one desk has one telephone on it, and the other desk has two telephones on it (side by side), then the comparison of the desks will include two ambiguous correspondences between the desk–telephone relationships. Dealing with such ambiguity is one of the most difficult issues of generalisation. None of the other systems discussed in this thesis (in chapter 2) dealt with this problem.

In developing ambiguity-resolution mechanisms for GRAM, as discussed in section 5.8, it has been necessary to distinguish between two types of ambiguity. The first is called *similar-similarity*, meaning that each pair of corresponding relationship/relatees is similar in the same way that the other competing pairs are, as is the case in the desk–telephone example above. The second type of ambiguity is called *different-similarity* meaning that each pair of corresponding relationship/relatees is similar in a *different* way from the other competing pairs. More specifically, some of the correspondences may be based on structure similarity of the relatees, while other correspondences may be based on context similarity.

It has also been necessary to make a distinction between *local* ambiguity resolution and *global* ambiguity resolution. The former deals only with the ambiguous correspondences of parent, neighbour, and subpart relatees for one particular cnote, while the latter deals with ambiguous correspondences amongst cnotes produced by the matcher. Currently GRAM only performs local ambiguity resolution, and leaves global ambiguity resolution as the responsibility of the larger learning system, for reasons discussed in section 5.8.

Various methods to resolve ambiguity have been developed in the GRAM system, including the formation of groups or multi-relationships to 'wrap up' the ambiguous items into a single entity; *multiple generalisation* to account for *all* of the ambiguous correspondences; and *best-only* generalisation which simply ignores all but one of the correspondences.

5.2.5 Structure and context interpretation must be considered.

The *interpretation* of structures and contexts affects how they should be generalised. For example, if a concept to be generalised to cover an instance has a *partial* interpretation, then any unmatched relationship/relatees in the instance should be ignored. This and several other aspects of the generaliser pertaining to interpretation are considered in section 5.9. If the interpretation is *disjunctive* then the generaliser must deal with the winning (and any marginally-losing) disjunct correspondences produced by the matcher, as discussed in section 5.7.

5.2.6 The cnote-graph may contain inconsistencies.

Section 4.3.5 of chapter 4 discussed how inconsistencies can be produced in the results of the matcher. An example of a portion of a cnote-graph containing inconsistencies is shown in Figure 5.3, where cnote *A1:B1* assumes *A10* is best matched with *B12*, but *A10:B12* assumes that *A1* is best matched with *B2*. Thus, the generalisation of *A1* and *B1* will be defined in terms of a generalisation of *A5* and *B7*, which will be defined in terms of the generalisation of *A1* and *B2*. GRAM's generaliser does not check for this kind of situation, and so it may seem that the concepts it produces could be nonsensical.

However, as discussed in section 4.3.5, the richness of object descriptions tends to prevent such situations as in Figure 5.3 from occurring unless there are actual ambiguities, in which case the cnote graph would not be as shown (since marginally-losing correspondences would also be included), and the ambiguity resolution mechanisms would deal with it, as discussed in section 5.8.

5.2.7 Objects can be generalised independently from other objects.

A characteristic of GRAM's generaliser is that it can perform a generalisation of two objects independently from the generalisation of their related objects (except for making use of the results of other generalisations). This is partly due to not requiring consistency, and partly due to the representation scheme which an object is defined in terms of its *relationships* to

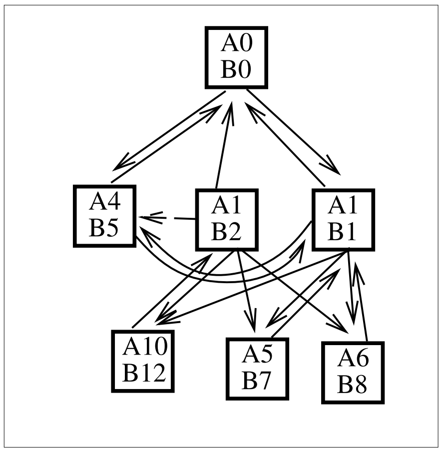


Figure 5.3: A cnote graph with inconsistencies

other objects, rather than in terms of a set of parts defined locally within the object description. This allows an object to be considered a separate entity that does not even include its subparts. Independence is also a result of storing each relationship with only one object, rather than it being common to both of the objects involved, thus enabling relationships to be generalised independently for each object correspondence.

5.3 The generalisation algorithm.

This section presents an outline of GRAM's generalisation algorithm, the details of which are examined more thoroughly in the rest of the chapter.

The basic generalisation process is very simple. To generalise two objects in an cnote, GRAM generalises the structure and context properties and generalises the corresponding relationships, with the relatee of each generalised relationship referring to a generalised relatee obtained by recursively invoking the generaliser on the original relatees. Thus the generaliser spreads outwards through the cnote graph. For example, to generalise objects *tvmain1* and *tvmain2* in Figure 5.2, the generaliser must recursively invoke itself to obtain generalisations of *tv1* and *tv2*, *aerial1* and *aerial2*, *box1* and *box2*, *screen1* and *screen2*, *lleg1* and *lleg2*, and *rleg1* and *rleg2*. These generalisations may also have to obtain generalisations of other pairs of objects.

Currently GRAM performs depth-first traversal, although this could easily be converted to a breadth-first traversal which would give the generaliser an interruptible 'any-time' behaviour. In terms of efficiency, the choice of traversal algorithm is not particularly important, since the generaliser is not performing a search. Rather, it is just following through the graph of winning correspondences that have already been identified by the matcher.

Although the basic generalisation strategy is straightforward, the full algorithm has a number of complexities due to the problems of circularities, ambiguity, unmatched relationship/relatees, and disjuncts. An outline of the algorithm is given in Figure 5.4, which we will now consider step by step.

The main parameter to the generaliser is a cnote. This cnote describes the comparison between two objects, *object1* and *object2*. Other parameters will be introduced later in the chapter.

The process begins by testing whether the generalisation has already been performed earlier (or is already currently being processed). This is necessary to prevent infinite loops. A cnote that has already been (or is currently being) processed has the resulting (perhaps incomplete) concept stored in it. Therefore, this concept can be immediately returned if it is available.

The next step tests whether the two objects are sufficiently similar to justify generalisation. This may involve reinvoking the matcher to obtain a more accurate similarity score, since the match may have been performed with a low spread. If the objects are not generalisable, then a null result is returned.

The generaliser then determines whether the two objects are sufficiently similar to justify modifying *object1* to cover *object2*, or whether a new concept should be created, with *object1* and *object2* as subconcepts. This involves obtaining a *fit-score* for *object2* with respect to *object1*. If modification is appropriate, then *newconcept* is set to be *object1*, otherwise *newconcept* is set to be a copy of *object1*. This step and the previous step are discussed further in section 5.5.

The *newconcept* is now stored in the cnote to prevent recursive cycles, and its *instance-count* is set to be the sum of the instance-counts of *object1* and *object2*. The rest of the algorithm generalises the features of *newconcept* (which is currently *object1*, or a copy of *object1*) to cover the features of *concept2*.

The next section of the algorithm is responsible for generalising the context of the objects. The context properties are processed by generalising the individual attribute values that describe the properties. Section 5.4 explains how this is done.

If the objects are described disjunctively, then the winning disjunct correspondences (produced by the matcher) are processed in the manner discussed in section 5.9.

The next step tests whether the non-disjunctive details of the contexts are similar enough that the parent and neighbour relationships can be generalised. If not, then the context of *object2* is added as a new disjunct of *object1*, or generalised to cover an existing disjunct. The problem of determining when and how to form disjuncts is discussed in section 5.7.

If, on the other hand, the parent and neighbour relationships can be generalised, then it performs three further steps. First, it processes each parent and neighbour relationship/relatee of *object1* as follows: If the relationship/relatee has a good unambiguous correspondence with a relationship/relatee of *object2*, then the attributes describing the relationships are generalised (including the instance-counts), and the two relatees are generalised by recursively invoking the GENERALISE procedure. The relationship of *newconcept* is set to refer to the new generalised relatee.

Second, all unmatched relationship/relatees of *object2* are processed, as discussed in section 5.6. This needs to take into account the *interpretation* of the contexts, since if *object1*'s context is *partial* then unmatched relationship/relatees of *object2* may be able to be ignored. The issue of how the generaliser deals with different structure and context interpretations is discussed in section 5.9.

Third, any ambiguous relationship/relatee correspondences are processed, as explained in section 5.8. This may involve forming multi-relationships or groups, or performing multiple generalisations to account for objects that match two or more other objects in different ways, thus satisfying 'multiple roles'.

The structures of the two objects are then generalised in the same manner as for contexts.

GENERALISE (cnote) (for *object1* and *object2*)

IF generalisation has already been done
  RETURN the concept stored in the cnote.

IF objects are not generalisable THEN
  RETURN null.

(section 5.5)

IF *object1* should be modified then
  Use *object1* as *newconcept*
ELSE
  newconcept ← a copy of *object1*,
    with *object1* and *object2* as subconcepts.

(section 5.5)

Store *newconcept* in the cnote (to prevent recursive loops).

Set instance-count to be sum of the two original instance-counts.

Generalise context:

Generalise *newconcept*'s context properties to cover object2's properties.

(section 5.4)

Process winning context disjunct correspondences (if any).

(section 5.9)

IF context is not similar enough to generalise relationship/relatees THEN
  Add *object2* as a context disjunct of *object1*,
  or generalise an existing disjunct to cover *object2*.

(section 5.7)

OTHERWISE
  FOR all parent and neighbour relationship/relatees of *object1*:
    IF it has a good unambiguous (or both-winning) correspondence with object2 THEN
      Generalise *newconcept*'s relationship to cover *object2*'s relationship.

(section 5.4)

      newrelatee ← GENERALISE (relatees-cnote).
      Set the generalised relationship to refer to *newrelatee*.

  Process unmatched relationship/relatees of *object2*.

(section 5.6)

  Process ambiguous relationship/relatee correspondences.

(section 5.8)

Generalise structure:

(as for context)

Figure 5.4: The GENERALISE algorithm.

5.4 Attribute generalisation

Since properties and relationships are represented as attribute vectors, the most basic component of the generaliser is the attribute-value generaliser. This section describes the different kinds of generalisation performed for each of the different kinds of attribute (*i.e.* *numerical*, *nominal*, *boolean*, *etc*, as defined in section 3.3.2).

Numerical attribute values are generalised simply by computing a new mean and standard deviation to account for the two contributing values, which may be generalised or ungeneralised.[1]

Directional attribute values are generalised in a similar manner to ordinary numerical attributes, except that modulo arithmetic is used.

Nominal attribute values are generalised by combining the probability distributions of the symbols in the contributing values. This is done by forming a union of their symbols and summing the instance-counts for each pair of corresponding symbols.[2] The instance-count for the generalised attribute value as a whole is the sum of the instance-counts of the two contributing values. Some examples of this are given below. An ungeneralised value is specified as a single symbol or a set of symbols (a *symbolset*) in which each symbol has an implicit instance-count of 1. A generalised value is specified with its overall instance-count at the start, followed by a list of symbols and their individual instance-counts. (Notice that since an instance may be a *symbolset* containing several symbols, the overall instance-count of a generalised value is not necessarily the sum of the instance-counts of its individual symbols.)

(1 (red:1))	+	(red)	→	(2 (red:2))
(4 (red:1, blue:3))	+	(green)	→	(5 (red:1, blue:3, green:1))
(4 (red:1, blue:3))	+	(red green)	→	(5 (red:2, blue:3, green:1))
(4 (red:4, blue:3))	+	(2 (red:1, green:1))	→	(6 (red:5, blue:3, green:1))

Currently GRAM's generaliser does not perform any 'climb-hierarchy' operations to generalise symbols (such as producing *polygon* from the symbols *rectangle* and *hexagon*). Instead it is assumed that all possible generalisations are (if necessary) included in the instance descriptions themselves. The combined instance-counts of two attribute values reflect what is common, as shown by the following example, which results in a generalised value for which *polygon* has 100% 'probability', and *rectangle* and *square* have 50% probabilities.

(1 (rectangle:1, polygon:1)) + (1 (polygon:1) (hexagon:1))

→ (2 (polygon:2, rectangle:1, hexagon:1))

[1] This is achieved by recording the sums, sums-of-squares, and instance-counts of the instance values.

[2] GRAM does not need to perform a 'climb-hierarchy' operation because it assumes all values at all levels of generality are present in the instance, such as *hexagon* and *polygon*. In this sense it is similar to Connell and Brady's Gray coding scheme described in section 2.7. However, it differs from their scheme because the generalisation is the *union* rather than the *intersection* of the values, and thus loses less information.

Boolean attributes are a special case of nominal attributes, where only two values, true and false, are allowed, and so they are generalised by summing the instance-counts, true-counts, and false-counts of the two values.

Position values are simply a pair of numerical values, and so are generalised by generalising each of the two values independently. It may seem that this is an over-generalisation. For example, generalising two positions (0.1, 0.1) and (0.2, 0.2) would produce a new position which will match well with positions whose x coordinate is (roughly) in the range 0.1..0.2 and y positions whose y coordinate is also (roughly) in the range 0.1..0.2. This generalisation has lost the fact that the x and y coordinates are the same, or more importantly, that the positions are both on a line in the direction of 45 degrees. However, since direction and distance are included explicitly as attributes anyway, this is not a problem.

Profile values are generalised in the same manner as positions, by independently generalising each of the values in the profile vector.

5.5 Determining what is to be generalised.

This section considers various methods for controlling and determining which objects are to be generalised. The first method is by the use of scope-restriction parameters that can be passed to the generaliser from an external source, such as a teacher, or the larger learning system, or from the generaliser itself via a recursive invocation. These can be considered ways of "focusing the attention" of the generaliser. The second method is by the use of various criteria to determine whether a pair of objects are sufficiently similar to justify generalisation. These criteria may also be passed as parameters.

5.5.1 Scope restriction can be achieved by marking the generalisable objects.

In order to control which objects are to be generalised (or more precisely, which cnotes are to be processed), GRAM allows instance objects to be marked as *in-scope* or *out-of-scope*. This is a similar method to that used by the matcher, except that the matcher also needs to mark the concepts, so that both object-graphs are restricted, otherwise mismatches would occur. The generaliser only needs to restrict the instance graph, since it does not have to perform any search: correspondences are already established.

For example, if a teacher pointing to a handdrill wants a robot to generalise its existing *handdrill* concept to cover the new instance, ignoring the context of the handdrill (*i.e.* the room, the workbench, other tools, *etc*), then s/he only needs to indicate the restricted scope of the instance, rather than restricting the scope of concept-memory.

In the case of concepts being matched with concepts, only the second concept and its *in-scope* related concepts are marked. Likewise, when generalising two instance graphs, only the second is marked.

The most useful form of scope-restriction is to restrict generalisation to the *substructure* of an observed instance, rather than its context, since many of the concepts that a robot system would be required to learn are relatively concept-independent, such as *handdrill* (as above), *chair*, *telephone*, *bicycle*, *etc*.

An example of the effects of using scope restriction is given in Figure 5.5, where the substructure portion of the instance graph is marked, as indicated by the shaded object nodes. If we assume that the original concepts can be modified, rather than new ones created, then the only concepts affected are *A1*, *A2*, *A3*, *A4*, and *A5* (if we also assume that these are the only acceptable correspondences with the marked objects of the instance graph). This is illustrated by figure (a) in which the dark-shaded concept nodes have been modified by generalising them to cover the corresponding B objects. All of these generalisations occur by spreading from the *seed* cnote, which in this case happens to be *A1:B1*.

Although the out-of-scope An objects are not generalised, the *relationships* from *A1*, *A3*, and *A5* to the out-of-scope objects *A6*, *A7*, *A8*, *A9*, and *A10 are* generalised to cover the relationships from *B1*, *B3*, and *B5* to the out-of-scope objects *B6*, *B7*, *B8*, *B9*, and *B10*, if the correspondences are sufficiently strong.

Notice also that *A5* now has a neighbour relationship with *B11*, since there was no corresponding neighbour of the original *A5*. Thus the generalised and ungeneralised object graphs have been linked together via a neighbour relationship. This issue of dealing with unmatched objects is discussed in section 5.6.

If it is necessary to create *new* concepts for all generalisations, rather than modifying existing concepts, then new concepts are only created for cnotes involving an *in-scope* instance, as shown in (b) of the figure. If a relationship of an original concept has a relatee concept that is not matched with an *in-scope* instance, (as in the case of the *A1–A6* relationship), then the generalisation of that concept includes a copy of the original relationship, which will still refer to the original unchanged relatee concept, as shown by the relationship from *A1+B1* to *A6*.

An alternative scheme is to actually generalise the relationship to cover the instance relationship (assuming there is an unambiguous winning correspondence) but without generalising the relatee. In this scheme, the scope-restriction is being interpreted in a slightly weaker manner, since relationships from in-scope objects to out-of-scope objects are processed.

If a robot system is operating in a relatively unfocused manner (without a teacher or specific task to be achieved, such as when 'wandering' or 'pondering') then generalisation scope may be left unspecified, so that the generaliser just spreads throughout the cnote graph produced by the matcher. In fact, in a full learning system, matching and generalisation could be performed concurrently, with the generaliser requesting more results from the matcher as required, and results of the matcher causing the generaliser to be invoked when a good classification is obtained.

5.5.2 Scope restriction can be achieved by specifying the required spread.

Another convenient way of specifying the generalisation scope is by the use of a *required-spread* parameter that specifies how far the generaliser should spread through the cnote graph. One spread value can be given for substructure-spread and one for context-spread. This can be interpreted to be a way of specifying the 'effort' to be applied, or the amount of detail considered important for generalisation. Again, this is similar to the use of the required-spread parameter by the matcher.

The generaliser uses the required-spread parameter to determine which instance-objects are to be marked as in-scope or out-of-scope. For example, in the handdrill example above, a context-spread of 1 and a structure-spread of 3 would cause the generaliser to mark subparts that are 3 deep in the decomposition hierarchy, and to mark context objects that are directly related to the seed object *or* to a marked subpart. This is illustrated in Figure 5.6, where the dark-shaded object is the seed object, and the light-shaded objects are those that are in-scope.

If existing concepts in memory are to be generalised to cover the objects in this instance-graph, the concepts *person*, *hand* (or perhaps *clenched-hand*), and *workbench* can only have their properties generalised, with no further spread, although since *hand* is a subpart concept of the *person* concept, *person* will implicitly be generalised more than just by property generalisation. Using this low-context-spread restriction, the system will learn more about the context of

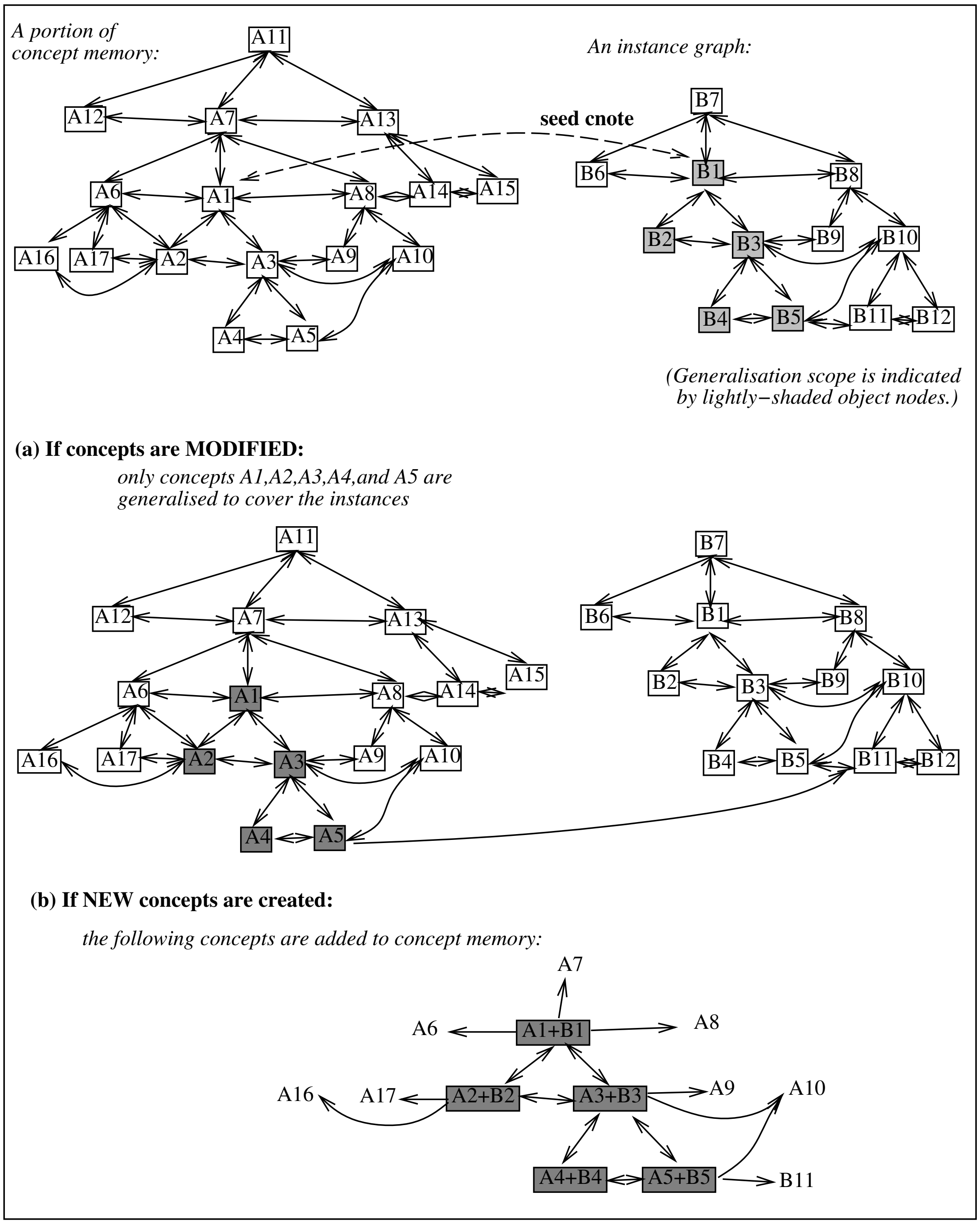


Figure 5.5: Scope restriction.

handdrills, such as what kinds of rooms they appear in, what kinds of people use them, what kinds of hands hold handdrills (*i.e.* usually a clenched hand), *etc*, but without investing effort into generalising details of the context that are not directly related to the handdrill.

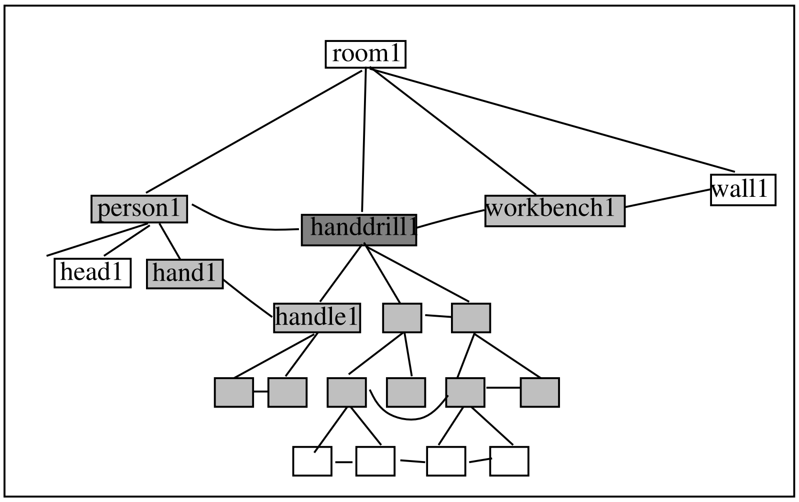


Figure 5.6: Spread restriction

5.5.3 Proximity-scores and fit-scores determine whether to generalise or modify.

Section 4.3.8 discussed the distinction between fit-scoring and proximity-scoring, and mentioned that proximity-scores are used to determine whether two objects are similar enough to justify forming a generalisation, and fit-scores are used to determine whether the second object 'fits' the first object well enough to justify modifying the first object's description or whether a new concept should be created with the original descriptions unchanged but linked to the new concept as subconcepts.

The threshold cutoff values for generalisability and modifiability are therefore another means for controlling what cnotes are processed by the generaliser. These values can be either passed as parameters to the generaliser, or obtained from globally predefined defaults.

The generalisability and modifiability parameters must be able to be overridden, since some concepts, such as *furniture* or *hand-tools*, are formed from instances that are significantly different, and must be formed on the basis of explicit instruction from a teacher, or from a larger system that deals with functional knowledge.

The generaliser might leave the concept unchanged

If the fit-match score is very high, and if a large number of instances of the concept have already been seen, then it might not even be necessary or desirable to generalise the concept at all. Thus a system operating in a familiar environment day after day would not need to continually modify its concepts on the basis of every object it observes. This is perhaps the most important means for controlling the generalisation spread. In the example of the handdrill given earlier, the neighbouring concepts *person*, *hand*, *workbench*, *etc* would most likely be left unchanged, even without the scope-restriction.

5.5.4 The matcher may need to be reinvoked.

As the generaliser spreads through the cnote network, it may reach a cnote that was evaluated by the matcher with low-spread effort (perhaps due to being on the fringe of the match scope) and hence the lowerbound of the score might be too low to justify generalisation, even if the estimated similarity score is high. Two possibilities can occur here. Either the generaliser can simply bottom-out, leaving the original concept unchanged, or it can request the matcher to perform a more thorough comparison based on the *required-spread* parameter of the generalisation process. The decision depends on task requirements, and is thus an additional parameter to the generaliser.

5.6 Dealing with unmatched parents, neighbours, and subparts.

This section explains how the generaliser processes a relationship/relatee that is unmatched, or more precisely, that has a winning correspondence whose score is too low to justify generalisation.

The simplest situation to deal with is illustrated in Figure 5.7, in which *X0* is to be generalised (by modification) to cover *Y0*. In this situation, the relationships *X0–X3* and *X1–X3* do not have any acceptable correspondences with the relationships of *Y0* and *Y1*. Generalisation is straightforward: The relationships and objects that *do* correspond well can be generalised (with their instance-counts incremented from 1 to 2), and the extra unmatched relationships to and from *A0* and *x1* are left unchanged (with their instance-counts remaining as 1). The unmatched features are *not* dropped from the description (in contrast to Winston's system), but instead become optional features. The resulting modified X objects are shown at the bottom of the figure.

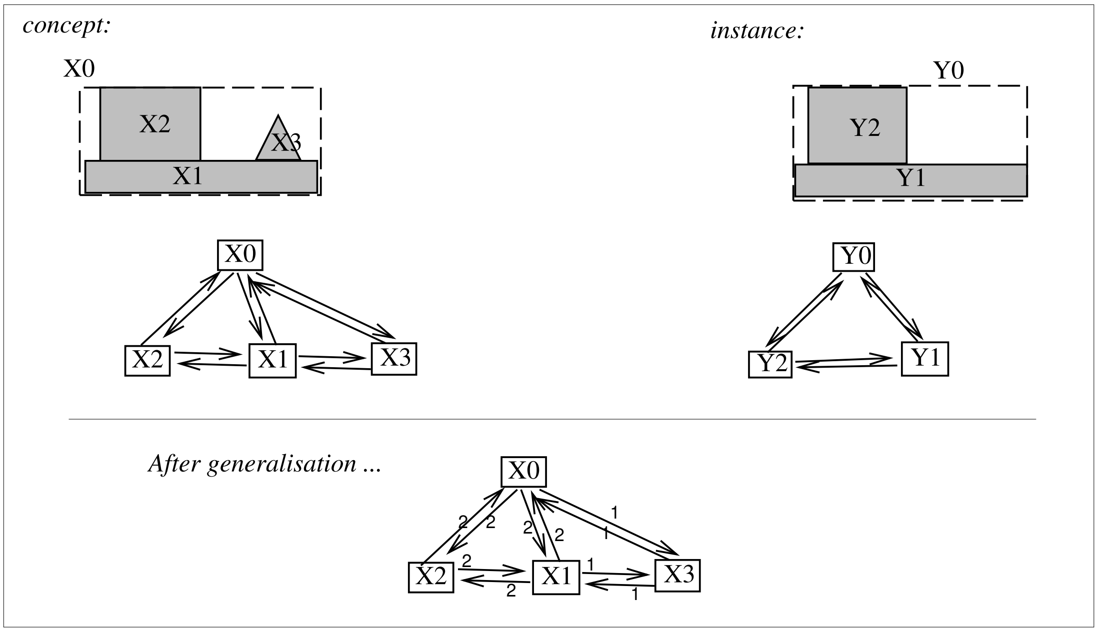


Figure 5.7: An unmatched instance-part

A more difficult situation is when new relationship/relatees have to be *added* to an object description. For example, in Figure 5.8, the *B0–B3* subpart relationship/relatee has no acceptable correspondence with the subpart relationship/relatees of *A0*, and so the generalisations of some of the A objects must have new relationship/relatees added. Two alternative methods could be applied here, as described in the following sections. At this stage it is not clear which method is best, since more evaluation in the context of a full learning system needs to be performed to determine this. Therefore, this thesis simply describes the alternative methods that are supported by GRAM, and indicates their overall advantages and disadvantages.

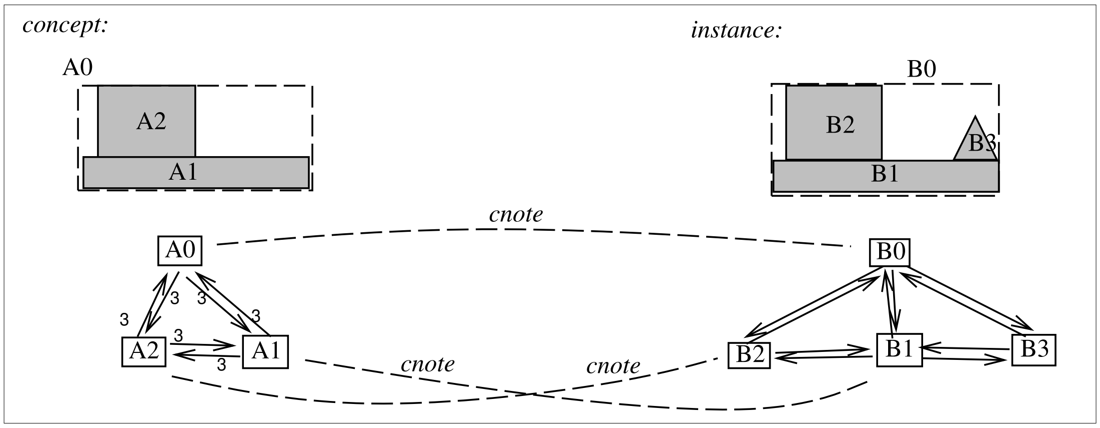


Figure 5.8: An unmatched instance-part

5.6.1 Method-1: The generalisation refers to the original unmatched object.

The first method involves copying the unmatched relationships and adding them to the corresponding generalised objects. The new generalised relationships refer to the *original* unmatched objects. This is illustrated at the top of Figure 5.9 in which new concepts *A0+B0*, *A2+B2*, and *A1+B1* are created as generalisations of the An and Bn objects. *A0+B0* has a copy of the *B0–B3* relationship (with an instance-count of 1), which refers to the original *B3* object. Likewise, *A1+B1* has a copy of the *B1–B3* relationship. However, the *B3* object does *not* have relationships to *A1+B1* and *A0+B0*, but instead remains unchanged, with relationships to *B0* and *B3*.

This method is very simple for the generaliser, but leads to problems for the matcher. This is because the generalised objects (*A0+B0 etc*) are now defined in terms of two object graphs: the graph of the other generalised concepts, and the graph of the original Bn objects. This means that to match a new instance with the concept, the matcher has to compare the instance with both graphs. For example, to match *A0+B0* with *C0* shown in the lower half of Figure 5.9, the matcher must compare *B3* with *C3*. This then requires the neighbours *B2* and *C2* to be compared, which leads throughout the Bn and Cn graphs. At the same time, the matcher must compare *C1*, *C2*, and *C3* with the objects in the An graph. If the objects *A0*, *B0*, and *C0* were much more complex, then the relationship between *A0+B0* and *B3* would lead the matcher into a considerable of work which is mostly redundant, since the most of the Bn objects are subconcepts of the An-Bn objects.

One way to reduce this redundant matching is to make use of the AKO hierarchy, as explained in section 4.4.8 of chapter 4. For example, since *B2* is a subconcept of *A2+B2*, then if the matcher has already compared the new object *C2* with *A2+B2* then this result could be used as an estimate for the desired *B2:C2* comparison. Conversely, if *B2* and *C2* were matched first, then when the matcher tries to match *B2* with *A2+B2*, the *B2:C2* result could be used as a lower-bound on the score.

However, another problem with this method is that an explicit generalisation of the unmatched object is not produced. In the example above, the optional subpart *B3* of the new concept *A0+B0* is not generalised by making its relationships refer to the new An+Bn concepts, and thus there is no explicit 'transfer of information'.

This method may seem to lead to a further problem: if the An+Bn concepts are later generalised with more instances, then generalisation could lead to object *B3* being generalised, which would therefore alter *B0* and the other original instance objects that are defined in terms of B3. However, this is only a problem if B3 is modified (rather than by being generalised as a new concept), and modification only occurs when it is acceptable.

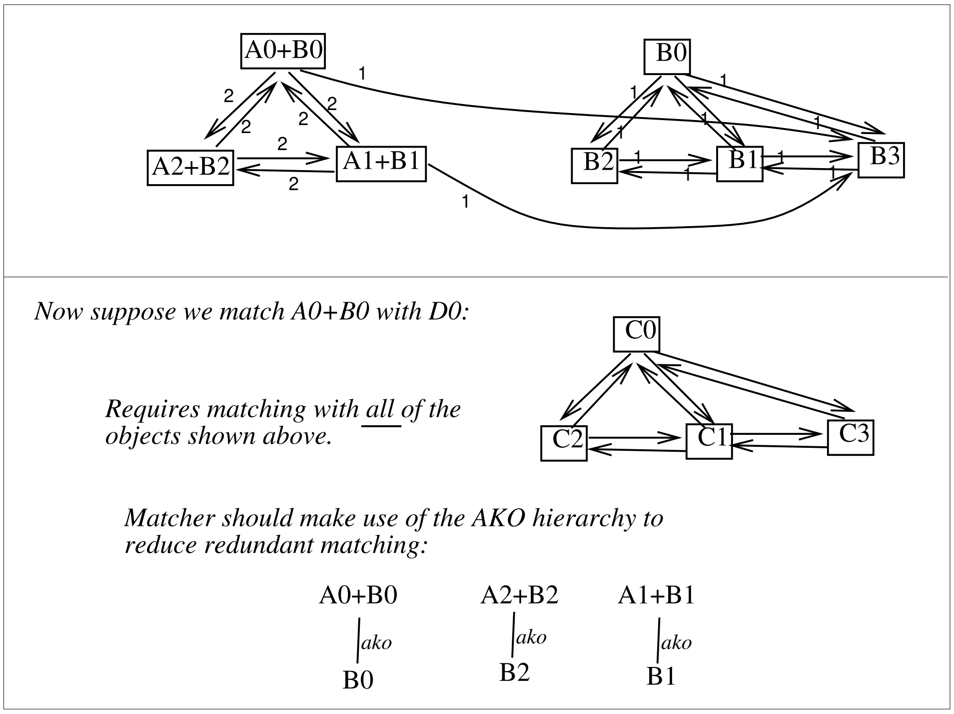


Figure 5.9: Method-1: Add relationships referring to the unmatched objects.

5.6.2 Method-2: Unmatched objects are copied.

The second method for dealing with an unmatched object is to create a *copy* of it, altering the relationships of the copy so that their relatees are the new generalised concepts. This is illustrated in Figure 5.10, where a concept *newB3* is created by copying *B3* and creating relationships to refer to and from the other An+Bn concepts. Thus *newB3* is generalised because it is now defined in terms of generalised objects.

This is a more complex process than the first method, and also leads to a larger and more complex concept-memory, especially if the unmatched object has unmatched substructure and/or unmatched context (which in most cases it will) since this must also be copied. The AKO hierarchy is made more elaborate since the new description must be specified as a superclass of the original description. However, these complications are necessary to resolve the problems of the first method.

One difficulty in this method is determining what relationships to create for the new object. It is straightforward in the situation above, since the relatees of *B3* (*i.e.* *B0* and *B1*), both have unambiguous winning correspondences with *A0* and *A1* respectively, and so the new object, *newB3*, can be given relationships to the generalisations of these, namely *A0+B0* and *A1+B1*. However, if there were ambiguities, these would have to be resolved. For example, Figure 5.11 shows a situation in which the neighbour *B1* of the unmatched triangle *B3* could be corresponded with either of *A2* or *A5*, and there is no way to choose between them. However, this ambiguity means that it does not particularly *matter* which correspondence is chosen. In fact, in this situation the objects *A1*, *A4* and *B4* are likely to be generalised (by the ambiguity-resolution mechanism) to form a single typical-member concept for a group, and so the copied version of *B3* can refer to the components of this concept rather than just to either the *A2+B1* or *A5+B1* generalisation.

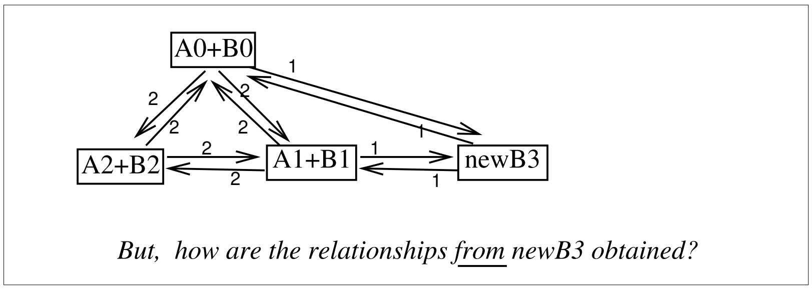


Figure 5.10: Method-1: Create a *copy* of the unmatched instance.

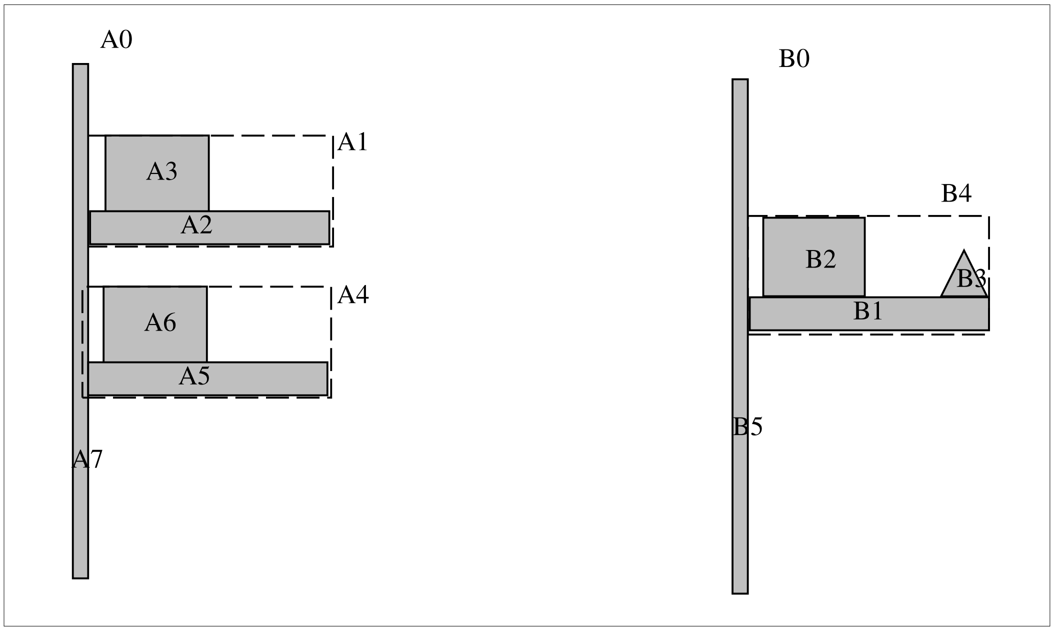


Figure 5.11: Unmatched and ambiguous.

5.7 Partial similarities and disjunct formation.

In this section we explore when and how disjunctive concept descriptions are created and generalised by GRAM. We first consider the two main kinds of partial similarity that necessitate disjunct formation, and then consider the issues of creating disjunctive descriptions and generalising them with future instances.

5.7.1 Generalisation (by disjunct formation) may be justified by structure-only or context-only similarity.

Generalisation can sometimes be justified on the basis of a partial similarity between two objects, even if the overall similarity score is not particularly high. 'Partial similarity' is used here to mean that two objects are similar in some aspects but not others. The two kinds of partial similarity that justify generalisation in GRAM are context-only similarity (which leads to structure disjunction) and structure-only similarity (which leads to context disjunction).

For example, in Figure 5.12 objects *A2* and *B2* clearly correspond, based on their high *context* similarity (and also their overall shape), even though their structures differ significantly. The structures of these two objects cannot be generalised by generalising their properties and merging their relationships. Instead, the new concept must have a disjunctive structure, of which the disjuncts are the structures of the subconcepts *A2* and *B2*, as shown at the bottom of the figure. (*A0*, *A1* and *A3* are sufficiently similar to *B0*, *B1*, and *B3* respectively, that they could perhaps be *modified*, but for the purposes of the discussion it is assumed that new concepts are created.)

In Figure 5.13 objects *B2* and *C2* correspond only on the basis of high *structure* similarity, since the context similarity is poor. The resulting generalisation has a generalised structure and a disjunctive context, as shown at the bottom of the figure. The context disjuncts of *B2+C2* are the contexts of the subconcepts *B2* and *C2*. Notice also that *B1+C1* and *B3+B3* each have two relationships to *B2+C2*, since these are considered insufficiently similar to generalise.

