Order, Chaos and Algorithms

Rod Downey
School of Mathematics, Statistics and Computer Science
Victoria University
PO Box 600
Wellington
New Zealand.

May 22, 2006

1 Introduction

In these notes I will look at some very exciting parts of mathematics related to order in apparent chaos, and surprising connections with applications in algorithm design. I assume that the reader has a basic background with the basic terminology of graph theory and has that elusive quality "mathematical maturity."

These notes have quite uneven levels of depth. They were prepared for a course to talented undergraduates at the *Nangyang University of Technology* for May 2006. The idea is that several different courses could be made from the notes. For example, the flow material could be covered quickly, only pointing at the main ideas, such as max-flow/min cut and applications, for really advanced students. Then the idea would be to move on to the WQO type material and treewidth. On the other hand, with less advanced students more time could be spent in a thorough treatment on this easier material, with less time on the more advanced cutting-edge material; but allowing the less advanced student to have a *tour d'horizon*.

2 Network flows

In this section, we develop some basic machinery which allows us to begin work in *topological graph theory*, an area where the emphasis is on connectivitity and "shape". The material is of interest in itself, and might be familiar with students who have done basic operations research.

2.1 Basic Results

First we give a typical problem. We want to send oil from s to t, via a sequence of pipes of varying capacity. We wish to maximise this flow.

To model this problem, we associate a directed graph with the pipes. Let the verticies be s and t together with the places where pipes intersect. We give weights to the arcs (edges) which are the capacities of the pipes. Figure 1 gives an example of such a weighted-digraph below.

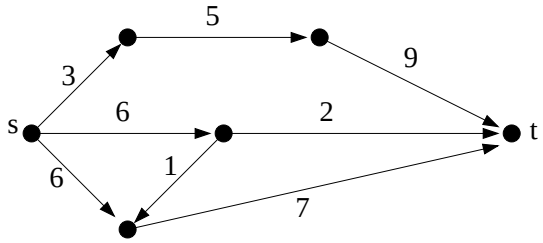


Figure 1: A graph of an oil pipe sytem

For the rest of this section, let s be *the source* and t be *the sink*. The graphs should have no loops or multiple edges. The vertex s to vertex t should be strongly connected. All weights should be positive.

Next we develop some notation. Let N be a network, $V(N)$ be it's vertex set and $E(N)$ be the edge set of N. Let $x \in V(N)$. The set $\{v \in V(N) : (x, v) \in E(N)\}$, of verticies *after* x will be denoted by $A(x)$. The set $\{u \in V(N) : (u, x) \in E(N)\}$, of verticies *before* x will be denoted by $B(x)$.

In Figure 2, $A(x) = \{p, b\}$ and $B(x) = \{b, a\}$.

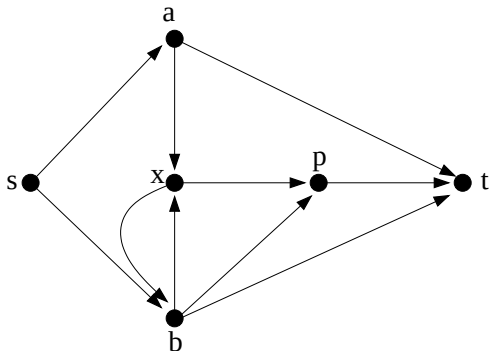


Figure 2: A network

Convention is to use *lower case* letters for individual verticies, e.g. p,s,t,a,b. Also, we use *upper case* letters for sets of verticies, e.g A, B,C.

Now we describe the collection of edges between to sets of verticies. Let A and B both be subsets of $V(N)$. Then

$$(A, B) = \{(x, y) : x \in A \text{ and } (x, y) \in E(N) \text{ and } y \in B\}.$$

An example follows, using the network in Figure 2. Suppose $X = \{s, a, p\}$ and $Y = \{x, b\}$. Then

$$(X, Y) = \{(s, b), (a, x)\}$$

and

$$(Y, X) = \{(x, p), (b, p)\}.$$

Suppose that h is any function, from verticies to verticies, e.g flow or capacity. Then for sets A and B, both subsets of $V(N)$, we define

$$h(A, B) = \sum_{(x,y)\in(A,B)} h(x, y).$$

So in the example above, letting $h(a, x) = 3$ and $h(s, b) = 7$, we find that

$$h(X, Y) = h(s, b) + h(a, x) = 10.$$

Some trivial consequences follow. We abuse notation and write a for $\{a\}$ when the context makes clear which choice was intended.

1. $(V(N), a) = (B(a), a)$.
2. $(a, V(N)) = (a, A(a))$.
3. $h(X, Y \cup Z) = h(X, Y) + h(X, Z) - h(X, Y \cap Z)$.

Now we come to a key definition. Let N be a network, where every edge (x, y) has a weight $C(x, y)$, called the *capacity* of that edge. A *flow* f in a network N is a function $f : V(N) \times V(N) \to \mathbb{R}$ which satisfies the following three conditions

1. $f(x, y) \geq 0$ for all $(x, y) \in E(N)$. So all flows are non-negative.
2. $f(x, y) \leq C(x, y)$, for all $(x, y) \in E(N)$. So the flow cannot exceed the capacity.
3. There is a fixed positive value q such that, for every $x \in V(N)$

$$f(x, A(x)) - f(B(x), x) = \begin{cases} q & \text{if } x = s \\ -q & \text{if } x = t \\ 0 & \text{otherwise} \end{cases}$$

So, apart from the source and the sink, the flow into any vertex should equal the flow out. We call the value in condition (3) the *value* of the flow.

We continue to build up notation. For any set of vertices X, we write $\overline{X}$ for $V(N) - X$. If $s \in X$ and $t \in \overline{X}$ then we call $(X, \overline{X})$ a *cut*. Next we extend the idea of edge capacity to the capacity of a cut. This will calculate how much you can send from one part of a network to rest. Let $C(X, \overline{X})$ be the *capacity of the cut* $(X, \overline{X})$, and we calculate it as follows

$$C(X, \overline{X}) = \sum_{(x,y) \in (X, \overline{X})} C(x, y).$$

Similarly, for a flow f, we denote the *flow of the cut* $(X, \overline{X})$ by $f(X, \overline{X})$ and calculate it as follows,

$$f(X, \overline{X}) = \sum_{(x,y) \in (X, \overline{X})} f(x, y).$$

Theorem 2.1. *Suppose f is a flow with value q. Then if $(X, \overline{X})$ is any cut, the following two statements hold*

1. *$q = f(X, \overline{X}) - f(\overline{X}, X)$. So the amount out of X less the amount into X is the amount leaving the source and entering the sink.*
2. *$f(X, \overline{X}) \geq 0$ and $q \leq f(X, \overline{X})$.*

Proof. First we will show that statement (1) is true. Recall from the definition of cuts that $t \notin X$. Also, if $a \neq s$ and $a \in X$ then, as flow in equals flow out,

$$f(a, V(N)) - f(V(N), a) = 0 \tag{1}$$

Let $\hat{X} = X - \{s\}$. Combining Equation 1 with the fact that neither s not t are members of $\hat{X}$, we see that

$$f(\hat{X}, V(N)) - f(V(N), \hat{X}) = 0.$$

As $(X, \overline{X})$ is a cut, we know that $V(N) = \overline{X} \cup X$ and $\overline{X} \cap X = \emptyset$.

Now we calculate the difference of $f(X, V(N))$ and $f(V(N), X)$.

$$\begin{aligned}
f(X, V(N)) - f(V(N), X) &= f(\hat{X} \cup \{s\}, V(N)) - f(V(N), \hat{X} \cup \{s\}) \\
&= \Big[f(\hat{X}, V(N)) + f(s, V(N)) - f(\hat{X} \cap \{s\}, V)\Big] \\
&\quad - \Big[f(V(N), \hat{X}) + f(V(N), s) - f(V(N), \hat{X} \cap \{s\})\Big] \\
&= \Big[f(\hat{X}, V(N)) - f(V(N), \hat{X})\Big] + [f(s, V(N)) - f(V(N), s)] \\
&= 0 + [q - 0] \\
&= q
\end{aligned}$$

To complete the proof of statement (1), we use the difference calculated above.

$$\begin{aligned}
q &= f(X, V(N)) - f(V(N), X) \\
&= f(X, X \cup \overline{X}) - f(X \cup \overline{X}, X) \\
&= f(X, X) + f(X, \overline{X}) - f(X, X) - f(\overline{X}, X) \\
&= f(X, \overline{X}) - f(\overline{X}, X)
\end{aligned}$$

We use statement (1) to show that statement (2) is also true. From above, $q = f(X, \overline{X}) - f(\overline{X}, X)$. Rearranging, we see that

$$f(X, \overline{X}) = q + f(\overline{X}, X).$$

From the definition of flow, $f(\overline{X}, X) \geq 0$. Hence $q \leq f(X, \overline{X})$. Again, from the definition of flow, $f(X, \overline{X}) \geq 0$. □

Corollary 2.2. *The maximum value a flow of a network can take, is no larger than the minimum value the capacity of a cut can take.*

Proof. Let $(X, \overline{X})$ be a cut. Now we look at some flow f in the graph. From Theorem 2.1, $q = f(X, \overline{X}) - f(\overline{X}, X)$. As f is a flow, $f(X, \overline{X}) \leq C(X, \overline{X})$. So

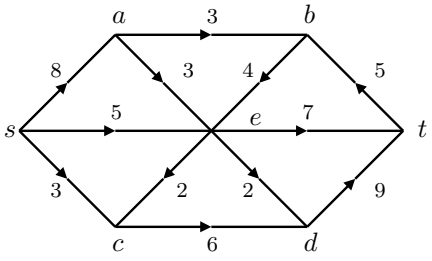
$$q \leq C(X, \overline{X}) - f(\overline{X}, X).$$

As $f(\overline{X}, X) \geq 0$, it follows that $q \leq C(X, \overline{X})$.

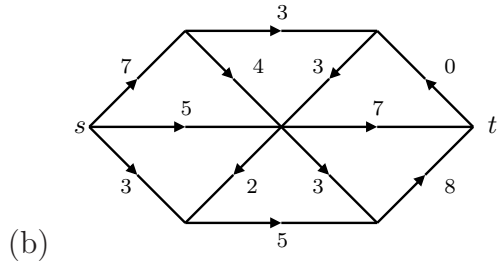
So the value of any flow in the network is no larger than the capacity of our cut $(X, \overline{X})$. Hence, over all the flows and cuts, the maximum flow is no larger than the minimum cut. □

3 Exercises 1

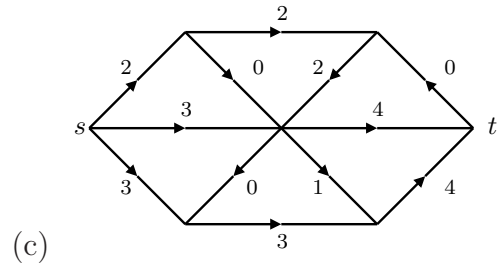
1 In the below Figure (a) represents a transportation network with source s and sink t. Figures (b), (c) and (d) represent "flows," with the flow on a edge represented by the number on the arc. Determine which are actually flows, and in the case that they are, determine the value of the flow.



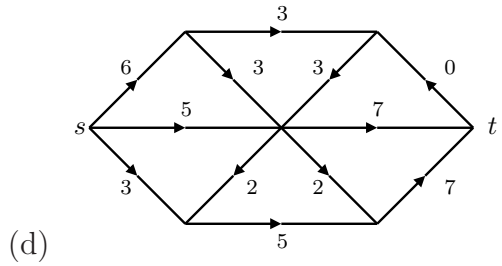
(a)



(b)



(c)



(d)

2 Find the capacity of each cut in the transportation network (a).

3 Both $(X, \overline{X})$ and $(Y, \overline{Y})$ are minimum cuts in a certain transportation network N. Prove that both $(X \cup Y, \overline{X \cup Y})$ and $(X \cap Y, \overline{X \cap Y})$ are minimum cut in N.

4 The Ford Fulkerson Algorithm

The summary of the algorithm below is

"accentuate the positive and eliminate the negative!"

This will be done across a minimum cut.

One of the key idea used for this algorithm help maximize a given flow,

is how to extend a given flow. Let $s = a_1$ and $t = a_n$. A undirected path of vertices $a_1 a_2 \ldots a_n$, with no repetitions, from s to t, is called an *augmenting path* if

1. One of (a_i, a_{i+1}) or (a_{i+1}, a_i) is a directed edge in the network.
2. If $(a_i, a_{i+1}) \in E$ then $f(a_i, a_{i+1}) < C(a_i, a_{i+1})$
3. If $(a_{i+1}, a_i) \in E$ then $f(a_{i+1}, a_i) > 0$.

Below is an example of an augmenting path in a network flow. Remember that the first number is the capacity of the edge, and the second number is the flow through the edge. An augmenting path is s, d, b, a, t. Note that,

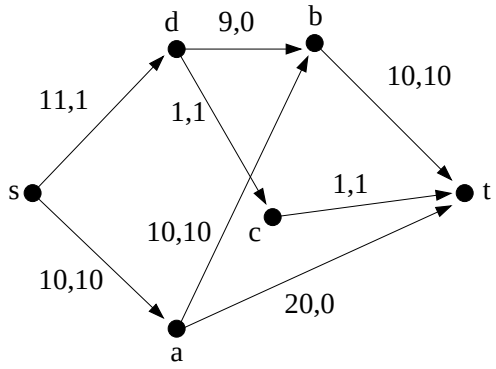


Figure 3: Before the path

even though $f(a, b) = C(a, b)$, in the path the orientation is (b, a), so we only require that $f(a, b) > 0$. The table below explains how we can use the augmenting path to increase the value of the flow.

	f(X, Y)	C(X, Y)	possible net change
(s, d)	1	11	10
(d, b)	0	9	9
(b, a)	-10	0	10
(a, t)	0	20	20

As the smallest possible net change is 9, that means we can augment the flow in Figure 3 by 9 units along the augmenting path. The new, augmented flow, is in Figure 4 Note that we lowered the arc (b, a) by 9, as the path goes

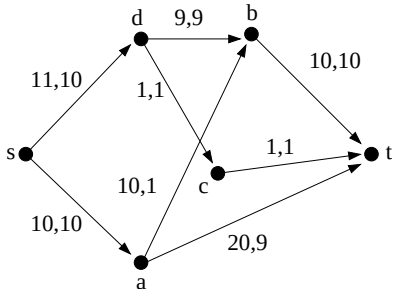


Figure 4: After the path

the other way.

4.1 Ford-Fulkerson algorithm

First we state the algorithm, later we will prove it's correctness.

We will label some vertices, with a triple (v, d, a). The first ordinate, v, is a vertex. The second, d, is a direction (either + or −). The third, a, is the amount by which we can increase the flow.

Step 1. Label the source s with $(s, +, \infty)$. So we've come from s, in a positive direction, with no limit to how much the flow could increase.

Step 2. If the sink has been labeled, go to step 5.

Step 3. Check to see if the flow is maximal already. If we have scanned every labeled vertex, then the flow is maximal. Stop.

Step 4. We now scan (or process) a vertex. Let x be the last vertex labeled, but not scanned. Suppose the label of x is $(y, \pm, a)$. Our next move depends on whether $(y, x) \in E$ or $(x, y) \in E$.

If $(y, x) \in E$, then $f(y, x) < C(y, x)$ and the label is $(y, +, a)$,

Otherwise , $(x, y) \in E$, then $f(x, y) > 0$ and the label is $(y, -, a)$.

Now we look for other vertices to label. Vertex w can be *seen* by x if w has not been scanned already and either

1. $w \in A(x)$ and $f(x, w) < C(x, w)$. So in this case, there is capacity to extend our augmenting path. Label w by $(x, +, b)$ where

$$b = \min\{a, C(x, w) - f(x, w)\}$$

2. $w \in B(x)$ and $f(w, x) > 0$. So there is room to redirect the flow. Label w by $(x, -, b)$ where

$$b = \min\{a, f(w, x)\}$$

Once all vertices seen by x have been labeled, declare that x is scanned and go to step 2.

Step 5. As we came from step 2, the sink is labeled $(y, +, b)$ for some b. We can improve the flow by b. Retrace the path, using the labels. For each edge in the path, lower or raise the flow by b depending on whether the label is $+$ or $-$. For example, the edge (y, t) corresponds to the label $(y, +, b)$, so the flow along (y, t) is increased by b. If a label on the path is $-$, then decrease the flow by b.

4.2 Verification of Ford-Fulkerson

Lemma 4.1. *If there is an augmenting path*

$$s = x_1, x_2, \ldots, x_n = t$$

then the flow can be increased by the algorithm.

Proof. For each i satisfying $1 \leq i < n$, let

$$S_i = \begin{cases} C(x_i, x_{i+1}) - f(x_i, x_{i+1}) & \text{if } (x_i, x_{i+1}) \in E \\ f(x_{i+1}, x_i) & \text{if } (x_{i+1}, x_i) \in E \end{cases}$$

and $b = \min\{S_i\}$.

We define a new (augmented) flow g as follows.

$$g(x,y) = \begin{cases} f(x,y) & \text{if neither } (x,y) \text{ nor } (y,x) \text{ is on the path} \\ f(x,y) + b & \text{if } (x,y) \in E \text{ and, for some } i, x = x_i \text{ and } y = x_{i+1} \\ f(x,y) - b & \text{if } (x,y) \in E \text{ and, for some } i, x = x_{i+1} \text{ and } y = x_i \end{cases}$$

First we show that g is, in fact, a flow. To show this, we need to confirm that for every vertex, apart from the source s and the sink t, the flow in to this vertex is the same as the flow out. Let x be the vertex of interest. If x is not on the path, then the flows through x haven't changed from f. As f is a flow, the flow in and out of x are unchanged.

Suppose that, for some j, $x = x_j$. So x is on the path. We can ignore the edges that are not on the path. As we used the path to construct g, only two edges on the path, using x, have a different flow from f. Let (z_1, x_j) and (x_j, z_2) be the edges on the path that contain x. There are four cases, depending on which orientation that edges take in the network.