5.7.2 Some examples of disjunct generalisation and formation.

In the above examples, the original descriptions were non-disjunctive. The following sequence of examples show how disjunct generalisation and formation occurs when one of the descriptions is already disjunctive. They also give more examples of the use of fit-scores and proximity-scores to determine generalisability and modifiability.

Figure 5.14 shows an object *D0* which is to be generalised to cover the *A0+B0* concept shown in Figure 5.12. In this case there is a structure-only similarity between *A2+B2* and *D2*, since *D2*'s structure does not match either of the structure disjuncts of *A2+B2*. This may seem to suggest that a new concept *A2+B2+D2* should be created, as for the earlier situation in Figure 5.12. However, the variance of the structure of *A2+B2* is high, due to the disjunction, and this causes GRAM to compute fit-scores primarily on the basis of context similarity. Therefore, the fit-score of *D2* with respect to *A2+B2* is actually quite high, and so *A2+B2* can be modified to

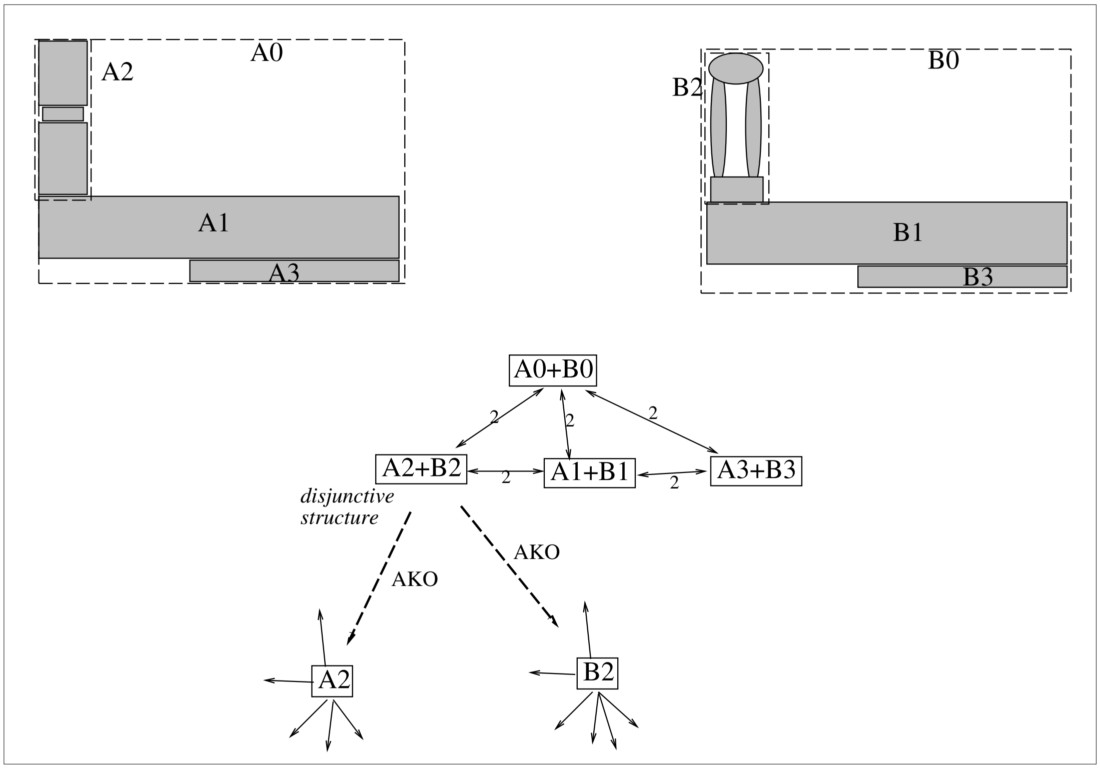


Figure 5.12: Context-only similarity.

cover *D2*. The results of the generalisation of *A2+B2* are shown at the bottom of the figure. (If fit-scoring did not take into account disjunction in this manner, then every new object such as *D2* would force the creation of new *'A2+B2+...'* object.)

Figure 5.15 shows another object *E0*, whose subcomponent *E2* is similar to the disjunct *D2* of *A2+B2*. Therefore, there is no need to add a new disjunct to *A2+B2*, since the structures of *D2* and *E2* are generalisable. The only significant difference between them are the small circles in *E2*. The fit-score is not high enough, however, to justify modifying *D2*, and therefore a new concept *D2+E2* is created, to which *D2* and *E2* are linked as subconcepts. *D2+E2* is *not* defined disjunctively though, since a full description of the substructure (including the optional small circles) is specified non-disjunctively in *D2+E2*. *D2* and *E2* are just specialisations of *D2+E2*.

Suppose the next object observed is *F0*, as given in Figure 5.16. In this example, *A2+B2* and *F2* have similar structures (since the disjunct *D2* matches *F2* very closely) but different contexts. The fit-scores for all of the Fn objects with respect to the An+Bn objects is poor, due to the differing contexts, and so new concepts must be formed. The new concept *A2+B2+F2* is interesting because it is now defined by a disjunctive structure *and* a disjunctive context. (Recall from chapter 3 that the subconcepts of such a concept define both the structure and the context disjuncts). The new disjunct (subconcept) *D2+F2* has a disjunctive context defined in terms of *D2* and *F2*.

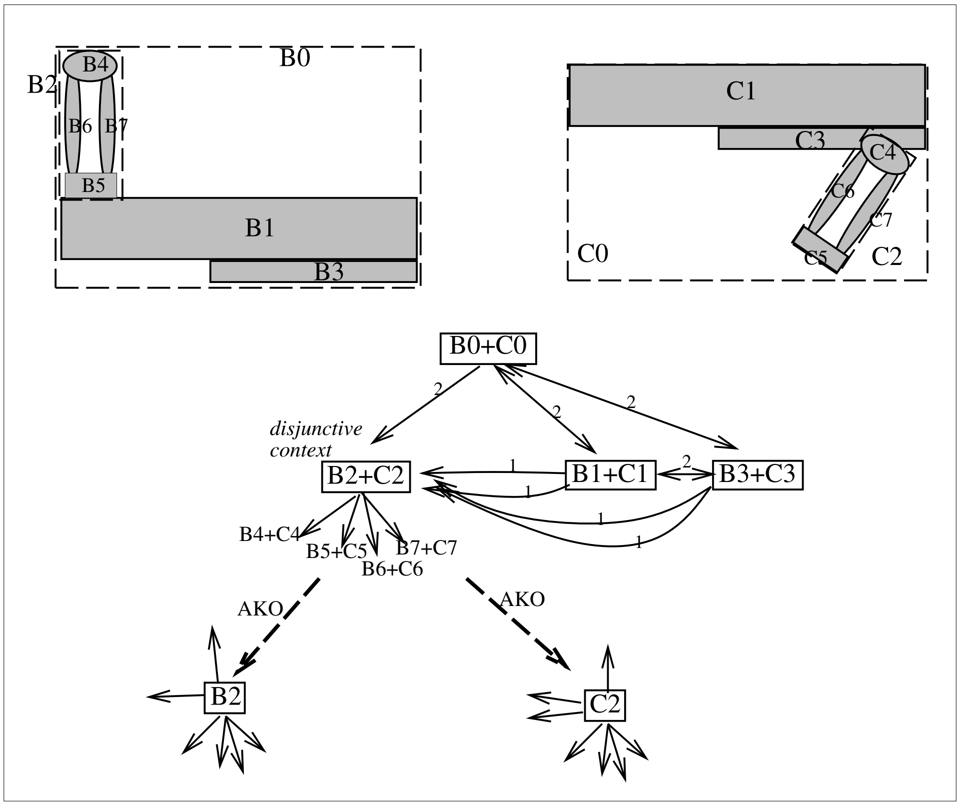


Figure 5.13: Structure-only similarity.

The next object to be considered is *G0*, in Figure 5.17. In this case, the structure of *G0* does not match any of the disjuncts' structures, and its context also does not match any of the disjuncts' contexts. Therefore, it does not have a sufficiently high score to justify generalisation.

5.7.3 *Import-from* relationships could be created.

When disjuncts are formed, it is often desirable to define them using *import-from* relationships (as defined in section 3.4.1) since this can reduce redundancy and allow a greater transfer of information amongst the disjuncts. For example, concept *B2+E2* in Figure 5.16 could be defined by importing its context from the other disjunct *A2*.

However, the current version of GRAM does not create *import-from* relationships, since it involves issues pertaining to the problem of memory organisation in the larger learning system. Therefore, although the representation and matcher support *import-from* relationships, the generaliser does not.

5.7.4 Disjuncts could be converted to an 'any' interpretation.

If a disjunctively-defined concept acquires, through generalisation, a large number of structure [or context] disjuncts, then it may be desirable to give the structure [or context] an '*any*'

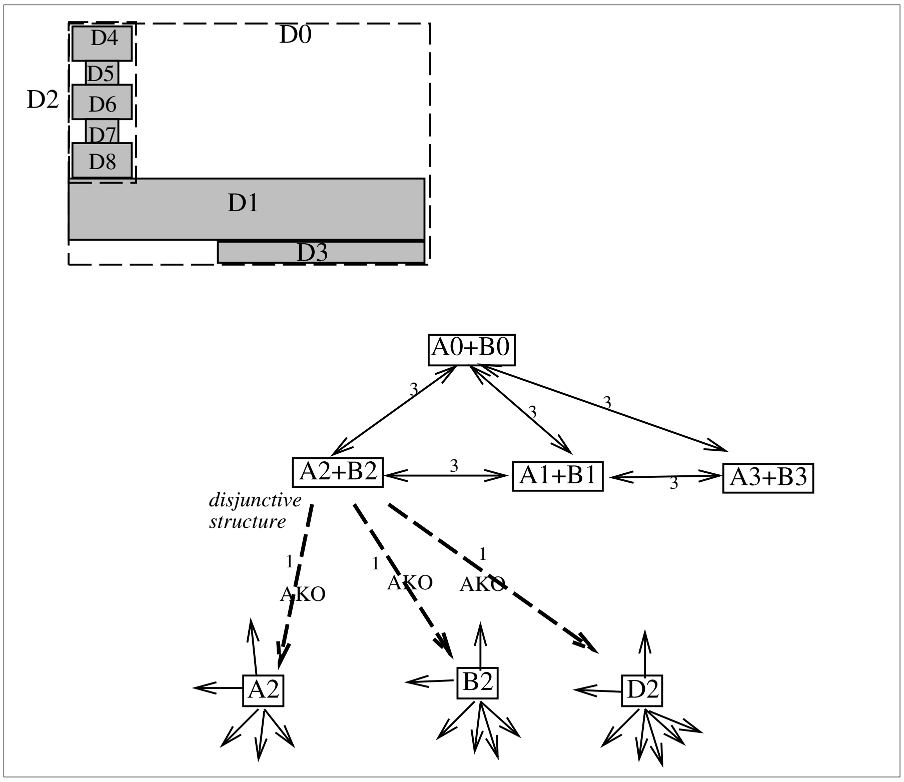


Figure 5.14: Creating a new structure disjunct.

interpretation. This means that future matching will ignore the structure [or context] when computing the overall similarity score, so that the concept will be matched solely on the basis of its context (or structure]. However, the subconcept disjuncts could still be retained as ordinary subconcepts, or even as disjuncts if the context [or structure] is defined disjunctively in terms of some or all of those subconcepts.

As with the operation of creating *import-from* relationships in section 5.7.4, this operation is not done by the GRAM generaliser, but is the responsibility of the memory-organisation component of the larger learning system. This and other operations for 'cleaning up' or optimising concept-memory could be performed during idle 'sleep' time, rather than during the process of generalisation itself.

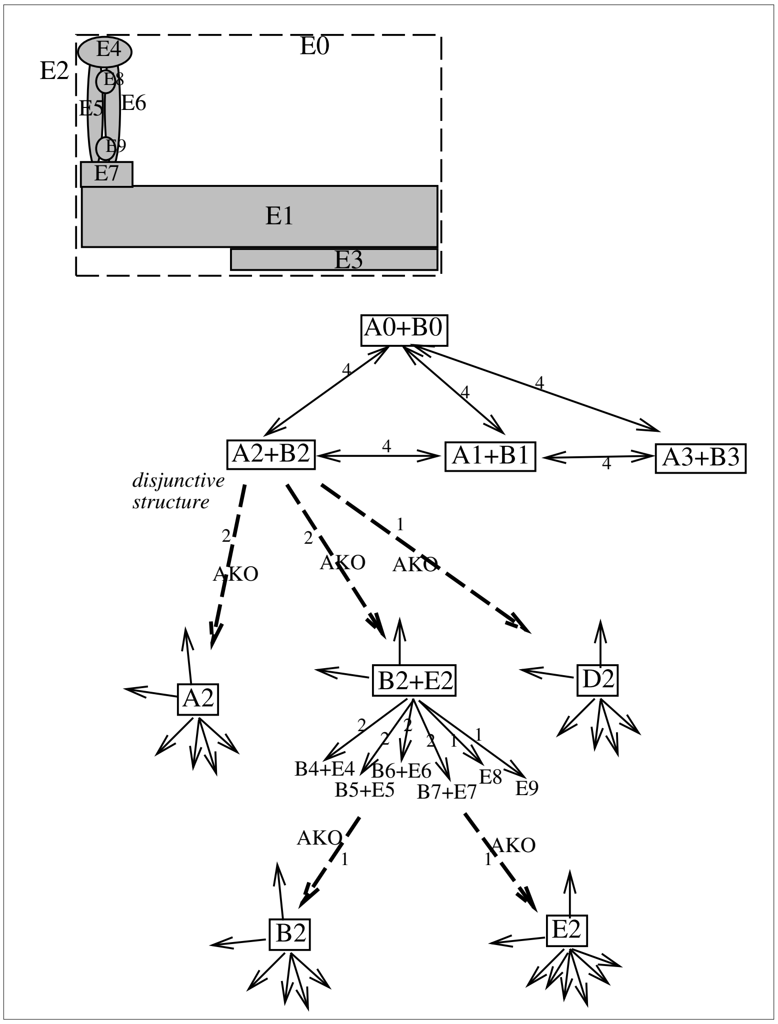


Figure 5.15: Generalising an existing structure disjunct.

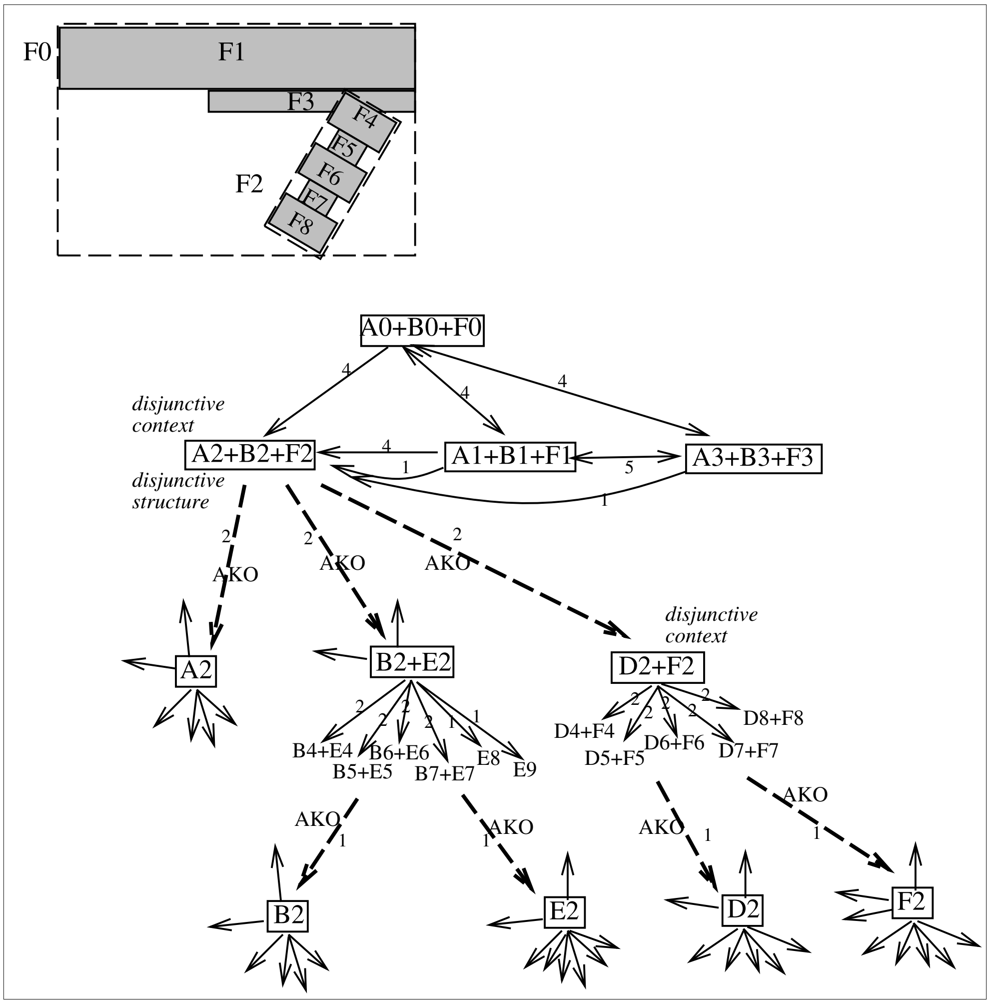


Figure 5.16: The new concept is disjunctive in both structure and context.

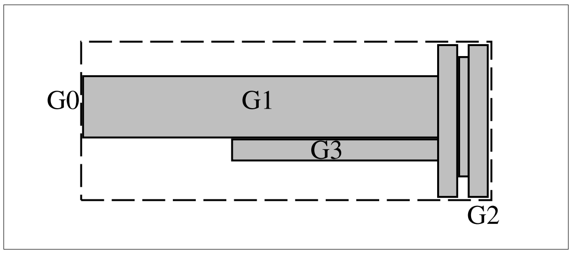


Figure 5.17: A new object is too different to justify generalisation.

5.8 Ambiguity.

In most of the examples we have looked at in this chapter so far, there have been clear-winning unambiguous one-to-one correspondences between the parent, neighbour, and subpart relationship/relatees of the objects being generalised. This section addresses the problem of ambiguity, where it is not obvious which correspondences should be selected for generalisation.

The section identifies several different kinds of ambiguity that can occur in GRAM's domain, and discusses the most desirable output of the generaliser for each kind of ambiguity situation. It then considers the issues in achieving this, and explains the ambiguity-resolution mechanisms used in the GRAM system.

5.8.1 *Similar-similarity* ambiguity and *different-similarity* ambiguity.

Ambiguity comes in two main forms, namely *similar-similarity* ambiguity and *different-similarity* ambiguity, each of which requires a different kind of generalisation. *Similar-similarity* occurs when several correspondences all score equally well in all respects. This situation occurs when several similar concepts, or several similar instances, are similar to each other, since any correspondences with them will necessarily compete with each other. An example of this is given in Figure 5.18 where *A3* matches *B3* in a similar way to its match with *B4*, because *B3* and *B4* are similar to each other.

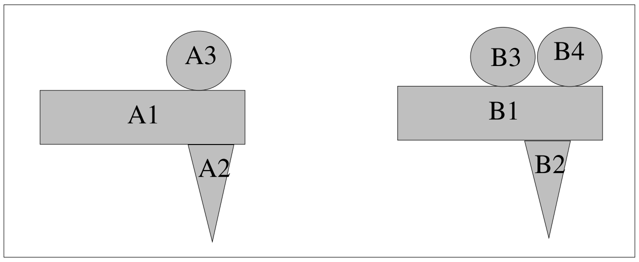


Figure 5.18: Ambiguity due to 'similar-similarities'

Different-similarity, on the other hand, occurs when several correspondences score equally well overall, but have different scores for the finer details. In other words, the items are partially similar in *different* ways. This is illustrated in Figure 5.19 where the overall similarity scores for the *A4*:*B3* and *A3*:*B3* correspondences are approximately the same, but the former has a good structure similarity and poor context similarity, and the latter has the converse. In this situation, the set of concepts, and/or the set of instances, are *not* similar to each other. In the example, *A3* and *A4* are quite different, but *B3* is partially similar to both of them, in different ways.

Similar-similarity ambiguity is resolved by forming a multi-relationship.

The desired resolution of similar-similarity ambiguity is to merge ambiguously matching relationships and relatees into a single multi-relationship with a generalised *howmany* count

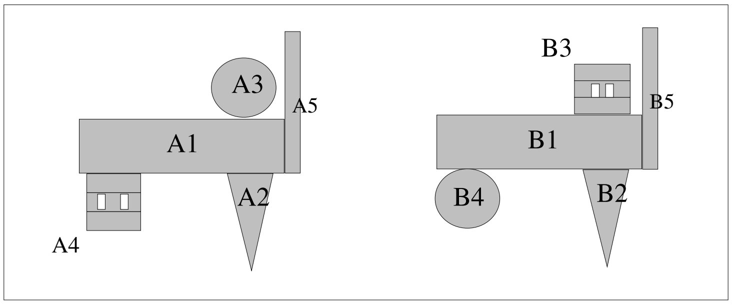


Figure 5.19: Ambiguity due to 'different-similarities'

and a single generalised relatee, perhaps also forming a *group* of which this is the typical member. For example, the result of generalising the objects in Figure 5.18 above is as shown in Figure 5.20, which includes a generalisation of the circles *A3*,*B3*,and *B4*, and a multi-relationship to it from *A0+B0* and also from *A1+B1*, both with generalised *howmany* counts indicating the range 1..2. The *A3+B3+B4* concept also has two neighbour relationships to itself, one which refers to the ball on the right (obtained from *B3*) and the other which refers to the ball on the left (obtained from *B4*) both of which have instance-counts of only 1, since only one of the contributing instances had each of the relationships. A grouped object has not been formed in this particular example.

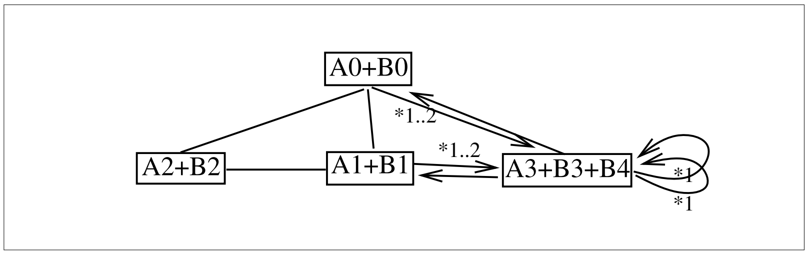


Figure 5.20: Similar-similarity ambiguity resolution.

Different-similarity ambiguity is resolved by multiple generalisations.

The desired resolution of *competing*-similarity ambiguity cannot involve multi-relationships or grouping, since the objects involved are not similar. A distinct generalisation is produced for each of the ambiguous correspondences, since each captures distinct and potentially important information. For example, in the situation of Figure 5.19 the best generalisation would specify that there is 'something' at the top right of *A1+B1*, something at the bottom left, a circle attached somewhere, and a square-thing attached somewhere. The generalised objects in Figure 5.21 capture this information. Notice that the relationships between *A5+B5* (the vertical rectangle) and *A3+B3* (the generalisation of the two circles) has an instance-count of only 1, since only *A3* has a direct relationship with *A5*.

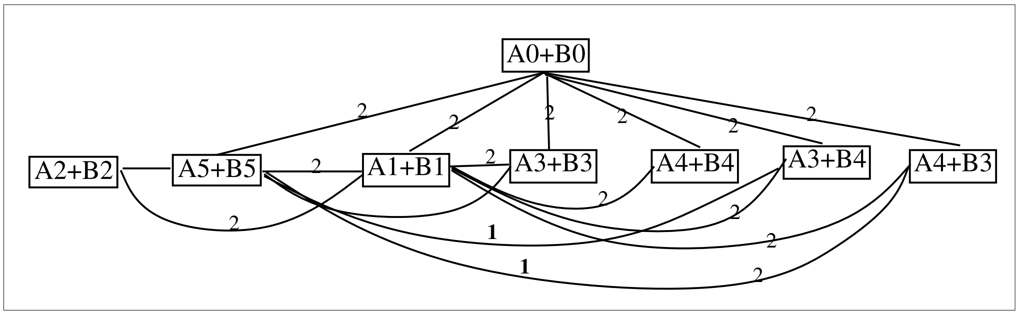


Figure 5.21: Different-similarity ambiguity resolution.

Vertical ambiguity.

A special form of different-similarity ambiguity is illustrated in Figure 5.22 and is called *vertical ambiguity. A2* matches *B4* well on context, but poorly on substructure, and it matches *B2* (which is a subpart of *B4*) well on structure but poorly on context. Thus *A2* ambiguously matches two objects that are along the same branch of the object decomposition hierarchy.

The key characteristic of vertical ambiguity is that there is a composite object X which consists of a *large* subpart *X1* and one or more small attachments, such that if, in another observed object, the object corresponding to X is non-composite, then that object will also match *X1*.

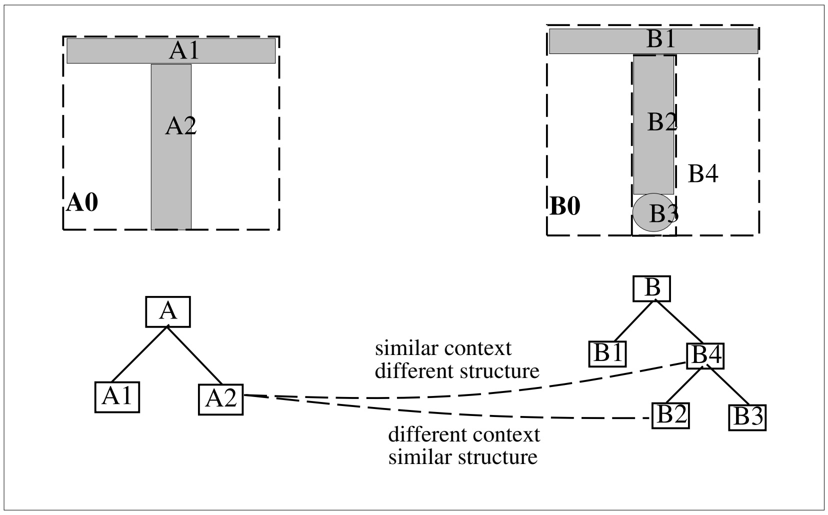


Figure 5.22: Ambiguity due to 'different-similarities'

Vertical ambiguity should also be resolved in the same way as for ordinary different-similarity (by producing a separate generalisation for each correspondence) since each correspondence is distinct and captures important information. For the example above, this would result in the object *A0+B0* (the whole 'T') having two subpart relations, one to *A2+B2* (the generalisation of both solid vertical rods), and the other to *A2+B4* (the generalisation of the solid A rod and the composite B rod), as shown in Figure 5.23. The result differs from the resolution of ordinary different-similarity because one of the generalisations becomes a subpart of the other.

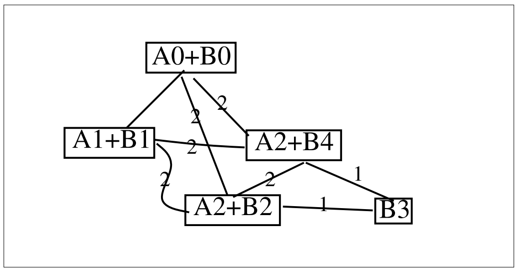


Figure 5.23: Ambiguity due to 'different-similarities'

Multi-relationship ambiguity is a special kind of similar-similarity.

A special case of similar-similarity ambiguity is *multi-relationship ambiguity*, where two or more relationship/relatees of one object match the generalised multi-relationship of the other object, as illustrated in Figure 5.24. Object *A1* has a multi-relationship to *A2*, and *B1* has three ordinary relations to *B2*, *B3*, and *B4*. When matched, *A2* ambiguously matches all of *B2*, *B3*, and *B4*. The resolution of this form of ambiguity is straightforward, since the relatee of the multi-relationship can be generalised to cover all of the other relationship/relatees, and the *howmany* count updated accordingly, as shown at the bottom of the figure.

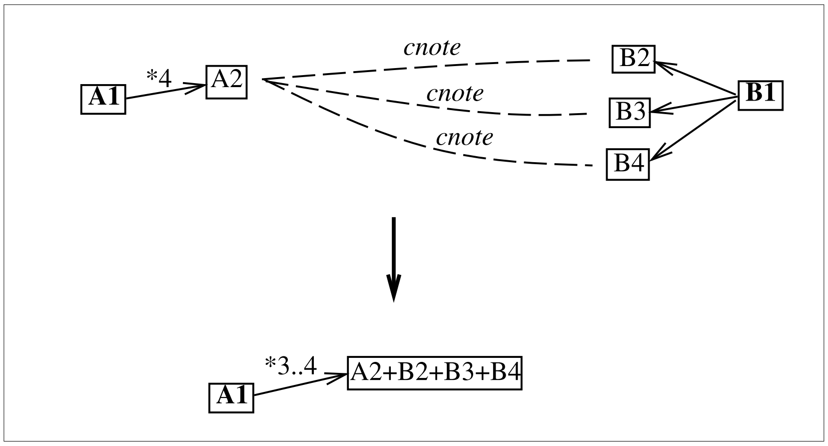


Figure 5.24: Multi-relationship ambiguity.

'Both-winning' correspondences can be generalised individually.

Sometimes in a situation of similar-similarity ambiguity there may be some relationship/relatee correspondences that are winning correspondences for both of the two objects involved, even though they are only marginal winners. In such a case it is reasonable to generalise each of these pairs of relationship/relatees individually and include the generalisations explicitly in the resulting generalisation, in addition to the multi-relationship.

For example, the top hinges of the two doors in Figure 5.25 match each other better than any of the other hinges, and so their generalisation can be included as relatees of the generalised door, and likewise for the bottom two hinges.

One of the effects of allowing ‘both-winning’ correspondences to be generalised even when they are only marginal winners, is that atypical relatees will be retained in the generalisation, without requiring any special-purpose mechanism for dealing with atypical members explicitly. The ends of ‘chain’ groups, such as the endmost books in a bookshelf, are examples of this. There is no need for methods such as using the FST (first) and MST (most) generalisation operation employed by Michalski [Michalski, 1983].

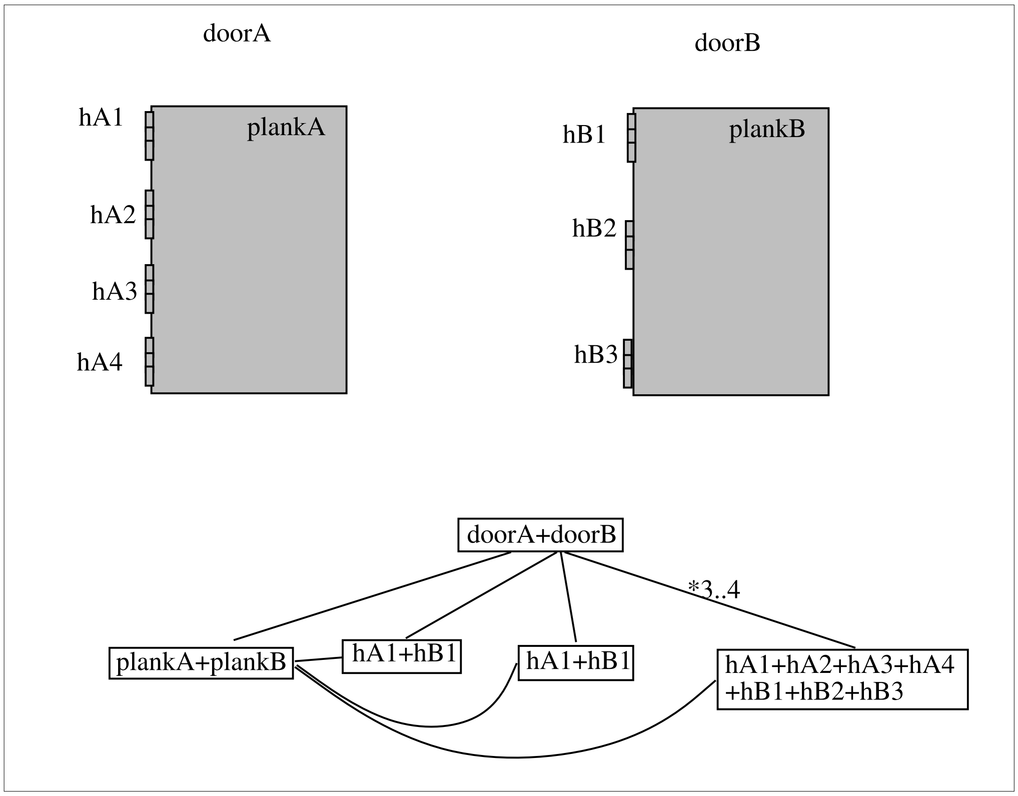


Figure 5.25: Multi-relationship ambiguity.

5.8.2 Local and global ambiguity.

Ambiguity can also be characterised as *local* and/or *global* ambiguity. Local ambiguity refers to ambiguity amongst parent, neighbour, and subpart relationship/relatee correspondences for a particular cnote. Global ambiguity refers to the ambiguity amongst all cnotes generated during the match process.

For example, suppose the matcher has matched *roomA* and *roomB* in Figure 5.26, and in doing so it has matched all of the potplants in *roomA* with all or most of the potplants in *roomB*. If we consider the correspondence between the two desks, there is local ambiguity (with respect to the desks) between the potplant correspondences, since either of the two potplants on top of *deskA* could be corresponded with the potplant on *deskB*. The potplants that are not directly related to the desks are excluded from the consideration of local ambiguity. (The potplant to the left of *deskA* can be considered unmatched, unless an explicit neighbour relationship between *deskB* and *ppA2* was formed.) Local ambiguity of the potplants with respect to the desks is based not only on the similarity scores for the potplants, but also the relationships from the desks to the potplants. Therefore, local ambiguity resolution would choose to merge only the potplants that are on top of the desks. It would produce a concept which could be interpreted as "the kind of potplant typically found on desks". The generalised desk would have a multi-relationship to this concept with a *howmany* count of 1..2.

Global ambiguity of the potplants, on the other hand, is not associated with any particular correspondence between the room components, but instead refers to the ambiguity between all potplant cnotes that were generated when comparing the rooms. Therefore, global ambiguity resolution could deal with all pairings of potplants, including ppA1, ppA4, ppB1, and ppB2, which might not even be noticed by local ambiguity (except perhaps with respect to the *floor* correspondence). Although it could also notice the ambiguity between the potplants on the desks, it would not place any extra significance on the relationships to the potplants from the desks. This relationship would just be treated as one feature of the context description of the potplants. Therefore, global ambiguity resolution is able to initiate the creation of larger and (from a global perspective) more complete groups of objects than would local ambiguity resolution. In the pot-plant example, it could invoke the group-constructor to produce a single generalisation of all 7 potplants. The generalised desk could then have a multi-relationship to this concept with a *howmany* count of 1..2, although it might also have a multi-relationship to the more specialised concept created locally with respect to the desk correspondence.

Since global ambiguity is not associated with a particular cnote, its resolution is the responsibility of the larger learning system, which must take into account the AKO hierarchy as a whole when determining what new concepts, groups, and multi-relationships to create, and how to reorganise the AKO hierarchy if necessary. Therefore, it has not been addressed fully in this thesis. Currently GRAM only performs local ambiguity resolution. In a future version of GRAM, local and global ambiguity resolution should be integrated, since it is not usually appropriate to form groups and/or multi-relationships on the basis of local ambiguity alone. Rather, local ambiguity should be the trigger for invoking the global mechanism.

5.8.3 Vertical and horizontal AKO ambiguity.

A special form of global ambiguity is where an observed instance matches more than one concept down the same branch of the AKO hierarchy. For example, an observed chair may match the concept *seat*, *chair*, and *armchair*, as shown in Figure 5.27 (a). This kind of ambiguity, which can be called *vertical AKO ambiguity*, occurs for almost every classification, and clearly

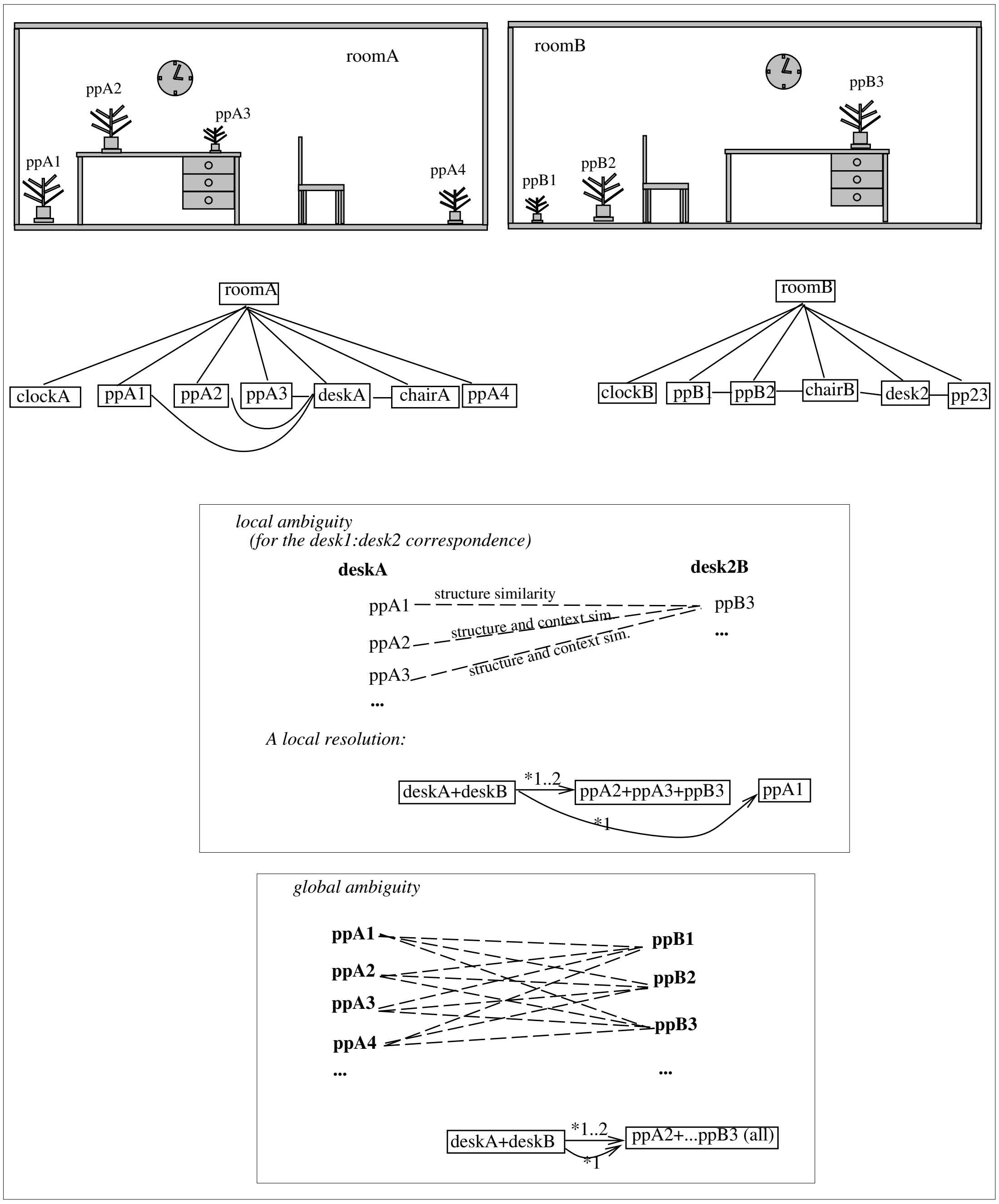


Figure 5.26: Local and global ambiguity.

should not be resolved by merging all of the concepts into a single concept (at least not usually). Rather, each should be generalised separately to cover the new object.

All other forms of ambiguity involving concepts in the AKO hierarchy can be called *horizontal AKO ambiguity*. For example, Figure 5.27 (b) shows a situation where a new object matches several subconcepts of the same superconcept equally well. This indicates that one of the subconcepts could be generalised to cover all the others, and these others could then be removed from memory, or made to be subconcepts of that subconcept. Alternatively, a new concept could be created which is a generalisation of, and a superconcept of, all of the ambiguously matched subconcepts.

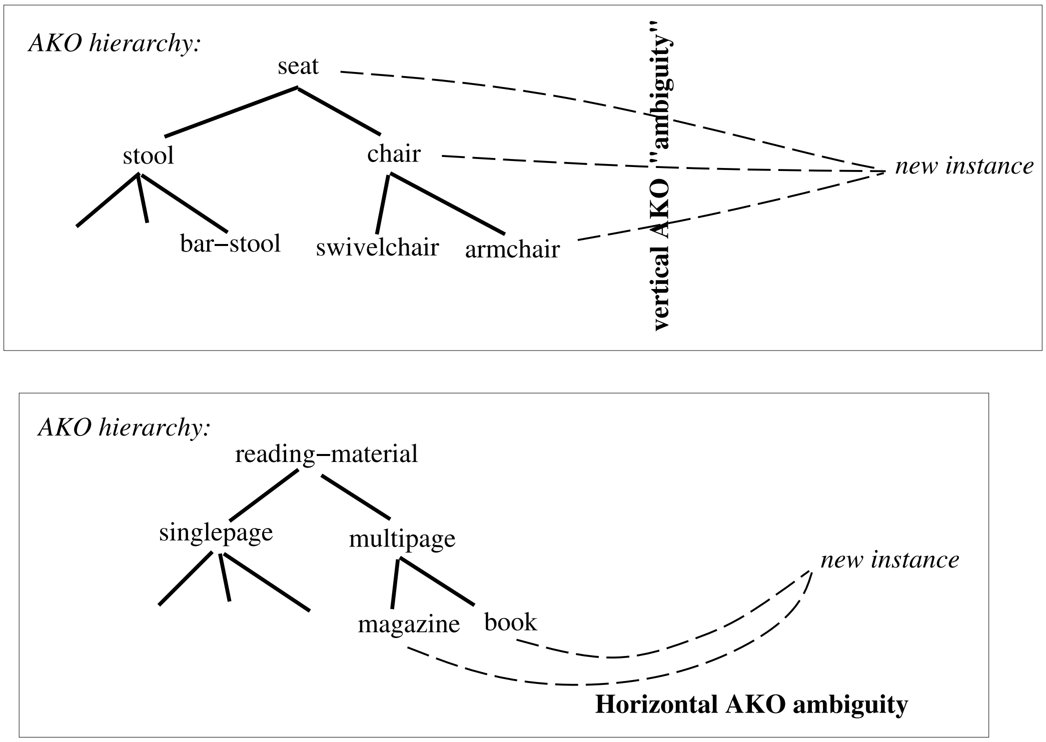


Figure 5.27: Vertical and horizontal AKO ambiguity.