(z_1, x_j) and (x_j, z_2) are also the directed edges in the network. Then $g(z_1, x_j) = f(z_1, x_j)+b$, and the flow in to x is increased by b. Also, $g(x_j, z_2) = f(x_j, z_2) + b$, and the flow out of x is increased by b. Hence the flow in and flow out are still equal.

(z_1, x_j) and (z_2, x_j) are the directed edges in the network. So the second edge in the path was against the network direction. Again, $g(z_1, x_j) = f(z_1, x_j) + b$, and the flow in to x is increased by b. In this case, $g(z_2, x_j) = f(z_2, x_j) - b$, so the flow in to x is decreased by b. Overall, the flow in unchanged and the flow out was not affected. So the flow in and out are the same.

(x_j, z_1) and (x_j, z_2) are the directed edges in the network. Similar above.

(x_j, z_1) and (z_2, x_j) are the directed edges in the network. See above.

So we have confirmed that g is a flow. Clearly, the value of g is b units bigger than f's value. So g increases the flow. □

Now we have to check what happens when the algorithm stops. We must confirm that the flow is, indeed, *maximal*. We remark that if we assume that all the capacities are integer valued, then since we increase the flow each time, we must stop. We will tacitly make this assumption, though it can be eliminated for flows with *rational* weights (by multiplying to make them integers) and then for real weights (using rational approximations). These distract from the main ideas so we don't discuss further.

Lemma 4.2. *Suppose that a transportation network has a flow from s to t of value q. Then either there is an augmenting path, or there is a cut whose capacity is q.*

Proof. Run the Ford-Fulkerson algorithm. This does a complete search for an augmenting path.

Suppose that no such path exists. Then the algorithm gets stuck on step 2, before it gets to t. So t is not labeled. But every vertex that is labeled is also scanned, in the search for an augmenting path. Let X be the set of vertices scanned.

We claim that $(X, \overline{X})$ is a cut of capacity q. Clearly $s \in X$ (s is scanned first) and $t \in \overline{(X)}$ (as t is not labeled, it cannot be scanned). Let $x \in X$ and $y \in \overline{X}$. There two possibilities, depending on the orientation of the edge. In each case, recall that the algorithm stopped.

1. Suppose that $(x, y) \in E$. If there is any spare capacity, then x could see y and we would have scanned y. So there is no spare capacity and $f(x, y) = C(x, y)$.
2. Suppose that $(y, x) \in E$. If there is any flow along this edge, then x would see y and we would have scanned y. So there is no flow along this edge and $f(x, y) = 0$.

Together, these two cases show us that $(X, \overline{X})$ is a cut. Now we calculate

the capacity of this cut.

$$\begin{aligned} q = &\text{ the capacity of } (X, \overline{X}) && \text{(the value of the flow is the capacity)} \\ &= f(X, \overline{X}) - f(\overline{X}, X) && \text{(from the definitions)} \\ &= C(X, \overline{X}) - 0 \\ &= C(X, \overline{X}) \end{aligned}$$

□

Corollary 4.3. *In a network, the maximum flow value is equal to the minimum cut capacity.*

4.3 An example of Ford-Fulkerson

Below we give an example of the Ford-Fulkerson algorithm run on a particular network. We use a table to conveniently keep track of the labels.

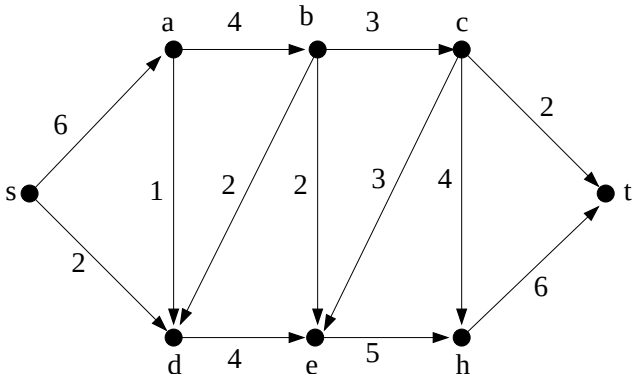


Figure 5: Ford-Fulkerson example, with no flow

vertex processed	vertex labeled	label
	s	[s+, ∞]
s	a	[s+, 6]
	d	[s+, 2]
a	b	[a+, 4]
d	e	[d+, 2]
e	h	[e+, 2]
h	t	[h+, 2]

So there is an augmenting path (reading backwards)

$$(s, d, e, h, t).$$

The new flow is shown in Figure 6.

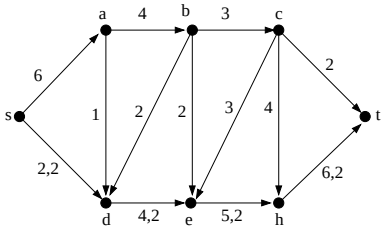


Figure 6: Ford-Fulkerson example, after 1 iteration

vertex processed	vertex labeled	label
	s	[s+, ∞]
s	a	[s+, 6]
a	b	[a+, 4]
	d	[a+, 1]
b	c	[b+, 3]
	e	[b+, 2]
c	t	[c+, 2]

So the augmenting path from this iteration is

$$(s, a, b, c, t).$$

The new flow is shown in Figure 7.

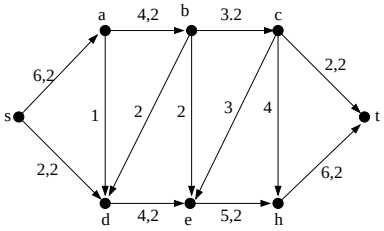


Figure 7: Ford-Fulkerson example, after 2 iterations

vertex processed	vertex labeled	label
	s	[s+, ∞]
s	a	[s+, 4]
a	b	[a+, 2]
	d	[a+, 1]
b	c	[b+, 1]
	e	[b+, 2]
c	h	[c+, 1]
h	t	[h+, 1]

So the augmenting path is

$$(s, a, b, c, h, t).$$

The new flow is shown in Figure 8.

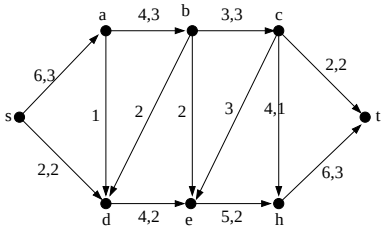


Figure 8: Ford-Fulkerson example, after 3 iterations

vertex processed	vertex labeled	label
	s	[s+, ∞]
s	a	[s+, 3]
a	b	[a+, 1]
	d	[a+, 1]
b	e	[b+, 1]
e	h	[e+, 1]
h	t	[h+, 1]

So the augmenting path is

$$(s, a, b, e, h, t).$$

The new flow is shown in Figure 9.

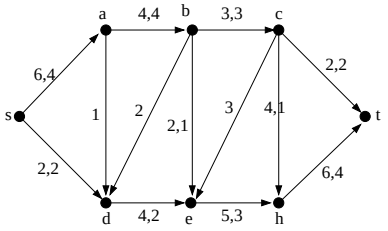


Figure 9: Ford-Fulkerson example, after 4 iterations

vertex processed	vertex labeled	label
	s	[s+, ∞]
s	a	[s+, 2]
a	d	[a+, 1]
d	e	[d+, 1]
e	b	[e-, 1]
	h	[e+, 1]
b	Nothing	
h	t	[h+, 1]

So the augmenting path is

$$(s, a, d, e, h, t).$$

The new flow is shown in Figure 10.

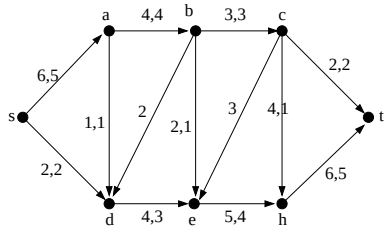


Figure 10: Ford-Fulkerson example, after 5 iterations

vertex processed	vertex labeled	label
	s	[s+, ∞]
s	a	[s+, 1]
a	Nothing	

So there is no augmenting path. So the maximum flow is the flow in

Figure 10 and a minimum cut

$$(\{s, a\}, \{b, c, d, e, h, t\})$$

is described in Figure 11.

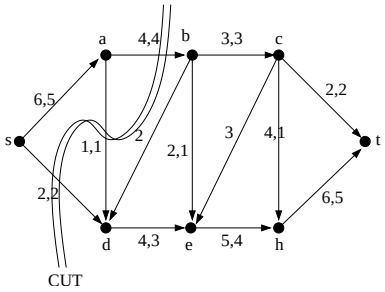
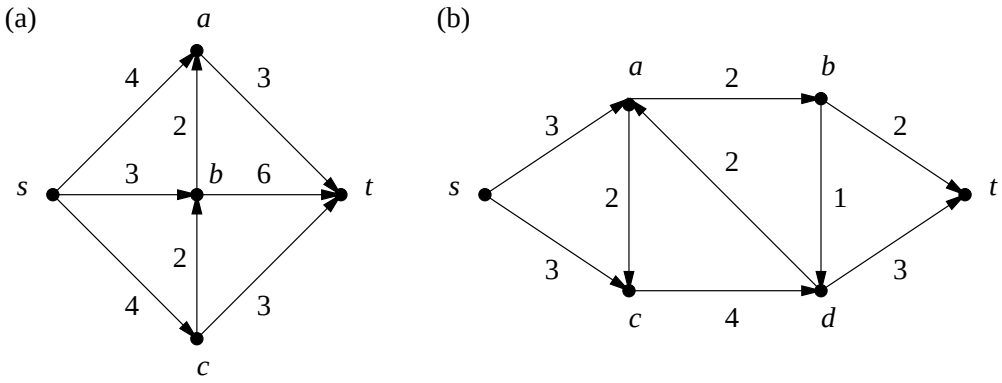


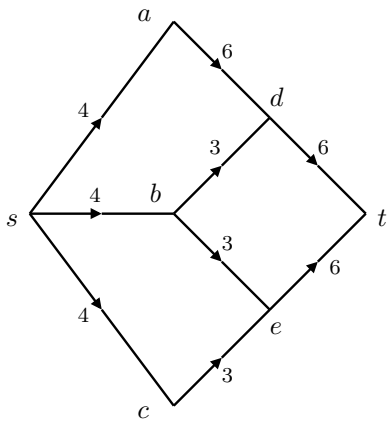
Figure 11: Ford-Fulkerson example, finished with a min-cut

5 Exercises 2

1 Use the Ford-Fulkerson Algorithm to maximize the flow of the following network.



2 Use the Ford-Fulkerson Algorithm to maximize the flow of the following



6 Some applications

6.1 Supply and Demand

Suppose we have a collection of factories that supply a collection of retailers. The basic problem is to determine if all the factories can produce enough to supply all the retailers, through the distribution chain.

In this problem, we have a set S of sources $x_1, x_2, \ldots, x_\alpha$ and a set T of sinks $y_1, y_2, \ldots, y_\beta$. The sets are mutually exclusive, $S \cap T = \emptyset$. Every source x_i has an associated number $s(x_i)$ which is the maximum supply from x_i. Every sink y_i has an associated number $d(y_i)$ which is the demand at y_i. We say *supply meets demand* if we can find a flow in the network whose output is at least equal to the demand.

The flow must meet the following constraints.

1. If $x \in S$ then $f(x, V) - f(V, x) \leq s(x)$
2. If $y \in T$ then $f(V, y) - f(y, V) \geq d(y)$
3. For all other vertices, $z \notin S \cup T$, $f(z, V) - f(V, z) = 0$
4. $0 \leq f(a, b) \leq c(a, b)$, as per usual.

Let N be a supply-demand network with vertex set V and let $X \subseteq V$. We define

$$s(X) = \sum_{x \in S \cap X} s(x)$$

and

$$d(X) = \sum_{y \in T \cap X} d(y)$$

Theorem 6.1. *Let N be a supply-demand network with vertex set V and let $X \subseteq V$. Supply can meet demand if and only if, for every cut $(X, \overline{X})$,*

$$d(\overline{X}) - s(\overline{X}) \leq c(X, \overline{X})$$

Proof. Suppose that supply meets demand. Let $(X, \overline{X})$ be a cut. The total demand from retailers in $\overline{X}$ is $d(\overline{X})$. The maximum supply from suppliers is $s(\overline{X})$. The rest of the demand must be produced from suppliers in X. Given any flow f, the net amount that flows from X to $\overline{X}$ is $f(X, \overline{X}) - f(\overline{X}, X)$. This flow cannot exceed the capacity $C(X, \overline{X})$. So the supply still required must be less than or equal to the capacity available from X. Hence $d(\overline{X}) - s(\overline{X}) \leq C(X, \overline{X})$.

The other direction is omitted. □

This theorem tells us how to find out whether supply can meet demand in supply-demand network. First, set up a network N' with one source (a *super source*) and one sink (a *super sink*). In N', the super source has edges to the suppliers and the retailers have edges to the super sink. The capacity of the new edges are the values of the functions s or d respectively.

If the flow fills all the edges into t, then there is a solution to the supply-demand problem. If not, the problem is unsolvable. So, in effect, we find a maximum flow (in N') that saturates all new edges from the retailers to the super sink. The flow that meets the supply-demand problem is just the saturating maximum flow of N' when it's restricted to the original network.

Below, in Figure 12, is a diagram of how the procedure works.

6.2 Matching theory

Next we look at a problem associated with bi-partite graphs. Recall that a graph G is bipartite if the vertices of G can be partitioned into two cells X and Y such that all edges are incident with a vertex from each cell. Equivalently, a graph is bipartite if all it's circuits contain an even number of vertices. Also, a graph is bipartite if it is two colourable.

A typical matching problem is the following. There are 6 people and 6 jobs. Some people can only do some jobs. Each job needs just one person. Is it possible to match every person with a job.

A *matching* in a bipartite graph is a collection of edges for which no two

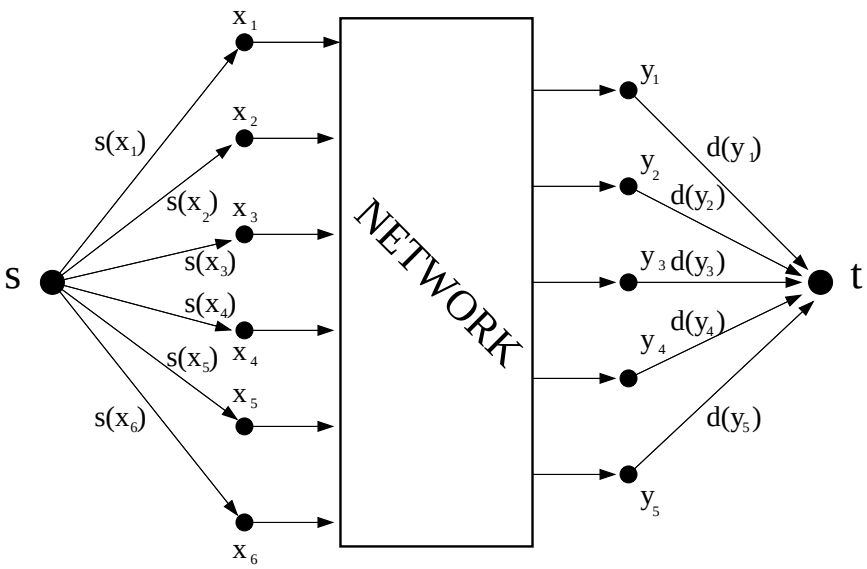


Figure 12: Supply-demand network

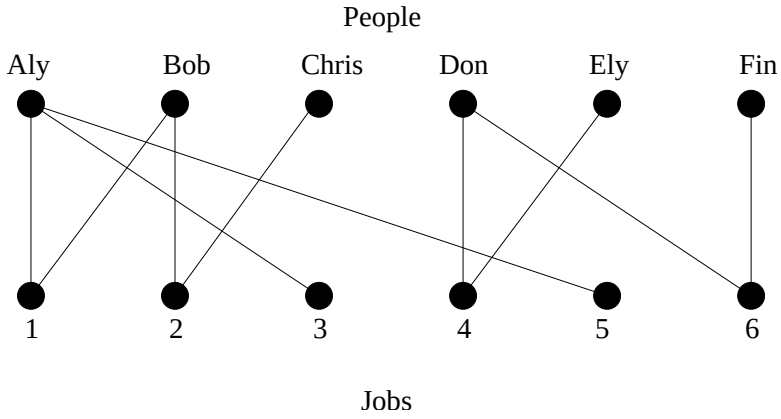


Figure 13: People and jobs

edges are incident to a common vertex. A matching in the example above is

$$\{(Aly, 1), (Bob, 2), (Don, 4), (Fin, 6)\}.$$

A matching is *maximal* if it cannot be extended. So M is a maximal matching if there is no larger matching N that contains M as a subset.

A matching is *maximum sized* if there is no larger matching. So no matching in the graph has more edges.

A matching is *complete* if every vertex is incident with an edge in the matching.

In the example below, we give a matching that is maximal, not maximum sized and explain why the bipartite graph has no complete matching. The matching is the edges that are emboldened.

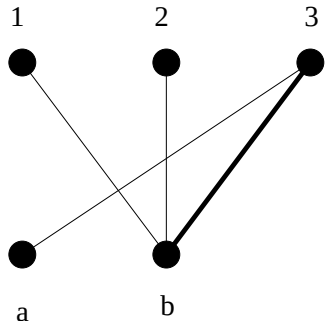


Figure 14: A bipartite graph

This matching $\{(3, b)\}$ is maximal as we cannot extend it (all other edges are incident with either 3 or b.) It's not a maximum sized matching as there are larger matchings, such as $\{(3, a), (1, b)\}$. The bipartite graph cannot have a complete matching as there are only 2 vertices $\{a, b\}$ in one part and 3 vertices $\{1, 2, 3\}$ in the other.

So the problem is to find a maximum sized (and hopefully complete) matching in a bipartite graph. To do this, we will use network flows.

6.3 Using network flows

The method is as follows. Let G be a bipartite graph with parts A and B. Orient the edges of G, so that all edges go from A to B. Add two vertices, s and t, to the digraph and add directed edges as in the supply-demand case. So, for each vertex a in A, there is a directed edge (s, a). Similarly, for each vertex b in B, there is a directed edge (b, t). Give every edge a weight (or capacity) of 1. Then the resulting digraph is a network.