5.9 Structure and context interpretation affects generalisation.

This section considers how the generaliser needs to behave differently for the different kinds of structure and context interpretation, such as *complete*, *partial*, *any*, *etc*. These were defined in the representation chapter, and a summary can be found at the end of section 3.4.3 on page 102.

Most of the examples in this chapter are '*complete+complete*' generalisations, where both of the two descriptions being generalised specify all permissible and required relationships and relatees. In these situations, any relationships that are only present in one of the descriptions are included in the new concept. One exception to this is when the two descriptions differ significantly in their structure or context so that a disjunction is created, in which case the new description only retains relationships that are common to both descriptions, since the non-shared relationships are specified in the disjuncts. The resulting generalisation therefore has a *partial* interpretation.

When generalising two descriptions of which one already has a *partial* interpretation, the generaliser operates slightly differently. In particular, if the generaliser is generalising a concept and an instance, and the concept has a *partial* interpretation and the instance has a relationship that has no acceptable correspondence in the concept description, then the relationship is ignored, and the resulting generalisation remains *partial*.

If, on the other hand, a relationship in the partial concept description does not have any acceptable correspondence with an instance relationship, then it is *not* dropped immediately, but instead is only dropped if the instance-count-ratio drops below some parameterised threshold value and if a sufficient number of instances have already been observed. For example, suppose that *X1* in Figure 5.28 is to be generalised to cover *Y1*. The *X1–X2* and *X1–X3* relationships match the *Y1–Y2* and *Y1–Y3* relationships respectively, so the instance-counts of the former relationships are incremented. The relationship of *X1* to *X4* is unmatched, but since it has a reasonably high instance-count-ratio it is retained in the description. However, if we suppose that dropping occurs if less than two-thirds of at least 18 instances have the relationship, then the unmatched relationship to *X5* must be dropped.

For the extreme value of the 'drop-relationship' parameter for which dropping occurs when an instance-count-ratio is less than 1 and at least two instances have been observed, then all concepts acquired by the system will either be *complete* descriptions with no optional relationships, or *partial* descriptions. This leads to simpler concepts, but the matcher must make greater use of the AKO hierarchy (by matching subconcepts) when classifying or finding faults in observed objects, since optional details of concepts will only be specified in their subconcept descriptions. This thesis does not state whether this is better or worse than having a high-tolerance threshold for optional components, but rather just points out the difference, so that the larger learning system can use either approach.

Complete descriptions can periodically be 'pruned' by a clean-up module (which has not been implemented) which checks for concepts that are defined in terms of a multitude of relationships mostly having low instance-count-ratios, and removes all but the high-instance-count-ratio relationships, changing the interpretation to *partial*, as illustrated in Figure 5.29.

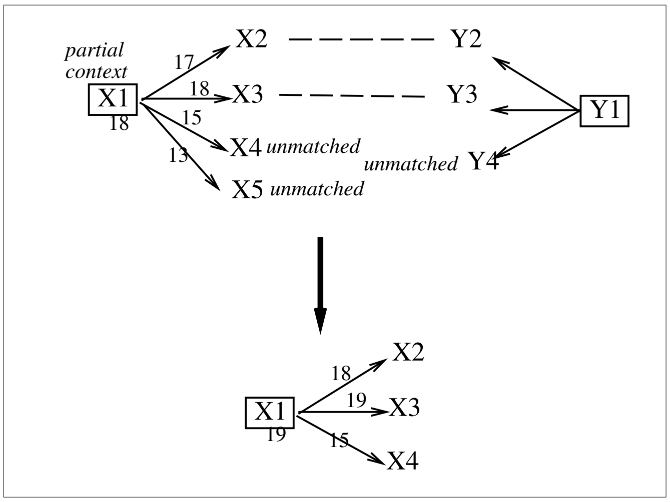


Figure 5.28: Dropping relationships from a partial description.

This not only saves memory, but also reduces the effort required by the matcher to process such descriptions. This may seem to lose special-case information that could be important for making predictions, such as, for example, losing the fact that a door might occasionally have a “Do not disturb” notice on it. However, such information will usually be retained in subconcept descriptions. For example, the concept *motel-room-door* could have an explicit optional relationship with the concept *do-not-disturb-notice*, while the superconcept *door* might not, due to the instance-count-ratio being too low.

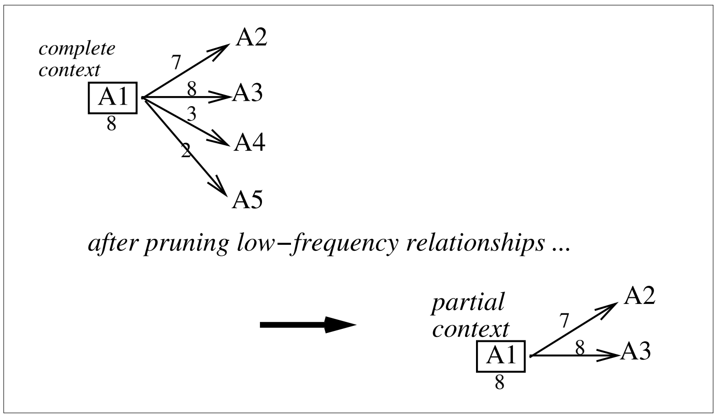


Figure 5.29: Cleaning up by removing low-frequency relationships.

Although it is not possible for a *partial* description to be changed back to a *complete* description (since details have already been lost) it may be changed to an *any* interpretation. This occurs when all or most of the relationships have become no longer common to most instances, and have thus been removed, either by the clean-up module described above, or during generalisation.

In a full robot system, functional knowledge would, of course, be important to help justify retaining or dropping relationships, but the simple threshold-cutoff method is the best that is possible for a syntactic system such as GRAM.

If a structure or context has a *disjunctive* interpretation, then the generaliser uses the disjunct correspondences produced by the matcher. If the winning correspondence are good enough, then the generaliser is recursively invoked to generalise the corresponding disjunct subconcepts. If a new concept is created, then this is added as a new disjunct (subconcept). Otherwise the original disjunct is modified. (There is no danger of generalising a disjunct with an instance twice, as could potentially occur if a concept has a structure *and* context disjunction, since the cnote describing the disjunct correspondence indicates whether generalisation has already been performed.) The same method is applied to structures and contexts with *import-from* specifications.

If the interpretation is *partial+disjunctive*, or *partial+typical*, or *partial+imported* then a combination of the methods discussed above is applied.

Chapter 6

The Instance Constructor

This chapter discusses the Instance Constructor, whose role is to produce a description of an observed scene or object. GRAM distinguishes between three stages of this 'perception' process. The first and most primitive stage is *block approximation* which involves identifying a set of simple blocks that characterise the scene at multiple levels of detail. The second stage is *object-graph formation* which involves creating a hierarchy of composite and primitive objects linked by parent, neighbour, and subpart relationships. The formation of objects may be based not only on the set of blocks, but also on other factors such as grouping, connectedness, and topology. The third and most abstract stage of perception is *classification* which involves classifying the objects in the object graph as instances of previously learned concepts. Classification may also enable the object-graph to be augmented further on the basis of the expected features of these concepts, especially if the instance was initially observed only partially or at a coarse level of detail.

The first stage, *block approximation*, is performed by a low-level vision system. This has not been developed in the thesis since there has already been considerable research done on identifying distinct two or three-dimensional blocks, pieces, or regions from images [Chin, 1988]. Therefore, GRAM merely assumes that a vision system is available which will produce a set of rectangular, elliptical, or polygonal shapes described with respect to a coordinate system. Various information is specified, such as the position of the center of each block, the dimensions and orientation of its rectangular bounding box, the shape type, and the number of edges.

The second stage, *object-graph formation*, is the main subject of this chapter. It describes various criteria that could be used to justify the formation of a composite object from a set of smaller objects and the selection of which parent, neighbour, and subpart relationships should be made explicit. It also discusses the kinds of mechanisms needed to find and create composite objects, in particular the *group-finding* mechanism. The process of finding and creating composite objects is called *object-formation*, and it is the subject of sections 6.1 and 6.2, the latter of which addresses the group-finding problem. The process of selecting parent, neighbour, and subpart relationships for each object is called *relationship-selection*, and is the subject of section 6.3. Only the group-finding and relationship-selection mechanisms have been implemented (to some degree) in the current GRAM system.

The third stage, *classification*, is discussed in chapter 1, and involves indexing and matching objects with concepts. Indexing is an area of future work since it requires mechanisms for building and maintaining concept-memory.

These three stages of perception and the three systems or processes responsible for them, as shown in Figure 6.1, are overlapping and interdependent. This is because perception must proceed not only in a bottom-up fashion where the results of the block-approximation vision system govern the results of the higher levels, but simultaneously in a top-down manner, where the higher levels drive the lower levels, such as when expectations based on previously learned concepts drive object-graph construction and even block-perception. Furthermore, the creation of *groups* in the description involves both the matcher and the generaliser to produce the typical-member concept, and hence the process of perception requires the inter-dependent participation of *all* components of the GRAM system. However, it is useful for purposes of discussion and system development to make a distinction between the three levels.

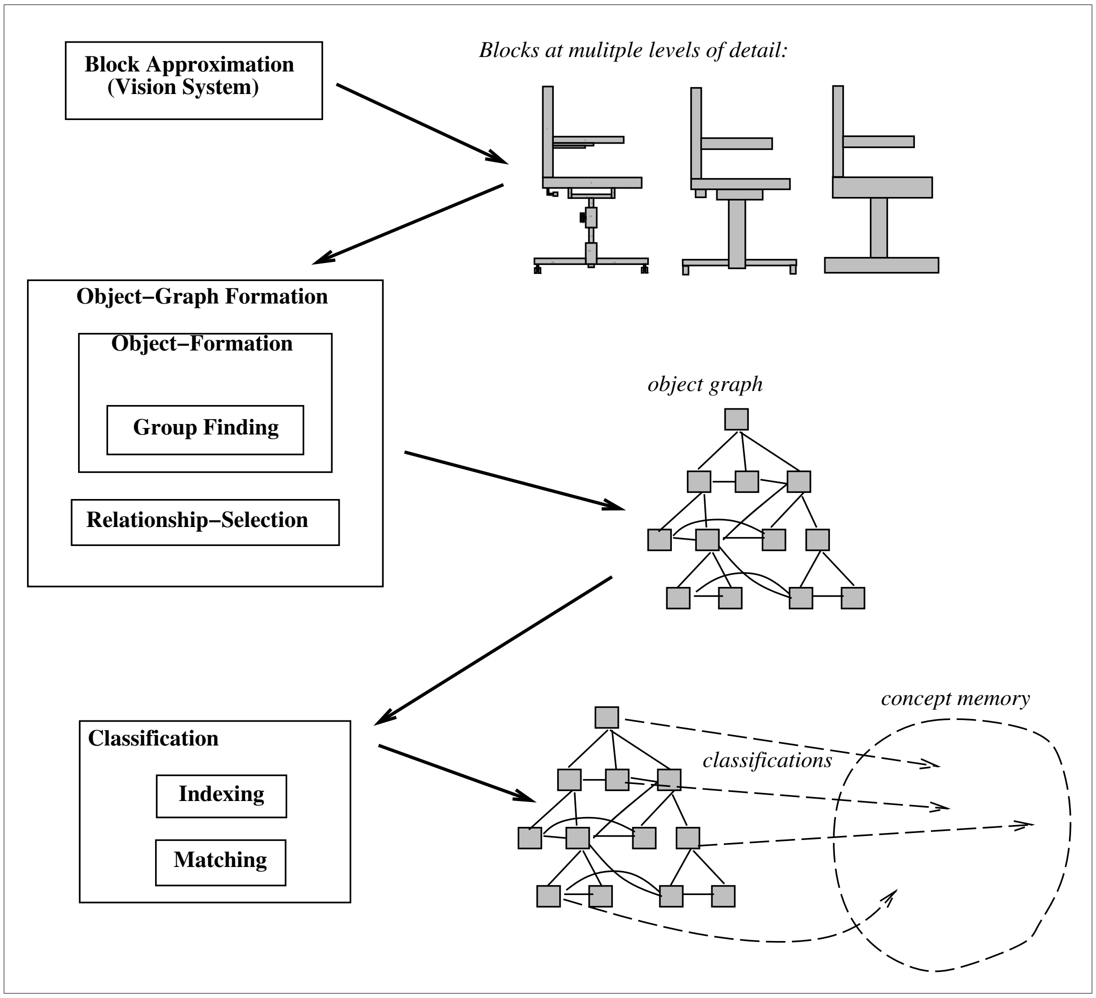


Figure 6.1: Stages of perception.

The most important aspect of the design of a perception system is the choice of *criteria* used to justify the selection and formation of various descriptive entities. The perception *mechanisms* are obviously important but are secondary and are really just ways of operationalising the criteria. Without first identifying the criteria (on the basis of overall domain and task requirements), the mechanisms cannot be justified. Therefore, the main contributions of this chapter are the sets of criteria for justifying object formation (especially grouped objects) and relationship selection. A secondary, although important contribution, is the group finding mechanism. The chapter distinguishes between two basic group-finding search strategies, "Seed-Expansion" and "Propose-and-Prune", of which the former is used by GRAM's group-finder.

6.1 Object-Formation

Chapter 3 discussed the reasons for representing a scene or object in terms of a hierarchy of parts at multiple levels of approximation and abstraction. Each part is either a composite or primitive object. A brief review of these reasons is given below, since they provide a basis for justifying the criteria and mechanisms of object-formation that are presented in this section. The reasons are as follows:

- An object can be recognised efficiently at a coarse level of detail by just considering a few abstract or approximate components and their relationships.
- Memory usage can be reduced by 'summarising' an object in terms of more abstract or approximate components or 'chunks'.
- Efficiency and effectiveness of the matcher is improved by allowing correspondences between coarse details to guide and constrain the search for correspondences at finer levels of detail.
- Two objects may be able to be generalised on the basis of corresponding abstract or approximate features, even if they differ significantly in their finer details. If
- Multiple levels of approximation and abstraction enable the fault-finder to report differences at both a coarse and fine level of detail, depending on task requirements.
- A composite object captures constraints and properties of a collection of several components as a whole, and these features might not otherwise be representable.
- GRAM creates concepts by generalising instance objects, and therefore, if multiple levels of abstraction were not supported, it would not be possible to create abstract concepts from composite instance objects.

On the basis of these reasons, this section presents various specific criteria which could be used to justify the formation of an object to characterise an observation. It also discusses the kinds of mechanisms needed to find and create objects. The group-finding mechanism has been partially implemented in the current GRAM system, and is discussed in detail in section 6.2.

In a full parallel system, each of the object formation mechanisms would independently search for candidate sets of composite of primitive objects that could be combined into a single composite object. If a set is sufficiently strongly supported by one or several of the mechanisms, then a composite object would be created. New objects might then be created on the basis of these, and then again from those objects, repeating until no further composition can be done.

6.1.1 Object-Formation Criteria

There are many criteria that could be used to justify the formation of composite objects that partition an object or scene into larger chunks. A factor that underlies all of them is that there should be a clear boundary between the components of a composite object and non-components. More specifically, there should be a clear distinction between the collective and/or individual features of components and non-components. If this were not the case, then the boundary would be arbitrary and descriptions would not be sufficiently consistent to support effective matching and generalisation.

Figure 6.2 lists the criteria that are or could be used in the GRAM system, and also shows simple illustrations of their meaning. These are discussed below. Each of these criteria provides a way of identifying useful boundaries or partitions between sets of components within an observed object or scene. Some of these criteria are similar to the Gestalt "Laws of Perceptual Organisation" which were proposed fifty years ago [Ellis, 1939], but the criteria presented here are more specific, since we are concerned with actually building a real perception system.

The discussion also mention the various kinds of object-formation mechanisms that are or could be based on each of them.

(a) Blockness

The simplest and most important form of composite-object-formation is based directly on the blocks produced by the low-level vision system. Each block is an approximation of some portion of the observed object at some level of detail. If a block has a clear and simple boundary then this justifies the formation of a composite object. Primitive objects are created from blocks that are not decomposable into any finer level of detail.

For example, Figure 6.2 (a) shows three objects with portions of their substructure marked by dotted boxes. In the left two objects, the marked regions indicate strong blocks which should be perceived by the vision system, since their boundaries are clear and simple. In the rightmost object, the marked region should not be perceived as a block by the vision system, since its boundary does not form a simple shape that distinguishes it from the rest of the object. If you make your eyes go out of focus, it does not stand out as a distinct component.

'Holes' are perceived by the vision system in much the same was as ordinary solid objects, since GRAM represents a hole as an ordinary object with a *density* property of zero, and a *block-type* property with the value 'hole'. Therefore, holes can be combined into composite 'hole' objects in the same way as solid objects. The middle object in Figure 6.2 (a) includes a primitive hole.

A real vision system would require edge-detectors and so forth, and could perceive fuzzy blocks directly by detecting edges at a coarse level of detail, without having to first detect the smaller blocks of which it is composed. However, fuzzy blocks could also be detected in a bottom-up fashion by clustering groups of smaller blocks into larger blocks. For development purposes, a simple system was implemented to do this which, in effect, observes a scene at a number of different levels of detail, each defined by some tolerance factor. For each level of

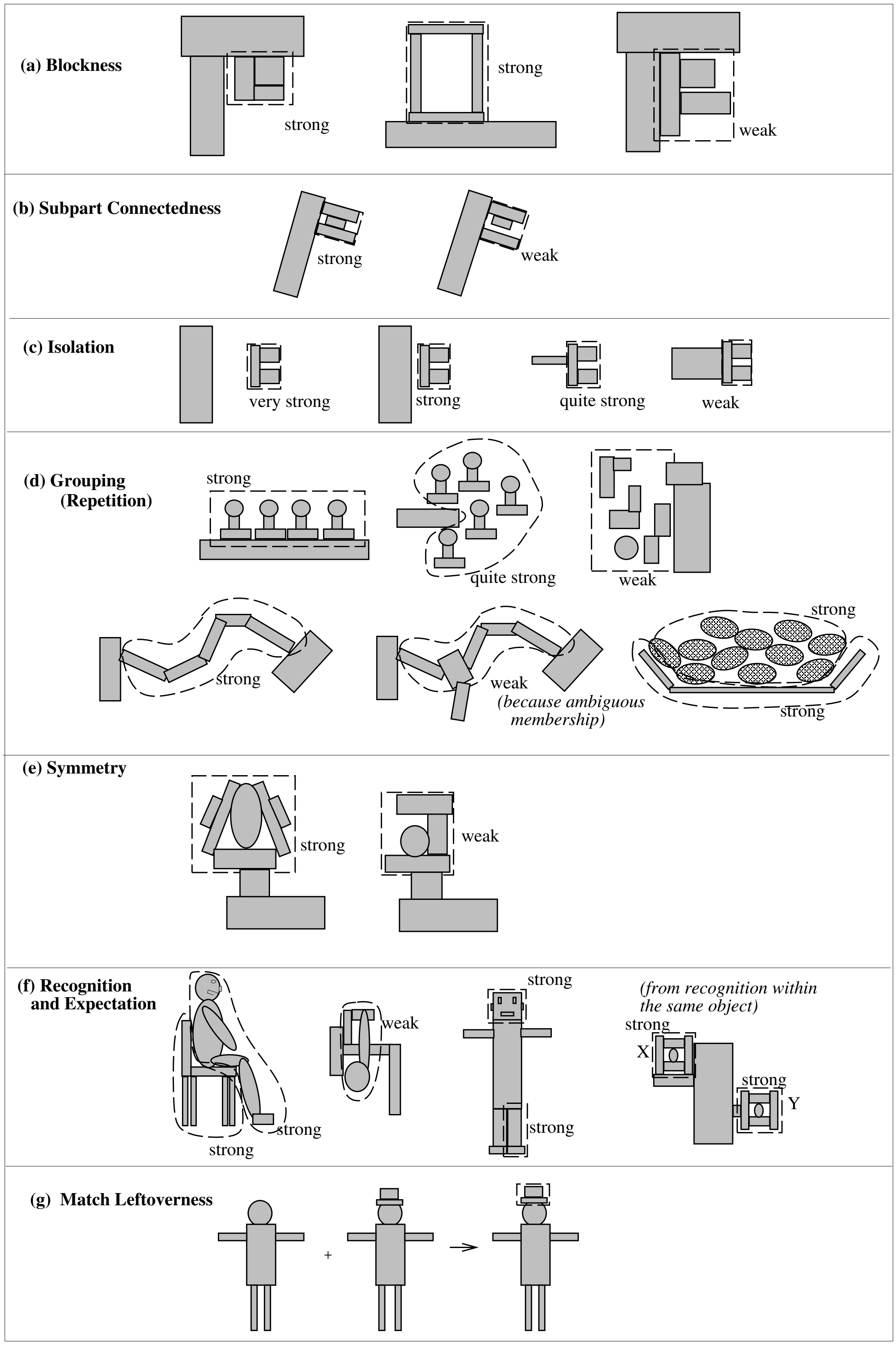


Figure 6.2: Partitioning Criteria.

detail, it merges blocks if the empty space within their rectangular bounding box is smaller than the tolerance factor defining that level. Thus a series of 'views' of the object is obtained, each representing the object at a particular level of detail. An example of a series of views produced for a chair is given in Figure 6.3. However, composite objects should not necessarily be created for all of the blocks. Rather, objects are only created for those blocks that persist through several levels of detail, and thus can be considered 'stable'. Stable blocks are most likely to be perceivable in other similar instances, and are therefore more important to represent explicitly to support matching and generalisation. However, weaker blocks might also be used if they satisfy the other object-formation criteria discussed below.

Unfortunately the above system is limited in that it deals only with rectangular blocks that are all aligned with the axes of the coordinate system, as in Figure 6.3. Therefore, since a real low-level vision system has not been available, most of the work on GRAM has used input obtained directly from a simple graphics package [1] in the form of a set of primitive shapes, such as rectangles, ellipses, and polygons, with blocks indicated explicitly by the teacher.

(b) Subpart Connectedness

One of the simplest criterion for object-formation is connectivity. If a set of objects are *not* connected, then they are less likely to be functionally or structurally dependent on each other, since their relative position and orientation are not subject to direct structural constraints. Therefore it is less appropriate to combine them into a single composite object than if they were connected.

For example, Figure 6.2 (b) shows two proposed composite objects (indicated by the dotted boxes), where the first satisfies the subpart-connectedness criterion, and the second is not. The composite objects shown here might, of course, be still justifiable on the basis of other criteria.

Currently GRAM only considers parts to be either connected or not connected. An extension to GRAM's representation could include more types of connection which would give varying degrees of justification for object-formation, such as fixed joins, articulated joins, or mere contact. In fact, several blocks perceived by the vision system might even be portions of the same piece of material, such as the base, stem and the bowl of a wine glass. This form of connection can be called a *same-piece* connection. Same-piece or fixed-joins are the strongest justification for combining objects into the same composite object, since they imply the strongest functional inter-dependence and structural constraint. On the other hand, the weak *contact* connection between a person and a chair does not strongly justify the formation of an object consisting of both the person and the chair, although other object-formation criteria, such as blockness and isolation (discussed below) might.

Often it is not possible to know whether the connection between two objects is *same-piece*, *fixed-join*, or *contact*, unless they are observed to move in relation to each other, or unless prior domain knowledge is available. This thesis suggests that in the absence of such information, the system should presume that the objects are *fixed-joins*, since composite objects formed

[1] *Idraw* for X-windows on a Unix system, which is a demonstration application for the Interviews project.

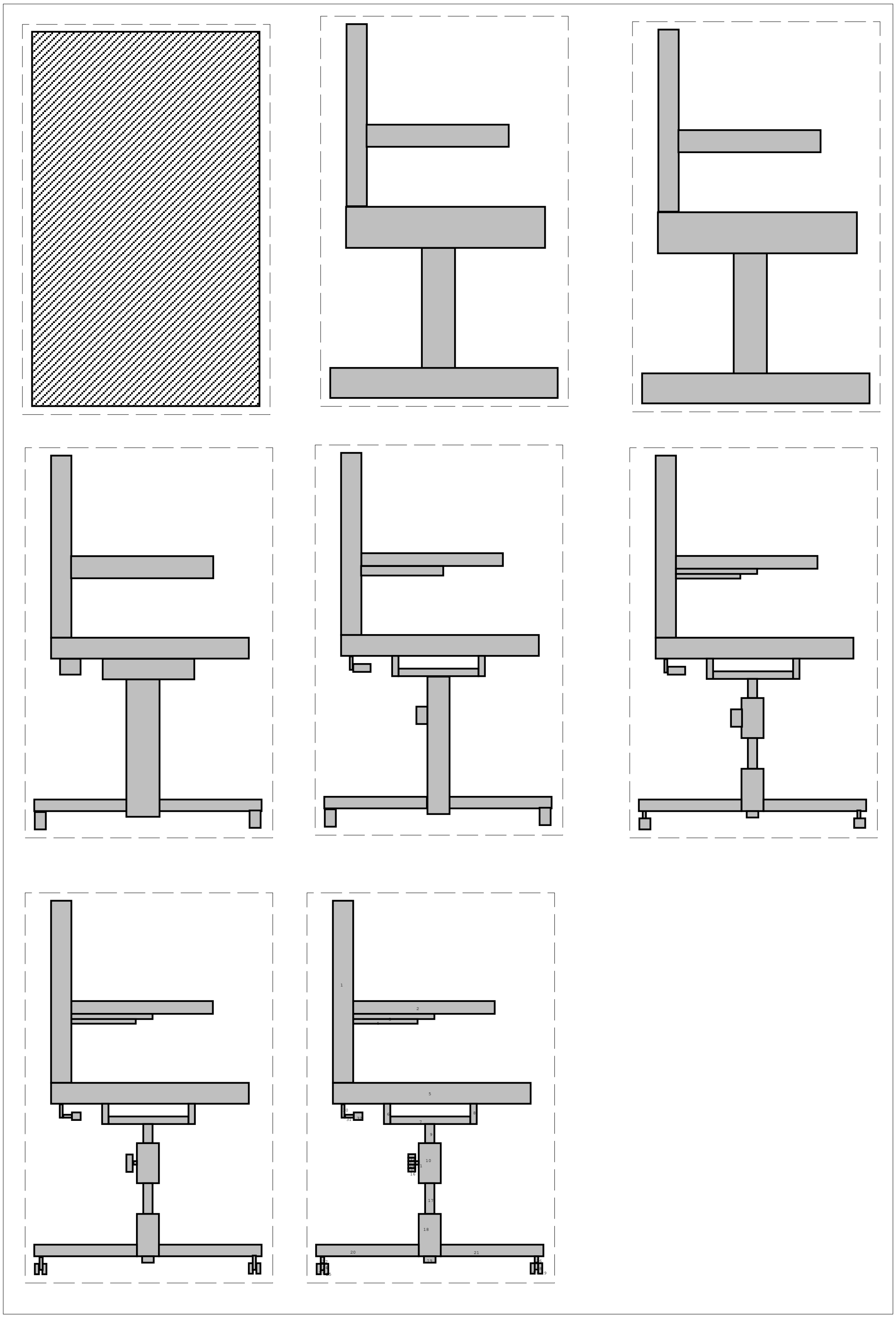


Figure 6.3: A series of ‘block’ views at different levels of detail.

incorrectly on the basis of this can always be removed later from concept memory, while it is more difficult or impossible to later create composite objects from already-generalised objects in order to resolve mismatches. In other words, it is better to err on the side of excess redundant descriptive entities.

Currently GRAM does not use this criterion, since its input comes directly from the teacher. However, this criterion would be easy to implement by computing the maximum distance between pairs of subblocks of the blocks provided by the vision system.

(c) Isolation

Th *isolation* criterion is closely related to the previous criterion. It causes the instance constructor to favour object-formation for candidate composite objects which are distant from, or at least separated from or very weakly connected to, other neighbouring objects, where 'weakly connected' means that only a small area of the surfaces or edges are involved in the connection, such as a chair on a floor. For example, Figure 6.2 (b) shows three objects ranging from strong to weak isolation.

As with the previous criterion, this is not currently used by GRAM, but could be easily implemented by computing the minimum distance between the candidate object and other objects, or, if the object is connected to other objects, by computing the degree of connectivity.

(d) Grouping (or repetition)

The grouping criterion justifies the formation of a composite object from a set of similar and similarly related objects. A strong group is one in which there is a clear distinction between members and non-members, and where the members have strong similarity of structure, context and inter-member relatedness. Groups might be an unstructured *cluster*, or a linearly ordered *chain*, or any of the other kinds of groupings such as *array*, *loop*, or *collection*. Groups have already been discussed in some detail in the representation chapter, and section 6.2 describes specific kinds of grouping criteria and mechanisms for finding groups.

(e) Symmetry

Symmetry of a set of objects or blocks is another justification for object-formation, since it suggests a collective dependence between the objects involved. The most common form of symmetry is when a set of objects are co-linear, such as the base, stem and bowl of a wineglass, or the components along a drive shaft. A chair is a more complex symmetrical object, and this symmetry could help to distinguish a chair as a distinct object prior to recognition within a crowded room. Figure 6.2 (d) shows two artificial composite objects, one with strong symmetry and the other with weak symmetry. Brady's "Smoothed Local Symmetries" system [Connell and Brady, 1987] also used this principal for identifying the important regions of an image.

One simple way to test for symmetry (introduced by [Winston, 1975]) is to match the set of subpart objects of a proposed composite object against itself, after first performing a mirror-image transformation of their parent, neighbour, and subpart relationships. This is somewhat expensive computationally, so to avoid doing this for all possible sets of objects, it could be done only for composite objects that have already been proposed (at least weakly) on the basis of other criteria, and whose overall shape and density profile is also symmetrical, which can be determined by much simpler computation. The GRAM matcher has not yet been applied to this task.

(f) Recognition and Expectation

If a sub-component of an object or scene is recognised as being an instance of a known concept, then that may cause the system to expect to see instances of the concept's parent, neighbour, or subpart concepts, and this may give justification for creating new composite objects.

For example, in Figure 6.2 (f), the *person* and the *chair* might not initially be identified as being distinct objects on the basis of other object-formation criteria. However, the *head* is a distinct block, and could be recognised. Since a head is expected to be attached to a torso, and close to an arm, these could also be recognised. From these, other components could be recognised. These classifications would provide sufficient justification to create a new composite object in the instance graph, consisting of the classified person components.

This demonstrates how object-formation cannot always be completed before classification, since the two processes are inter-dependent.

In the second example in the figure, the indicated composite object is not justified on the basis of such recognition.

In the third example, the formation of the legs as distinct objects is clearly not justified on the basis of any of the criteria discussed previously. However, the humanoid as a whole could be recognised via recognition of the head and face structure, in the manner described above, and on this basis the legs could be recognised by expectation, and this justifies formation of distinct objects in the instance graph.

The fourth example is closely related to the repetition, or grouping, criterion, since the formation of the composite part X indicated on the left side is justified because it matches the composite part Y on the right side. If we assume that Y was created on the basis of other criteria, and if we assume that every instance object can be immediately treated as a concept in its own right, then part X on the left will be recognised as an 'instance' of it. This also assumes that X has either been already proposed (weakly) as a distinct object on the basis of blockness and/or symmetry, or is formed on the basis of expectation after recognising one of its subcomponents, such as the ellipse in the middle, as matching one of Y's subcomponents.

Chapter 2 described the Labyrinth system which employs a bottom-up recognition process to classify primitive parts so that it can classify composite parts composed of them, and then continue up the part hierarchy until the object as a whole can be classified based on the classifications of its direct subparts. A limitation of this forward-chaining-style approach is that it assumes that subpart object-formation has already been done prior to classification of the parent part. With the recognition-expectation criterion, GRAM can not only do bottom-up recognition, but can also use a classification of a parent part to guide the formation of its subparts, using a more backward-chaining expectation-driven approach. For example, a bicycle might be recognised on the basis of its overall shape and properties, and this could guide the formation (*i.e.* perception) of its subparts.

(g) Match Leftoverness

In Figure 6.2 (g) we see a person without a hat and a person with a hat. If the system does not know the concept *person*, then there is no justification for treating the hat as being distinct from the head. However, when matching the hatless person with the hatted person there are two unmatched objects, and this can be used to suggest a composite object comprised of them, since their 50% absence and 50% presence indicates a co-dependency and a structural distinctness from the head. Other criteria such as connectivity need to be combined with this criterion, and since the two hat parts *are* connected, and are also aligned along their central axes, they can be combined into a single object in the generalised person, as shown in the figure.

This criterion does not really belong in the instance constructor, since it is a task of the matcher (as is also the case for the recognition-expectation criterion). However, since it pertains to object formation, it is included in this discussion. Match-leftoverness has not yet been implemented, since the current matcher is not able to cope with new composite objects being formed during the matching process itself.

(h) Function, behaviour, and knowledge about construction.

A full robot system would also partition a scene into composite objects on the basis of knowledge and reasoning about function and behaviour, and also of knowledge and reasoning about construction, whether by human beings or by other parts of nature. For example:

- If a chair has a rubbish bin sitting on it, then the system might treat the chair and rubbish bin as a single object, possibly creating a new variety of chair concept. However, if knowledge about the function of a chair is available, then the system will know that the rubbish bin is a separate object.
- If the system observes a bicycle and sees that the pedals are moving in relation to other parts of the bicycle, while their internal structure remains unchanged, this indicates that the pedals are distinct objects. Likewise, if the concept *chair* was not previously known by the system, and it then saw a chair leaning against a desk, it might think that the chair and desk were one object. However, as soon as the chair is moved relative to the desk (or even just when it is observed separately from the desk) the distinction is clear.
- If a system observes a bicycle for the first time, but knows something about physics and how objects are constructed, it might be able to guess that the frame is one piece of material, or perhaps several very strongly connected pieces, and therefore can be treated as a single composite object.

Function, behaviour, and construction knowledge is far beyond the scope of this thesis, and is only mentioned here for completeness.

6.2 Group Finding

Group finding involves finding collections of component objects within a scene or object that are sufficiently similar that they can be represented in summary form as a single object with a multi-relationship to a *typical-member* concept.

This section considers two aspects of the group-finding process. Firstly, and most importantly, it looks at various kinds of criteria that can be used to justify group formation. Secondly, it describes two mechanisms, "Seed-Expansion" and "Propose-and-Prune".

The reasons why grouping is important were discussed in chapter 3, but it is worth reviewing these briefly in order to provide motivation and justification for GRAM's group-finding criteria. These reasons, listed below, are refinements of the general reasons discussed in section 6.1 for representing objects at multiple levels of detail.

To improve match efficiency: Two groups can be matched as single entities, thus avoiding the need to match each of their individual members.

To enable generalisation: Grouping enables two collections of similar parts to be put into correspondence and generalised (as single entities), even if they have different cardinalities. This results in a generalisation that is defined in terms of a *variable* number of instances of the generalised typical-member concept.

To reduce memory usage: Since a collection of similar parts can be summarised in terms of a generalised description of the typical member, often the descriptions of individual member subparts can be removed, thus reducing memory requirements.

For constraint discovery and representation: A group is really an n-ary similarity relationship between several items, and therefore group finding can be considered to be a way of explicitly noticing and representing important regularities and constraints between a collection of objects. The kinds of regularities that the group-finder notices should be those that are likely to have functional or structural significance, indicating that the grouping has a common or collective function, or an underlying unifying cause or constraint.

As a form of concept acquisition: Since the process of grouping involves noticing repetition of similar items and forming a generalisation of them, group formation is a process of concept acquisition within a single scene (as opposed to concept acquisition from a series of scenes observed at different times). Concept acquisition enables predictions to be made about similar objects observed in the future, or elsewhere in the scene. This also means that individual members (if retained in the description) are being implicitly generalised, since they are being classified as instances of the typical-member concept, and thus future matching with individual members will be more tolerant of differences that are acceptable to the implicitly inherited typical-member features.

It should be noted that GRAM's group-finder is not intended to model the way humans find groups, but only to support the above requirements. However, since the system is intended to operate in a human world, the kinds of descriptions it produces should contain at least roughly the same kinds of information that humans seem to consider important. We know so little about this matter that the requirement must be treated somewhat loosely.

A typical-member concept may be formed without a grouped object.

Sometimes it may be inappropriate for a proposed grouping to be represented as a single new object, even though the grouping is sufficiently strong to justify generalising the objects into a new concept. If this is the case, then other objects can be related to the concept via multi-relationships, but cannot be explicitly related to the a group as a whole since it is not represented. This situation arises when the grouping is an unstructured *cluster* or *collection*, and for which the grouping as a whole does not satisfy the other object-formation criteria, particularly 'blockness'.

6.2.1 Grouping Criteria

Section 3.5 described a variety of types of groups, such as *chains*, *clusters*, *arrays*, *etc*, each of which was distinguished by the kinds of features that were common to the members. We now consider in more detail the kinds of criteria used to actually discover and justify such groups. All of these criteria are based on the overall reasons for grouping given above, such that if a proposed group of objects satisfies enough of these criteria strongly, then the group can be considered a strong group, and therefore one worth representing explicitly.

Before describing GRAM's grouping criteria, the kinds of criteria and group-finding mechanisms proposed by Winston [Winston, 1975] are outlined, since his early system has been a motivating factor in the work on group-finding in this thesis.

A key idea proposed by Winston is that all members of a group should have "equal right to membership". On this basis, he used the specific requirement that each member of a stable group should have at least 80% of the features that are common to more than half of the members. His 'common-features' group-finding algorithm worked by proposing a generous grouping, and then pruning out atypical members – those that do not satisfy the above requirement.

However, Winston also suggested that there cannot be just one universal group-finding mechanism, but many, each based on different demands, such as for finding collections of items that fit together like jig-saw pieces, or groups based on a single common property (such as all red objects in the room), or groups based on overall properties which do not characterise individual members but the group as a whole (such as its overall shape).

Winston also discussed sequences (equivalent to GRAM's *chains*) as the most simple form of group. The criteria for a sequence as outlined briefly in his paper were that a sequence must have at least three members which are linked, in sequence, by the same relation, such as SUPPORTED-BY or IN-FRONT-OF. A sequence also must not have junction points, and the consecutive members must not dramatically change in size or relative position.

One of the (perhaps intentional) problems with Winston's group-finding system was that the sequence-finder and 'common-features' mechanism were distinct processes, each using very different membership criteria. The sequence-finder did not incorporate the '80%' rule, and conversely the 'common-features' mechanism did not take into account relationships *between* the members. Therefore, these two grouping criteria could not *both* contribute to the formation of a group.

In GRAM, these mechanisms have been integrated more closely as a result of first elaborating and refining the criteria proposed by Winston, and only then considering what grouping mechanisms are needed. This approach follows the overall methodology of this thesis of identifying criteria before mechanisms. Whether a single general-purpose grouping mechanism can be developed to account for all of the grouping criteria, or whether multiple distinct and perhaps cooperative mechanisms are preferable, is a secondary issue.

We will now consider the various specific grouping criteria in more detail. The discussion often refers to the bookshelf in Figure 6.4, as this illustrates many different kinds of groupings. The grouping criteria are summarised in the list below.

- Strong similarity of structure, context, and inter-member relationships.
- Clear membership boundary.
- 'Tightness' of inter-member relationships.
- Parent non-groupedness.
- Cardinality.
- Structure complexity of members.

Members should be similar to each other with respect to structure, context, or inter-member relationships.

The most important and obvious criteria for grouping is that the members are similar to each other. The more similar they are, the stronger the group. Three kinds of similarity can be distinguished as follows:

Structure similarity is the similarity of the members' structure in isolation, independent of their context. External *context* similarity takes into account the relationships between members and non-members, such as the relationship between each of the books on a shelf, and the shelf. *Inter-member relationship* similarity takes into account the relationships between members themselves.

For example, each of the two stacks of bricks supporting the shelves satisfies structure similarity and inter-member similarity, since the bricks are structurally identical and they are organised as a linear (and almost connected) sequence. The context of the bricks varies, however.

As a another example, the bottles on the bookshelf all have very similar structure, moderately similar context in that they all sit on a shelf, but their inter-member relationships are not regular.