Below is an example, of a bipartite graph and it's associated network. Remember, each edge in the network (Figure 16) has capacity 1.

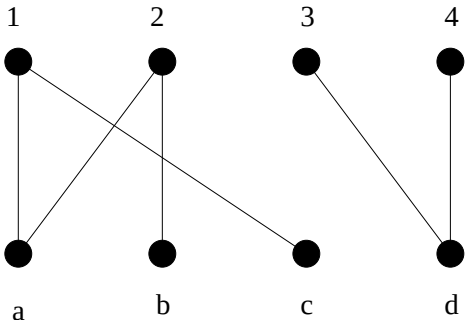


Figure 15: A matching

Suppose that f is a flow in the network N associated with a bipartite graph G. Then the collection of edges in N with non-zero flow, when considered as (undirected) edges in G, is a matching. Conversely, for every matching in G, there is a flow in N that has non-zero flow precisely on the (directed) edges that are in the matching.

So a maximal flow in N gives a maximum sized matching in G.

Let $(X \cup Y, E)$ be a bipartite graph with parts X and Y. For each subset X' of vertices of X, let

$$N(X') = \{y \in Y : (x, y) \in E \text{ for some } x \in X'\}.$$

We call $N(X')$ the *neighbours of* X'. For example, in Figure 15, $N(\{1, 2\}) = \{a, b, c\}$.

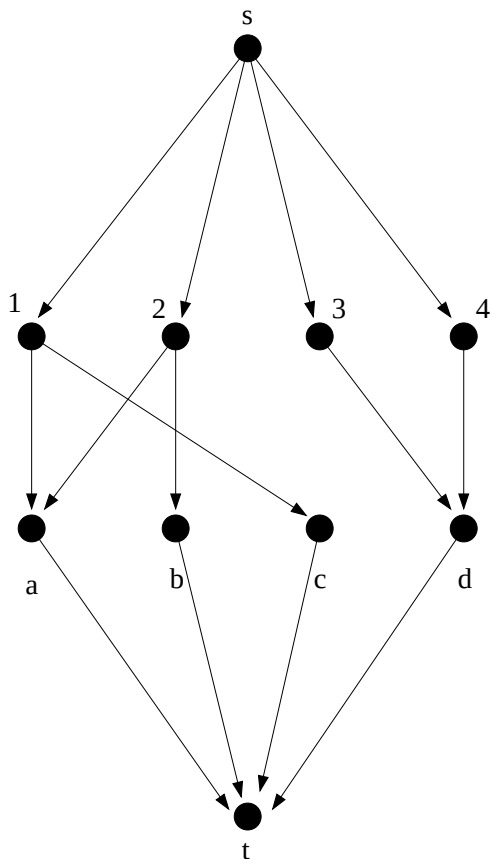


Figure 16: The network associated with Figure 15

Theorem 6.2 (Halls Marriage)**.** *Let $G = (X \cup Y, E)$ be a bipartite graph. Then G has a complete matching if and only if, $|X| = |Y|$ and, for every $X' \subseteq X$, $|N(X')| \geq |X'|$.*

One direction is easier than the other. Can you guess which one?

Proof. First the easier direction. Suppose that G has a complete matching. Then

$$|X| = |Y| .$$

Now for the inequality. Let X' be a subset of X and let M be the collection of edges in the complete matching that are incident with a member of X'. We'll use M to count some members of $N(X')$. As M is a subcollection of the matching,

$$|M| = |X'| .$$

Let Y' be the collection of vertices, in Y, that are incident with edges of M. Then, again as M is a subcollection of the matching,

$$|Y'| = |M| .$$

But Y' is a subset of $N(X')$, so

$$|Y'| \leq |N(X')| .$$

Combining all these together, we see that

$$|X'| \leq |N(X')| .$$

Now for the more difficult part. Suppose that $|X| = |Y|$ and, for every subset X' of X, $|N(X')| \geq |X'|$. We will use network flows to show that the graph has a complete matching. Let N be the network associated with the bipartite graph.

Recall that, by the max-flow min-cut theorem, the maximum value a flow can take is equal to the minimum capacity a cut can have. Remember that a flow of maximum value in the network will result in a maximum-sized matching in the bipartite graph. So combining these two facts, if we can

show that any cut in the network has capacity at least $|X|$, then there will be a complete matching in the bipartite graph.

Let $n = |X|$. Let $(A, \overline{A})$ be a cut in the network N. Let $X' = A \cap X$ and $Y' = A \cap Y$. So $A = \{s\} \cup X' \cup Y'$. Figure 17 below, may help with the next calculation.

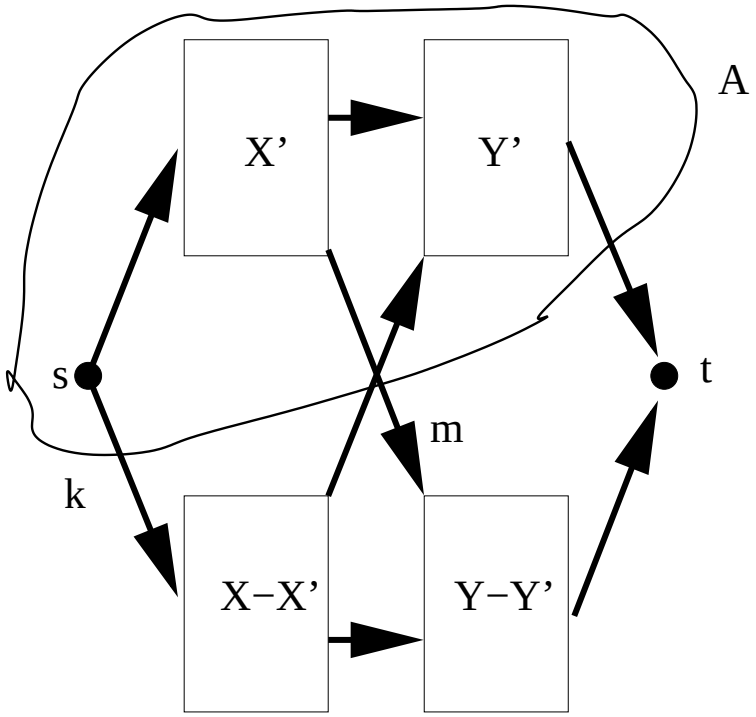


Figure 17: Help for Halls marriage theorem

The capacity of the cut is

$$C(A, \overline{A}) = |X - X'| + |Y'| + C(X', Y - Y').$$

Let $|X - X'| = k$. So there are k vertices in $|X - X'|$. So, by the construction, there are k edges from s to $X - X'$ and $n - k$ edges from s to X'.

Recall Halls' condition on X', that $|N(X')| \geq |X'|$. So there are at least $n - k$ neighbours of X'. Hence there must be at least $n - k$ edges from X' to

Y. Suppose that m neighbours are in $Y - Y'$, then there must be $n - k - m$ reminding neighbours in Y'.

Since there are m neighbours in $Y - Y'$ there must be at least m edges from X' to $Y - Y'$. Since there are at least $n - k - m$ members of Y' there must be at least $n - k - m$ edges from Y' to t. This gives a total of at least $k + m + (n - k - m) = n$ edges out of the cut. □

We also present another algorithm, which is quicker, but does not necessarily find a maximum-sized matching. It will find a maximal matching.

6.4 First-fit back-tracking algorithm

The algorithm runs in $|X|$ steps. At each step k, match x_k to some y_k not already matched. If you can find the match, then extend your matching. If you cannot do match x_k to anything new, then rematch.

Figure 18 below might help?

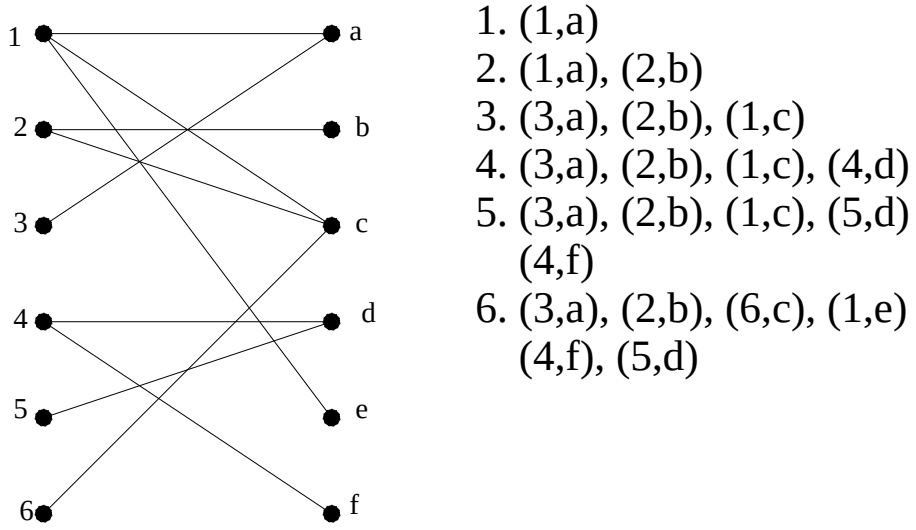


Figure 18: An example of first-fit back-track

So the matching found here is

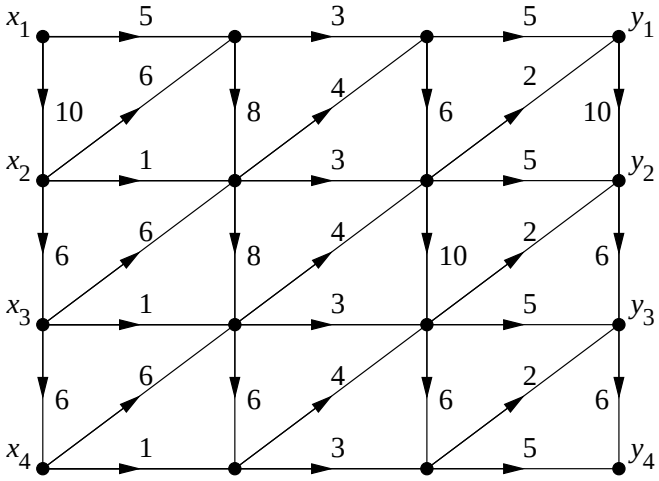
$$(1, e), (2, b), (3, a), (4, f), (5, d), (6, c).$$

The algorithm will find a matching or it may collapse. Remember it doesn't guarantee a maximum-sized matching, just a maximal matching. By chance, the example in Figure 18, gives a complete matching.

7 Exercises 3

1

- Goods are manufactured at 4 factories x_1, x_2, x_3 and x_4 and are to be transported to 4 stores y_1, y_2, y_3 and y_4, through the transport network with capacities shown in the figure below.

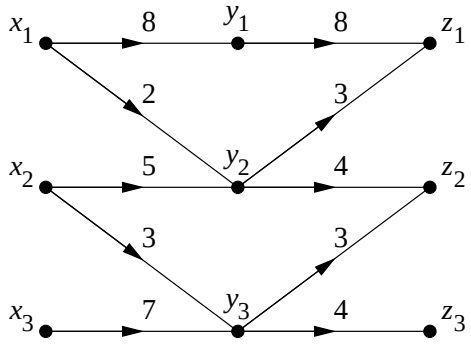


Factory x_1 can make at most 12 units per month.
Factory x_2 can make at most 12 units per month.
Factory x_3 can make at most 1 unit per month.
Factory x_4 can make at most 1 unit per month.
Retailer y_1 needs at least 3 units per month.
Retailer y_2 needs at least 10 units per month.
Retailer y_3 needs at least 3 units per month.
Retailer y_4 needs at least 10 units per month.

Can supply meet demand? If so, how many goods should each factory produce? If not, find a partition $(X, \bar{X})$ of the vertices for which $d(\bar{X}) - s(\bar{X}) > c(X, \bar{X})$ is false.

- In the network below, x_1, x_2 and x_3 are manufacturers; y_1, y_2

2 In the network below, x_1, x_2 and x_3 are manufacturers; y_1, y_2 and y_3 prepare the product for sale and then distribute it to the retailers z_1, z_2 and z_3. The capacities of the links are shown.



Factories x_1, x_2 and x_3 can produce 8, 4 and 9 units per week respectively. No distribution plant can operate economically if the weekly input is less than 5 units; y_1 and y_2 are each capable of processing 7 units per week, while y_3 can process 6. Of the retailers, z_1 needs 9 units per week, z_2 needs 5 and z_3 needs 4. Can supply meet demand while satisfying the constraints on y_1, y_2 and y_3?

3 Suppose that there are 5 committees. Committee A's members are a, c, e; B's are b, c; C's are a, b, d; D's are d, e, f and E's are e, f. Our goal is to have each committee send a different representative to a convention.

(i) Set up a bipartite graph to describe this situation. Use a network flow algorithm to determine if it is possible to chose such distinct representatives.

(ii) Use the backtracking algorithm to do the same.

(iii) Make a flow that corresponds to the partial matching A, e; B, b; C, a; D, f. Apply Ford-Fulkerson to this partial flow.

8 Menger's Theorems

8.1 Connectivity

Recall that a graph G is connected iff for all vertices x and y or G, there is a path joining x to y. Similarly if G is a directed graph then G is connected if there is a directed edge from x to y in G. We say that G is n-connected if for all such x and y there are n disjoint paths connecting x to y. We can also specialize this to two particular vertices v and w of G. Namely v and w are called n-connected iff there are at least n (edge) disjoint paths connecting v to w in G.

Suppose that G is a nontrivial graph with an isthmus. Then G can't be 2-connected! Menger's theorems tells us that such *separations* determine connectivity.

A set of edges E is called a *n-edge separation* or *n-edge cutset* iff the removal of these n edges disconnects the graph. Similarly for a fixed v and w we say that E is a n-cutset or n-separation of v and w iff the removal of these n edges disconnects v and w.

Theorem 8.1 (Menger)**.** *If the minimum sized v, w cutset in a directed (multi-)graph G has size n, then v is n-edge connected to w in G.*

Proof. Let $G = (V, D)$ be a directed graph and v, w in G as above, satisfying the hypotheses of the theorem.

Now regard v as the source and w as the sink, and give each edge capacity 1. Let E be a minimal v, w edge cutset in G. Thus if all but one of the edges is removed then is a path from v to w and yet if we remove them all then there is no such path. Suppose that U is the set of all vertices x such that there is a path from v to x in $(V, D - E)$, and let C be the partition $(U, V - U)$. Then C is a cut. Moreover the only edges from U to $V - U$ are in E and hence the capacity of C is $|E| = n$. Summarizing, if the minimum v, w cutset has size n then the minimum cut capacity is n. Now by the Ford-Fulkerson Theorem there is a flow of size n from v to w.

We need to prove that this means that the flow constitutes n *disjoint* paths from v to w. This is an easy induction as we now see. Let S be the set of all k such that that there are k disjoint paths from v to w in G. Clearly 1 is in S. Assume that m is in S. Then there these m paths constitute a flow of size m. Remove the edges of these m paths. Call the result $\hat{G}$. Then there is still a flow of $n - m$ in $\hat{G}$. Hence if $m < n$, there is a path from v to w in $\hat{G}$, and this is disjoint from S. □

Menger's Theorem has a more standard vertex separation version. We say that a set of vertices S is a n-vertex cutset or n separation in G if the removal of the n vertices (and of course the edges from them) of S disconnects G.

Theorem 8.2 (Menger)**.** *If the minimum sized v, w vertex separation in a directed graph G has size n, then v is n-(vertex-)connected to w in G.*

Proof. Take each vertex x of G and replace it by an edge x_1x_2 and then for any edges xy of G replace it by x_2y, and any edge px replace it by px_1. Thus we "split" the vertices. This forms G'. A vertex cutset in G is a edge cutset in G'. □

Menger's Theorem is also true in the undirected case.

Theorem 8.3 (Menger)**.** *If the minimum sized v, w cutset in in undirected graph G has size n, then v is n-edge connected to w in G.*

Proof. Take G and for each edge $e = xy$ of G replace it by a pair of directed edges xy and yx. call the result G'. Notice that then an n-edge cutset in G will be a v, w-cutset of size $2n$ in G'. Now, again note that the reasoning of the first theorem shows that the maximum capacity of a cut in G' is $\leq 2n$ and at least n. Thus by Ford -Fulkerson there is a flow of size n from v to w in G', and for each such cut $C = (U, G' - U)$, this flow has all edges from U to $G' - U$ saturated, and all from $G' - U$ to U zero. Below we will show that in any such flow we can only use one of xy and yx, and hence the value of the flow will be n and we can lift the flow back to the undirected case.

Specifically, for any other edge pq which has been replaced by two pq and qp, suppose that some path of the flow uses both pq and and other uses qp. Let these be

$$v = x_0 \dots p(pq)q \dots w, \text{ and}$$

$$v = x_0 \dots q(qp)p \dots w.$$

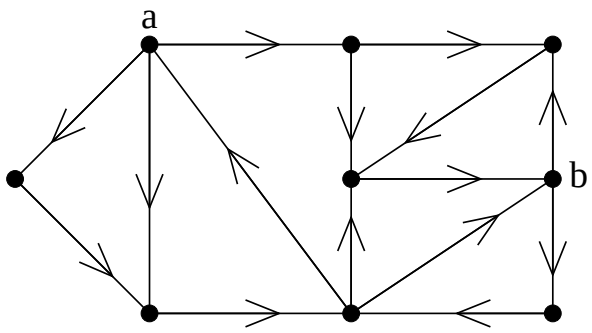
Then we can replace these paths by two that use neither pq nor qp. Namely, start at v, travel to p using the first path, and then to w using the second, and for the other one, travel from v to q using the second, then then from q to w using the first. Thus we can eliminate all simultaneous occurrences of both pq and qp, meaning that the paths will correspond to edge disjoint paths of G. □

You should prove for yourself the vertex form of Menger's Theorem for undirected graphs.

8.2 Exercises 4

1 Prove the vertex version of Menger's Theorem for undirected graphs.

2 *Use one of the Network Flow Algorithms* to find the largest n such that a is n edge connected to b in the directed graph below.



3 Find the maximum sized vertex disjoint paths in from a to b in the figure of Question 2 above.

9 Preflow-Push algorithms*

I won't have time to discuss these efficient algorithms, but will include for completeness. For more on this I refer the reader to Ahuja-Orlin [1].