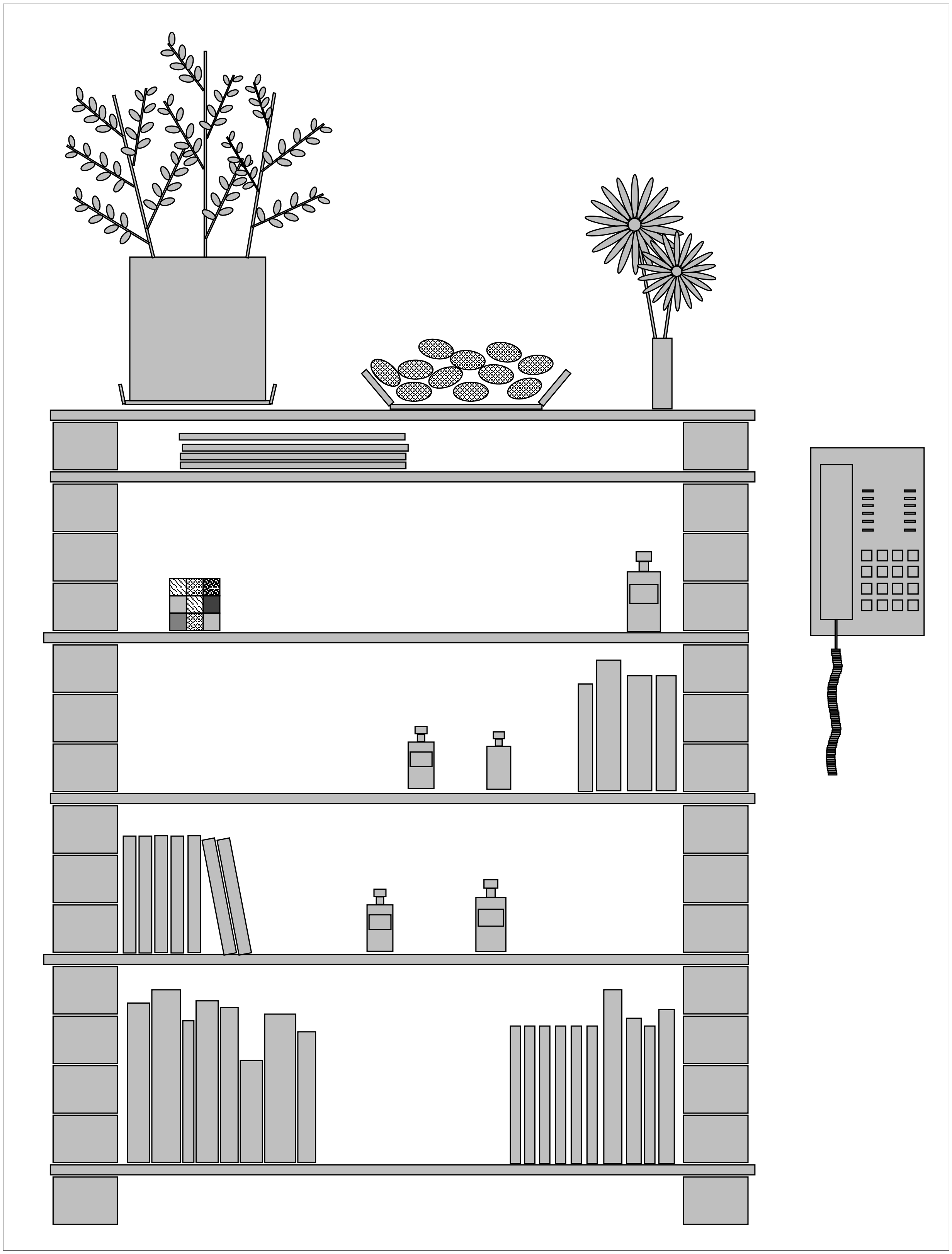


Figure 6.4: A bookshelf containing groups.

On the basis of this, the bottles are clearly generalisable to form a new concept, and multi-relationships could be created from the shelves to the *bottle* concept, but it is less justifiable to represent the bottles as a single group object, since such a group does not have any overall structural regularity or blockness.

If inter-member relationships form an ordered sequence, such as for the row of books or stack of bricks, then this justifies a *chain* grouping, which is considered an especially strong form of grouping since orderedness is often a functionally significant feature. On the other hand, the cookies in the bowl on the bookshelf do not form an ordered sequence, but each cookie is related to at least one other cookie by contact. In an *array* grouping, such as the windows of a building, or squares on a chequer board, or the blocks in the Rubic's cube on the bookshelf, the relationships between members are more regular although they do not form a single linear sequence. Rather, there are two different kinds of inter-member relationships, one vertical, one horizontal pairs.

This introduces the problem of deciding *which* inter-member relationships should be considered when evaluating and creating a group. The simplest method is to consider only the most tightly related (*i.e.* the closest, most aligned, *etc*) neighbours of each member object. In a *chain*, the tightest relationships for each member will normally be the relationships with the next and the previous members in the chain. In a *cluster*, there may be several equally tightly related neighbours. In an *array* there will be four equally tight relationships to the members to the left, to the right, above, and below.

In Winston's system, structure and context were not distinguished, and member similarity was measured in terms of the number of common relationships such as (ON-TOP-OF X) or (HAS-PROPERTY green). The representation was much simpler than GRAM's, and group members were single blocks rather than potentially very complex structures, and so it was simpler to match the proposed group members with each other. His system formed a "common features list" for the proposed group, and the size of this list indicated the strength of the group. Individual candidate members could be compared with this list to determine their atypicality. In GRAM, the equivalent of the "common features list" is a structural generalisation of the members (*i.e.* a concept), and the strength of the grouping is indicated by the *variance* of the concept, since that indicates the regularity of the members. The lower the variance, the stronger the group.

A group should have a clear boundary between members and non-members.

Similarity is not, however, a sufficient criteria on its own for justifying a group. For example, four of the apples in a large bowl of thirteen apples may be very similar to each other but should not be represented as a group. Another criteria is required, namely that there is a clear *boundary* between members and non-members. This does not mean that the members should appear in a distinct spatial region of the scene (although this may be one aspect of it) but rather that there are no non-members that have as equal a right to membership as the members. This is illustrated in Figure 6.5, showing two groupings of apples, one with a strong membership boundary and one with a weak boundary.

Winston's 'common-features' group finding system did not make this criteria explicit because the system worked by proposing an initial overly generous group and then pruned it, and so the clear-boundary criteria was implicitly satisfied. His sequence-finder *did* explicitly embody this criteria however, by preventing a sequence from containing members with a sudden change of size or relative position. Such sudden changes indicate the boundary.

The requirement of a clear-boundary ensures that members are not being missed out. A clear boundary also means that the group is more likely to be able to be matched with other instances of the same concept. If there is a fuzzy boundary, then other instances might include more or fewer items as members of the corresponding group, and this would not be matched as successfully. Such a group is therefore less useful for supporting the matching and generalising process.

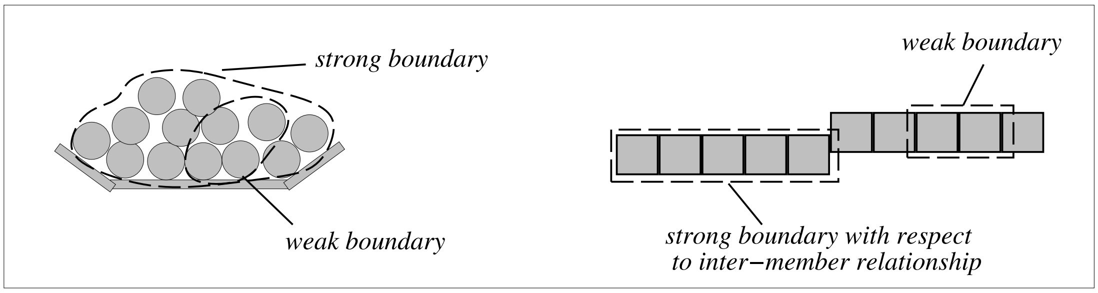


Figure 6.5: Strong and weak group boundaries.

The strength of the boundary between members and non-members must be based on the variance of the generalisation of the members, which is a measure of its regularity. More specifically, the difference between a non-member and the typical-member concept is measured as a ratio of the variance, or in other words as a *fit-scoring* comparison. Thus, if a grouping is highly regular, then even if a non-member is only a little different from the members, the group boundary would be considered strong if this difference were significantly larger than the variance. For example, the two groups of petals (one group for each flower) have reasonably clear boundaries, even though some of the petals overlap. This is because the petals in each group have highly regular inter-member relationships for a large number of instances, and also because they collectively form a circular block. So the overlapping petals from the other group clearly do not belong.

The interestingness or 'tightness' of inter-member relationships increases the group strength.

The discussion above has implied that if group members have very similar structure, non-member relationships, and inter-member relationships, and if the group has a clear boundary, then it is a strong group. However, this is not quite true because it is also important that the relationships are 'interesting', meaning that they are more likely to have functional or behavioural significance in the physical domain, or in other words, capture important constraints. Since functional knowledge is not available, the term *tightness* is more specifically what is required in GRAM's structural domain. The tightness of a relationship is measured in terms of proximity, alignment, and 'visibility' (meaning that the two objects do not have other objects between them). For example, in Figure 6.6 (a) the ellipses 4,5,6, and 7 clearly form an 'interesting' group because not only do the members have similar structure and inter-member relationships, but the inter-member relationships are tight. In (b), on the other hand, the objects 4,5,6, and 7 also have similar structure and inter-member relationships, but these relationships are less tight, and so the group is considered less interesting.

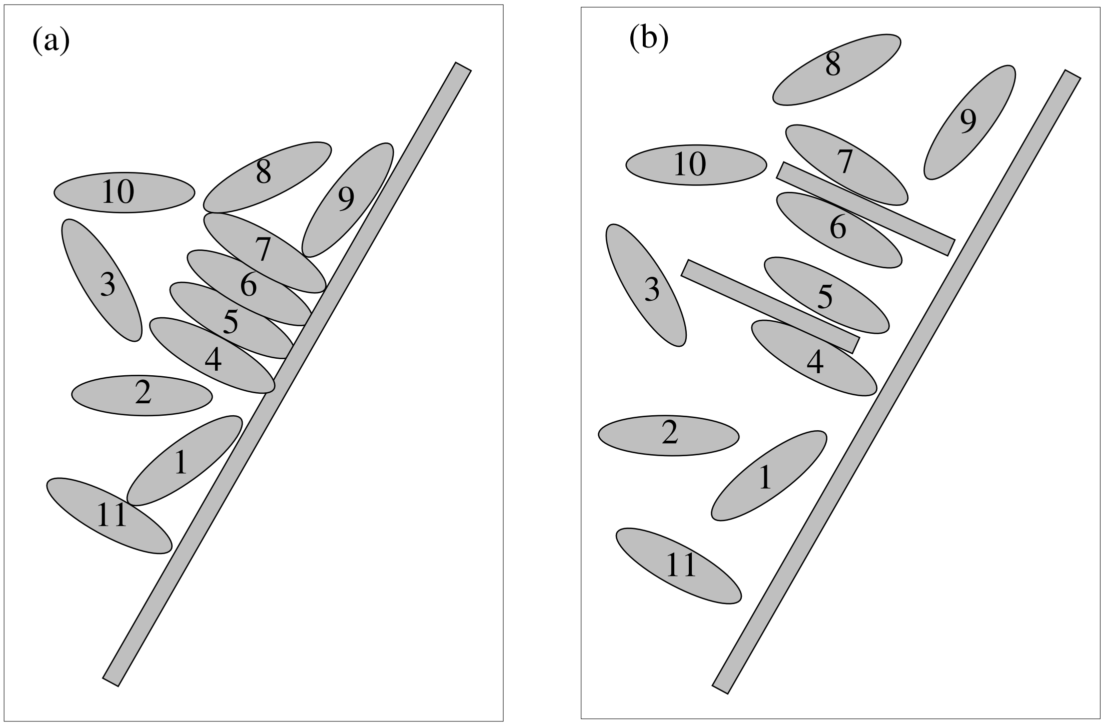


Figure 6.6: Relationship 'tightness'.

The group must have sufficient cardinality.

Since one of the purposes of group formation is to decrease memory usage and increase match efficiency, a grouping with lots of members should be considered stronger or more justified than a grouping with only a few members. However, the effectiveness of the matcher and generaliser is usually helped by explicitly representing even small groups, since they will enable one-to-one correspondences between sets of objects with different cardinalities. Therefore, the current version of GRAM does not use cardinality as a grouping criteria at all, except that a group must have at least 2 members, or 3 members if it is to be explicitly represented as a *chain*.

Member structure should be sufficiently complex if the grouping is justified by structure alone.

If we were to focus on memory usage and match efficiency, then not only the cardinality of the group would be relevant, but also the *complexity* of the individual members. The complexity of an object can be simply measured in terms of the number of subcomponents it has. Forming a grouping of complex structures is more beneficial in terms of memory usage and match efficiency because the typical-member concept is a summary of a larger quantity of information. This advantage is only achieved, of course, if the individual members are removed from memory, or at least ignored by the matcher.

However, for the same reasons of match effectiveness discussed above for cardinality, GRAM normally ignores complexity, and treats a group of single blocks as being just as worthwhile representing as a group of complex structures. An exception to this is when the proposed group is a *collection* – that is a dispersed set of similar objects which are not organised into a *cluster* or *sequence*. The reason is that GRAM's representation does not include a wide variety of attributes such as texture, colour, edges-shape details, material, *etc*, and therefore highly similar primitive objects (with no substructure) can often appear all over an object or scene. For example, a piece of a bicycle frame might be grouped with a chair-leg if there was no requirement for sufficient complexity to ensure that the similarity is 'interesting'.

A group need not be formed if the members parents are grouped.

If the members of a proposed group are all subcomponents of members of an equally strong group with the same cardinality, then the grouping may be redundant. For example, it is not necessary to form a grouping of the backs of several chairs if the chairs have already been grouped. This criterion is called *parent-non-groupedness*.

For example, in Figure 6.7, the group of objects *U*, *V*, *W*, and *X* is not worth representing explicitly since their parent objects, *A*, *B*, *C*, and *D* form an equally strong group. However, objects 1 to 6 do form a worthwhile group, since only some of their parent objects are grouped.

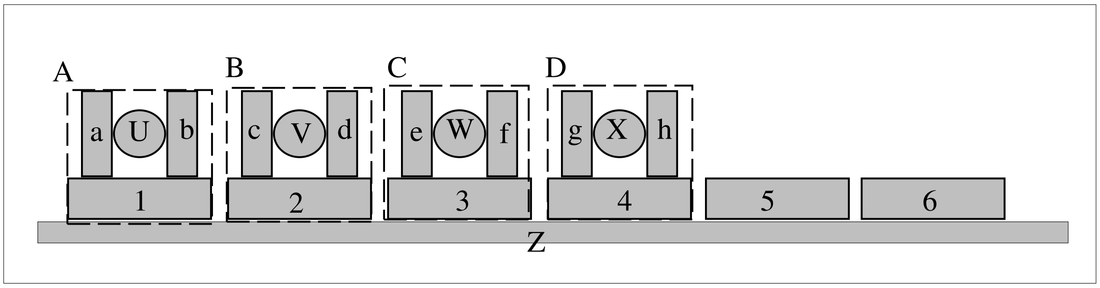


Figure 6.7: Parent-non-groupedness.

The term *parent-non-groupedness* is slightly misleading, because it actually should refer to grandparents, or any super-parents. For example, in the case of the chair backs, each chair-back may be composed of several subparts which may not have a direct parent relationship with their chair object, and we do not want groups of these pieces (one piece from each chair-back) to be made into a group.

Task specific grouping may be justified on the basis of a single common feature.

A grouping may be formed from a collection of objects that all share a single (or perhaps just a few) common features, such as category, colour, shape, a distinct subcomponent, and so on. The members of such a group can otherwise be quite different, or may be similar but such that there is no clear boundary with non-members except on the basis of their particular common feature. For example, the group might consist of "all plastic objects on the bench", or "all chairs in the room that have arms".

There are as many ways of forming such groups as there are combinations of object attributes. For example, the desk I am working at could be grouped into plastic objects, objects that are about 3cm along one dimension, objects which contain a circular component and a red component, and so on. Therefore, the *single-common-feature* grouping criteria must be task specific, rather than being included in the general automatic group-finder, and so will not be considered further.

6.2.2 Group Finding Search Strategies.

This section explores the problem of how to find sets of objects that satisfy a sufficient number of the above grouping criteria sufficiently strongly that a group-object, or at least a typical-member concept, should be formed. We are not concerned with the details of assigning numerical importance to the various criteria, but only with identifying the kinds of search strategy that can be used and the factors that must be taken into account during the search.

Since the search algorithm must be based on the grouping criteria, this section considers each of these criteria as a way of identifying what kinds of search heuristics and strategies it suggests.

Following this, the section considers how these strategies can be integrated into a single group-finding mechanism by making the distinction between two overall search approaches, *Propose-and-Prune* and *Seed-Expansion*.

Search heuristics and strategies suggested by the grouping criteria.

(a) Structure similarity: *Any pair of objects in a scene may be structurally similar, suggesting that all pairs must be considered.*

The criteria of *structure similarity* suggests that we need to consider objects that are similar, independent of their context or relationships between each other. This does not provide much constraint on the search, since the objects could appear distributed all over the scene, as in the case of the bottles on the bookshelf on page 254. It would obviously be expensive to perform a complete comparison of all pairs of objects in a scene, remembering that an 'object' includes composite and primitive components at all levels of detail. Therefore we need some way to prune this search.

One way is to assume that we only need to consider grouping objects of roughly similar absolute size, and to make use of the decomposition hierarchy to prune the search on this basis. More specifically, if two objects have sizes that are too dissimilar to warrant being members of the same group, then there is no point considering groupings consisting of the larger of the two objects and any of the subpart descendents of the other smaller object. For example, in an observed office in which the desk is significantly larger than the potplant, there is no need to compare the desk and the potplant for structural similarity, or to compare the desk with any of the bits of the potplant, since they must obviously be smaller too. This very simple heuristic reduces the search enormously, since every pair of objects considered and found to be too dissimilar in size, means *n* other comparisons need not even be considered, where *n* is the number of subpart descendents of the smaller object, and may be large.

A second way of reducing the search is to make use of an indexing system that enables concepts in memory to be directly accessed on the basis of particular features. If the system treats every observed object as a concept in its own right and adds information to the feature indexes to make it directly accessible (as shown in Figure 6.8), then the group-finder can simply scan the objects in the object graph and whenever the features of an object are indexed to one or more other objects in the object graph, then this suggests a possible grouping. The search is therefore linear, since each object is only considered once. The success of this approach depends on the effectiveness of the indexing (or associative memory) mechanisms. Efficiency could also be improved by creating a separate 'short-term memory' index rather than using the indexes of main concept memory.

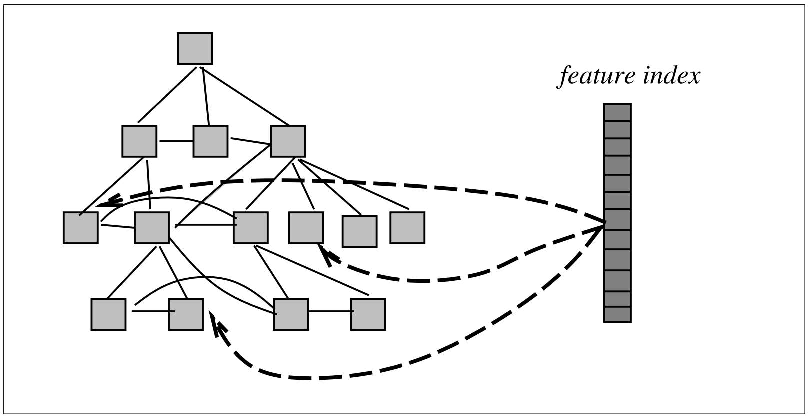


Figure 6.8: Group-finding by indexing from features.

The simplest way of identifying groups of objects with similar structure is when the objects have already been classified as being instances of the same concept. For example, a group of pots on a shop shelf can be identified not by having to compare each of the pots, but because they have all been classified as being pots. Such groups can be found simply by recording references in the concept description to all recently observed instances. Each object in the scene can be checked to see whether it has been already classified, and if so, all of the other instances in the scene are immediately available for group proposal.

The last two search strategies have not been implemented, since they require mechanisms that are part of the larger classification system which has not been the focus of this thesis.

(b) Context similarity: *Consider grouping objects that are related to some other object in the same way.*

The criteria of *context similarity* suggests a more constrained search since potential group candidates must be similarly related to at least one common (or at least similar) neighbour or parent. Therefore, a simple search strategy is to consider grouping the neighbours and subparts of each object. This only needs to be done if the relationships are sufficiently similar. For example, all of the keys on a keyboard (except the space-bar) have similar size and orientation relative to the keyboard as a whole, and so these can be considered for grouping on the basis of possible context similarity, and then a more thorough comparison can be performed. Similarly, all of the bricks in each of the two shelf support stacks of the bookshelf are related in a similar way to the stack as a whole (assuming that the brick stack already exists as an explicit ungrouped object formed on the basis of 'blockness'). On the other hand, the *arm* and *head* subparts of a *person* object have significantly different relationships with the person as a whole, and so they would not be considered for grouping.

This technique was used by Winston in his 'common-features' grouping mechanism: a grouping was proposed on the basis of a common relation with some other object. However, GRAM combines all of Winston's relations (such as ON-TOP-OF, BIGGER-THAN, *etc*) into a single descriptive entity called a parent, neighbour, or subpart relationship. Therefore, the group-proposition strategy must involve comparing relationships and considering grouping the relatees for which the similarity scores are sufficiently high.

(c) Inter-member relationship similarity, and relationship 'tightness': *Traverse the neighbour relationships.*

The third form of similarity, *inter-member relationship similarity*, and the requirement that the relationships be sufficiently *tight*, is the simplest to incorporate into a search strategy. This is because the criteria for 'tightness' are a stronger version of *neighbourliness* criteria. If we can assume that neighbour relationships have already been created, then groups based on similar and sufficiently tight inter-member relationship similarity can be found by just considering groups of objects that are neighbours in the object graph, where the neighbour relationships are similar.

One way of identifying potential groups of neighbours is to somehow traverse neighbour relationships, accumulating objects that are related in a similar manner.

Another approach, more in the form of the object feature-indexing approach mentioned above, is to directly index from relationships to pairs of objects, thus finding groups of pairs of objects that share some relationship feature. However, GRAM's current representation would have to be enriched, by adding features such as *colour, texture, material, edge-shape*, *etc*, to ensure that this method would not produce large numbers of spurious sets of similar relationships.

(d) Groupedness of members' parent objects: *Use the decomposition hierarchy*.

Another grouping criteria discussed earlier was that a grouping proposal is weakened if the members of the group are all subparts of objects which are themselves already grouped just as strongly, as in the case of the seats of a row of chairs. This suggests that the search algorithm could make use of the decomposition hierarchy, by working top-down through the levels, abandoning the group search for subparts of already grouped objects.

However, a simpler way which is not constrained and complicated by the top-down strategy is to evaluate proposed groups of *larger* objects before evaluating proposed groups of *smaller* objects. When evaluating a proposed grouping, the system must check whether the parents are already grouped just as strongly, and adjust the group score accordingly. By processing larger before smaller, unnecessary group formation is reduced, and the need to re-check parent-groupedness later is removed.

The search heuristics.

A summary-list of the search heuristics considered above is given below:

- Use the decomposition hierarchy, and object size comparison, to constrain the search.
- Use the object-feature indexing mechanism to propose groupings.
- Use object classifications to propose groups. If several objects have all been classified to the same concept, then they may form a group.
- Use the parent, neighbour, and subpart relationships of each object to propose groupings of relatees.
- Traverse neighbour relationships to propose groupings of similarly related objects.
- Index from neighbour relationship features to propose groupings of pairs of objects related in a similar way.
- Create groups of large objects before evaluating groups of small objects.
- Check for parent groupedness before forming a group.

Two basic search strategies: *Propose-and-Prune*, **and** *Seed-Expansion*

Now that we have considered various specific group-finding heuristics and strategies, each based on a particular grouping criteria, we can now look at how to integrate them into a single algorithm.

In developing a group-finder it has been necessary to distinguish two different overall search strategies. The first – *Propose-and-Prune* – involves proposing a generous group (on the basis of some common feature) and then pruning it until a stable group is obtained which has a strong membership boundary. This process is illustrated in Figure 6.9 (a). The feature-indexing, common classification, and object-relatees heuristics above clearly support the *proposal* component of such a scheme. Winston's 'common-features' algorithm mentioned earlier is precisely of this form, although there was not an emphasis on having a strong boundary, and the group evaluation did not take into account the variety of grouping criteria presented in section 6.2.1. An outline of an algorithm for the *Propose-and-Prune* strategy is given in section 6.2.4.

The second strategy – *Seed-Expansion* – involves identifying two (or perhaps three) objects that could potentially belong in a group, and then expanding this 'seed' group by adding new members until a clear membership boundary is reached, or until it is decided that the grouping is not good enough. This process is illustrated in Figure 6.9 (b). Winston's other method, for finding sequences, appears to take this form, although that is not made explicit. He talks about finding "sets of objects that are chained together" and "terminating chains at junction points .. or size differences", but does not describe this in the context of a search algorithm or an expansion process. Also, and more importantly, he only uses this strategy for finding sequences, rather than for other non-ordered groups. GRAM's Seed-Expansion algorithm (presented in the next section) is intended for any kind of group.

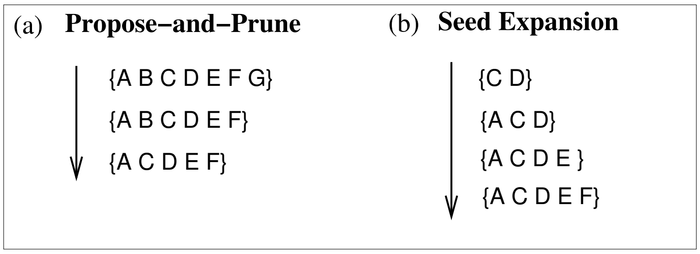


Figure 6.9: Propose-and-Prune and Seed-Expansion.

6.2.3 The Seed-Expansion Algorithm

The seed expansion algorithm is outlined in Figure 6.13. The first step is *seed-group-proposal* which involves finding pairs or triples of objects which could potentially expand into a group. The second step is *seed-expansion* which focuses on a particular seed-group and incrementally adds the best new member to the group until a good group is found, or until the grouping is abandoned.

Seed-group-proposal is based on the various criteria and strategies already discussed above, such as structure similarity (via feature indexing or common classification), neighbour relationships, common parent or neighbour, and similar and tight neighbour relationships (via feature indexing or graph traversal). Currently GRAM only produces seed pairs, not triples.

These seed-groups (in the set *SeedGroups*) are then sorted according to a priority score which is primarily based on how likely they are to be expanded into a good group, determined by the strength of the various grouping criteria for each seed object. The priority score is also partially based on the *size* of the objects so that pairs of larger objects will be processed before pairs of smaller objects, to help satisfy the 'parent-non-groupedness' criteria.

The seed-group with the highest priority is then chosen, and a candidate grouping consisting of its two (or three) members is created by producing a typical-member generalisation of the objects. Then a 'fringe' set of candidate new members is created by finding all other seed-groups which overlap this seed-group (*i.e.* contain one of its members). For example, in Figure 6.10, if the seed chosen happened to be {B C}, then the fringe would be {A, F, H, and D} obtained from the other seed-groups that contain B or C.

Each of these candidate members is evaluated by matching the candidate object (and its relationships with current group members) with the typical-member generalisation of the group. A fit-scoring comparison is done because we want to determine how well the object belongs to the concept. Some candidate members can be immediately rejected from further consideration on the basis of this.

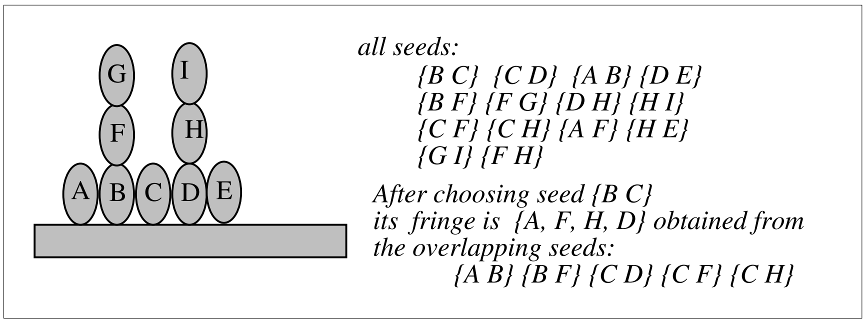


Figure 6.10: A simple seed-expansion example.

The best new candidate member is selected from the fringe set, and if adding it to the group would lower the strength of the group by a significant amount (relative to the current strength) then this indicates a clear boundary. If the current grouping is sufficiently strong, then it is added to *FixedGroups* so that it can be later added to the object graph as a grouped object.

The new member is then added to the group so that further expansion can continue. This is done even if there is a clear boundary and a new group was added to FixedGroups, since a larger and weaker – but still valid – group could be found. For example, in Figure 6.10, after expanding {B C} to form the group {A B C D E}, expansion could continue as a weaker *cluster* group resulting in {A B C D E F G H I}.

After each new member is added, the fringe-set of candidate members is updated by finding any other seeds-groups that contain the new member and also contain another object that is not already in the group or in the fringe-set.

Group expansion finishes when the group is no longer sufficiently regular, or when it is weaker a group in *FixedGroups* of which it is a subgroup.

This latter condition helps prevent unnecessary expansion, and often prevents seed-groups from being expanded at all. For example, in Figure 6.10, after forming the group of {A B C

D E} from the seed-group {B C}, the seeds {A B}, {C D} and {D E} will not be expanded, since they will be already contained in a FixedGroup, and are not stronger (in the sense of their similarity and relationship tightness). On the other hand, in Figure 6.11, if the first seed chosen happened to be {4,5}, it would be expanded first into the group of increasing-height objects {1,2,3,4,5,6}, and then further expanded into the weaker group {1,2,3,4,5,6,7,8,9,10,11}. Although the seed-group {8,9} (for example) will now be contained within a FixedGroup, it will still be expanded because it is stronger than the containing group, since the tightness of the inter-member relationship of the containing group is lower. (Tightness of a generalised relationship takes into account the variance of the relationship).

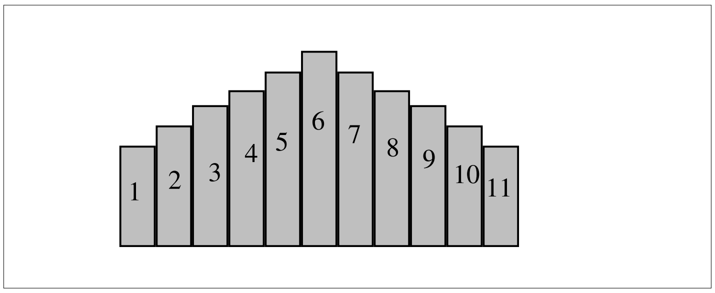


Figure 6.11: Groups within a weaker group.

Seed pairs are also ignored if they do not satisfy the ‘parent-non-groupedness’ criteria: For example, in Figure 6.12, after forming the group of {A,B,C,D}, the seed {1 2} is ignored because the parents already belong in a group, and {1 2} could not expand into a stronger (*i.e.* more regular) group. However, since seeds {4 5} and {5 6} do not satisfy this condition, they are not ignored, and so could be expanded to find the group {1,2,3,4,5,6}.

After group expansion has completed, the next best seed-group is then selected, and the expansion process begins again. This continues until there are no more seed-groups.

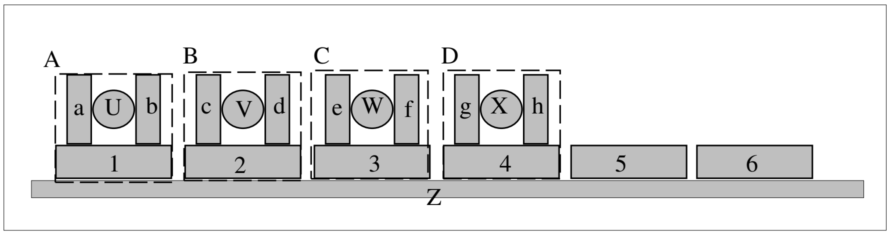


Figure 6.12: Parent-groupedness

An alternative algorithm is to expand groups in a competitive manner: at each iteration, the strongest group is selected, and one new member added to it. Thus a group will only be expanded when it is currently the best group. When its strength drops, some other group will have its turn at being expanded. However, this algorithm leads to wasted effort because several subgroupings of a group would be expanded simultaneously. For example, initial groups each consisting of two apples from a bowl would be expanded simultaneously, competing with each other until one group was completed and encompassed all the others. There would have to be continual testing whether one group already included another group. It is more efficient to simply expand one group at a time until completion, thus making expansion of contained seeds unnecessary.

FIND-GROUPS (scene)
  FixedGroups ←{}.
  SeedGroups ←all pairs (or triples) of objects that could expand into a group.
  Sort *SeedGroups*.
  WHILE *SeedGroups* is not empty:
    seedgroup ←pop strongest seedgroup from *SeedGroups*.
    Create a set of candidate 'fringe' members for *seedgroup*.
      EXPAND-GROUP (*seedgroup*)

EXPAND-GROUP (*group*)
  IF *group* is sufficiently regular to continue expanding THEN
  AND stronger than all containing groups in *FixedGroups* THEN
    best-new-member ←best next candidate member to add
    IF *group* has a sufficiently strong boundary
    AND is sufficiently strong THEN
      Add *group* to *FixedGroups*.
    Add *best-new-member* to *group*.
    Add new candidate 'fringe' members to the group, based on *best-new-member*.
    EXPAND-GROUP (*group*)

Figure 6.13: Seed-Expansion Algorithm

Seed-expansion for *chain* groups.

The seed-expansion algorithm as given above does not distinguish between *chain* groups and other kinds of groups, such as *clusters*. However, *chain* groups are a special case because they are ordered, and this must be considered explicitly in the algorithm. New members must be only added to the *ends* of the chain, and if a new member could potentially be added as a neighbour of a middle member then this would indicate a fork in the chain, and should force abandonment of the chain, although the group it may continue to be expanded as a *cluster* rather than a *chain*. Figure 6.14 shows a situation where there is a fork in a chain. If the grouper is expanding the group of objects {2,3,4,5,7}, then object 6 might be considered as the next candidate to add, since the relationship between 4 and 6 is typical of the current group. However, because object 4 is not an end-member, this indicates a fork, and so the chain expansion should be abandoned.

On the other hand, the sequence of objects {1,2,3,4} could be represented as a chain, although the membership boundary is weak since the relationships between 4 and 6, and between 4 and 5, are both typical of the {1,2,3,4} group.

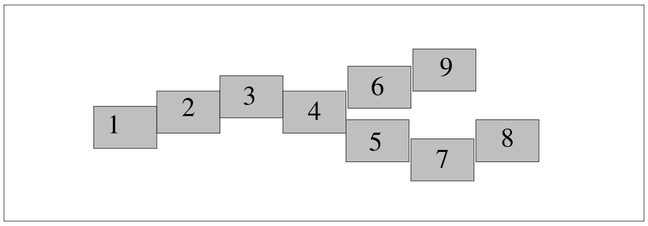


Figure 6.14: A chain with a fork.

An example of seed-expansion.

The following example shows the seed-expansion process for the objects in Figure 6.12. The proposed seed-groups shown are given below, ordered by priority score. (Note that the differing contexts of the objects at the left and right ends reduce the scores of the seed-groups involving them.)

{B C} {A B} {C D}
{2 3} {3 4} {4 5} {1 2} {5 6}
{V W} {U V} {W X}
{b c} {c d} {d e} {e f} {f g} {a b} {g h} (these are weak)

The first seed-group chosen is {B C}. Seed-Groups that have sufficient neighbour relationship tightness are always assumed to be a potential chain at the beginning of expansion, and this assumption is only abandoned when a fork is found. The candidate fringe members of {B C} are A and D, obtained from the overlapping seed-groups {A B} {C D}. None of these imply a fork, and so the group can remain as a *chain*.

Object A is added to the group to form {A B C}, ordered by the neighbour relationship. he ordering is simple to determine because the overlapping seed-group {A B} indicates that A should be added to the 'B' end of the chain.

The fringe of this new group is now just D, which still supports chain-ness, and the group is still strong enough for expansion. Since D fits the group well, there is not a clear boundary to justify fixing the group as it is, so D is added to the other end, resulting in the chain-group {A B C D}. Now there are no further candidate members, which is obviously a strong boundary and so the group can be added to the instance graph as a new object, and expansion is abandoned.

The seed-groups {A B} and {C D} are then compared with this new group and found to be no stronger, and so they are not expanded.

In addition, the seed-groups {1 2}, {2 3}, {3 4}, {U V}, {V W} and {W X} will not be expanded since both objects of each pair are subparts of consecutive members of the {A B C} group, and are not significantly stronger. Therefore, the only seed-groups that will be expanded at this point are as follows:

{4 5} {5 6} {b c} {c d} {d e} {e f} {f g} {a b} {g h}

Seed-group {4 5} is selected, and expanded in the same way as above until the group {1 2 3 4 5 6} is obtained and added as a new group object. The seed-group {5 6} is ignored because it is contained within this.

The remaining seed-groups are somewhat weak, and would not be created if lower acceptability threshold parameters were chosen for the group-finding system. However, they have been included here to illustrate how they do not get removed on the basis of the 'no-parent-groupedness' requirement, since the objects in each pair are *not* subparts of consecutive members of the {A B C D} group. Therefore, the best seed-group {b c} is expanded into the group {a b c d e f g h}.

The same process occurs for *clusters* and other types of groups, except that the chain-ness assumption is dropped at some point during expansion, and so the group ordering becomes unimportant and members are added at any place in a group.

Groups of groups.

The algorithm can be extended very slightly to also cope with groups of groups, such as the group of four rows of books in the bookshelf on page 254. Whenever a new group-object is formed, further seed-groups that contain that new group-object are created, so that groups can be formed from these.

6.2.4 The Propose-and-Prune Algorithm

The Propose-and-Prune algorithm has not been implemented in the GRAM system since the Seed-Expansion algorithm has proved sufficient for all of the kinds of groups that have considered. However, an outline of a Propose-and-Prune algorithm is given here.

Propose-and-Prune operates in the reverse direction from Seed-Expansion, beginning by obtaining a generously proposed grouping on the basis of the various criteria discussed earlier. For example, proposed groups might include a group of all objects that are related in a similar way to some particular object, or a group of objects which have all been indexed from the same object features or relationship features, or a group of objects with the same classification. Winston's system proposed groups on the basis of only the first of these criteria,

In GRAM-based Propose-and-Prune algorithm, a *generalised concept* would be created from the proposed members. This is more complex than Winston's system which creates a 'common-features' list.

Then, each member is matched with that concept. If one or several members are sufficiently atypical relative to the other members, then they are removed from the group. This requires the concept to be specialised to exclude these members. Then each remaining member is again compared with the concept, and pruning occurs again if necessary. The process continues until a stable group is reached, or until the group is too weak to be considered further.

An assumption of this method is that the initially proposed grouping has a clear boundary, since there is no way to test for this in the algorithm unless a set of 'fringe' non-members is identified, as in the Seed-Expansion algorithm. Therefore, the feature-indexing and common-classification proposal mechanisms should err on the side of too many proposed members, so that they can be pruned until a clear member-nonmember boundary is found.

One minor difficulty of implementing this algorithm in the GRAM system would be the task of specialising a typical-member concept to exclude atypical members. This can involve removing disjunctions, ungeneralising attribute values, and removing objects and relationships, none of which are supported by the existing generalisation mechanism.

6.3 Relationship Selection

The instance constructor is not only responsible for creating objects, but also for creating parent, neighbour, and subpart relationships between them. This section discusses the kinds of criteria that can be used to select which relationships should be made explicit in each object description.

First, the section presents four general requirements which form the basis for relationship selection. Following this, it considers a variety of specific relationship selection criteria. The mechanisms for searching for the best selection are also briefly discussed.

The four general selection requirements, all of which are specialisations of the overall requirements of the representation scheme discussed in chapter 3, are as follows.

1. Memory usage should be minimised by avoiding unnecessary relationships.

For a scene containing *n* objects, there are n^2 possible relationships, and this would require significant memory usage for a typical scene. A typical scene in a house or factory might be represented in terms of hundreds or thousands of objects, which would mean tens of thousands or millions of relationships. It is therefore desirable to minimise memory usage by only including relationships that are useful.

Reducing the number of relationships also reduces the computation time for generating instance descriptions.

2. Relationships should capture structurally or functionally important information.

Concept descriptions are intended to consist of information that usefully characterises the concept, so that they can be used to identify the important faults or unusual features of an instance, or to make predictions about the details of an instance without having to observe (or even being able to observe) all of its details. For example, after recognising a bicycle from a quick glance, it is not useful to be able to predict that the back tyre is not at the expected distance and direction from the left handlebar, or to notice if it is not. On the other hand, we might want to be able to predict that the chain is connected to the front and back sprockets, and to notice if it is not.

Since man-made objects are primarily created serve some function, the information in a concept description should therefore be related to that function. However, GRAM does not deal with functional or behavioural knowledge, or even domain specific knowledge, and hence the measure of what information might be functionally important must be based on, or embodied in, general heuristics pertaining to structure.

3. Relationships should support "incremental spread" matching.

Since the GRAM matcher makes use of parent, neighbour, and subpart relationships to constrain and guide its incremental-spread comparison process, the choice of neighbour relationships is important to ensure that comparisons lead to good classification and generalisation performance.

If there are too few relationships made explicit, then some object correspondences might not be found. On the other hand, if there are too many explicit relationships, the matcher will be overloaded with candidate correspondences to evaluate and reject.

For example, in Figure 6.15 (a) the object X is only linked to one other object, via a parent relationship. Therefore, when matching this instance graph with a concepts in memory, a correct correspondence for X is only possible if its parent has been matched classified successfully. In example (b), on the other hand, object X has more relationships made explicit, and this enables the incremental-spread process to propose classifications for X on the basis of any of its neighbour and parent classifications.