9.1 The Goldberg-Tarjan Algorithm

The idea behind this algorithm is aggressive-passive. You push as much as you can into the network, then trim the excess. So while the algorithm is running, the flow in and out of vertices may not be equal. We call the difference at each vertex v the *excess at* v. Using a relabeling algorithm, we eventually makes the excess at each vertex 0. As the algorithm will not have a flow until the very end, we call the temporary amounts in the network a *pre-flow*.

First some terminology. Let f be a pre-flow in the network. The excess at x is denoted $e(x)$ and is computed by

$$e(x) = f(B(x), x) - f(x, A(x)).$$

We say x is *active* if $e(x) > 0$. The source and sink are never active. The *residual capacity* of an edge (x, y) is denoted by $r(x, y)$ and calculated by

$$r(x, y) = c(x, y) - f(x, y) + f(y, x).$$

The *residual network* is the collection of edges whose residual capacity is non-zero We denote this network by $R = \{(x, y) : r(x, y) > 0\}$. A distance function d is a function from V to $\mathbb{R}$ which satisfies two conditions. First, $d(t) = 0$. Second, if $(i, j) \in E$ and $C(i, j) > 0$ then $d(i) \leq d(j) + 1$. These two conditions will control the direction of the algorithm and make it try to push more towards the sink.

9.2 The actual algorithm

Preprocess. Set up the initial conditions. Set $d(s) = |V|$, $d(t) = 0$. For every other vertex x (not the source or sink), set $d(x) = 1$. For every edge (s, x), set $f(s, x) = C(s, x)$.

Process. While there is an active node, do the following. Select an active node i. We will try to push more pre-flow towards the sink. There are two situations in which we push.

1. Suppose that $(i,j) \in E$, $d(i) = d(j) + 1$ and $r(i,j) > 0$. Then there is capacity to push more from i to j. We push as much as the excess at the vertex and the residual capacity of the edge, will allow. So push $\min\{e(i), r(i,j)\}$ from i to j and change f accordingly.
2. Suppose that $(j,i) \in E$, $d(i) = d(j) + 1$ and $r(i,j) > 0$. Then there is capacity to push more from i to j. We push as much as the excess at i and the residual capacity from the edge (i,j), will allow. So push $\min\{e(i), r(i,j)\}$ from i to j and change f accordingly.

If we cannot push anything from i, then relabel i as follows. Replace $d(i)$ with

$$\min\{d(j) + 1 : j \in A(i) \cup B(i) \text{ and } r(i,j) > 0\}$$

Finish. Once we have finished, the pre-flow is a maximum flow.

We omit the proof for this algorithm, pointing only at Ahuja-Orlin [1] where preflow-push algorithms are discussed, and another (the excess scaling algorithm) is also discussed. What follows is an example of how the algorithm works in practice.

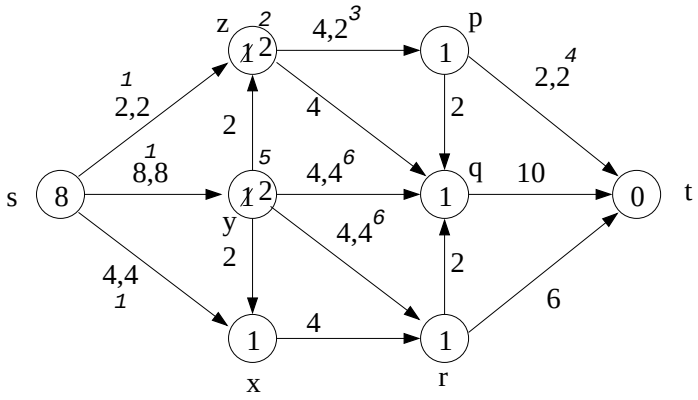


Figure 19: The first 6 steps

it	z	y	x	p	q	r	sz	sy	sx	zp	zq	yz	yx	yq	yr	xr	pq	rq	pt	qt	rt	ac
1	1	1	1	1	1	1	2	8	4	0	0	0	0	0	0	0	0	0	0	0	0	x,y,z
2	2	1	1	1	1	1	2	8	4	0	0	0	0	0	0	0	0	0	0	0	0	x,y,z
3	2	1	1	1	1	1	2	8	4	2	0	0	0	0	0	0	0	0	0	0	0	x,y,p
4	2	1	1	1	1	1	2	8	4	2	0	0	0	0	0	0	0	0	2	0	0	x,y
5	2	2	1	1	1	1	2	8	4	2	0	0	0	0	0	0	0	0	2	0	0	x,y
6	2	2	1	1	1	1	2	8	4	2	0	0	0	4	4	0	0	0	2	0	0	x,q,r
7	2	2	1	1	1	1	2	8	4	2	0	0	0	4	4	0	0	0	2	4	0	x,r
8	2	2	1	1	1	1	2	8	4	2	0	0	0	4	4	0	0	0	2	4	4	x
9	2	2	2	1	1	1	2	8	4	2	0	0	0	4	4	0	0	0	2	4	4	x
10	2	2	2	1	1	1	2	8	4	2	0	0	0	4	4	4	0	0	2	4	4	r
11	2	2	2	1	1	1	2	8	4	2	0	0	0	4	4	4	0	0	2	4	6	r
12	2	2	2	1	1	2	2	8	4	2	0	0	0	4	4	4	0	0	2	4	6	r
13	2	2	2	1	1	2	2	8	4	2	0	0	0	4	4	4	0	2	2	4	6	q
14	2	2	2	1	1	2	2	8	4	2	0	0	0	4	4	4	0	2	2	4	6	

In this case the algorithm managed to push the entire initial amount through to the sink. Note that steps 2, 5, 9 and 12 were relabel steps. Also note that we did two pushes in step 6. This example was very simple and required no trimming as such.

Below is a simpler example that does some trimming.

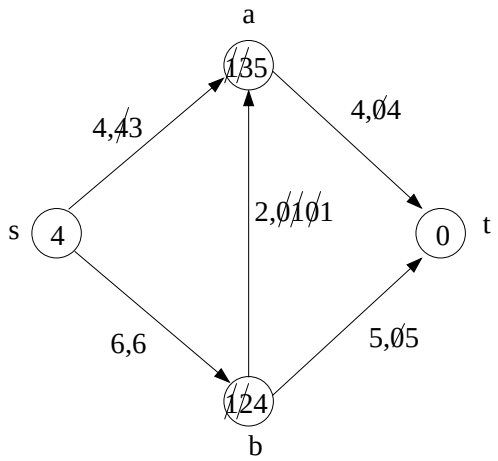


Figure 20: The entire algorithm, on a small example.

it	a	b	sa	sb	ba	at	bt	ac
1	1	1	4	6	0	0	0	a,b
2	1	1	4	6	0	4	5	b
3	1	2	4	6	0	4	5	b
4	1	2	4	6	1	4	5	a
5	3	2	4	6	1	4	5	a
6	3	2	4	6	0	4	5	b
7	3	4	4	6	0	4	5	b
8	3	4	4	6	1	4	5	a
9	5	4	4	6	1	4	5	a
10	5	4	3	6	1	4	5	

Notice a few things. At stage 10, the label (5) of vertex a was high enough to push pre-flow back to s (with label 4). Stages 3 through 10 were pushing pre-flow back and forth between vertices a and b. We did two pushes at stage 2. We relabeled at stages 3, 5, 7 and 9.

9.3 Projects

(i) What is the current situation regarding the "best" network flow algorithm. is running time all that matters? How do data structures affect

the running times? All of these algorithms have choices in them as to where to go next. Does it matter? Does it matter *on average*?

(ii) What is the situation of *min flow*. What about negative edge weights?

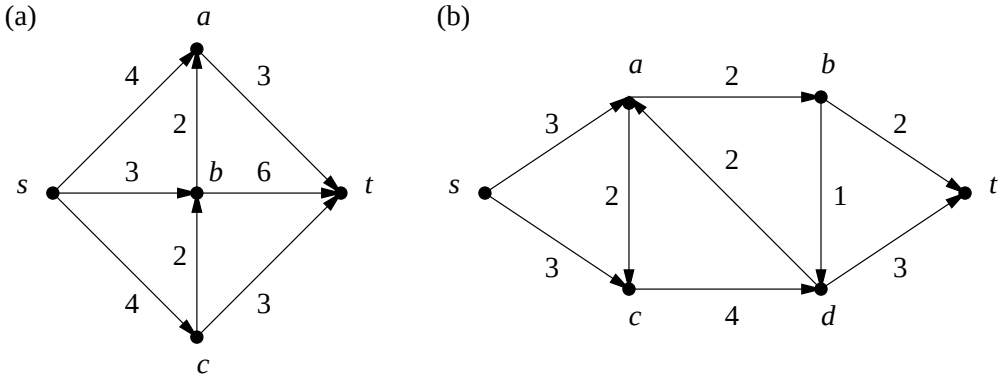
(iii) What about matching in *infinite graphs*. Is there such a result as Menger's theorem for infinite graphs? (ref Erdös conjecture and the workk or Aharoni and Diestel) what about Hall's theorem? (ref König's duality theorem)

(iv) How does matching in general graphs work. How it it useful. How is it used in *reduction algorithms* such as *crown reductions?* What about minimal/maximal matchings in weighted graphs? What about preferential matching such as *stable marriage*. How does this relate to voting schemes and *Arrow's Theorem*?