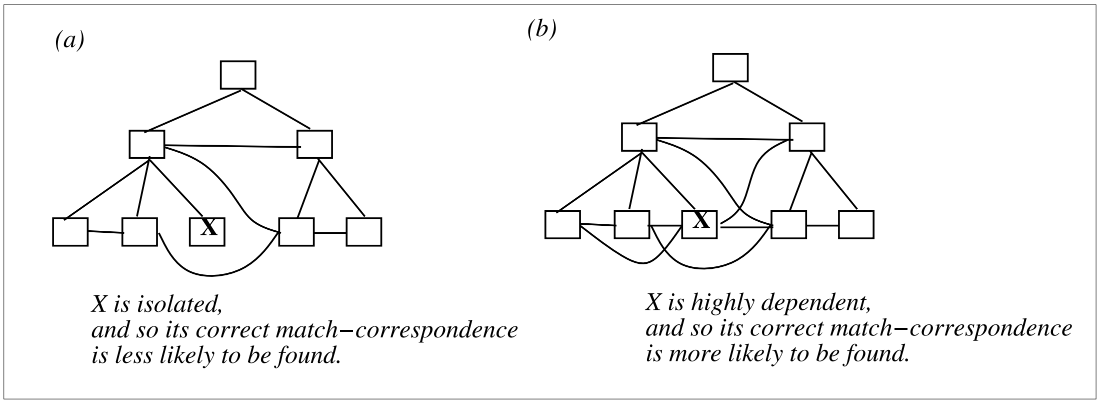


Figure 6.15: Relationships support the matcher.

4. Selected relationships should help to convincingly support a match correspondence and to resolve ambiguities when matching and generalising.

An object should include sufficient explicit relationships that a correspondence proposed and evaluated by the matcher can be convincing. The more relationships that are made explicit, the more convincing a correspondence is if a high similarity score is obtained. If only one or two relationships are included, then a high similarity score might not mean very much.

An object should also include relationships that help resolve ambiguity when generalising. Chapter 4 showed that the matcher does not have to address ambiguity, since it does not attempt to find one-to-one correspondences. However, the generaliser *does* have to deal with ambiguity. Therefore, we do not want ambiguity to result simply from a description lacking relationships that could resolve the ambiguity. As a simple example, if the two chairs in Figure 6.16 are to be matched, then the correspondences between the legs A1 and A2, and B1 and B2, might be ambiguous if the relationship between the legs of each chair is not made explicit.

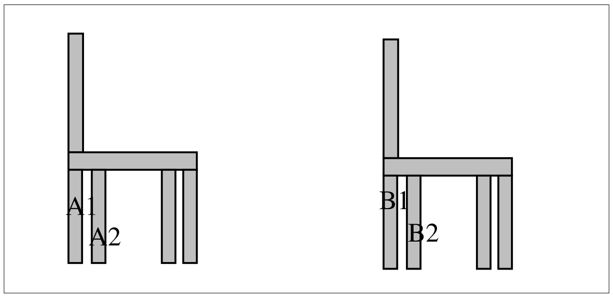


Figure 6.16: Relationships may help reduce ambiguity.

6.3.1 Criteria for Selecting Neighbour Relationships.

On the basis of the general requirements given above, we can now identify a number of specific criteria for relationship selection, beginning with those for neighbour relationships.

It should be remembered that neighbour relationships are *directional*, in the sense that each relationship is associated with and recorded in the description of only object, even though an identical relationship may also be associated with and recorded in the description of the relatee. *Both* neighbour relationships are described in terms of position, orientation, *etc* relative to *both* objects involved. However, the issue being considered here is whether to explicitly include a neighbour relationship in the description of a particular object, independent of whether it is made explicit in the description of the relatee.

Each of the selection criteria discussed below are listed in Figure 6.17, along with simple examples. A candidate relationship is given a score for each criterion, and these scores are combined into a single score for a relationship. If the score is above a certain threshold, then the relationship is made explicit.

Proximity.

The proximity of objects is the most obvious and most important criterion for selecting neighbour relationships, since the closer two objects are together, the more likely they are to be functionally dependent on each other. For example, the mouse of a computer is close to the mouse pad, and so that relationship can be considered more important than the relationship between the mouse and the keyboard, or the office door handle. In Figure 6.17 (a), the *A*–*B* relationship is considered stronger than the *A*–*C* relationship according to the proximity criterion.

In GRAM, the proximity criterion is measured in terms of the distance between the objects as a ratio of the largest dimensions of the *neighbour* (remembering that each relationship is associated with just one object). So in Figure 6.17 (a), the *C*–*A* relationship is considered more important (in the description of *C*) than the *A*–*C* relationship (in the description of *A*). From the point of view of *C*, *A* is a significant neighbour, just as the sun is a significant neighbour of the earth because it defines its location in the universe. But from the point of view of *A*, *C* is not as significant, just as the earth is not as significant to the sun. In other words, larger

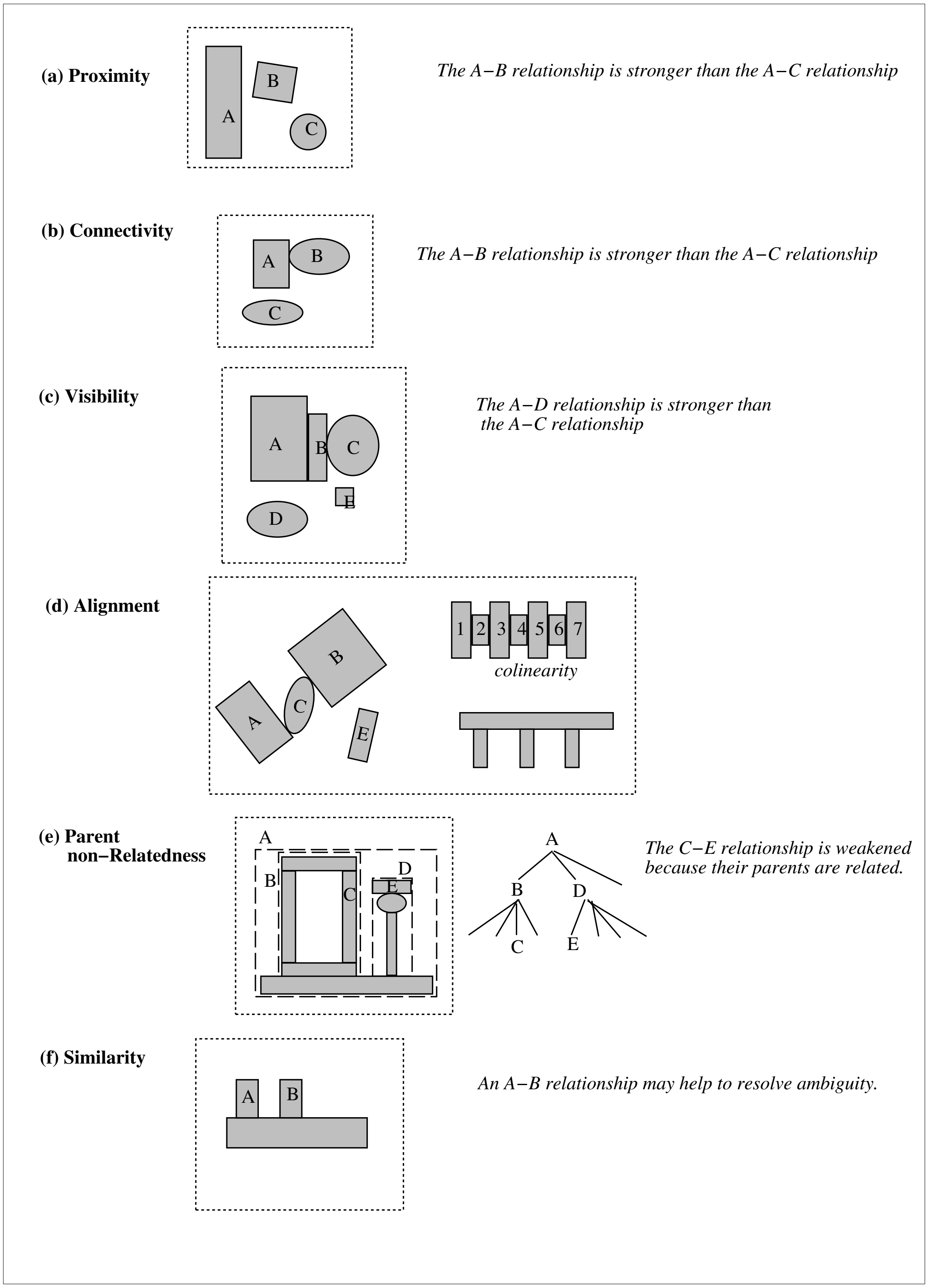


Figure 6.17: Neighbourliness Criteria.

neighbours are considered to be more significant, and hence the proximity criterion is also a *size* criterion.

Connectivity.

If two objects are connected, then this is an especially strong indication that they are dependent on or constrained by other. Connections are particularly important for indicating how an object is constructed. Figure 6.17 In Figure 6.17 (b), the *A–B* relationship is considered stronger than the A–C relationship.

Visibility.

The *visibility* criterion measures how much of a neighbouring object B would be visible from an object *A* if object *B* had an eye on it. For example, for the chair in Figure 6.18, the legs *1* and *2* are clearly visible from each other, but leg *1* and back *5* are not visible to each other because the seat is in the way. The thesis suggests that the more invisible a neighbour is to an object, the less useful that relationship is, because the dependence between the objects is mediated by the objects between them.

For example, legs *1*, *2*, *3*, and *4* are constrained by their position, size, and orientation *etc* relative to the seat 6, and so their relationship with back *5* is less important. In the case of leg *1*, however, the *alignment* criterion described below might outweigh the invisibility criterion.

Figure 6.17 (c) shows some more examples. The *A–C* relationship and the A-D relationships score equally well for proximity, but *A–C* is much weaker on visibility.

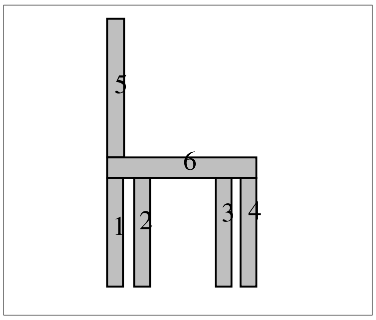


Figure 6.18: Neighbour visibility.

The visibility criterion is very useful for pruning large numbers of relationships between objects that might otherwise be considered good neighbours on the basis of other factors such as proximity.

The precise definition of ‘visibility’ is not given here. It could be based on visibility from the center of the object (as in the current version of GRAM), or from a range of positions on it, or numerous other possibilities. Each scheme will give slightly different results, and will vary in its computational complexity. The emphasis in this chapter is on the *kinds* of criteria required, rather than the specific details.

Alignment

Objects whose edges and/or axes are aligned, in the sense of being co-linear or parallel, are often functionally dependent on each other, even if they are not particularly close, as in the case of shelves in a bookshelf, sides of a drawer, the legs of a chair, the components along a drive shaft, or the centrally aligned parts of a handdrill. Figure 6.17 (d) shows several examples of parallel and co-linear objects.

Many co-linearities and parallelisms are *not* significant though, such as between a book on a desk and a shoe on the floor. However, these probably would not even be easily noticeable anyway. GRAM assumes that the low-level vision system is able to detect clear and obvious alignments, and that all noticed alignments should contribute to justifying neighbour relationships.

Parent non-Relatedness

On the basis of the criteria given so far, a desk might be described in terms of neighbour relationships to the lead of a pencil that is on top of it, a chair leg, and a back book cover. Such relationships are unnecessary because the relationship between the desk and the whole pencil and whole chair and whole book are sufficient to characterise the desk and to support recognition. The position, size, orientation, *etc* of the individual subcomponents of the pencil, chair, and book are sufficiently constrained by their relationships within their parent objects. Similarly, it is not desirable or necessary that a desk in a building be explicitly related to the building next door, even though this relationship may have a strong score based on the *proximity* criterion. The relationship between the two buildings is sufficient. This criterion is important otherwise there would be a huge conglomeration of redundant relationships which increase memory usage and decrease match efficiency.

Figure 6.17 (e) shows another example of this, where the relationship between C and E scores poorly on this criterion because the parents of C and E are related at least as strongly.

To account for this, an additional criteria is needed, called *parent-non-relatedness*. This criterion helps to avoid making relationships explicit between objects for which the parent (or super-parent) of one object is strongly related to the other object or one of its parents (or super-parents). Of course, other criteria such as connectivity and alignment (discussed below) might still lead to the inclusion of the relationship.

The relationship between the desk (or desk-top) and the pencil-lead, would score low on this criterion. On the other hand, the desk and the pencil do not have any related parent objects, and so would be included. They have the *same* parent object, that is, the room as a whole, or a *desk+contents* object, but they do not have any *related* parents.

One consequence of this criterion is that each object will tend to be related to large rather than small objects, since relationship selection is based on a kind of climb-the-hierarchy process. However, this criterion does not prevent relationships to small objects: for example, if a desk has a single pin on it, then a relationship between the desk and the pin would be made explicit, even though the pin is very small. If the pin is in a pin cushion, then this would not be the case.

Another consequence of this criterion is that relationships between direct subparts of a parent object are likely to be included. So, for example, the relationships between the parts of a chair would be made explicit, but the relationships between the parts of the chair and other objects, or parts of other objects, are less likely to be included unless they are particularly strongly related. This means that descriptions of concepts such as “chair-leg” are less likely to be cluttered with inconsequential context information. Instead, information about the expected context of a chair-leg (beyond the boundaries of the chair) will be captured in the context description of the chair as a whole. This helps to improve the classification process since to recognise a chair in an office, the matcher is more likely to be able to find high-scoring correspondences between the legs of the chair and the legs of the chair concept even if the chair is in an unusual context.

Similarity

Earlier, in Figure 6.16, we saw that ambiguity can be resolved by explicit relationships between objects, such as the two chair legs. In this situation the two objects were very similar, and this caused the ambiguity. Likewise, in Figure 6.17 (f) we see two objects A and B which are similar, and so should have an explicit neighbour relationship created between them in order to help resolve ambiguity during future matching.

Therefore, another criteria that contributes to neighbour relationship selection is the similarity of the two objects. This would seem to require matching every pair of objects in an observed scene, but in fact it is not necessary to match a pair of objects whose relationship scores sufficiently poorly on the other criterion that even a perfect similarity would not bring the overall selection score above the required threshold. Also, matching can be done very efficiently by first only performing a rough match, and then doing a more detailed match only if they are sufficiently roughly similar. In fact, a rough comparison alone is sufficient for this criterion.

Mismatch during matching or generalisation.

The above criteria are all data-driven. There is also an expectation-driven criterion that involves the matcher, since relationships can be created not only during instance construction, but also during matching when resolving mismatches. When a concept description contains a relationship that is not present in the description of an instance with which it is being compared, the matcher can request the creation of a relationship, even if it scores poorly on all of the criterion given above. This is considered further in section 4.4.7.

6.3.2 Criteria for Selecting Subpart Relationships.

The criteria for selecting subpart relationships are more straightforward than for neighbour relationships, since there are already more constraints due to the decomposition structure that has been created by the object-formation process. Firstly, relationships can only be created between an object and its subcomponents, and secondly, a set of subpart relationships will already have been specified by the object-formation mechanism, since it works by identifying sets of objects to combine into a single composite object.

Therefore, the subpart relationship selection task assumes that objects are already represented as some kind of subpart hierarchy, and the issue is to determine whether the hierarchy can be refined by adding or removing relationships. The discussion in this section refers to the examples in Figure 6.19.

Avoid redundant subpart relationships.

The simplest criterion for optimising the hierarchy is to avoid all redundant subpart relationships. If a subcomponent of object X is also a subcomponent of another of X's subcomponents, then according to this criterion there is no need to include an explicit relationship to it. For example, a subpart relationship between a desk and drawer-handle should be excluded because there is already a subpart relationship with the drawer. Similarly, in Figure 6.19, the description of "Bob" (as a whole) should not include the subpart relationship to his left forearm, since it is already a subcomponent of the left arm. Details about the forearm are not kept directly in the description of Bob, but only in the description of the arm.

This minimises the complexity of the hierarchy, and thus reduces memory requirements and increases match efficiency because there are fewer relationships to deal with.

The other criteria below serve to refine this minimal hierarchy by causing additional indirectly related subparts to be included as direct subparts, or even to remove subparts.

Bypass Weak Subparts.

If an object is 'weak' in the sense that it was only barely considered worth creating by the object-formation system, then additional subpart relationships should be added to bypass this object. For example, in Figure 6.19 the 'Door-X' hierarchy consists of an object *handle+catch* consisting of the handle and the catch, but this object is somewhat weak on the basis of object formation criteria. If another door is observed, say *door-Y*, it is quite possible that a corresponding object will not be included due to its object-formation score falling just below the threshold. This means that a mismatch would occur, and the correspondences between the handles and the catches would have to be found via neighbour relationships. Therefore, in this situation it is worth creating an explicit subpart relationship between *door-X* and the handle, and between *door-X* and the *catch*. This enables *door-X* and *door-Y* to be matched more successfully.

Another example of this is the buttons on *TV1* in Figure 6.19: The *array* of nine buttons and the large button below it could be combined into a single part, but this is a somewhat weak composite object. Therefore the *Bypass Weak Subparts* heuristic would suggest that the array and the large button should also be direct subparts of *tvmain1*, as indicated by the two heavy lines on the figure.

Bypass Doubtful Parent.

A slightly different situation arises when a subpart is strong, but one of that subpart's subparts does not clearly and unambiguously belong in it. For example, it is clearly useful to have an *aerial* object in the TV description, but it is not so clear as to whether the base (*abase*) should be

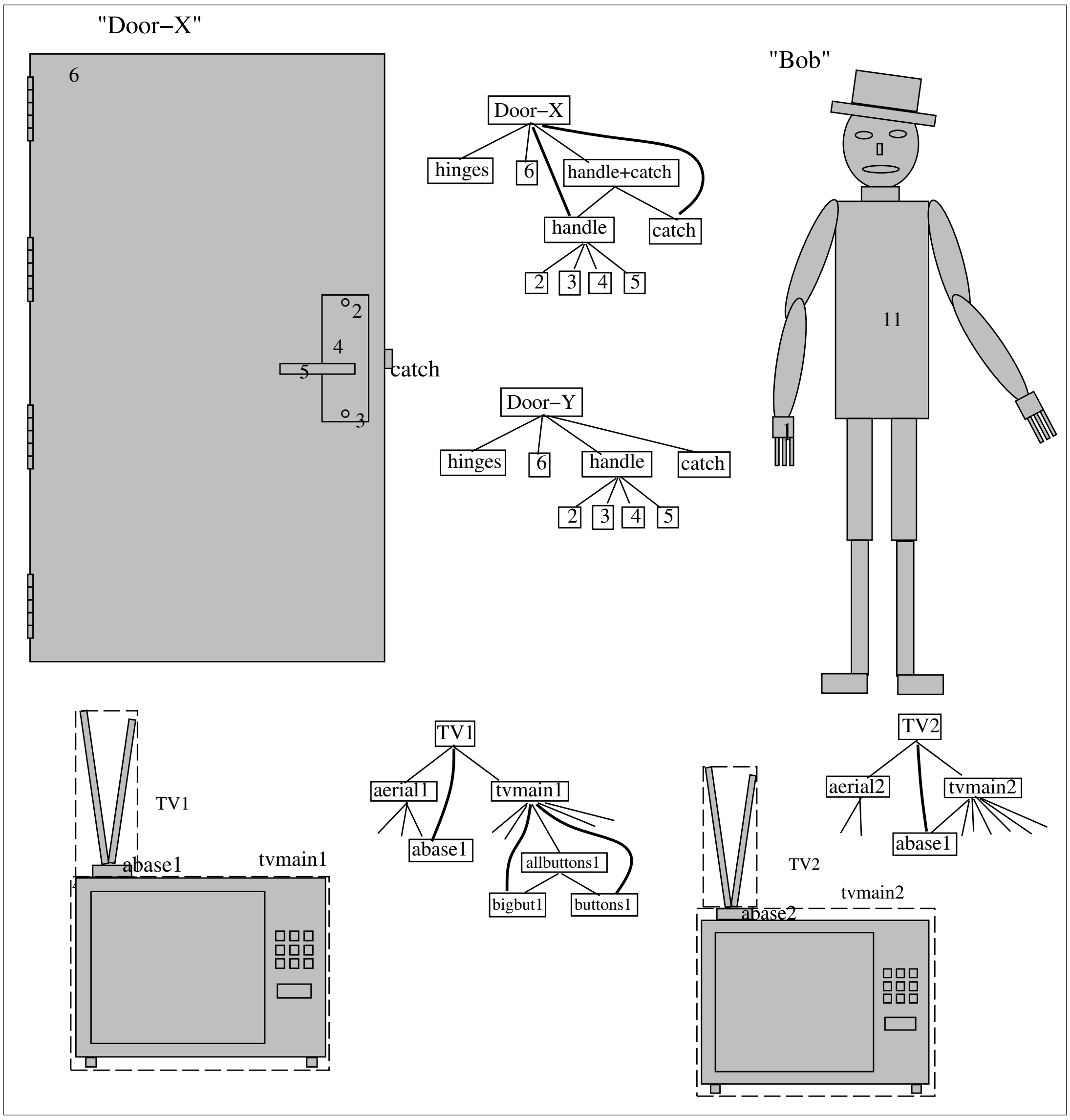


Figure 6.19: Subpart selection situations

included within this, or treated as part of *tvmain*, or perhaps both. In other words, the boundary between *abase* and *tvmain* is not clear. This ambiguity indicates that two observed TVs might be partitioned differently (depending on threshold parameters of partitioning criteria), as shown in the two alternative part hierarchies in the figure. In such a situation the correct correspondence of the aerial bases might be harder to find.

The *Bypass Doubtful Parent* heuristic suggests the creation of a direct relationship from an object X to an object Z if Z is a subpart of an object Y, Y is a subpart of X, and Z is only ambiguously or weakly a subpart of Y. In the *TV* example, subpart relationships between *TV1* and *abase1*, and between *TV2* and *abase2* could be created, as indicated by the heavy line.

Mismatch during matching and generalisation.

As in the case of neighbour relationship selection discussed earlier, subpart relationships may also be created during the matching and generalising process in response to expectations of the concept description being considered. This is discussed in the matcher and generaliser chapters.

6.3.3 Criteria for Selecting Parent Relationships.

The selection of parent relationships uses the same selection criteria as for subpart relationships above, but inverted, thus giving almost symmetric results.

6.3.4 Search strategies for selecting relationships.

The above criteria for evaluating relationship selection do not specify how to search for candidate relationships. Subpart and parent relationships are not a problem, since the search is strongly constrained by the subpart hierarchy that is already given. However, the search for neighbour relationships is more difficult because the search space is much less constrained. The current implementation simply does an exhaustive search of all possible neighbour pairings, computing selection scores based on the above criteria. This is clearly impractical and unnecessary for a large scene, but has been sufficient for the purposes of this thesis. Several techniques could be used in future to improve this, as follows.

The first method is to make use of the decomposition hierarchy. If two parts higher up in the hierarchy are found to have a low selection score, then pairs of their subparts must also score even lower for most of the criteria. The alignment and similarity criteria might still hold, but these pairs could be found by special visual alignment mechanisms and feature indexing, respectively. For example, the front wheel and the seat of a bicycle score poorly on the neighbour relationship selection criteria. Therefore it is unnecessary to consider the pairings of their subparts, such as the front axle and the seat tightener, which must necessarily have much poorer proximity, parent-non-relatedness, and visibility.

Another method depends on having a real robot eye which scans the observed scene. Scanning could be done by moving outwards from the part being considered, and not extending outwards further than is necessary. At a lower resolution it could extend further, in order to find large neighbours, while at a finer level of detail it would scan to a more limited range. The level of detail is still governing the search, as in the previous method, but not by using the decomposition hierarchy.

These methods are not discussed further, since the intention here is only to indicate that the search for relationships need not be an unreasonably expensive task.

Chapter 7

Evaluation

This chapter gives initial steps towards substantiating the claims of the thesis by evaluating the performance of the implemented GRAM system. It also discusses the main limitations of the system and identifies the areas of the described system that have not yet been fully implemented, or which could be extended and improved in future work.

The three main claims of this thesis, which were given at the beginning of chapter 1, are the following:

1. Complex physical objects can be matched effectively by using structural descriptions only, without requiring functional knowledge.
2. The effectiveness and efficiency of matching descriptions of complex physical objects can be improved by exploiting the structural relationships between the components of the objects.
3. Complex objects can be matched effectively without maintaining global consistency while searching for correspondences between their parts.

The first three claims are addressed in sections 7.1 and 7.2 which describe the results of matching complex descriptions of two bicycles, and also of matching all pairs of 27 much simpler object descriptions.

The fourth claim is considered in section 7.4 which shows an example of how grouping can significantly reduce the size of a description of a complex household object.

The performance of the generaliser is discussed in section 7.3 which presents the results of generalising the two bicycles.

7.1 Effectiveness of the Matcher

7.1.1 Matching identical descriptions of the same object

Any good structural-object matcher should be able match two identical descriptions and find all correct correspondences between the subcomponents. GRAM's matcher was therefore given two identical descriptions[1] of *BIKE1* in figure 7.1, with the decomposition hierarchy shown in figure 7.2. For every part, the highest scoring correspondence found by GRAM was correct, indicating that identical objects can be matched successfully by GRAM without functional knowledge, and without enforcing global consistency during the search.

It is doubtful whether a system such as Labyrinth (discussed in section 2.8) could achieve this since Labyrinth does not represent contextual information in its object descriptions, and therefore would not be able to disambiguate between components with similar substructure, such as the pedals, tyres, sections of the mudguards, and so on. Wasserman's MERGE (section 2.5) could also not achieve this, since it requires parts to be partially pre-classified by name.

7.1.2 Matching different descriptions of the same object

An important aspect of the effectiveness of a matcher is its robustness, especially the ability to cope with non-canonical part decompositions of the objects being matched, since the corresponding components may be on different levels of the two hierarchies, or may be subparts of different parent parts. Section 4.2.4 states that GRAM can successfully deal with this situation by exploiting the contextual neighbour relationships between components. This claim is supported by the results of matching the description of *BIKE1*, pictured in figure 7.2, with a different description of the same bicycle, pictured in figure 7.3. Several of the corresponding components of these two descriptions are at different levels of the hierarchies, such as the front fork (*FFORK*), and/or have different parents, such as the top of the chain (*CHAINTOP*).

The author produced a list of 74 required winning correspondences, with 6 composite parts of *BIKE1* considered unmatchable (including *SEAT*, *PEDETC*, *HBARFORK*, *FWMID*, and *FRONT*, and this was compared with GRAM's results. 73 out of 74 of the winning correspondences produced by GRAM were correct, giving a performance of 98.7%. GRAM also determined that the six parts above were unmatchable, producing no continuable cnotes for them.

The one incorrect correspondence was between *BPEDBAR* of one bike and *TPEDBAR* of the other bike, with an axis correspondence of 180 degrees. Hence GRAM found the rotational similarity of the two pedals, which is not unreasonable. The score was marginally higher than the correct correspondence, which GRAM also found. The higher score was primarily due to the fact that *BPEDBAR* and *TPEDBAR* were only evaluated with a spread effort of 1. In other words, only the properties and relationships were considered, and not their relatees. If another iteration of the match algorithm were applied, thus giving more effort to

[1] GRAM works directly from the postscript data produced by a graphics package called IDRAW and a text file that specifies the decomposition hierarchy. The selection of neighbour relationships is performed by GRAM.

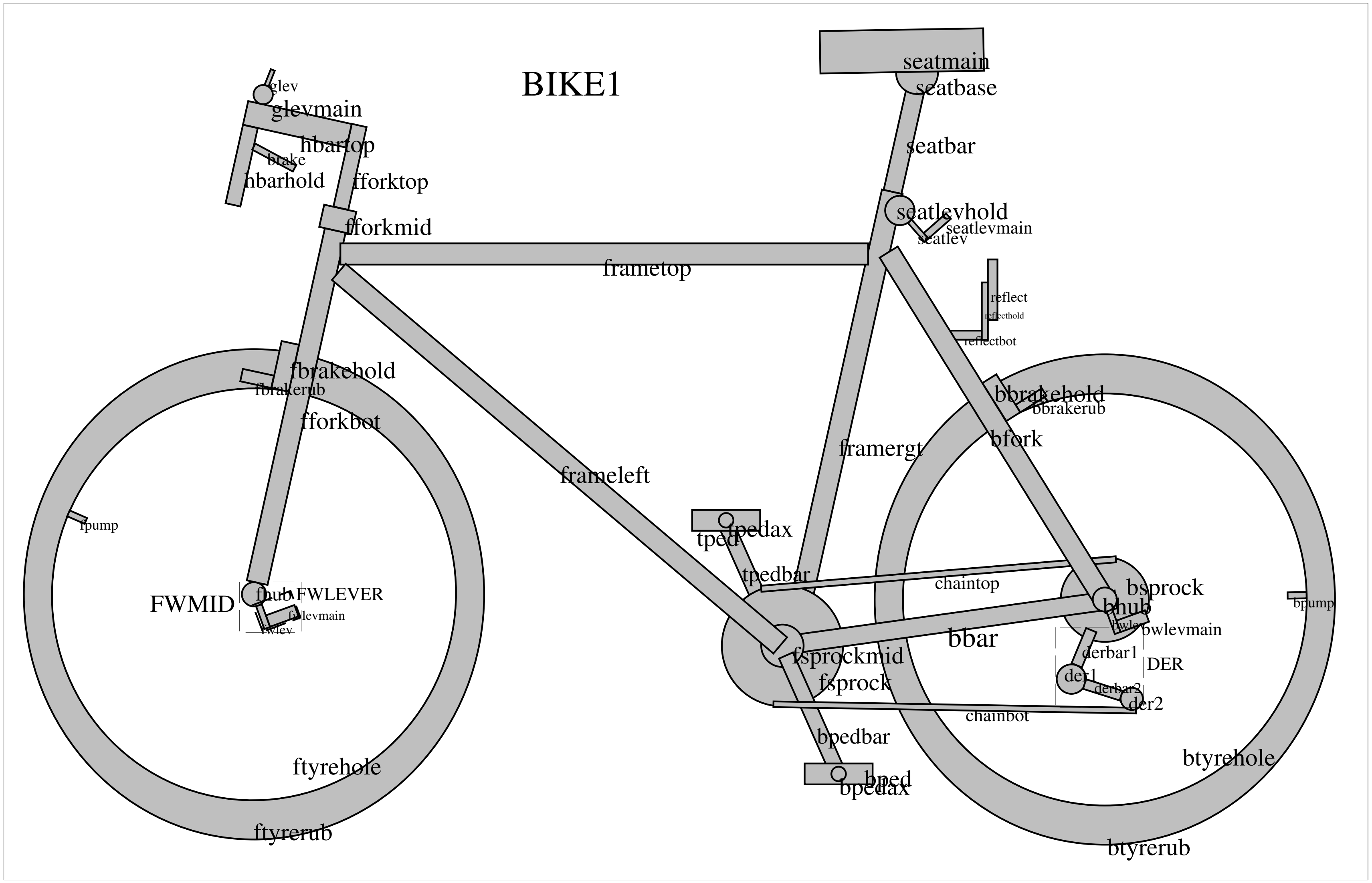


Figure 7.1: BIKE1

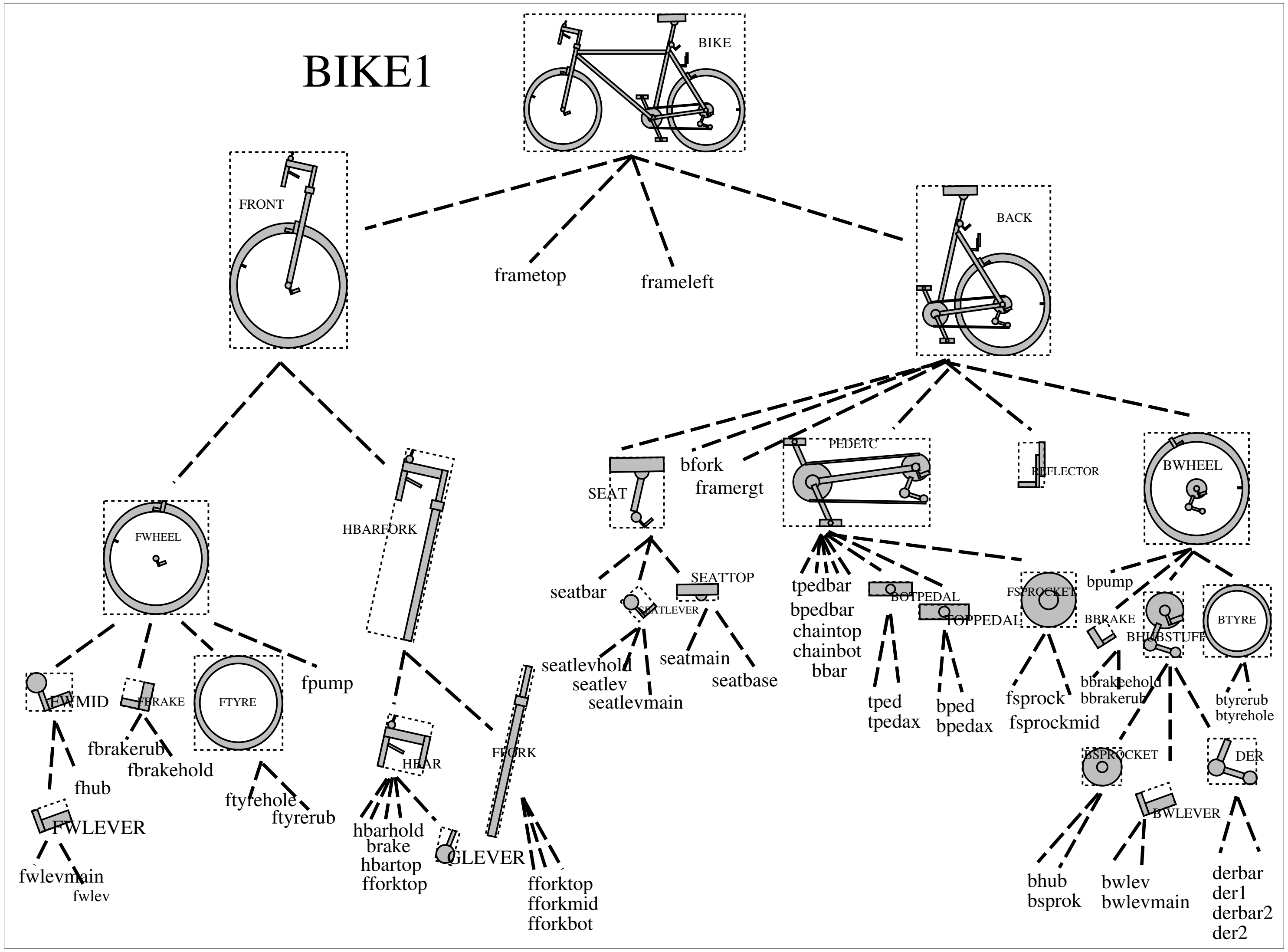


Figure 7.2: Decomposition hierarchy for BIKE1.

the *BPEDBAR-TPEDBAR* comparison, its score would have dropped below that of the correct correspondence.

These results demonstrate the robustness of GRAM's matcher and its ability to exploit neighbour relationships to deal with non-canonical descriptions, especially those that lead to the level-hopping problem.

7.1.3 Matching two different bicycles

BIKE1 in Figure 7.1 (with the decomposition hierarchy shown in Figure 7.2) was also matched with a different bicycle, *BIKE2*, shown in figure 7.4. *BIKE2*'s decomposition hierarchy was as shown in figure 7.5, which is similar to that used for *BIKE1*. The author identified 69 winning correspondences that should be found by a matcher, and compared these with the results produced by GRAM. [2]

When a spread effort of 6 was applied, 65 of the 69 required correspondences were correctly proposed by GRAM as winning correspondences, giving a performance measure of 94%. The 4 incorrect correspondences were as follows: *FHUB* of bike2 was best-matched with *FWMID* of bike1 (composed of FHUB and *FWLEVER*), when it should have been best-matched with *FHUB* of bike1 (and vice versa). This was because although the substructure of bike1's *FWMID* and bike2's *FHUB* were considered quite different, their contexts were considered more similar than bike1's *FHUB* and bike2's *FHUB*. Bike1's *FWMID* and bike2's *FHUB* both have very similar relationships with the wheel and front fork, but bike1's *FHUB* has wheel lever parts attached to it, which bike2's *FHUB* does not. Although not unreasonable, this does seem to indicate a problem in the similarity metrics which will need to be explored in future work.

The other three required correspondences that were not found were between *GLEVMAIN*, *GLEV*, and *HBAR* of each bike. After applying a spread effort of 6, GRAM had no correspondences for these parts (for either bike1 or bike2), either because the scores of the correspondences found earlier were too low to justify keeping the cnote in memory, or because the spread never even proposed any correspondences because there were no sufficiently strong correspondences from which to propose them.

Two additional incorrect winning correspondences were created when more effort was applied to the match, which indicated a further problem in the matcher. The first was for bike2's *TPEDBAR* which was best-matched with bike1's *BPEDBAR*, although bike1's *TPEDBAR* was correctly best-matched with bike2's *TPEDBAR*, and the two *BPEDBAR's* were correctly reported as a winning correspondence. The incorrect *BPEDBAR*:*TPEDBAR* correspondence scored only very marginally higher than the correct *TPEDBAR*:*TPEDBAR* correspondence, with less

[2] 11 parts of *BIKE1* were considered by the author to be unmatchable, or rather, only matchable with parts of *BIKE2* that are better matched with other components). These are *FWLEVER*, *FWLEVMAIN*, *FWLEV*, *FWMID*, *REFLECTOR*, *REFLECT*, *REFLECTBOT*, *REFLECTHOLD*, *BWLEVMAIN*, *BWLEV*, *BWLEVER*.
Of *BIKE2's* 100 parts, 31 were also considered unmatchable: the mudguards and stand (and their components), *FFORKTBOT*, *FFORKMIDT*, *FFORKMIDB*, *FFORKBOTMID*, *HBAR3*, *HBAR2*, *HBAR1*, *FHUBOUT*, *FHUBIN*, *FBRAKE1*, *FBRAKE2*, *BBRAKE1*, and *BBRAKE2*.

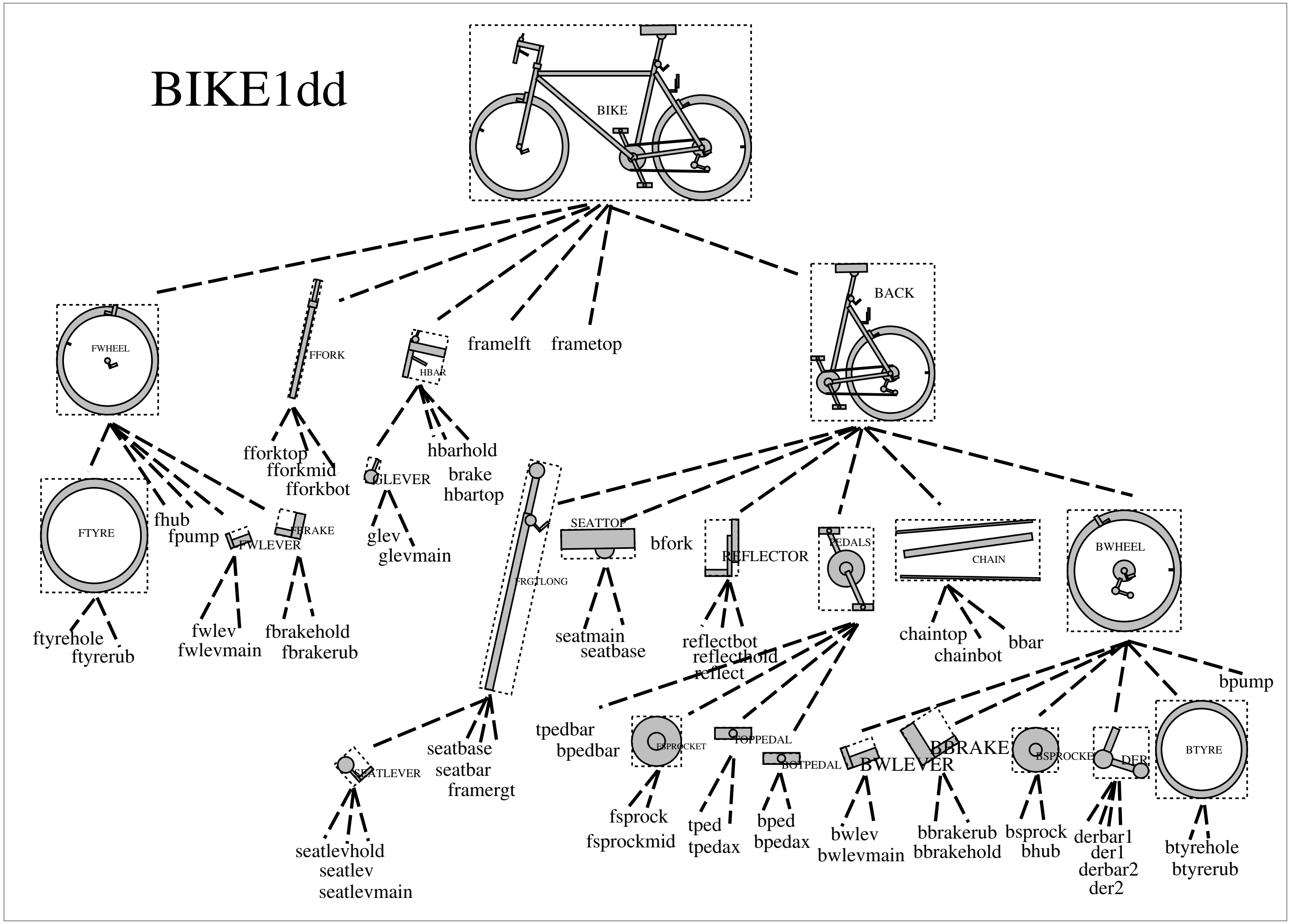


Figure 7.3: BIKE1dd: Same bicycle as BIKE1, but different decomposition

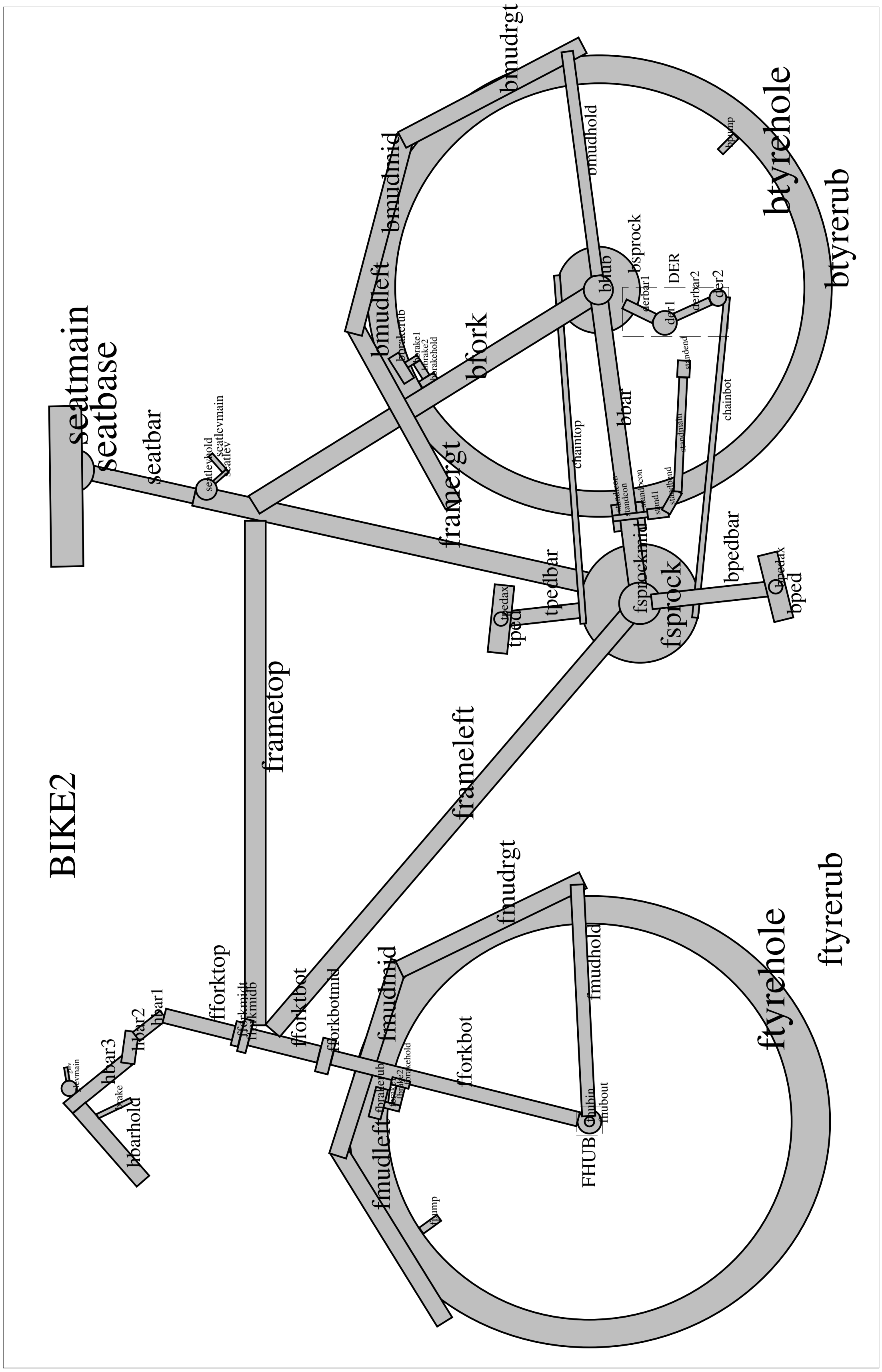


Figure 7.4: BIKE2

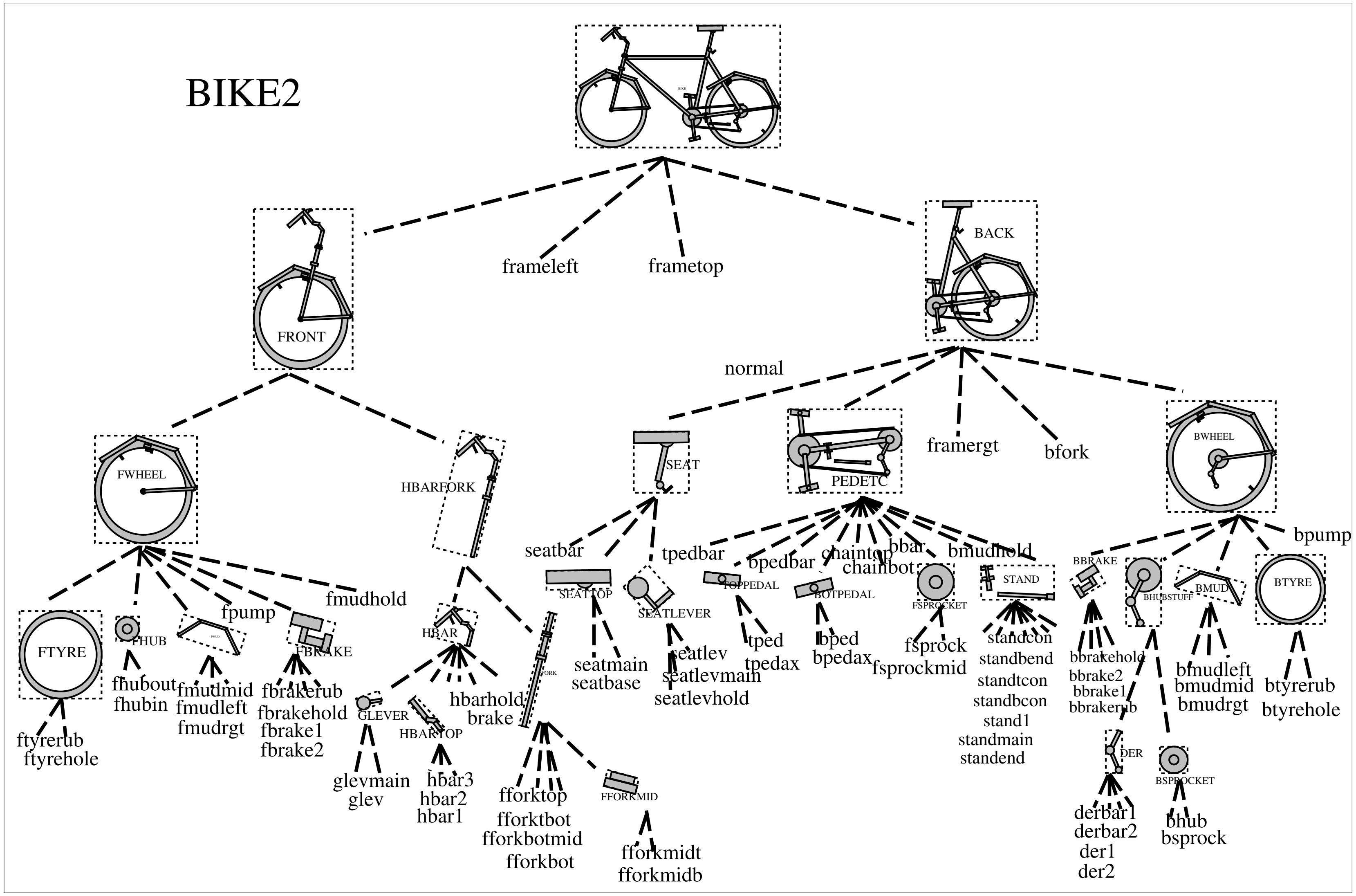


Figure 7.5: Decomposition hierarchy for BIKE2.

than 0.01 difference. However, what is important here is that the *BPEDBAR*:*TPEDBAR* correspondence was only evaluated to spread-1, and thus could not significantly take into account the contextual differences. The *TPEDBAR*:*TPEDBAR* correspondence, on the other hand, had already 'survived' through an effort of 5, and was based on a significant evaluation of contextual similarity.

Another incorrect correspondence that was added when more effort was applied was almost identical to the above, but involved *TPED* and *BPED*.

The *BPEDBAR*:*TPEDBAR* and *TPED*:*BPED* problems are essentially the same as the *BPEDBAR*:*TPEDBAR* mentioned in section 7.1.2 in the case of matching the two different descriptions of *BIKE1*, except that this time there was no rotation of 180 degrees involved. This error seems to indicate that the spread effort applied to a correspondence should be taken into account before making use of the results of the matcher. In fact, GRAM does this when using the results of the matcher: the generaliser does not operate on cnotes that have been evaluated to less than some minimum spread effort, and requires the matcher to be reinvoked before it can produce a generalisation from the correspondence. In the situations above, the generaliser would reinvoke the matcher, which would very quickly lower the scores of the incorrect *TPED*:*BPED* and *BPEDBAR*:*TPEDBAR* cnotes, thus resolving the problem.

Section 4.1 on page 124 stated that the matcher should employ an 'any-time' algorithm which allows useful results to be obtained even if only a small amount of effort was applied, or if the match is interrupted. The graph in Figure 7.6 shows the results of the matcher as it is progressively applying more effort, and also shows the similarity score produced at each step. The five correspondences found at spread-2 were *BIKE1+BIKE2* (*i.e.* the given seed correspondence), *FRONT+FRONT*, *BACK+BACK*, *FRAMETOP+FRAMETOP*, and *FRAMELEFT+FRAMELEFT*. As more effort was applied, more correspondences between parts further down the hierarchy were found, since the spread was able to access them. This indicates, not surprisingly, that the matcher does tend to first find correspondences at a coarse level of detail, providing useful results even from low spread effort. Also, and more importantly, the similarity score in the *BIKE* example converges quite quickly.

It should also be noted that the matcher does not have to start from the root parts of the part-graphs. For example, if the matcher was applied to the *SEAT* of a bike and a learned bicycle seat concept, many of the bicycle parts could be recognised via spreading, even if the bicycle was partially occluded. A strictly top-down matcher, starting from the root of the bike part graph, might terminate quickly due to significant dissimilarities that prevent further spreading.

7.1.4 Matching large numbers of objects against each other

The previous experiments tested the effectiveness of GRAM at finding the correct correspondences between components of large objects. Another experiment involved matching each of the 27 much simpler objects, shown in figure 7.7, against each of the others. The purpose was to test whether pairings of objects that belong in the same category score higher than all other pairings involving one of the objects and another object from a different category. This

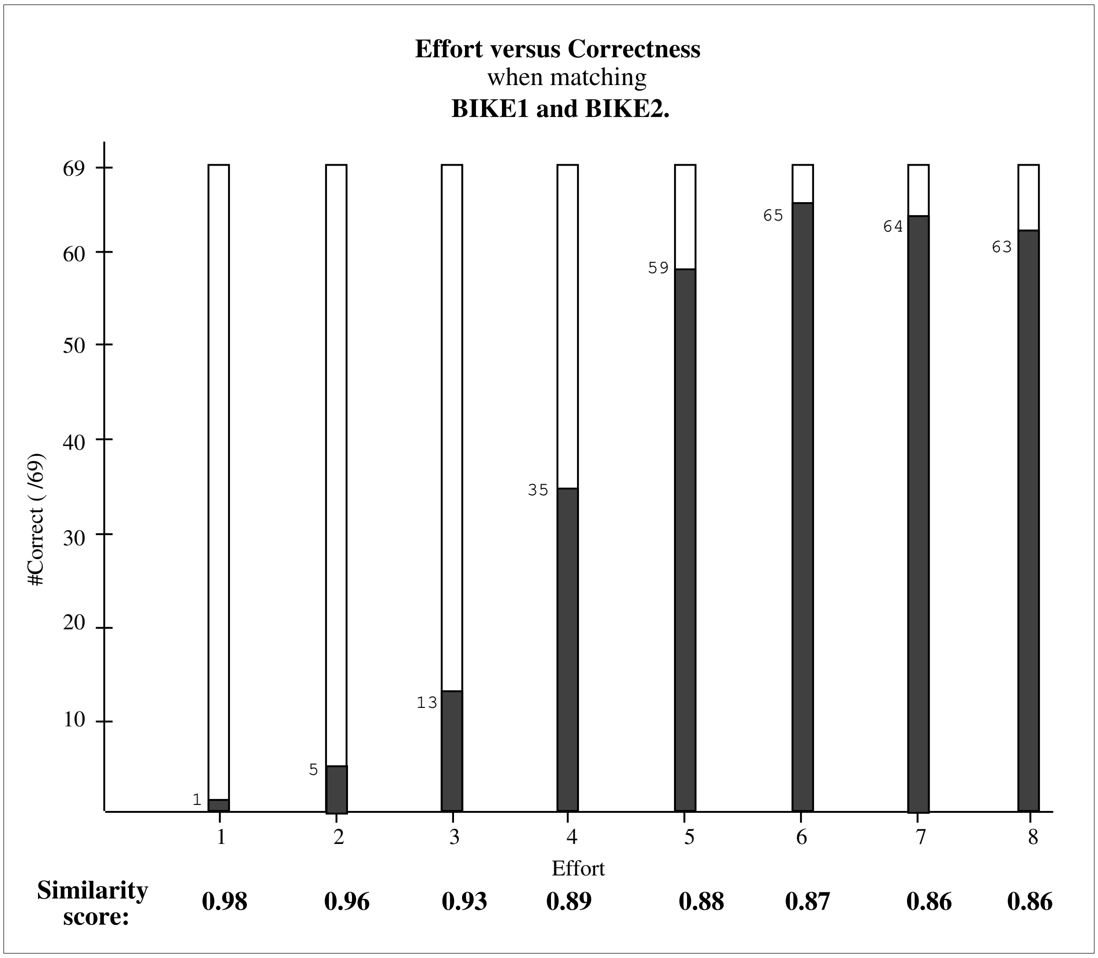

Effort versus Correctness
when matching
BIKE1 and BIKE2.
69
60
50
40
30
20
10
#Correct (/69)
1
5
13
35
59
65
64
63
1
2
3
4
5
6
7
8
Effort
Similarity score:
0.98
0.96
0.93
0.89
0.88
0.87
0.86
0.86

Figure 7.6:

experiment dealt only with ungeneralised instances, but does provide some measure of GRAM's classification ability.

The results are shown in the tables below, which show the scores of all pairs of objects from the same category, and the highest scoring competing pairs of objects that are from different categories. These results indicate how much distinction GRAM makes between the correct and incorrect classifications.

Out of the 31 intra-category pairings, 24 were found by GRAM to be higher scoring than all other competing inter-category pairings, giving a performance of 77%. The only mistakes involved the *TVS* and *CLOCKS*, while the objects from all of the other categories were correctly best-matched with other members of the same category. Since there are 351 possible pairings altogether, GRAM is clearly performing quite well, although this experiment would ideally be done on a much larger number of objects, and compared with results from other systems. However, only one of the systems described in section 1.2 produced a comparable experiment. The PARVO system [Bergevin and Levine, 1993] performed a similar kind of experiment for which 15 out of 23 (or 65%) objects scored higher when matched with the correct object model than with other object models.

The problem with the *TV1* and *TV3* match was that the substructure of the root node of the part graphs is significantly different: *TV1* has two sets of legs, and no aerial, while *TV3* has an aerial and no legs. Therefore, only one out of the three combined subparts have good correspondence scores. According to the similarity scheme created for GRAM , the two objects *are* significantly different. However, GRAM was also used to produce a *generalised* television from four instances, one of which did not have legs (although all had aerials), and therefore was able to learn the optionality of legs, and thus be more tolerant of missing legs. Thus, the low *TV1* and *TV3* score only seems to be a failure of GRAM because we (as humans) already know about TVs.

The low score for *TV2* and *TV3* seems less justified, since both have the main body and aerial. However, because both of the legs are unmatched, and because they each contribute roughly 25% of the score for the root cnote, they pull down the contribution of the higher scores for the main body and aerial correspondences. It is not clear how to resolve this. The current implementation of GRAM does not use the relative lengths of the subparts to weight their score contributions, and this would help to some degree. It would also be possible to give a higher weight to subparts with more complex substructure, but this would not be justified in other situations, such as giving more weight to the drawers of a desk than to the desktop. So, without artificially tweaking the system to give a good result for this particular example, it would seem that the low score is justified. However, as pointed out above, the generalisation process is the primary means by which the relative importance of subcomponents is learned, so that in the case of the TV concept, the presence and similarity of the main body would contribute most to the scores of future instances.

The other problem with almost all of the results of this experiment is that although most of the correct pairings scored higher than competing incorrect pairings, they did not score *much* higher, even for objects that humans would consider to be completely different, such as a TV and a handdrill. This seems to be partly a consequence of GRAM's scheme of not enforcing global consistency, since it allows any relatee of one part to play the corresponding role of any relatee of the corresponding object, even if it plays a different role relative to some other correspondence. For example, the aerial of TV2 might be matched (albeit weakly) with the trunk of a cactus, since the relationship between the trunk and the plant pot is similar to the relationship between the aerial and the TV body. Simultaneously, the leg of TV2 might be also matched with the trunk, but rotated by 180 degrees, since its relationship with the TV body is similar to the relationship between the trunk and plant-pot upside down. The allowance for inconsistencies means that scores tend to be higher than they would be if consistency were maintained, although such scores are still low and are unlikely to justify generalisation.

The problem with the five low clock-pairing scores is similar, but more justified, since there are more obvious differences between the clocks involved.

Headphones	
pairing	score
HEADPHONE1 : HEADPHONE3	0.84
HHEADPHONE1 : HEADPHONE2	0.77
HEADPHONE2 : HEADPHONE3	0.74
best other pairing	0.63

Cacti	
pairing	score
CACTUS2 : CACTUS3	0.92
CACTUS1 : CACTUS2	0.75
CACTUS1 : CACTUS3	0.75
best other pairing	0.58

Lamps	
pairing	score
LAMP1 : LAMP2	0.70
best other pairing	0.61

Clamps	
pairing	score
CLAMP1 : CLAMP2	0.90
best other pairing	0.63

Handdrills	
pairing	score
HANDDRILL1 : HANDDRILL2	0.64
best other pairing	0.63

Sewing Machines	
pairing	score
SEWMACHINE2 : SEWMACHINE3	0.80
SEWMACHINE1 : SEWMACHINE3	0.69
SEWMACHINE1 : SEWMACHINE2	0.66
best other pairing	0.61

Distributor Caps	
pairing	score
DISTCAP1 : DISTCAP4	0.75
DISTCAP3 : DISTCAP4	0.71
DISTCAP2 : DISTCAP4	0.67
DISTCAP1 : DISTCAP3	0.66
DISTCAP2 : DISTCAP3	0.66
DISTCAP1 : DISTCAP2	0.64
best other pairing	0.61

TVs	
pairing	score
TV1 : TV2	0.65
** TV1 : CLOCK4	0.63
** TV1 : CLOCK2	0.60
** TV1 : CLOCK1	0.59
TV1 : TV3	0.56
** 19 pairs!	0.6..0.5
TV2 : TV3	0.50

Clocks	
pairing	score
CLOCK1 : CLOCK2	0.90
CLOCK2 : CLOCK3	0.81
CLOCK1 : CLOCK3	0.81
CLOCK3 : CLOCK4	0.72
CLOCK4 : CLOCK5	0.70
** various pairs	0.63..0.5
CLOCK1 : CLOCK4	0.50
CLOCK2 : CLOCK4	0.49
CLOCK3 : CLOCK5	0.49
CLOCK1 : CLOCK5	0.44
CLOCK2 : CLOCK5	0.42

Although the results presented in this section indicate that there are some problems in the matcher, the results support the claim that complex objects can be matched effectively without having functional knowledge, and without maintaining global consistency during the search and when evaluating similarity. Of course, functional knowledge and global consistency could undoubtedly improve the matcher, but at a cost. The results above demonstrate that they are not *necessary* to obtain a reasonably good performance when finding part correspondences.

The main criticism of the GRAM matcher based on the above results is that although it tends to correctly find the best correspondences, the scores for incorrect correspondences seem

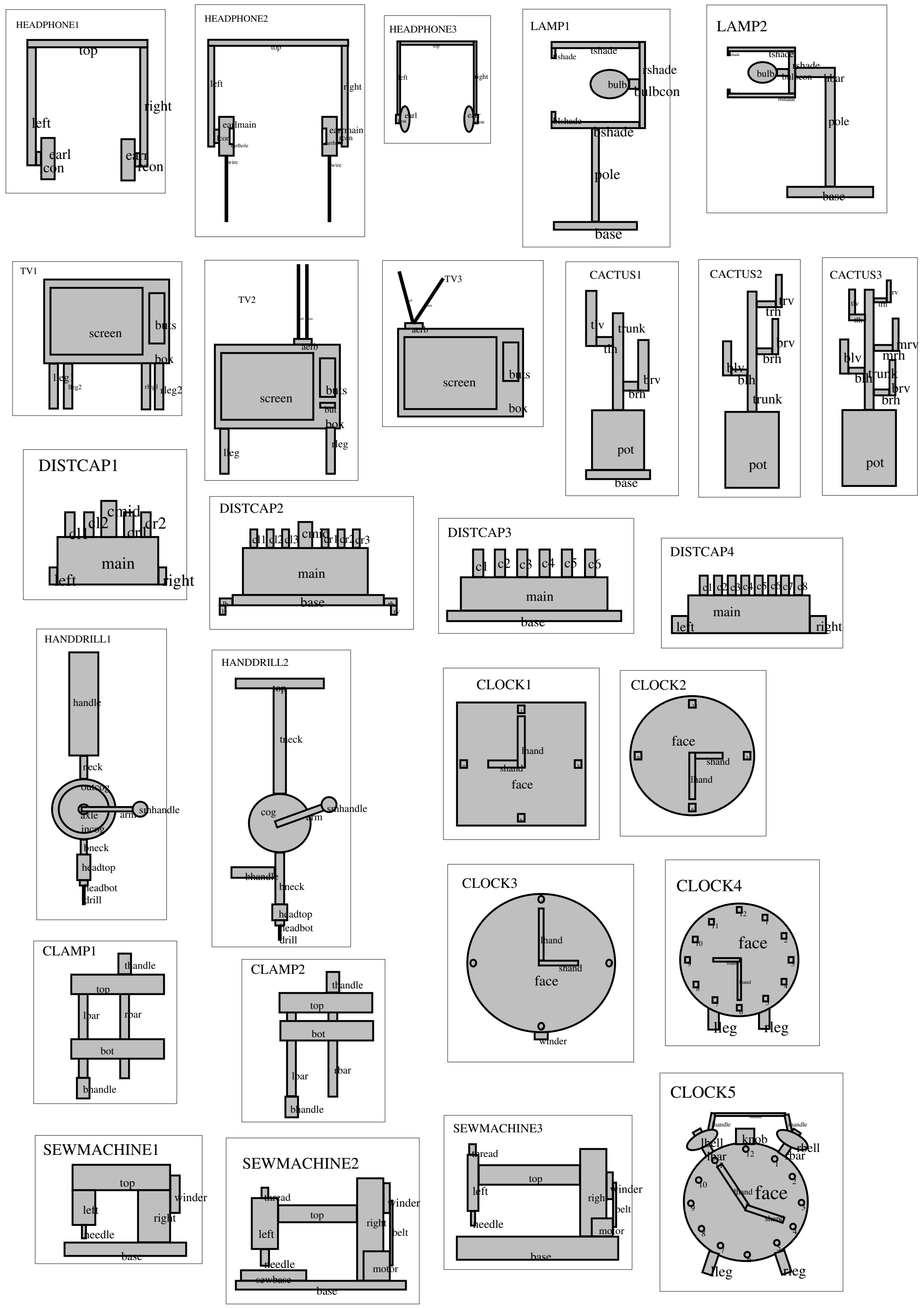


Figure 7.7: 27 objects in nine categories.

unreasonably high due to the strategy of not enforcing bindings. It is clear that enforcing bindings could help resolve this problem. For example, the strangely high score of 0.63 for *TV1* matched with *CLOCK2* would never occur, since no consistent set of correspondences can be found for which the interrelationships match well.

However, enforcing consistency does cost more. Even if some kind of effective greedy algorithm could be developed (of the kind developed earlier in this project [Andreae, 1993]) it would require a sophisticated backtracking mechanism to be able to find the kinds of correspondences that the current system finds in such a robust manner. More importantly, it requires a representation which enables global consistency to be meaningful. GRAM's approach of representing concepts in terms of other concepts, rather than in terms of a hierarchy of locally defined and distinguished parts, is not particularly amenable to enforcing global consistency, since it does not make sense to require a concept to be matched with only one instance. If global consistency were enforced (somehow), only one chair in an office could be classified as a chair. Therefore, consistency could only be enforced in the reverse direction to prevent each instance being matched with more than one concept, unless the concepts are on the same branch of the concept hierarchy.

Future work on GRAM could involve exploring a compromise between the two approaches: each object or concept could be described not only in terms of its relationships and relatees, but also in terms of relationships between its relatees. Each concept would, therefore, be a richer and more constrained description, such as depicted in figure 7.8, requiring local consistency when matching it against another concept or instance. Labyrinth employed this idea in its basic form, although only for substructure. This scheme would reduce the potential for parallelism, and would require a more complex algorithm, but would also enable more accurate similarity scores to be produced. Although each cnote would require more effort to evaluate, more cnotes could be rejected quickly. The algorithm would require special purpose mechanisms to detect and deal with parts that play multiple roles.

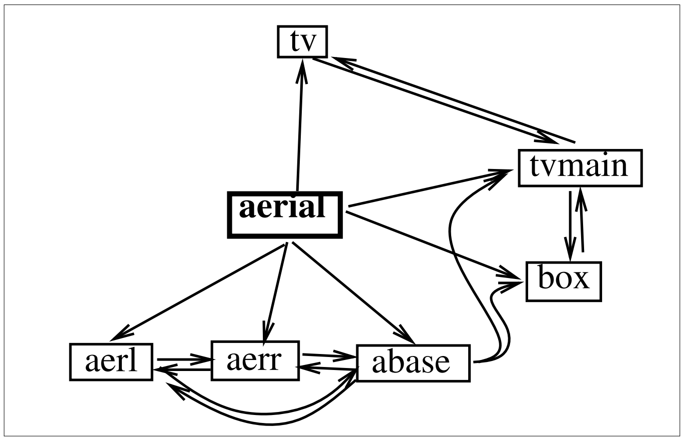


Figure 7.8: Description of an 'aerial' concept in an extended representation.

A more thorough analysis of the current system may reveal other ways to improve GRAM's performance, such as taking into account the relative 'winningness' of cnotes when evaluating their scores, thus enforcing a kind of 'soft' consistency while still allowing objects to play multiple roles.

The results of GRAM could perhaps be improved by refining the mechanisms for attribute normalisation and comparison. For the current system there has been minimal analysis of whether the similarity scores produced for different kinds of attributes are meaningfully comparable. They need to be more rigorously normalised so that a measure of, say, 0.7 for a size comparison has the same meaning as a score of 0.7 for a position or shape comparison.

7.2 Efficiency of the Matcher

This section evaluates the efficiency of the matcher by measuring the time taken to perform the match. Limitations in memory, time-sharing, and a very poor garbage collector in the Lisp used, meant that elapsed times did not give a correct indication of the cost. Therefore, the times presented are the actual CPU time used, excluding garbage-collection time.

The first graph in figure 7.9 shows the times taken to match various components of the bicycles, ranging from the small *BHUBSTUFF* part to the bike as a whole. The results shown are for different amounts of spread effort, ranging from spread-3 for *BHUBSTUFF* to spread-6 for the bikes as a whole, as indicated on the graph. The amount of effort chosen is the effort level above which the correctness did not increase significantly. The results for each comparison all have a correctness of around 93% or above, and further effort did not raise the percentage more than 1 or 2 percent. The number of parts, the correctness, and the average number of relationships for each part ('avrl') are all shown on the graph for each of the four objects.

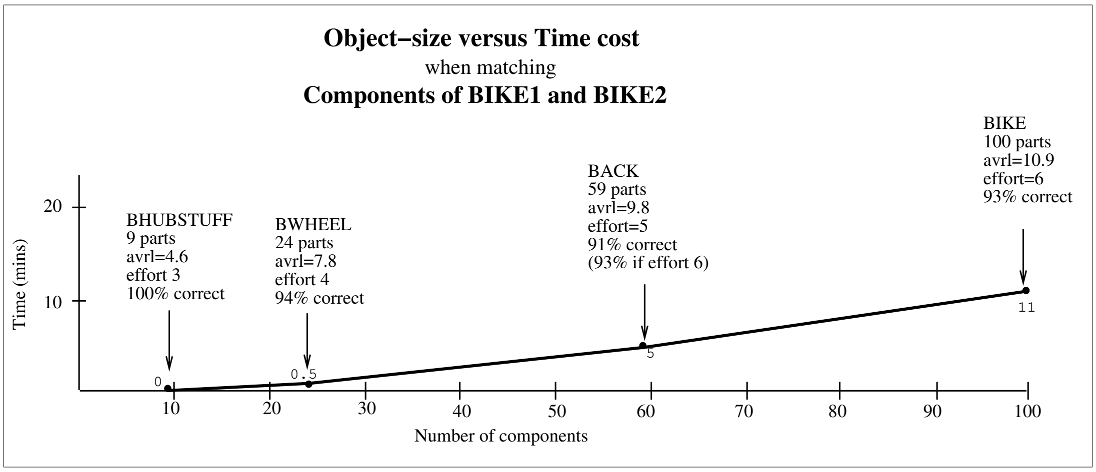


Figure 7.9: Object size versus Time

It should be noted that the efficiency of the implementation could be improved significantly by many basic coding improvements, and by compiling the code with the option of maximum-speed/minimum-safety, instead of the inverse. Coding the system in C rather than Lisp could also help significantly. Therefore the actual times shown in the graph are not particularly important in this discussion. The use of parallelism would also make a huge improvement, as is discussed later in this section.

The significant aspect of this graph is that it indicates that the GRAM matcher is *linear* in object size. This result is confirmed by the graph in Figure 7.10 which shows the number of spread-1 cnotes that were created for each of the different matches. The creation of spread-1 cnotes is the elementary operation for cost analysis because, firstly, the most expensive aspect of matching is comparing all pairs of relationships of two parts when performing an spread-1 comparison, and secondly, because the average number of relationships of a part within an object is relatively constant for large objects, rather than being dependent on object complexity. The latter is due

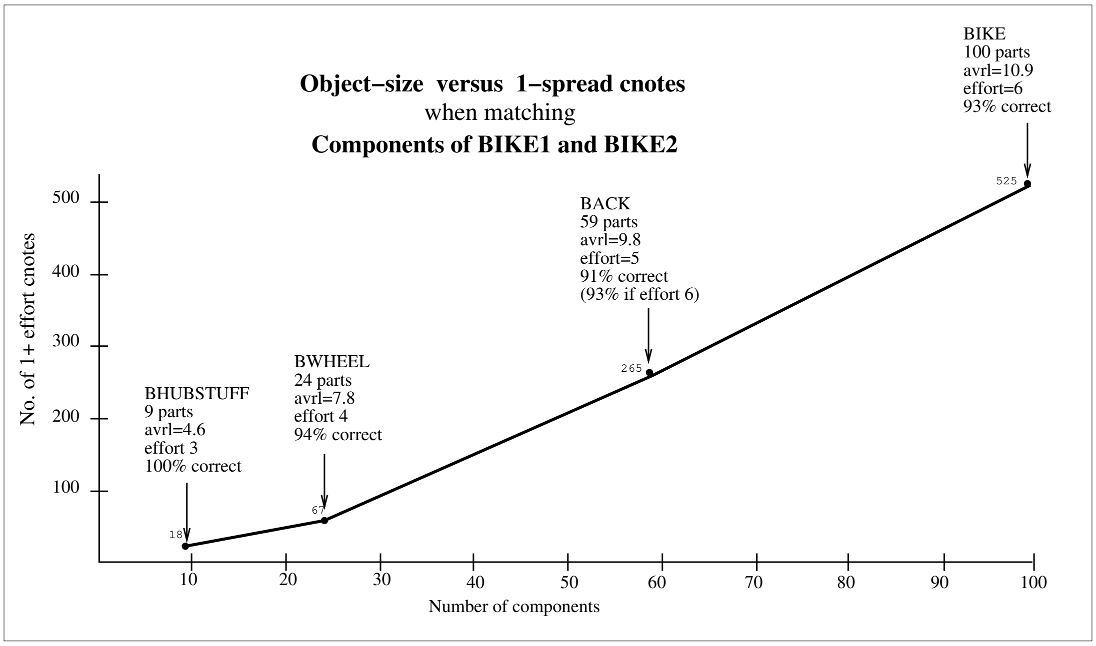


Figure 7.10: Object size versus 1-spread cnotes

to GRAM's instance constructor which constrains how many relationships are created for each part, and is confirmed by the graph in Figure 7.11. This graph shows the average number of relationships for the parts of several components of *BIKE2*, treating each one as a separate object, ignoring contextual relationships to the rest of the bicycle. It can be seen that the average number is converging to approximately 13, although this would have to be verified for other objects. For any objects more complex than about 100 parts, each cnote requires about 200-300 relationship correspondences to be evaluated, since all combinations of parent, neighbour and subpart relationships must be considered, for each of four axis correspondences.

The reason for the linearity of the size-versus-time graph is that the algorithm only spreads from plausible correspondences, and each part of an object usually only has a few plausible correspondences (when constrained by both substructure and context), even in a complex object such as a bedroom. Since the major cost of evaluating each additional cnote is the spread-1 evaluation, which is basically independent of object size for objects larger than about 100 parts, the algorithm is roughly linear.

7.2.1 Comparison with an exhaustive and 'all-pairs' strategies

The efficiency of the GRAM matcher can be contrasted with the efficiency of two alternative approaches. Firstly, an exhaustive comparison of two objects that requires global consistency would require roughly $n!$ sets of correspondences to be evaluated, where n is the number of parts in each of the objects. Since each set requires the evaluation of $n * 4$ cnotes, where 4 is the number of axis correspondences that need to be considered, the algorithm be of the order of $4 * n * n!$. Even an object with 7 parts would require 141,000 cnotes to be evaluated. In

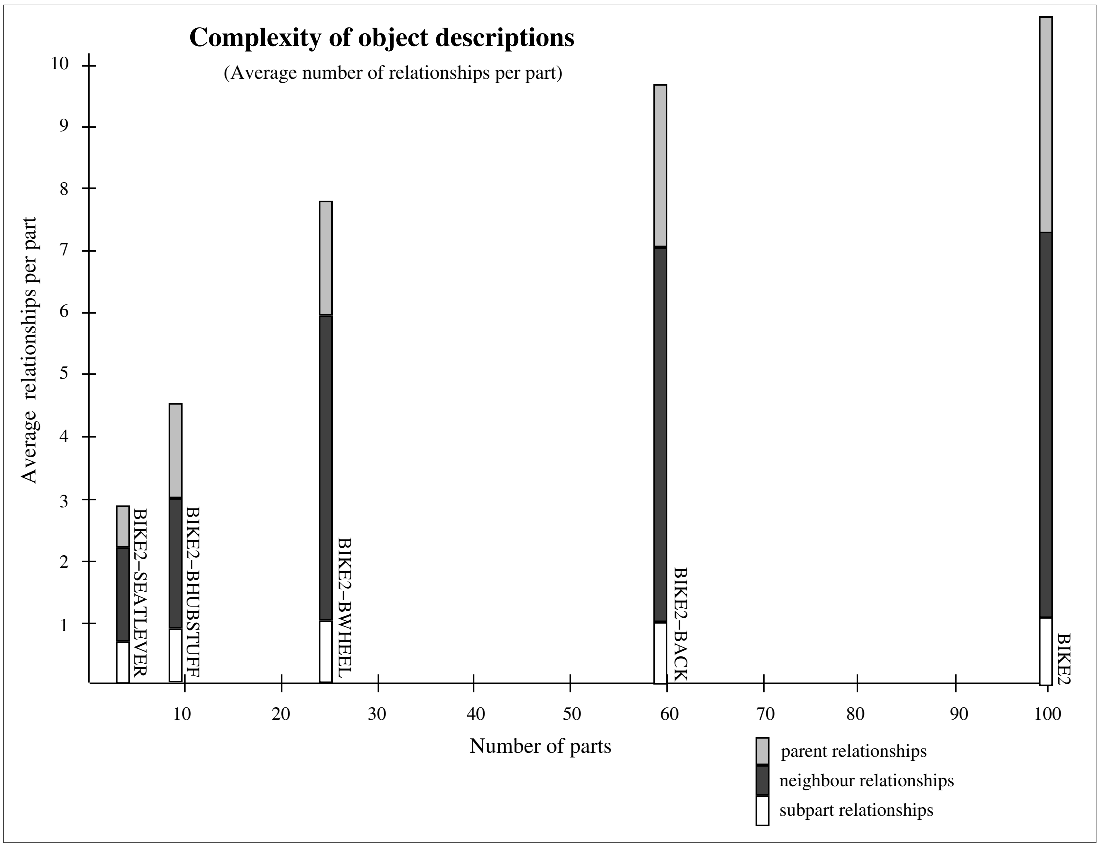


Figure 7.11: Object Description Complexity

contrast, the spread-6 comparison if *BIKE1* and *BIKE2* only requires 525 cnotes. If we let n be 90 (the average of the size of *BIKE1* and *BIKE2*, 80 and 100 respectively), then the number of cnotes is only $6 * n$.

Secondly, a comparison could be done by matching all pairs of parts, without requiring global consistency. If such an approach was as efficient as GRAM's approach then it would indicate that the search algorithm does not provide much advantage. However, the all-pairs match requires the creation of $n * n * 4$ cnotes, or approximately 36,000 cnotes, which is 64 times as many as for the GRAM match, and grows quadratically with the size of the objects involved. As an experiment, a simple program for performing an all-pairs comparison was implemented. First a 0-spread match was applied to all pairs of parts, and this achieved 62% correct winning correspondences. This required about twice as many 0-spread cnotes to be produced as the GRAM algorithm did when applying a 6-spread effort, the latter of which found almost all of the correct correspondences anyway. However, it should be noted from this result that a 0-spread match is still quite effective, with very low time cost since 1-spread cnotes are much more expensive. Thus if GRAM's property attributes were enriched further, then it might be worth doing an all-pairs 0-spread match (potentially in parallel) before applying the spreading activation algorithm, since the GRAM algorithm could then make use of these results from the beginning of the search to help prune relationship correspondences more quickly.

When a 1-spread match was applied to all pairs of parts, requiring the creation of 36,000 cnotes, each with an average of 220 relationship comparison notes, the system ran out of memory and aborted. This confirmed that it is an advantage to employ a search algorithm such as used in GRAM, which results in a large reduction in 1-spread cnotes and, more importantly, seems to have linear growth with the size of the objects.

These results confirm the claims of this thesis that efficiency of matching can be improved by exploiting the structure of objects to guide the search, and by relaxing the requirement for global consistency. This claim was not made and demonstrated for complex objects (or even simple objects) by the other matching systems discussed in section 1.2.

7.2.2 Efficiency of matching identical objects

The final efficiency test to be discussed is the cost of matching two identical descriptions of the same object, *BIKE1*. The time required to match the descriptions with effort 7 was 90 seconds, and further effort would have added almost no more time, since 100% correct correspondences were already found. The more useful and impressive statistics are the number of cnotes created, as shown below:

Object size	80 parts
Time	1.5 minutes
0-spread cnotes	115
1-spread cnotes	88

These costs are significantly less than the costs of matching *BIKE1* with *BIKE2* because the search could directly find correct correspondences from each new cnote created, without having to bypass unmatchable components or discover the correct correspondences via level-hopping.

Since *BIKE1* has 80 parts, these results suggests that the number of 1-spread cnotes is O(n) when matching identical objects, which is a very positive result. Further experiments with other large everyday objects would be necessary to determine whether this is typical.

7.2.3 A summary of the bicycle matching results

To summarise the above results of bicycle matching, the following table specifies the effort applied, the correctness, and the number of 1-spread cnotes created for each of the three experiments: (a) *BIKE1* matched against an identical description of itself, (b) *BIKE1* matched against a significantly different description of itself, and (c) *BIKE1* matched against *BIKE2*

BIKE1 matched against:

	BIKE1	BIKE1(diff)	BIKE2
Object sizes	80:80	80:80	80:100
Effort	7	8	6
Correctness	100%	98.5%	94%
	(74/74)	(73/74)	(65/69)
1-spread cnotes	88	588	525

7.2.4 The matcher is conducive to a parallel implementation

A significant feature of the matching algorithm is that it is conducive to parallelism. All of the 250 or so relationship correspondences for a cnote could potentially be evaluated simultaneously, at least to a 1-spread, if not further, since they can be evaluated independently. This is a major advantage of not enforcing global consistency. Also, the attributes within each attribute vector characterising the properties and each relationship of an object could be evaluated in parallel.

7.3 Effectiveness of the Generaliser

The effectiveness of GRAM's generaliser is evaluated in this section by presenting the results of generalising *BIKE1* and *BIKE2* whose part hierarchies were shown in figures 7.2 and 7.5. Generalisation involves spreading through the cnote graph produced by the matcher, producing new generalised parts (or modifying an existing part) on the basis of each cnote with a sufficiently high score, as explained in section 5.3 on page 207. The results of the generaliser can be partially evaluated by counting how many of the correct winning correspondences found by the matcher led to generalisations, and by qualitatively judging whether the generalisations seem reasonable from a human point of view.

Section 7.1 stated that 65 out of the 69 required correspondences were found by the matcher. 57 of these were successfully generalised by the generaliser, and the remaining ungeneralisable parts of each bike were represented as optional parts in the new generalised description.[3] A few disjuncts were formed, such as a context disjunct for the two *DERBAR2*s, due to the differing orientation, position, and direction relationships of the neighbours relative to *DERBAR2*'s primary axis, and perhaps also due to the additional relationship with the bike-stand in bike2.

An interesting feature of the GRAM system is that even when a parent part is not generalisable, some of its subparts may be. For example, although the two *DER*s (the derailleurs) were considered not quite generalisable, all of their subparts were still generalised as a result of the generalisation of neighbouring parts, most probably the *BSPROCKET*. In other words, level-hopping and 'mismatch-bypassing' also occurs during the generalisation process due to GRAM exploiting neighbour relationships. This feature is not present in the other system's reviewed in chapter 2.

However, there were a few problems in the resulting generalised description that need to be pointed out. Firstly, the two *FWHEEL* parts were not generalised even though their context similarity score was high (0.84), because the structure similarity score was just below the generalisability threshold of 0.7, and the context score of 0.84 was just below the 0.85 threshold needed to produce a disjunctive description. This indicated that the generaliser should perhaps use 'winningness' to justify generalisation: although the context similarity score for the front wheels was just below the threshold, the front wheels scored much higher than any alternative correspondence. Furthermore, a number of the *subparts* of the front wheels were generalised, indicating that the front wheels should not merely be added to the new description as distinct optional parts. This issue needs to be explored further, after analysing the results of GRAM on a larger number of test cases.

A few other minor problems occurred due to bugs and minor oversights in the program, and further evaluation of the generaliser on other large and small objects needs to be done before its strengths and weaknesses can be more clearly identified.

[3]The part-graph of the generalisation has not been shown as it requires 6 pages.

7.3.1 Matching and generalising the generalised bicycle.

To test the generalised bicycle description, it was matched and then further generalised with another description of *BIKE1*, although only to spread-4 so it could only reach partially down the part hierarchy. The matcher successfully found the correct correspondences and generalised the larger parts which had been matched to at least effort 2.

Memory constraints meant that a full 7 or 8-spread match was not possible, and so this evaluation is somewhat limited. Furthermore, the lack of a third bicycle description meant that the generalised description was not tested by matching it against a different bike.

A small problem in the generalisation strategy was highlighted by this experiment. Relatee correspondences that scored high, but with insufficient effort to justify generalisation, led to both relatees being added to the new generalisation as optional parts. For example, although the *SEAT*s were generalised, the subparts could not be, and so the resulting *SEAT* had a duplicate set of subparts, some from the original generalised seat, and the others from the new instance. Instead, the generaliser should just ignore these instance parts, perhaps generalising the relationship, but not the relatee. The instance count of the part should however, be incremented, otherwise the subparts might become optional when they shouldn't be, as in the example of the *SEAT*'s subparts.

7.3.2 Disjunction

To demonstrate the formation of disjunctive descriptions, GRAM was given descriptions of the doors in figure 7.12. An initial concept was produced from *DOOR1* and *DOOR2*, and then each of the remaining doors was matched and generalised, one by one, with the concept description, resulting in the part-graph shown in Figure 7.13. Although the main doors were straightforwardly generalised, and the handles were all matched, the internal structures of the handles were quite different. Consequently, the structure of the *HANDLE* part is represented disjunctively by referring to four subconcepts, as indicated by the heavy dotted lines. One of these subconcepts is a generalisation of two of the handles which were considered sufficiently similar to generalise. The second name in some of the boxes is the name of the first instance from which the generalisation was formed, and the number alongside each of these names is the instance count. For example, the concept *C28*, which has a subpart/parent relationship with the generalised handle, *C25*, is an optional subpart of *C25*, with an instance count of 1, formed from the *LOCK* of *DOOR4*.

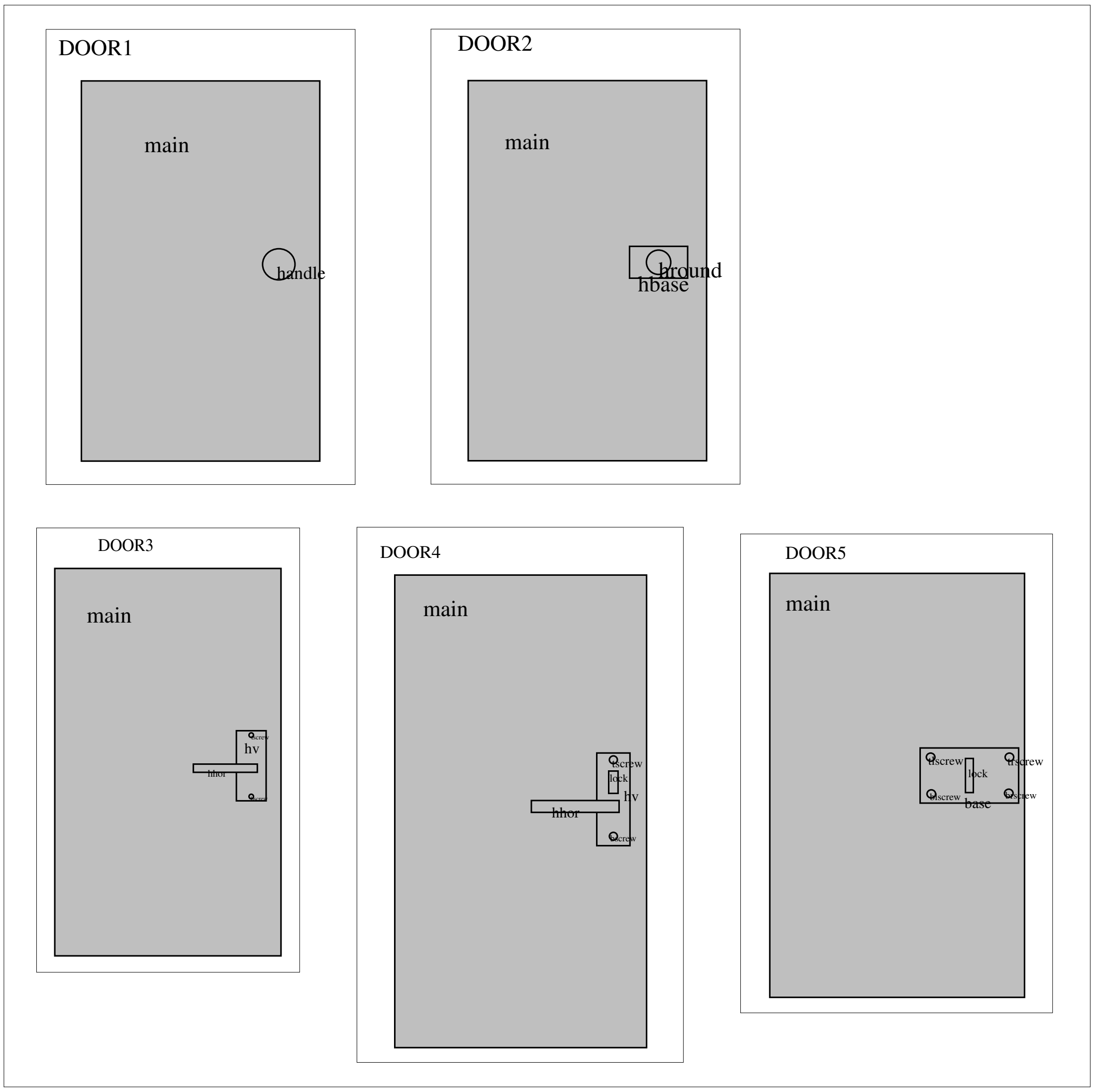


Figure 7.12: Five doors.

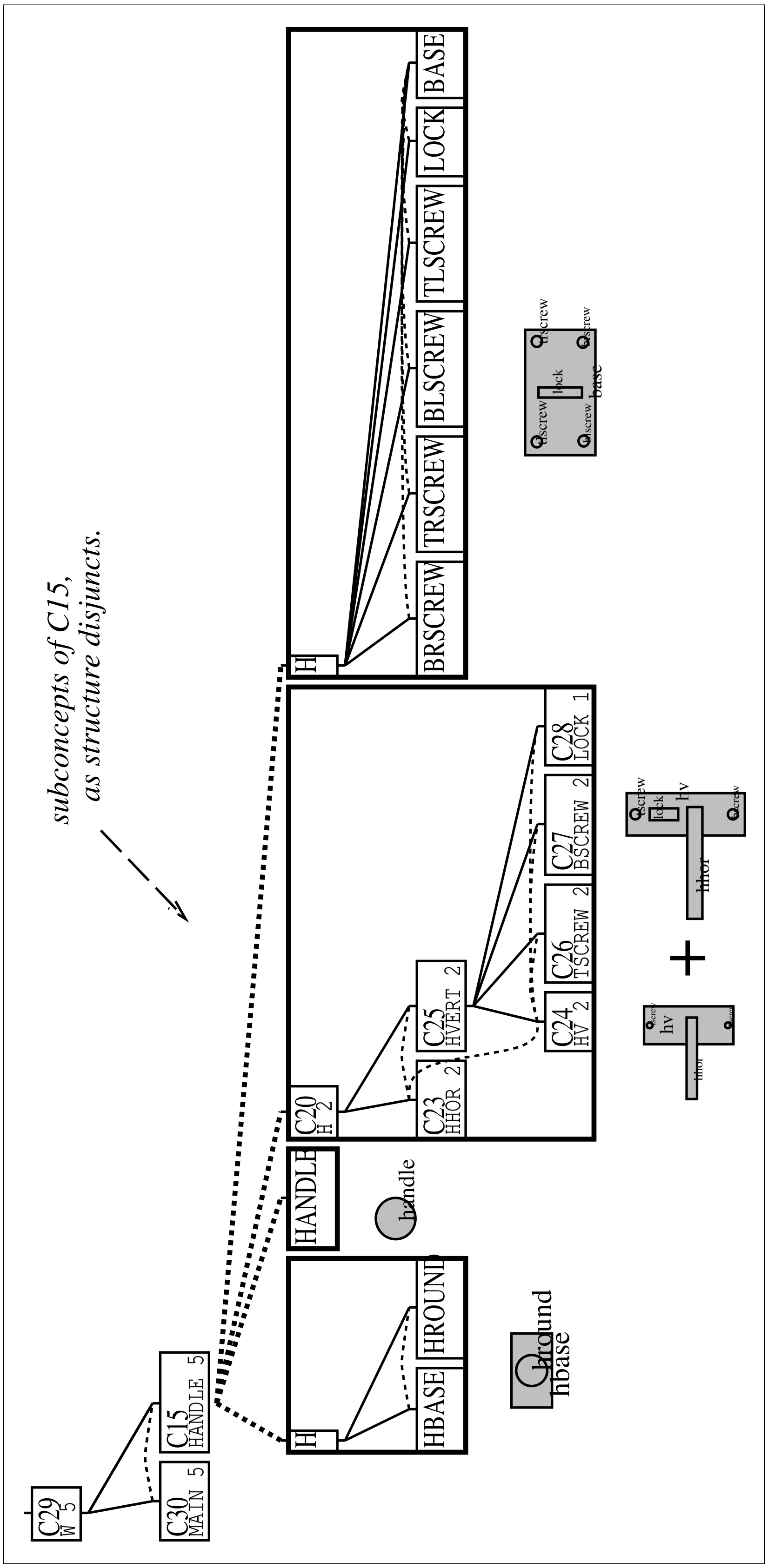


Figure 7.13: Generalisation of five doors.

7.4 Effectiveness of Grouping

One of the ways that GRAM exploits the structure of objects is to pre-process an instance description by forming groupings of similar objects. To partially demonstrate how this can improve match efficiency, this section presents the results of forming groups for the bookshelf in figure 7.14. A list of group members was given to the system, rather than being automatically found, so the results described here only present the performance of the group construction mechanism rather than the group finder[4]. The group constructor must match and generalise the members of each group to form the typical-member description, and then, under certain conditions, remove the individual members.

The bookshelf has been slightly simplified from the bookshelf given on page 23. Firstly, the potplant has been removed because it had a vast number of leaves which required manual input of all the groupings. Secondly, only half of the petals on the flowers have been included. This was because the current group constructor requires members of the group to all share the same axis correspondences, and this is not the case for all of the petals of the full flower due to the way GRAM selects the primary axis of a part. Completing the code to cope with this would be a relatively minor matter.

The top graph in figure 7.15 shows the description of the bookshelf before grouping, and the bottom graph shows the description after groups have been formed, and individual members removed. A readable version of the bottom graph is given in figures 7.16 and 7.17. Clearly there is a dramatic reduction in the size of the description, with the number of parts reduced from 157 to 41, and the number of relationships reduced from 1600 to 399. When the grouped bookshelf description was matched against an identical description the correct correspondences were found by GRAM, but unfortunately memory problems prevented results for the ungrouped bookshelf to be obtained, and so a comparison between the match performance is not available at present. However, it is obvious that the reduction in description size must significantly reduce the match time, as well as memory usage.

The details of the part-graph for the grouped bookshelf can be seen in figures 7.16 and 7.17 each of which shows one half of the part-graph. Notice that the instance-count for C21, the generalised bottle label at the bottom of figure 7.16, is 4 rather than 5, giving a further illustration of how GRAM acquires probabilistic descriptions.

---

[4]the implementation of the group-finder is incomplete, and currently only finds some of the groupings in the bookshelf.

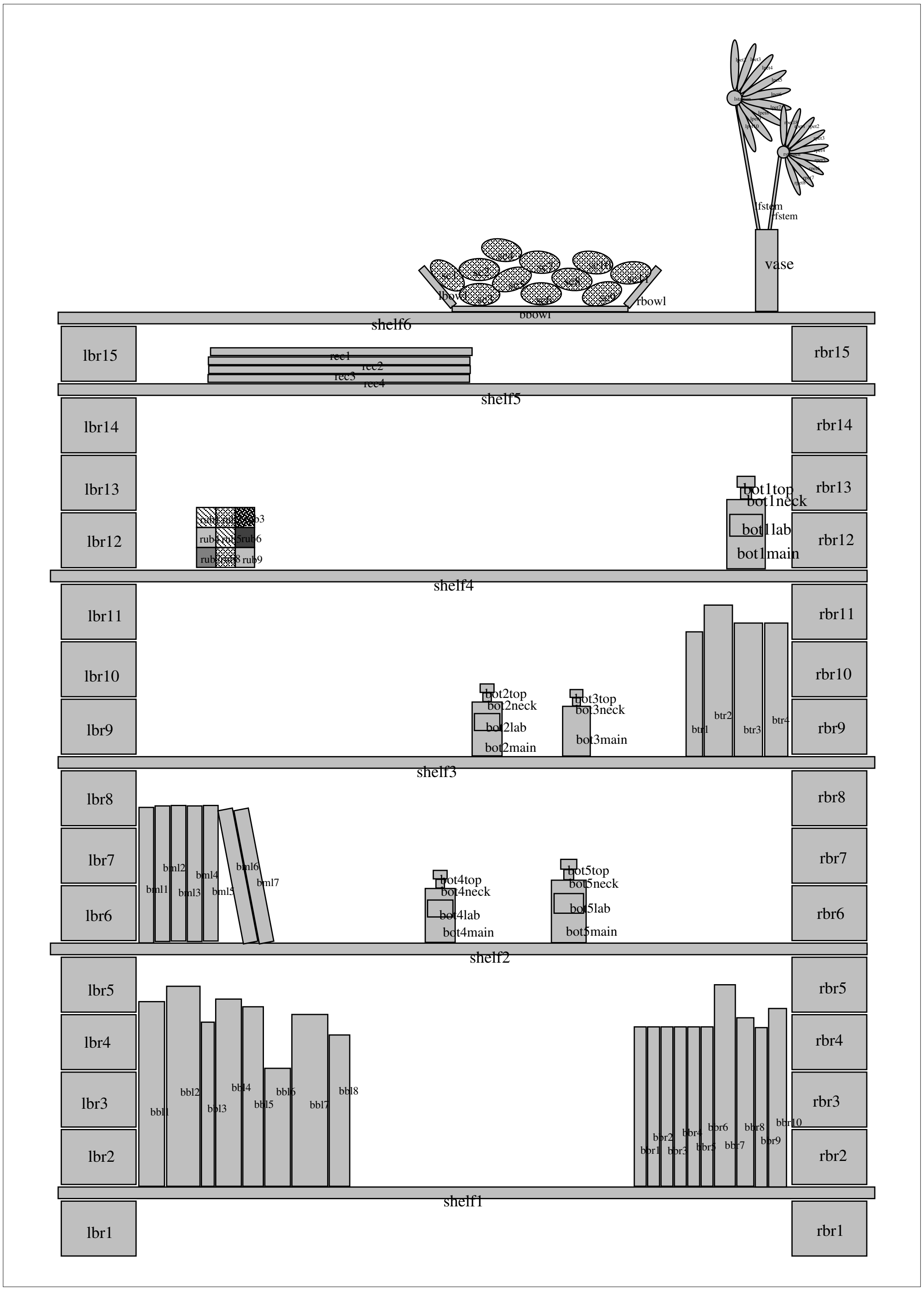


Figure 7.14: A bookshelf

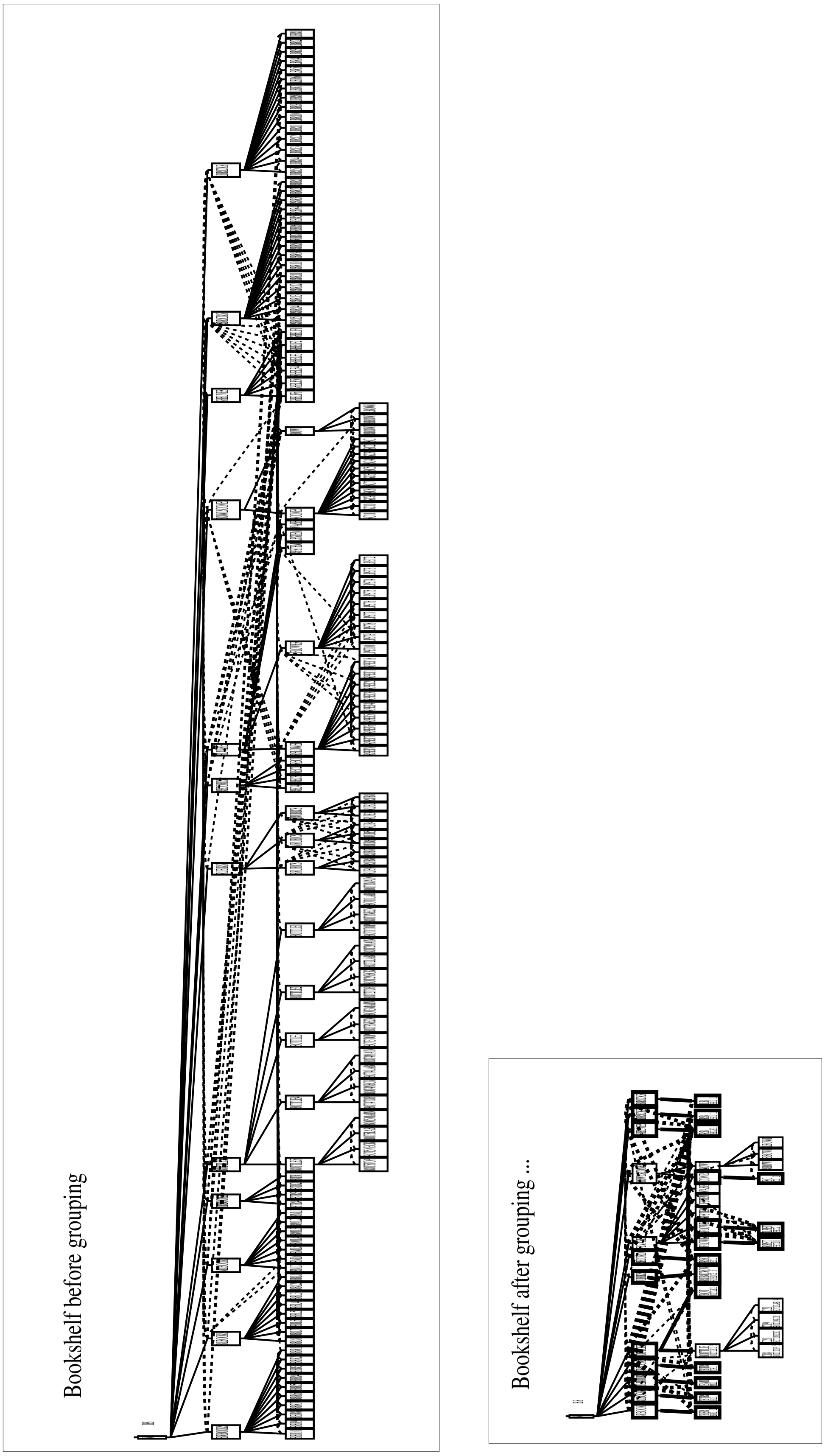


Figure 7.15: The effect of constructing groups

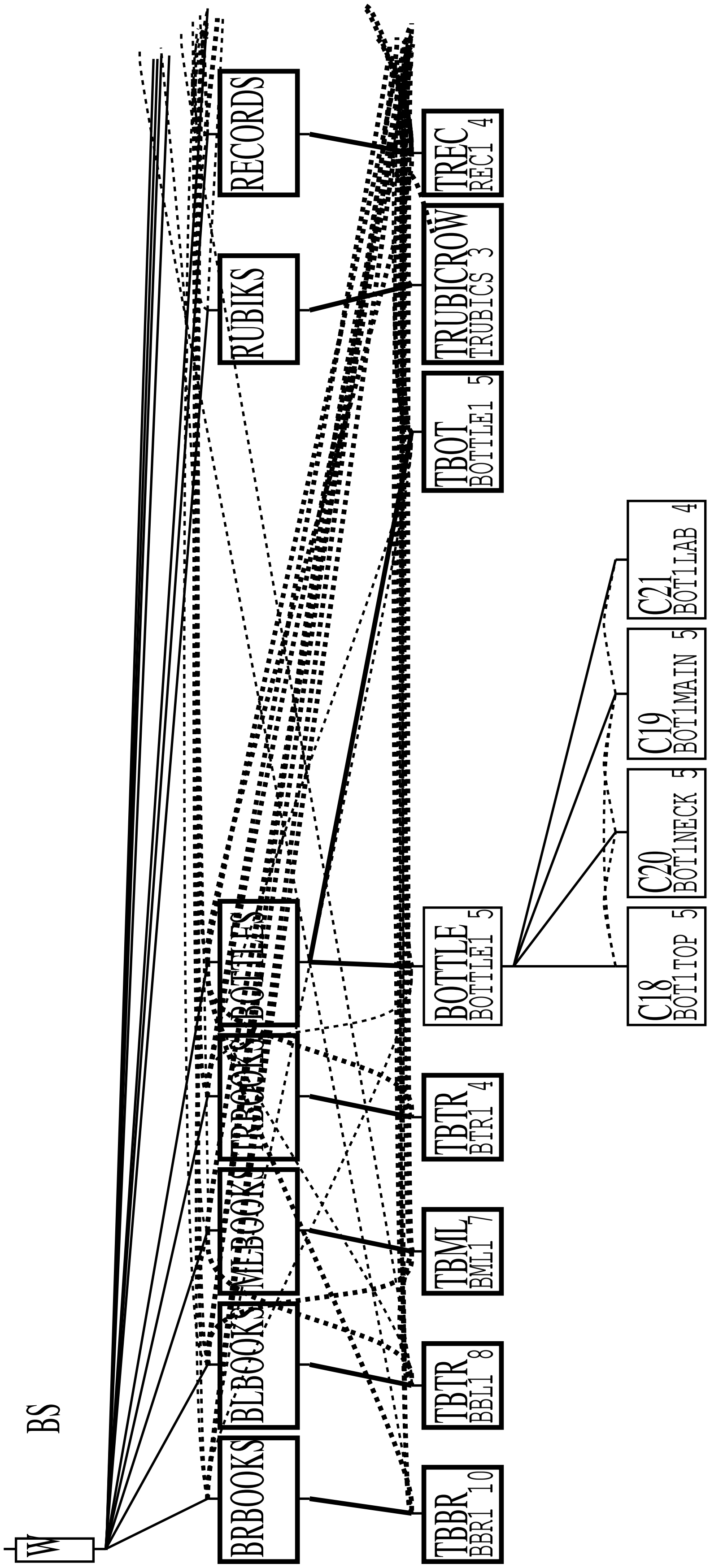


Figure 7.16: A grouped bookshelf

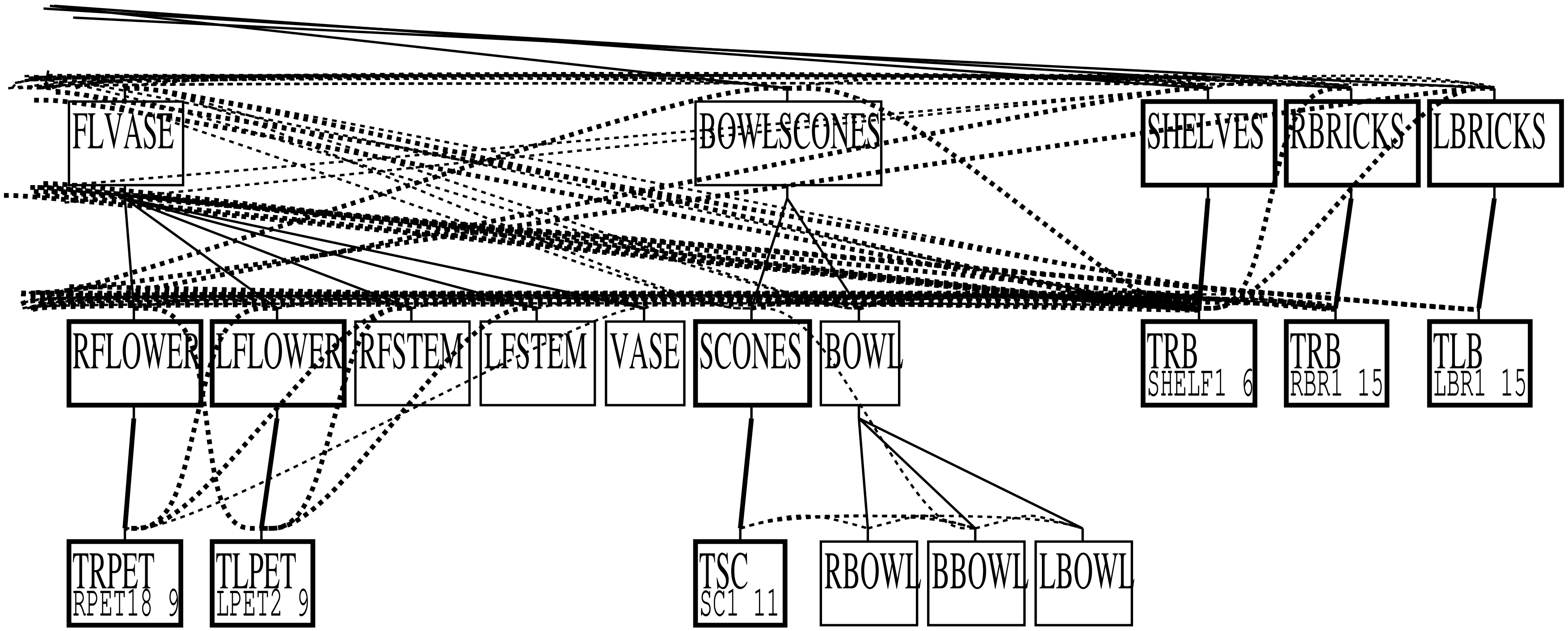


Figure 7.17: ... continued

7.5 Limitations and Future Work

Although the main ideas in this thesis have been implemented, there are a number of aspects of the system described in the thesis that have not been completed or could be improved and extended.

The representation scheme could be enriched to allow a two-dimensional equivalent of the "generalised cylinder" representation[5]. The main requirement for this extension is a low-level vision system that produces generalised cylinder descriptions. The representation scheme needs to be changed only minimally, by adding a few attributes, perhaps also including additional properties such as colour, texture, *etc*. The matching and generalisation mechanisms can be left essentially unchanged. However, it does introduce problems for coercion when comparing similar objects with different primary axes.

Extending GRAM to deal with three-dimensional object descriptions is another area to be considered. This will involve modifying and adding additional attributes that take into account the third dimensional axis. It also requires 24, rather than 4, alternative axis correspondences to be considered for each cnote, suggesting the importance of parallel computation. Enriching the set of object properties could enable these correspondences to be pruned more effectively prior to evaluating relationship similarities. As with the generalised cylinder extension, the main requirement for extending to three-dimensions is that GRAM will need input from a three-dimensional vision system or graphics package, rather than working with input from a simple two-dimensional drawing package.

A more elaborate extension that could be made to GRAM is to include explicit descriptive entities representing the edges of a object (and surfaces of a three-dimensional object). This would mean that an instance graph would contain two or more kinds of nodes (rather than just object nodes), and additional kinds of relationships, such as *edge-of* relationships. The design of the GRAM system is such that this extension may not require significant changes to the matcher and generaliser, since the "spreading" match process, guided and constrained by relationships, could still be applied. However, 'edge' entities would probably be best represented as local to each concept, rather than having additional 'edge concepts', so that an edge cannot be matched or generalised independently from the concept it belongs to.

Another possible extension to the GRAM system is to allow a single relationship to be described disjunctively. Currently it is possible for a concept description to include multiple relationships to the same concept, but it is not possible to explicitly describe a relationship disjunctively, except within a complete structure or context disjunction.

The matcher could be improved by performing a more thorough analysis of attribute similarity so that attributes values can be normalised and compared in a more consistent and effective way. The heuristics for pruning the search could probably also be improved after analysing the behaviour of the matcher more thoroughly. Estimates for upper and lower bounds could be made more accurate. It may also be worth making discontinued cnotes able to be recovered

[5] The Generalised Cylinder scheme represents objects in terms of a central spine and a cross-section that sweeps along the spine according to some function.[Brooks, 1981]

(by reevaluation) since the discontinuation and removal of a cnote is based on lower and upper bounds which are only estimates. The various expressions used to compute scores and weights also need to be investigated more thoroughly.

A limitation of the matcher is that it only coerces object descriptions based on rotations of 90, 180, or 270 degrees. Two triangles that require a 120 degree rotation to correspond correctly, cannot be matched accurately by GRAM. The coercion mechanism would need to be extended somehow to cope with such situations, perhaps taking into account the shape of the objects. Currently GRAM treats all parts, whether circles or polygons, as each being bounded by a rectangular box, and so is limited to 90 degree rotations.

Currently the fault-finder produces a rather detailed report only meaningful to the author. Future work could be done to produce a more summary report which identifies mismatches at whatever level of detail is required, in a more readable form with statements such as "*BWLEVER* is missing from the *BWHEEL* of *BIKE2*".

The mechanism for computing 'contents-similarity' scores has not be implemented. This mechanism needs to match all pairs of relatees, ignoring relationship comparisons, and to do this for every cnote would undermine the strategy of pruning correspondences on the basis of relationship similarity scores prior to comparing relatees. Therefore, contents-similarity scores would need to be only computed if there is justification, and this has not been explored sufficiently in this thesis. One possible justification is high context similarity and a strong winning margin relative to competing cnotes, but a poor structure match. Another justification is when a teacher, the environment, or reasoning mechanisms indicate that two objects correspond, but where the structure similarity is low.

Another difficulty with implementing contents-similarity is that it requires the subparts of the objects to be matched in isolation, ignoring context, as discussed in section 4.3.7. Although GRAM is able to perform nested scoping, by temporarily marking parts with a timestamp, an additional mechanism needs to be developed to enable the *scores* to be nested, so that the nested match will use the scores obtained from structure-only scoping, while the enclosing match will use the full scores. This may somewhat complicate the spreading-activation algorithm.

However, although contents-similarity has not been implemented, one of the main reasons for using it was to enable objects such as *rooms* to be matched and generalised. When GRAM was applied to the bedrooms in Figure 7.18, it found all of the correct correspondences anyway, and produced a correct generalisation, with an optional computer on the desk, a generalised grouping of desk drawers, and a disjunctively-described clock context.

There are a few aspects of the generalisation process described in the thesis which have not been fully implemented. One is that the generaliser does not invoke the grouper to resolve 'similar-similarity' ambiguity, or at least create multi-relationships. This depends to some degree on the group-finder, since it must evaluate the groupability of the ambiguous relatees. Only the basic mechanism of the group-finder has been roughly implemented so far, and so this form of ambiguity resolution has not been completed. The generaliser could be further extended to create new composite objects as a way of resolving ambiguity or apparent mismatches. Currently it is only able to create new relationships.

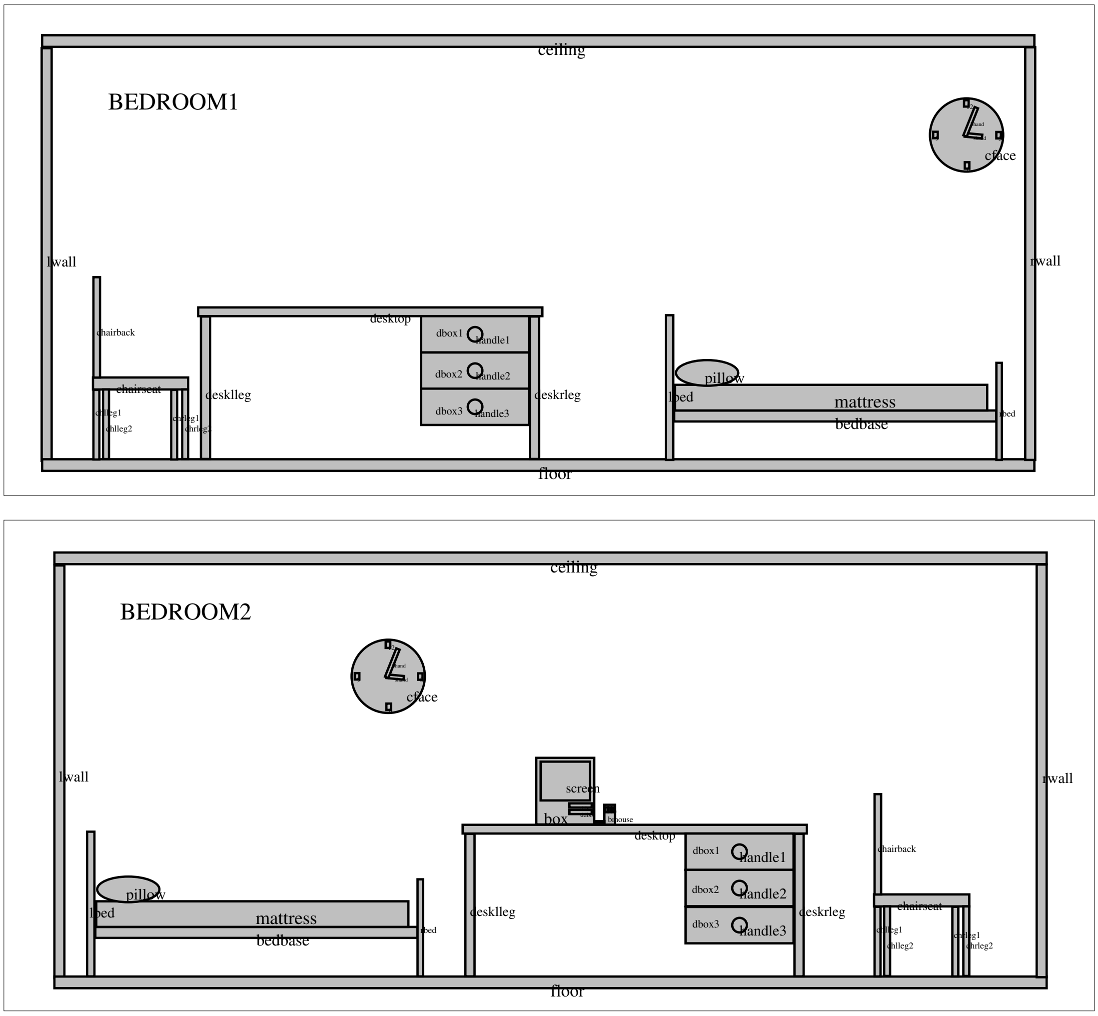


Figure 7.18: Two bedrooms

Also, the generaliser currently only uses *proximity* scores to determine generalisability, rather than *fit* scores. This is because only a simplified fit-scoring mechanism has been implemented which only considers the fit-scores of the properties and relationships of two parts, rather than spreading further outwards through the cnote graph. It also only computes fit-scores for numerical attributes and not for nominal attributes. This simplified mechanism has been tested by generalising eight very simple four-subpart chairs, and then matching them with a number of 'bad chairs' with faults such as a back tilted forwards. The simplified fit-scoring successfully noticed the incorrect tilt, due to the low variance of the generalised tilt, while proximity scoring considered the tilt acceptable.

The distinction between fit-scoring and proximity-scoring for the generaliser is only significant for concepts that have been formed from at least four or so instances, since the variance of numerical attributes formed from fewer instances is not considered reliable enough to be used in fit-scoring to determine faults. So the use of fit-scoring by the generaliser would not affect results such as those discussed for the bicycles.

Other minor extensions to the implementation are to enable it to merge disjuncts, correctly match and generalise multi-relationships with several normal relationships, and generalise grouped and ungrouped objects. The instance constructor cannot currently produce the 'coverage' profile attribute described on page 82. The disjunct-formation system also needs to be modified slightly so that the disjuncts of a concept have the same axis correspondence, or that the axis correspondence of the disjunct is recorded explicitly relative to its super-concept so that the matcher can coerce attributes appropriately.

Not all combinations of structure and context *interpretations* (section 3.4.3) are matchable or generalisable. Specifically, '*any*' and '*partial+typical*' interpretations are not dealt with, and it is not possible to match or generalise two generalised concepts that *both* have '*disjunctive*' or '*imported*' interpretations. Although GRAM can construct and match a description with an '*imported*' interpretation, it is not currently able to decide itself when the structure or context of an object should be replaced by an *import* reference.

GRAM's instance constructor needs to be extended to implement some of the composite-object creation mechanisms that have been suggested in the thesis, such as those based on symmetry and topology. The group-finding mechanism could also be extended to find sequences of alternating objects (called *cycles*).

The issue of two-way interaction between the matcher, the instance-constructor, and the low-level vision system has only been minimally addressed in the thesis. In a three-dimensional domain, where objects are always partially obscured, this issue is especially relevant. Ideas developed by [Brooks, 1981] could be applicable here, and the recent ideas discussed by [Winston, 1992] which show how recognition is possible from just a few two-dimensional models, could also be relevant.

Since GRAM's representation scheme and its matching and generalisation algorithms are amenable to a parallel architecture, future versions of GRAM could be implemented in this way. The GRAM scheme may also be conducive to some kind of connectionist implementation, since one of the difficulties of implementing a structured object learning system in a connectionist architecture is the problem of maintaining consistent bindings between components, which is not an issue in the GRAM system.

The most difficult and long-term extension of GRAM is to enable it to learn multiple concepts organised and indexed in a concept memory, perhaps drawing on and extending the ideas used in the Labyrinth system described in section 2.8. Although the matcher and generaliser do perform classification and generalisation of multiple concepts by 'spreading' along parent, neighbour, and subpart relationships, the thesis has not dealt with how to create, use, and maintain AKO hierarchies.

Clearly there is a great deal of scope for future work on the GRAM system.

Chapter 8

Conclusion

This chapter summarises the main ideas and conclusions presented in this thesis. The GRAM system has extended and integrated various ideas from other systems, and has also presented a number of new ideas for representing, matching, and generalising descriptions of complex physical objects.

A central contribution of this thesis has been to demonstrate that complex physical objects can be matched without functional or domain-specific knowledge. The thesis has also shown that the structural relationships of objects can be exploited to improve the effectiveness and efficiency of objects. Other systems (such as Labyrinth [Thompson and Langley, 1991] and MERGE [Wasserman, 1985]) exploited the decompositional nature of physical objects to enable top-down matching, but GRAM has extended this to make use of neighbour relationships, thus allowing the matcher to search in any direction through the object graph, guided and constrained by the object structure.

Another important contribution of the thesis is the idea that complex structured objects can be effectively matched and generalised without having to enforce consistency between correspondences. This allows the mechanisms and representation scheme to be simpler, more efficient, more robust, and more conducive to a parallel implementation.

A methodology that has been used throughout the thesis is that the choice of representation scheme, and matching and generalisation mechanisms, must be based on first identifying the *kinds* of situation that must be representable, matchable, and generalisable, rather than creating a mechanism and then identifying what it can and cannot do.

The following sections of this chapter summarise the specific contributions of the thesis for each component of the GRAM system: representation, matching, generalisation, and instance-construction.

8.1 Representation

8.1.1 Multiple levels of approximation and abstraction are important for matching and generalising.

The technique of 'chunking' an object into abstract or approximate components at multiple levels of detail, represented as a decomposition hierarchy, enables the matcher to be guided by the structure of the objects. This idea has also been applied in other systems such as Labyrinth, MERGE, and ACRONYM [Brooks, 1981]. It also enables two descriptions, that perhaps would be otherwise unmatchable and generalisable, to be matched and generalised at a coarse level of detail, as has been discussed by [Marr, 1982]. Approximation and abstraction also allows more information to be made explicit, since properties and relationships of the whole 'chunk' can be specified.

Multiple levels of detail are especially important in the GRAM system, since to match and generalise objects as complex as the bicycles shown in Figure 1.8 in chapter 1 without the guidance of a decomposition hierarchy is computationally very expensive. Systems such as CLUSTER/S [Stepp and Michalski, 1986a] and MARVIN [Sammut and Banerji, 1986] do not use decomposition hierarchies to guide the matcher, and this means that they are not able to efficiently deal with complex structured objects.

The PARVO system [Bergevin and Levine, 1993] demonstrates that classification of common physical objects can often be performed from coarse descriptions alone, without requiring finer details to be considered at all. The fact that humans can classify objects drawn as stick figures or simple cartoon images is further evidence. This is an important domain characteristic since it makes rapid classification possible.

Nevertheless, finer details are necessary for generalisation and fault-finding, and to enable the matcher to distinguish between specialised varieties of a class of objects.

8.1.2 The distinction between parent, neighbour, and subpart relationships helps guide and constrain the matcher.

Unlike representation schemes that allow arbitrary atomic relations between objects, (such as [Sammut and Banerji, 1986], [Winston, 1975], and [Connell and Brady, 1985]), GRAM instead distinguishes between just three types of relationship: (a) parent-relationships with enclosing parent objects, (b) neighbour-relationships with other objects that are connected, close, or otherwise interestingly related, and (c) subpart-relationships with subcomponent objects. These are not atomic relations, but are richly expressive descriptive entities consisting of both qualitative and quantitative information.

The main advantage of having just these three types of relationship is that each relationship acts as a richly labelled link in the object graph, where the pattern of links captures the structure of the physical object, and allows the matcher to exploit the structure of the object to guide the search for correspondences.

Other systems, (such as Labyrinth and MERGE) have made use of this principle to some degree by the use of subpart relationships to guide a top-down search, but GRAM's use of parent and neighbour relationships extends this principle further. In particular, neighbour relationships provide more "access paths" to an object, and these can cross levels of the decomposition hierarchy. This helps resolve the *level-hopping* problem in which correspondences between the components of two similar objects cannot otherwise be found because the components are on different hierarchical levels.

The richness of relationship descriptions enables the matcher to compute a rough similarity score for two descriptions merely by computing similarity scores of properties and relationships, without even comparing the descriptions of the parents, neighbours, and subparts. This means that an instance can be matched with a concept very cheaply, while still giving a reasonably good estimate of their similarity. A high similarity score would justify a more complete comparison.

Furthermore, an advantage of combining all the information about the spatial relationship between two objects into a single descriptive entity (represented as an attribute vector) is that the matcher can compute a single similarity score for two relationships, and hence more efficiently determine the best correspondences between the parent, neighbour, and subpart relationships of two objects, without even comparing the relatee objects.

8.1.3 Generalisation is simplified by giving each concept and instance its own set of relationships.

In GRAM, a relationship between two objects or concepts is duplicated, and one copy is stored with each object description. This enables concepts to be generalised independently. For example, if an observed instance matches a concept sufficiently well to justify generalisation, then each of its matched relationships can be generalised, even if it refers to a concept that is not itself matched well enough to be generalised. This characteristic of the representation supports GRAM's overall approach of the avoiding the need to deal with consistent correspondence bindings.

8.1.4 Physical objects are represented in terms of context as well as structure.

Many classes of physical object are defined not only by their structure (or 'form'), but also (or even primarily) by their context (or 'role'). For example, the *chair-leg* concept is defined largely by relationships with the concept *chair* and other chair components. Even if a concept is primarily defined by its structure, as for the concept *mug*, it is useful to include context information in its description since that can help to make predictions. For example, to find a mug in a kitchen it is useful to know that mugs are often on shelves in cupboards, or by the sink. GRAM's use of neighbour and parent relationships, in addition to subpart relationships, enables such knowledge to be represented.

8.1.5 Concepts can be conveniently defined by relationships to other concepts, rather than by a local part graph.

An earlier version of GRAM [Andreae, 1993] represented a concept as a complete part-graph, where all parts of the graph were local to that concept description. If the generaliser determined that it was necessary or desirable to represent a subgraph as a concept in its own right, then it had to be extracted out, with references from the original graph to the new concept. This process was somewhat complex, and many of the issues of deciding when and how to do the extraction were not resolved. The matcher was also complex, because it had to find a consistent set of one-to-one bindings between two potentially large part-graphs, which could contain embedded disjunctions and references to other concepts.

In the current version of GRAM, each concept description is much smaller, simpler, and more homogeneous, since it is defined only by a set of properties and a set of relationships to other concepts, with no local part graph. There is no distinction between concepts and parts of concepts. Consequently, the matcher does not have to deal with bindings, and the generaliser does not have to determine whether a subgraph should be extracted out as a global concept.

8.1.6 The explicit distinction between structure and context supports partial matching and a simple form of disjunction.

An important aspect of GRAM's representation is that it not only allows concepts and instances to be defined in terms of both structure and context, but it also explicitly distinguishes between these two types of information. One reason for this is to enable partial matching: The matcher can produce separate similarity scores for structure and context, and therefore is able to notice that two objects are similar in structure but not context, or *vice versa*. If the similarity is sufficiently strong, and is higher than scores for other competing correspondences, then this is considered sufficient justification for generalisation, even though the overall combined similarity score is poor. Without this distinction, a poor overall similarity score could mean that both structure and context similarities are poor, in which case generalisation would not be justified.

Another reason why the distinction is useful is that it enables a simple form of disjunction to be included in the representation scheme, without having to deal with arbitrary disjunctions of properties and relationships. More specifically, GRAM allows the structure and/or context of a concept to be defined disjunctively.

8.1.7 Explicit groups reduce memory usage, support efficient matching, and enable different-sized collections of similar objects to be generalised.

It is essential to be able to explicitly represent groups of similar objects (within a larger object or scene) as a single entity, characterised by its typical member, since groups are so pervasive in the physical world. An explicit group description not only reduces memory usage by summarising multiple descriptions into a single description, but also supports more efficient matching (since fewer items need to be compared) and enables a generalisation to be produced even if the groups of two descriptions have different cardinalities.

Therefore, an important contribution of the GRAM representation is that it allows groups to be explicitly represented, matched, and generalised. This was not supported by the other structure-learning systems reviewed in this thesis, except for Winston 's ARCH learner [Winston, 1975] and (in a limited way) Brooks's ACRONYM.

The idea of representing a group in terms of a generalised typical member was initially proposed and implemented by Winston. GRAM extends his representation to cope with more complex structures, a more unified representation scheme, and probabilistic generalisations. Most significantly, the description of the typical-member of a group is a concept that can have relationships to itself, which specify the typical inter-member relationships between the members of the group. This idea seemed to be partially included in Winston's system, but only by the typical member referring to a "another member" node, rather than by referring to itself, which is more compact and homogeneous.

This thesis has identified a number of different types of groups (such as chain, loop, array, and cluster) and GRAM's method of representing typical inter-member relationships allows these different types to be reflected implicitly in the set of typical-intermember relationships, and explicitly in the *group-type* property.

Items in a group sometimes contain groups themselves, and so it is necessary to allow the typical-member description to be a grouped concept. This is supported by GRAM, since a typical-member concept is just an ordinary concept.

8.1.8 Multi-relationships allow relationships to be grouped.

In addition to representing similar and similarly related objects as a single group object, GRAM also allows several similar relationships of a concept or instance to be represented as a single generalised *multi-relationship*, with a *howmany* property specifying how many relationships are being summarised.

This is important because for every group object, there are usually several other objects that are structurally related (as neighbours) to some or all of the members of the group. Although these relationships could be replaced with a single ordinary relationship to the group as a whole, this would lose information about how the object is typically related to the members of the group. Thus the multi-relationship is necessary. It is especially necessary if the related objects have not been explicitly represented as a group.

Brooks's ACRONYM supported this in a more restricted manner by having a 'quantity' parameter associated with an 'affixment' relation to a component. None of the other systems reviewed in this thesis supported multi-relationships.

As with groups, multi-relationships reduce memory usage, support more efficient matching, and enable a generalisation to be produced even if the multi-relationships of two descriptions summarise different numbers of relationships. One conclusion of this thesis is, therefore, that a representation for a complex structured domain should allow repeated features to be summarisable in a single descriptive entity.

8.1.9 Instance-counts are necessary to specify the degree of optionality of a component.

Many classes of physical object must be described in terms of optional parts, such as the aerial of a television set, the arms of a chair, or the keyhole in a door. If the optionality of a part cannot be represented, then either the part must be dropped from the object description altogether, or two subconcepts must be specified, one of which includes the part, and the other which does not. A more convenient method is to allow each part to be labelled as being optional or non-optional.

However, it is useful to also know the *degree* of optionality. For example, if only a small proportion of cars have a light on the top, then this should be explicitly specified to enable more precise prediction and similarity evaluation. Therefore, GRAM, like COBWEB [Fisher, 1987a] and Labyrinth, allows each concept feature to have an *instance-count* that indicates how many instances of the concept included that feature. Instance counts are associated with each property, each parent, neighbour, and subpart relationship, each *import-from* specification, and each concept as a whole.

8.1.10 The distinction between *contents* and *arrangement* is necessary in some domains.

Some concepts, such as types of rooms or places, may be defined primarily by their *contents*, with less importance placed on the *arrangement* of their contents than for objects such as chairs and handdrills. Originally it was thought that the representation scheme should explicitly distinguish between arrangement-dependent and arrangement-independent concepts. However, it was realised that the distinction is a matter of degree, rather than one or the other, and it has been found that the arrangement-independence of a concept is adequately specified by the *variance* of properties of objects and relationships: a generalised subpart relationship that has a high variance will usually match many of the relationships of an observed instance, and therefore the structure of the subpart itself will be the primary means for identifying the correct correspondence.

8.1.11 The *import-from* relationship provides a flexible way of reducing repeated information, and increasing information transfer.

Some structured concept learning systems allow a subconcept to inherit information from its parent concept, rather than repeating it redundantly. MERGE is one example of such a system. However, GRAM provides an *import-from* relationship which allows structure and context descriptions to be 'imported' from any other concept, not just from superconcepts. This not only reduces memory usage, but also means that if the concept from which the information is imported from is generalised, this is implicitly transferred to the concepts that import from it. Of course, this has the danger of over-generalisation, and the overall learning and memory organisation system must handle this, primarily by taking a cautious approach which only generalises an existing concept (rather than creating a new concept) if a new instance is sufficiently similar to it.

8.1.12 It is useful to explicitly distinguish between several different 'interpretations' of structure and context descriptions.

The thesis has discussed the distinction between *partial* and *complete* descriptions, which correspond to Stepp's distinction between *contains* and *is* semantics [Stepp, 1987b]. Rather than force all descriptions to have either one or the other interpretation, or leave it ambiguous, GRAM requires the interpretation of each structure and context description to be explicitly specified, and this means that the 'drop-feature' generalisation operation can be used without requiring that *all* descriptions have a *partial* interpretation.

GRAM also distinguishes between several specific varieties of *partial* interpretation, to indicate explicitly to the matcher that the description is disjunctive, grouped, or imports from another concept.

8.1.13 Structure and context disjunction can be conveniently specified by subconcepts in the AKO hierarchy.

A feature of GRAM that has not been employed by other systems, is the ability to represent the structure and/or context of a concept disjunctively, by referring to the subconcepts of the concept. Some subconcepts may implicitly characterise 'structure' disjuncts (such as types of chair structure), others implicitly characterise 'context' disjuncts (such as types of surroundings where chairs are located), while others may specify disjuncts that are specify a co-dependent combination of both structure and context (such as a dentists chair and its relationships with a dentists office).

In this scheme there is no need for additional descriptive entities for disjuncts, and the mechanism for forming disjuncts can be combined with the mechanisms for building AKO hierarchies.

8.1.14 Enriched representation of properties and relationships support partial matching.

It has been found during the development of GRAM that if the representation of properties characterising the structure and the context of an object is sufficiently rich, then an inexpensive partial comparison between a concept and an instance provides a good indication of whether further more expensive comparison should be performed.

One type of property that is especially useful in this respect is the *profile* property which gives a summary description of the arrangement of neighbours or subparts of an object. For example a *density* profile indicates the density within each of several grid regions within the bounding box of an object, and this enables the substructure of two objects to be roughly compared, without actually comparing the subparts.

Likewise, if the representation of relationships is rich, then a confident partial similarity evaluation can be obtained without matching the relatees of the two objects.

8.1.15 Important information should be made explicit, to prevent loss of information during generalisation.

One theme that has run throughout this thesis is that important information should be made explicit, rather than implicit. This is because implicit constraints can be lost during generalisation. For example, a description of a chair may include ratios of the lengths of each leg to the height of the chair as a whole, all of which are 0.47, but without any explicit specification that the lengths are all the same. When generalising with another chair whose legs are 0.54 of the chair height, then each leg length will be generalised, and the previously implicit constraint of equal leg lengths is lost. This illustrates why important information should be explicit in concept and instance descriptions. Many of the features of GRAM's representation scheme, such as groups, disjunction, instance-counts, neighbour relationships, *import-from* specifications, and the distinction between structure and context, help support this requirement.

8.2 Matching

8.2.1 A matcher can and should exploit the structural organisation of objects.

The most significant characteristic of the GRAM matcher is that it exploits the structural organisation of physical objects to a greater extent than the other structure-matching systems reviewed in the thesis. GRAM achieves this primarily by its "spreading activation" algorithm which traverses parent, neighbour, and subpart relationships to find and evaluate correspondences between concepts and instances.

By distinguishing between just three kinds of relationships (*i.e.* parent, neighbour, and subpart relationships) each of which is a richly expressive descriptive entity, each relationship acts as a direct link to another concept or instance. When comparing a concept with an instance, the most similar pairs of concept and instance relationships indicate correspondences between parent, neighbour, and subpart concepts and instances. Thus, these three kinds of relationships allow the matcher to efficiently search through the space of concept-to-instance correspondences.

This process is also made possible because descriptions can first be matched just using their properties and relationships, ignoring relatees. The similarity score from this partial match is a reasonable estimate, due to the richness of the property and relationship descriptions. Therefore, the matcher only needs to traverse relationships to perform comparisons of relatees if the partial match is sufficiently good. Thus, by enriching the representation of properties and relationships, the search is reduced.

8.2.2 Relationships enable direct indexing for classifying an instance.

Even though indexing has not been explicitly addressed in this thesis, in GRAM's "spreading activation" matcher, each relationship acts as a kind of indexing mechanism which leads directly to a hypothesised classification of an instance. Each hypothesised classification that has a reasonably good score on the basis of property and relationship similarity, can be used to directly hypothesise classifications of the instance's parents, neighbours, and subparts. In fact, this form of indexing is just as important as a mechanism that indexes directly from features to a concept. The latter kind of indexing may be necessary to initiate the process, but once one component of an observed scene has been given a hypothesised classification, the evaluation of this classification will often lead to classifications of other components of the scene, via the parent, neighbour, and subpart relationships. In future work on GRAM, both forms of indexing may be combined.

A limitation of the Labyrinth system (see section 2.8) is that it does not make use of its subpart relationships to support indexing, but instead classifies every observed instance independently. Wasserman's MERGE (in section 2.5) does not require indexing at all, because instances were already named. The PARVOS system [Bergevin and Levine, 1993] (see section 2.9) used direct indexing from features to concepts, but since it only matched concepts at a single coarse level of detail, it did not perform further classification of related objects in the manner employed by GRAM.

8.2.3 Efficient and effective matching of structural descriptions is possible without maintaining bindings between correspondences.

An interesting and surprising result of this thesis is that complex structured objects can be effectively matched without having to maintain a consistent set of correspondence bindings between concepts and instances.

In an earlier version of GRAM, it was originally thought that a consistent set of bindings was essential, and so each concept was represented as an entire local substructure part-graph. However, the new system has shown that the richness of object descriptions and the distinctness of classes of physical objects tend to enforce consistency.

Thus, the matcher does not prevent an instance from being classified as belonging to several concepts, which may or may not be on the same branch of the AKO hierarchy. This allows instances to play multiple roles. If there are several ambiguous classifications of an instance, then all should be reported by the matcher, and other concept–instance comparisons can select whichever classification of the parent, neighbour, and subpart instances suits it best.

A consequence of this approach is that each concept–instance comparison can be considered to be an independent process. Each comparison simply requires that the properties are similar, the parent, neighbour, and subpart relationships are similar, and the relatees are similar (as evaluated by a recursive application of the matcher, in a kind of backward-chaining manner). If some other classification assumes that the same instance is classified differently, that is not considered a problem. The matcher makes use of the similarity scores computed for other previously computed concept–instance comparisons, but it does not enforce consistency.

The independence of each comparison, and the non-necessity for ensuring consistency and correspondence bindings, means that the GRAM matcher is amenable to a parallel implementation. This is an important quality of a matcher that is to be employed in a real physical environment, since classification often must be very rapid.

8.2.4 Robust matching is made possible by searching in any direction through the object graph, starting from any hypothesised seed classification.

Several of the systems reviewed in this thesis have used a top-down search to compare two structured descriptions. GRAM is not restricted to this form of search, but can instead search in any direction via parent, neighbour, and subpart relationships. This means that the classification does not need to begin from the root node of an object description. Rather, a seed classification of a subcomponent (such as the classification of a bicycle seat) can lead to the classification of the bicycle as a whole. This is important in a real physical environment where an object to be classified may be partially obscured, preventing a direct indexing mechanism from accessing the required concept.

Various methods to allow the user or the larger system to control the match spread are provided by the matcher, such as by using a *spread-distance* parameter, or by explicitly marking (or *scoping*) the instance objects that are to be classified.

8.2.5 Efficient 'any-time' matching is possible by using a breadth-first 'iterative deepening' search.

It has been found that the most effective search strategy is a breadth-first search with 'iterative deepening'. A depth first search is not effective because the system should not spread outwards from a correspondence, trying to evaluate correspondences of relatees, unless the correspondence has already been found to be reasonably promising. Otherwise large numbers of spurious correspondences could be evaluated unnecessarily. Therefore, GRAM only spreads from correspondences for which the properties and relationships match sufficiently well. It then performs successively more thorough comparisons on the best parent, neighbour, and subpart correspondences.

This strategy ensures that the similarity score for each concept–instance comparison is always based on a particular *spread-distance*, which is increased incrementally until either the classification is considered acceptable or non-acceptable. Thus the algorithm is an 'any-time' algorithm which can be interrupted at any time, and still provide a usable similarity score that has a known level of accuracy. This is important for an autonomous robot, which should be able to classify to any required level of confidence, and be interrupted at any time while still having usable match results.

This approach is justified by the results of the PARVO system [Bergevin and Levine, 1993] which showed that common physical objects can often be classified on the basis of a coarse description alone.

8.2.6 The level-hopping problem is resolved by exploiting neighbour relationships.

A limitation in systems that employ a strictly top-down search is that two objects can only be matched if their corresponding components are on the same level of their decomposition hierarchies. If one object has been described with an additional composite object in the hierarchy, then the matcher may fail. This is called the *level-hopping* problem. Wasserman's MERGE employed a special mechanism to cope with this, by inserting "null nodes" into the part hierarchy, in all possible ways that could account for level misalignment of a certain distance.

GRAM does not need to employ explicit level-hopping techniques, since the use of neighbour relationships, which can cross levels of the hierarchy, enable correspondences to be found directly.

8.2.7 Instance-counts are important for syntactic recognition.

In a system such as GRAM which does not make use of functional knowledge to learn the relative importance of concept features, recognition of real-world objects can instead make use of *instance-counts* associated with generalised features, in order to make better predictions about the expected presence of a feature in a new instance.

GRAM uses instance-counts to weight the similarity scores of relationship comparisons, such that a concept relationship with a low instance-count, that has no similar relationship in the instance, does not reduce the overall similarity score, since it is optional. However, if there *is* a similar relationship in the instance, then the similarity score contributes fully to the overall score. This similarity-dependent and frequency-dependent weighting has not been employed in the other systems reviewed in this thesis.

8.2.8 A concept is a "probabilistic predictor" of parents, neighbours, and subparts

Since each concept has instance-counts indicating the expectedness of each parent, neighbour, and subpart relationship in an instance, a concept can be interpreted as a kind of 'probabilistic predictor'. More specifically, if the system is told, or hypothesises, that an observed instance is an instance of a particular concept, then the parent, neighbour, and subpart relationships are predictors of the presence of similar parent, neighbour, and subpart relationships and relatees in the instance.

This interpretation is useful because it enables concepts to be used for fault-finding or description completion. For example, if an object is only partially visible, but classifiable, then its missing features can be 'completed' via prediction. Similarly, if a feature is absent from a classified instance, then the significance of the absence can be measured on the basis of the instance-count. In general, a high instance-count (with respect to the instance-count of the concept) indicates a functionally significant feature that is expected to be present in most instances. Thus GRAM's instance-counts are a form of quantitative MUST-HAVE conditions. A similar approach was taken in Fisher's COBWEB system.

8.2.9 Mismatches can sometimes be confirmed or resolved by augmenting an instance description.

Mismatches between two descriptions can sometimes be resolved or confirmed by augmenting the instance description by generating missing relationship descriptions, or even by adding new composite objects (although the latter is not supported by the current system). This idea has not been included in other structure-matching systems. It is most similar to the techniques used in the SPARC/E system [Dietterich and Michalski, 1985] which augmented descriptions of a sequence of cards in order to support regularity finding.

Relationship augmentation can be done when two concepts are explicitly related, but where two instances that have been found to match those concepts (at least partially) are not explicitly related. A relationship between the instances can therefore be created, and compared with the concept relationship. This may either strengthen or weaken the classifications, but in either case it results in more accurate similarity scores. This shows how the process of matching can influence the vision system in an expectation-driven manner.

8.2.10 Fit-scores *versus* proximity-scores.

The distinction between *fit-scores* and *proximity-scores* has not been proposed by other systems. It was found necessary to make the distinction in GRAM because different components of the system require similarity to be measured in different ways.

A proximity-score measures the absolute similarity of two objects, and is based on the 'proximity' of two objects in object-space. A fit-score measures the *typicality* of an instance with respect to a concept, and is computed from the ratio of the absolute difference, to the variance of the concept. A fit-score for two descriptions might be much lower than the proximity-score. For example, an observed swivel-chair would have a reasonably high proximity-score with respect to the concept *four-legged-chair*, but a very low fit-score.

The generaliser uses proximity-scores to determine whether two objects are similar enough to justify producing a generalisation, and it uses fit-scores to determine whether an instance that has been matched with a concept is typical enough to justify modifying the original concept, rather than creating a new concept.

The matcher uses proximity-scores to determine the best correspondences between objects, and a fault-finder would need to use fit-scores to identify the faulty or unusual features of an object.

8.3 Generalisation

8.3.1 Generalisation is simplified by representing concepts as small independent descriptive entities.

Although concepts in GRAM are defined in terms of other concepts, they are independent in the sense that the correspondences between their parent, neighbour, and subpart relationships, and those of a matched instance, are selected independently from correspondences selected for other concept–instance matches. They are also independent in the sense that their relationship descriptions are not shared by the concepts that the relationships refer to. Every concept has its own distinct set of relationship descriptions. This means that the task of the generaliser is simplified, since each concept–instance pairing can be generalised independently from other pairings, even though results of other generalisations may be referred to.

The simplicity of concept descriptions in GRAM (as opposed to the complexity of concept descriptions that specify a complete part-graph) also simplifies generalisation, since there is no need to merge two potentially large graphs, which may require generalising or creating embedded disjunctions within the resulting generalised graph, and perhaps extracting out subgraphs as distinct concepts.

8.3.2 Various forms of ambiguity have been distinguished.

One of the difficulties in generalising two descriptions is when there is ambiguity in the correspondences between their parent, neighbour, and subpart relationships and relatees. Following the methodology of identifying the *kinds* of situation to be dealt with before designing mechanisms, the issue of ambiguity has been addressed by distinguishing between several kinds of ambiguity, each of which requires a different method of resolution.

These include *similar-similarity* ambiguity and *different-similarity* ambiguity (as discussed below); *vertical* ambiguity, in which the objects involved are along the same vertical branch of the decomposition hierarchy; *local* ambiguity, which refers to ambiguity between the correspondences of parent, neighbour, and subpart relationships and relatees for a particular object comparison; and *global* ambiguity, which refers to the ambiguity amongst all object correspondences produced by the matcher.

Similar-similarity ambiguity is resolved by forming a multi-relationship or a group.

Similar-similarity ambiguity refers to the situation where all of the pairs of competing correspondences are similar in the same way. This situation can be resolved by producing a multi-relationship or a group.

Different-similarity ambiguity is resolved by performing multiple generalisations.

Different-similarity ambiguity, on the other hand, is where the competing pairs are similar in different ways. For example, one pair may match well with respect to structure, while the other pair matches well with respect to context, even though both overall similarity scores are roughly the same. The concept or instance that is involved in both pairs is playing two roles, one contextual and one structural. This situation can be resolved by performing multiple generalisations, one for each pair.

8.3.3 The representation supports a simple form of disjunction creation.

When a concept is to be generalised to cover an instance whose structure (or context) differs significantly from the structure (or context) of the concept, the generaliser must create a disjunction of structures (or contexts). In a representation scheme that represents a concept as a complete part hierarchy, disjunction formation can be complex, since it requires embedded disjunctive subgraphs. In GRAM's representation, disjunction formation simply involves creating subconcepts. There is no need to deal with multiple levels of a hierarchy, since each concept is only one level deep, consisting solely of relationships with other concepts, which are generalised independently.

In an extended GRAM system, disjunction formation would be integrated into a larger mechanism that constructs and maintains the AKO hierarchy.

8.3.4 Over-generalisation is reduced by requiring a minimum match effort, a minimum fit-score, and a winning classification

The danger of over-generalisation is especially significant in a structured domain with a representation scheme such as GRAM's, since each concept may be referred to by many other concepts, and the generalisation of one concept will therefore implicitly generalise all of the concepts that refer to it.

GRAM reduces the chance of incorrectly generalising a concept by requiring that the fit-score is sufficiently high. This ensures that the concept is only generalised by small amounts, and a new concept is created if an instance is atypical. Also, GRAM requires that the matcher has applied some minimum amount of effort to the comparison. A rough partial comparison may be sufficient for classification in some tasks, but is not sufficient to justify generalisation. Therefore, the generaliser may either request the matcher to continue performing a more thorough comparison, or not perform generalisation.

The issue of over-generalisation has only been minimally addressed in this thesis, by the methods above. A more thorough consideration will be performed when developing a full learning system that deals with memory organisation.

8.3.5 Fault-finding is possible without using negative examples.

GRAM does not use negative examples to learn a concept. This is in contrast to Winston's system [Winston, 1975] and the CLUSTER/S system [Stepp and Michalski, 1986a]. Negative examples (especially near-miss examples) could certainly be useful for instructing GRAM in three alternative ways: (a) that a particular feature is important for the function of an object, although not essential for membership; (b) that a particular feature can be used to distinguish between similar categories (such as the 'back' distinguishes between chairs and stools), or (c) that a particular feature is essential for concept membership (such as the 'hole' in an 'arch') regardless of how similar the rest of the object is.

Near-miss negative examples are not readily available in a real physical domain such as a house or a workshop, unless specifically constructed by a teacher. It also necessary for the interpretation of the example to be made clear, since (a) and (c) above are very different: a near-miss example of a chair without a leg should not mean that future chairs without legs are not classified as being chairs, but rather that the absence of the leg is a severe fault. On the other hand, a near-miss example of an arch with a missing hole should perhaps be used to prevent future objects without holes from being classified as arches.

Although GRAM cannot use negative examples, its use of instance-counts does enable it to learn the same kind of information that can be acquired from type-(a) negative examples, since a high instance-count on a feature within an object that has been observed many times indicates that the feature is very important to that concept, while a low instance-count indicates lower importance. If 100 chairs with 4 legs have been observed, then a chair with a missing leg will be identified as faulty, although still correctly identified as a chair. Likewise, if 100 'arches' have been observed, all of which have supports that are separated (and have a 'hole', if GRAM could represent such 'empty parts'), then an object with a lintel and two touching supports would perhaps still be weakly classified as an arch, but the fault would be identified. This has been confirmed by giving the implemented GRAM system a series of simple arches, followed by a non-arch. It correctly noticed that the position and connectivity of the supports were outside the expected range.

Thus, near-miss examples are not *necessary* for enabling a system to identify faults, although they would enable the importance of a feature to be learned from a single example, rather than many examples, and they would also enable the stronger interpretation-(c) to be made if instructed to do so.

Currently GRAM is not able to represent the necessity of the *absence* of a part. It is not clear how useful this would be, apart from distinguishing between concepts such a chair and a stool as for interpretation-(b) above. Perhaps it is useful to be able to state that if there is a pencil case on a desk, it is not part of the desk, but the description of the desk concept could become cluttered with all sorts of non-components.

The issue of how negative examples could be incorporated into GRAM has not been considered in any depth in this thesis, and so is a possible subject for future research.

8.4 Instance Construction

8.4.1 GRAM's instance constructor augments primitive descriptions to support more efficient and effective matching and generalisation.

This thesis makes an explicit distinction between instance construction and low-level vision, and emphasises the importance of augmenting a description (produced by the vision system) in ways that explicitly support matching and generalisation. This is a kind of *constructive induction*, as discussed by [Michalski, 1983], although he did not address the issues of constructive induction to support exploitation of structure in a structured domain.

Two main aspects of this process have been discussed in the thesis: Firstly, abstract composite objects are created (from block descriptions produced by a vision system) to enable several objects to be processed by the matcher and generaliser as a single collective entity, and to enable properties that characterise the set of objects as a whole, to be representable. Secondly, explicit parent, neighbour, and subpart relationships are created to support the matcher by providing links between instances, and by enriching the description of each instance to ensure that a more confident classification can be obtained, and to ensure that generalisations produced will contain sufficient information to capture the important features of the concept.

The thesis has described sets of criteria for justifying the formation of composite objects and relationships. In particular, it has presented a set of criteria and a mechanism for finding and creating *group* objects.

8.4.2 Group construction during instance construction pre-empts group formation during generalisation.

Group creation is an interesting form of description augmentation since it involves matching and generalising within an instance. Although it is also possible to form groups when an instance is matched with a concept, as a way of resolving ambiguity, this can be preempted by the formation of groups during the instance construction process.

8.4.3 Groups are found using the Seed Expansion algorithm.

The thesis has presented an algorithm called Seed Expansion that finds groups of similar and similarly related objects. It proposes seed group by finding pairs of similar and interestingly related objects, and then expands the best seed by incrementally adding new members until a clear boundary between members and non-members is found, or until the group is found to be unacceptable.

None of the structure-learning systems reviewed in the thesis provide group-finding mechanisms, except for Winston's ARCH learner [Winston, 1975] from which may of the ideas of group representation were based. His paper described an algorithm that takes a "propose and prune" approach, which involves proposing a generous grouping, and then removing members until a stable group is found, thus working in the opposite direction from GRAM's expansion process. His paper also suggested that some kind of algorithm similar to seed-expansion was used to find sequences of objects, but the details were not given, and it was only used to find a restricted kind of sequence, and no other kinds of group.

Bibliography

[Andreae, 1993] Andreae, D. B. (1993). Representing and matching physical objects. *New Zealand Journal of Computer Science*, 4(2):3–13.

[Andreae, 1985] Andreae, P. (1985). *Justified Generalisation: Acquiring Procedures From Examples*. PhD thesis, M.I.T.

[Bareiss and Porter, 1987] Bareiss, E. and Porter, B. (1987). Protos: An exemplar-based learning apprentice. In *Proceedings of the Fourth International Workshop on Machine Learning*, pages 12–23.

[Bareiss et al., 1990] Bareiss, E., Porter, B., and Wier, C. (1990). Protos: An exemplar-based learning apprentice. In Kodratoff, Y. and Michalski, R., editors, *Machine Learning: An Artificial Intelligence Approach*, volume 3, chapter 4. Morgan Kaufmann.

[Bergevin and Levine, 1993] Bergevin, R. and Levine, M. (1993). Generic object recognition: Building and matching coarse descriptions from line drawings. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 3(1):19–36.

[Biederman, 1985] Biederman, I. (1985). Human image understanding: Recent research and a theory. *Computer Vision, Graphics, and Image Processing*, 32:29–73.

[Brady, 1983] Brady, M. (1983). Criteria for representations of shape. In Rosenfeld, editor, *Human and Machine Vision*.

[Brady, 1987] Brady, M. (1987). Intelligent vision. In Grimson and Patil, editors, *AI in the 1980's and Beyond*, pages 201–242. M.I.T Press.

[Brady et al., 1984] Brady, M., Agre, P., Braunegg, D., and Connell, J. (1984). The mechanic's mate. In O'Shea, editor, *ECAI-84: Advances in Artificial Intelligence*, pages 681–97. Elsevier Science Publishers.

[Brooks, 1981] Brooks, R. (1981). Symbolic reasoning among 3-d models and 2-d images. *Artificial Intelligence*, 17:285–349.

[Chin, 1988] Chin, R. (1988). Automated visual inspection: 1981 to 1987. *Computer Vision, Graphics, and Image Processing*, 41:346–81.

[Cohen, 1984] Cohen, B. Murphy, G. (1984). Models of concepts. *Cognitive Science*, 8(1):27–58.

[Connell, 1985] Connell, J. (1985). Learning shape descriptions: Generating and generalising models of visual objects. Master's thesis, Department of Electrical Engineering, MIT.

[Connell and Brady, 1985] Connell, J. and Brady, M. (1985). Learning shape descriptions. In *IJCAI-85*, pages 922–5.

[Connell and Brady, 1987] Connell, J. and Brady, M. (1987). Generating and generalising models of visual objects. *Artificial Intelligence*, 31:159–83.

[Cook and Holder, 1994] Cook, D. J. and Holder, L. B. (1994). Substructure discovery using minimum description length and background knowledge. *Journal of Artificial Intelligence Research*, 1:231–255.

[Dietterich, 1980] Dietterich, T. (1980). Applying general induction methods to the card game eleusis. In *Proceedings of AAAI-80*, pages 218–20.

[Dietterich and Michalski, 1981] Dietterich, T. and Michalski, R. (1981). Inductive learning of structural descriptions: Evaluation criteria and comparative review of selected methods. *Artificial Intelligence*, 16(3):257–94.

[Dietterich and Michalski, 1983] Dietterich, T. and Michalski, R. (1983). A comparative review of selected methods for learning from examples. In Michalski, R., Carbonell, J., and Mitchell, T., editors, *Machine Learning: An AI Approach*, volume 1. Tioga, Palo Alto, Calif.

[Dietterich and Michalski, 1985] Dietterich, T. and Michalski, R. (1985). Discovering patterns in sequences of events. *Artificial Intelligence*, 25:187–233.

[Dietterich and R., 1986] Dietterich, T. and R., M. (1986). Learning to predict sequences. In Michalski, R., Carbonell, J., and Mitchell, T., editors, *Machine Learning: An Artificial Intelligence Approach.*, volume 2. Morgan Kaufmann, Los Altos,Calif.

[Ellis, 1939] Ellis, W., editor (1939). *A Source Book of Gestalt Psychology*. Kegan Paul, Trench, Trubner & Co.

[Falkenhainer et al., 1989] Falkenhainer, B., Forbus, K. D., and Gentner, D. (1989). The structure-mapping engine: Algorithm and examples. *Artificial Intelligence*, 41(1).

[Fisher, 1987a] Fisher, D. (1987a). Improving inference through conceptual clustering. In *Proceedings of AAAI-87*, pages 461–5.

[Fisher, 1987b] Fisher, D. (1987b). Model invocation for three dimensional scene understanding. In *IJCAI-87*, pages 805–7.

[Fisher, 1988] Fisher, D. (1988). A computational account of basic level and typicality effects. In *AAAI-88*, pages 233–8.

[Gennari et al., 1989] Gennari, J. H., Langley, P., and Fisher, D. (1989). Models of incremental concept formation. *Artificial Intelligence*, 40(1-3):11–61.

[Haussler, 1987] Haussler, D. (1987). Learning conjunctive concepts in structural domains. In *AAAI-87*, pages 466–70.

[Hoffman and Richards, 1987] Hoffman, D. and Richards, W. (1987). Parts of recognition. In Fischler and Finschein, editors, *Readings in Computer Vision*, pages 227–239.

[Iba, 1979] Iba, G. (1979). Learning disjunctive concepts from examples. Technical report, M.I.T.

[Kubovy and Pomerantz, 1981] Kubovy, M. and Pomerantz, J. R., editors (1981). *Perceptual Organisation*. Lawrence Erlbaum Associates.

[Lebowitz, 1985] Lebowitz, M. (1985). Researcher: An experimental intelligent information system. In *IJCAI-85*, pages 858–862.

[Lebowitz, 1986] Lebowitz, M. (1986). Not the path to perdition: The utility of similarity-based learning. In *Proceedings of AAAI-86*, pages 533–7.

[MacGregor, 1988] MacGregor, R. (1988). A deductive pattern matcher. In *Proceedings of AAAI-88*, pages 403–8.

[Markman, 1979] Markman, E. (1979). Classes and collections: Conceptual organisation and numerical abilities. *Cognitive Psychology*, 11:395–411.

[Markman et al., 1980] Markman, E., Horton, M., and McLanahan, A. (1980). Classes and collections: Principles of organisation in the learning of hierachical relations. *Journal unknown*.

[Marr, 1982] Marr, D. (1982). *Vision*. W.H. Freeman, San Francisco, CA.

[Michalski, 1980] Michalski, R. (1980). Knowledge acquisition through conceptual clustering: A theoretical framework and an algorithm for partitioning data into conjunctive concepts. *Policy Analysis and Info. Systems*, 4(3):219–44.

[Michalski, 1983] Michalski, R. (1983). A theory and methodology of inductive learning. In Michalski, R., Carbonell, J., and Mitchell, T., editors, *Machine Learning: An AI Approach*, volume 1. Tioga, Palo Alto, Calif.

[Pomerantz, 1985] Pomerantz, J. (1985). Perceptual organisation in information processing. In Aitkenhead, A. and Slack, J., editors, *Issues in Cognitive Modelling*, chapter 6. Lawrence Erlbaum, London.

[Provan, 1987] Provan, G. (1987). Efficiency analysis of multiple-context truth-maintenance systems in scene representation. In *Proceedings of AAAI-87*, pages 173–7.

[Provan, 1988] Provan, G. M. (1988). Model-based object recognition: A truth maintenance approach. In *CAIA-88*, pages 230–5.

[Requiche, 1980] Requiche (1980). Representation of solid objects. *ACM Computing Surveys*, 12(4):437.

[Sammut, 1981] Sammut, C. (1981). Concept learning by experiment. In *Proc. of the Seventh IJCAI*, pages 104–5.

[Sammut and Banerji, 1986] Sammut, C. and Banerji, R. (1986). Learning concepts by asking questions. In Michalski, R., Carbonell, J., and Mitchell, T., editors, *Machine Learning: An Artificial Intelligence Approach.*, volume 2. Morgan Kaufmann, Los Altos,Calif.

[Stepp, 1987a] Stepp, R. (1987a). Concepts in conceptual clustering. In *IJCAI-87*, pages 211–3.

[Stepp, 1987b] Stepp, R. (1987b). Machine learning of structured objects. In *Proceedings of Fourth Int. Workshop on M.L.*, pages 353–63.

[Stepp and Michalski, 1986a] Stepp, R. and Michalski, R. (1986a). Conceptual clustering: Inventing goal-oriented classifications of structured objects. In Michalski, R., Carbonell, J., and Mitchell, T., editors, *Machine Learning: An Artificial Intelligence Approach.*, volume 2. Morgan Kaufmann, Los Altos,Calif.

[Stepp and Michalski, 1986b] Stepp, R. and Michalski, R. (1986b). Conceptual clustering of structured objects: A goal-oriented approach. *Artificial Intelligence*, 28(1):43–69.

[Thompson and Langley, 1991] Thompson, K. and Langley, P. (1991). Concept formation in structured domains. In Fisher, D. H. and Pazzani, M., editors, *Concept Formation: Knowledge and experience in unsupervised learning*, chapter 5, pages 127–161. Morgan Kaufmann, San Mateo, CA.

[Tversky, 1977] Tversky, A. (1977). Features of similarity. *Psychological Review*, 84(4):327–52.

[Tversky and Hemenway, 1984] Tversky, B. and Hemenway, K. (1984). Objects, parts, and categories. *Journal of Experimental Psychology: General*, 113(2):169–97.

[Ullman, 1989] Ullman, S. (1989). Aligning pictorial descriptions: An approach to object recognition. *Cognition*, 32:191–254.

[Wasserman, 1985] Wasserman, K. (1985). *Unifying Representation and Generalisation: Understanding Hierarchically Structured Objects*. PhD thesis, Columbia University.

[Wasserman and Lebowitz, 1983] Wasserman, K. and Lebowitz, M. (1983). Representing complex physical objects. *Cognition and Brain Theory*, 6(3):333–52.

[Whitehall and Reinke, 1987] Whitehall, B. and Reinke, R. (1987). Learning structural descriptions incrementally: The disjointness problem. Technical report, University of Illinois.

[Winston, 1975] Winston, P. (1975). Learning structural descriptions from examples. In Winston, P., editor, *The Psychology of Computer Vision*, chapter 5, pages 157–209. McGraw-Hill, New York.

[Winston, 1983] Winston, P. (1983). Learning physical descriptions from functional definitions, examples, and precedents. In *Proceedings of AAAI-83*, pages 433–39.

[Winston, 1984] Winston, P. (1984). *Artificial Intelligence*. Addison-Wesley, second edition.

[Winston, 1992] Winston, P. (1992). *Artificial Intelligence*. Addison-Wesley, third edition.