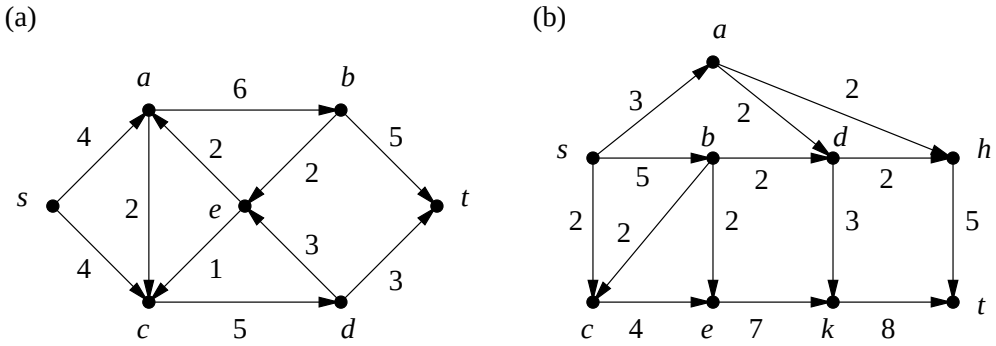
(iv) What about milti-source multi-sink networks. What is the complexity of k-cut? What about the planar situation?

10 Exercises 5

1 Use the Goldberg-Tarjan Algorithm to maximize the flow of the following (circle changes)



2 Use the Goldberg-Tarjan Algorithm to maximize the flow of the following (circle changes)



11 Treewidth

One of the major outgrowths from the beautiful work of Robertson and Seymour on "graph minors" and "immersions" was the identification of a new parameter which measures in a precisely defined sense how "tree-like" a graph is. The fundamental idea is that we should be able to lift many results from trees to graphs that are "tree-like." (Equivalent notions to tree decomposition have been subsequently, and apparently independently discovered in other contexts such as the artificial intelligence literature.)

In the present section we will look at a new parameter (treewidth) and associated parameters such as pathwidth. Then we will look at the uses of bounded treewidth for algorithm design.

The best known definition of treewidth comes from *partial k-trees.*

Definition 11.1 (k-tree)**.** (a) The class of k-trees is the smallest class obeying (i) and (ii) below.
(i) K_{k+1}, the complete graph on $k+1$ vertices, is a k-tree.
(ii) If G is a k-tree and H is a subgraph of G isomorphic to K_k, then the graph G' constructed from G by first adding a new vertex v to G and then adding edges to make $H \cup \{v\}$ a copy of K_{k+1}, is a k-tree.

(b) If G_1 is a subgraph of a k-tree, then G_1 is called a *partial k-tree.*

In Figure 21 we give examples of 2- and 3- trees.

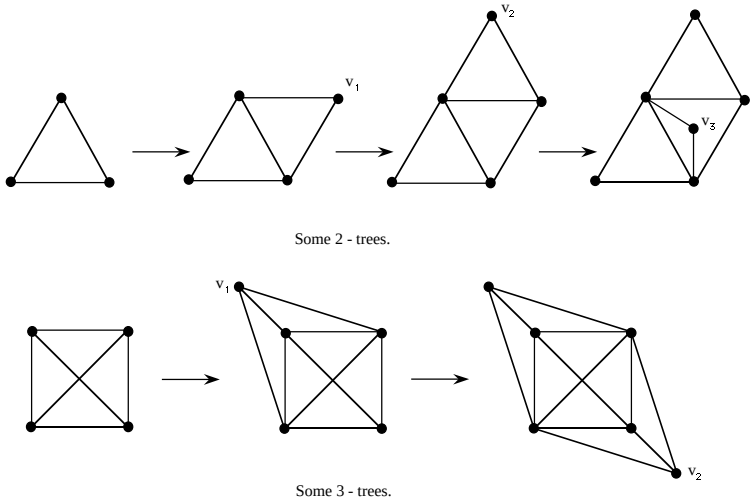


Figure 21: Examples of 2- and 3-trees

Definition 11.2 (Treewidth)**.** We say that a graph G has *treewidth* k if k is least such that G is a partial k tree.

For algorithmic purposes, Definition 11.2 is not the most useful. An equivalent characterization of treewidth is provided by the following.

Definition 11.3 (Tree Decomposition)**.** (a) A *tree-decomposition* of a graph $G = (V, E)$ is a tree $\mathcal{T}$ together with a collection of subsets T_x (called *bags*) of V labeled by the vertices x of $\mathcal{T}$ such that $\cup_{x \in \mathcal{T}} T_x = V$ and 1. and 2. below hold:

1. For every edge uv of G there is some x such that $\{u, v\} \subseteq T_x$.

2. (Interpolation Property) If y is a vertex on the unique path in $\mathcal{T}$ from x to z then $T_x \cap T_z \subseteq T_y$.

(b) The *width* of a tree decomposition is the maximum value of $|T_x| - 1$ taken over all the vertices x of the tree $\mathcal{T}$ of the decomposition.

If a tree decomposition of a graph G gives a path, then we say that the tree decomposition is a *path decomposition*, and use *pathwidth* in place of treewidth.

Figure 22 gives an example of a tree decomposition of width 2.

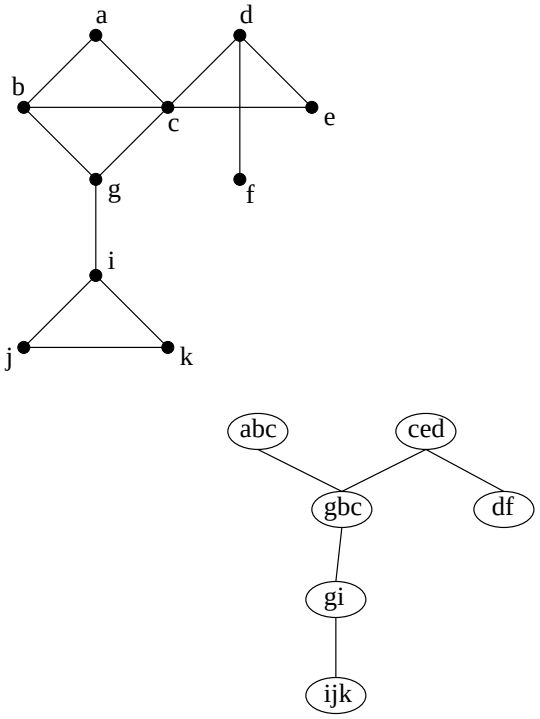


Figure 22: Example of Tree Decomposition of Width 2

The axioms above imply certain connectivity conditions on the graphs.

Lemma 11.4. *Let $\{T_x : x \in \mathcal{T}\}$ be a tree decomposition of G.*
(i) Let $x \in V(G)$. Then the collection of $y \in \mathcal{T}$ with $x \in T_y$ forms a subtree of T.
(ii) Suppose that C is a clique of G. Then there is some $x \in \mathcal{T}$ with $C \subseteq T_x$.

We leave the proof as an exercise (Exercise 1 below). The following basic result demonstrates that the two notions of width are identical.

Theorem 11.5 (Arnborg)**.** *A graph has treewidth k iff G has a tree decomposition of width k.*

Proof. Let G be a graph with a tree decomposition $\{T_x : x \in \mathcal{T}\}$ of width k. Thus $\max_{x \in \mathcal{T}} |T_x| = k + 1$. We prove that G is a partial k-tree, and in fact

G is a subgraph of a k-tree $K(G)$ in which every $\leq k$ element subset of a T_x is part of a k-clique. We use induction on trees $\mathcal{T}$. Let x be a leaf of $\mathcal{T}$. Let $\mathcal{T}'$ be the result of removing x from $\mathcal{T}$, and let G' be a graph corresponding to $\mathcal{T}'$. By hypothesis, G' is a subgraph of a k-tree $K(G')$ with the desired properties. Let y be attached to x in $\mathcal{T}$. Let $T_x \cap T_y = T'_x$ and $T_x - T'_x = T''_x$. If $T''_x = \emptyset$, then $T_x \subseteq T_y$ and hence $G = G'$. So suppose that $T''_x \neq \emptyset$. By property 2. of Definition 11.3, for all $z \neq x$, $T''_x \cap T_z = \emptyset$. It follows that T'_x is a proper subset of T_x, and hence as the width is k, T'_x is a subset of at most k elements of T_y. By hypothesis, this means that T'_x is a subset of part of a k-clique C of $K(G')$. The idea is now to simply add T''_x one node at a time. We use the following algorithm.

Algorithm 11.6. 1. Initially we let $K = K(G')$.
While $T''_x \neq \emptyset$ we repeat the following steps 2-5.

2. Pick a vertex v of T''_x.
3. Add v to K and also add all edges between vertices of C and v.
4. Let $T''_x \leftarrow T''_x - \{v\}$.
5. If there is any vertex $c \in C - T_x$, let $C \leftarrow [C - \{c\}] \cup \{v\}$.

By induction on the algorithm, one can see that after each iteration, C is a k-clique of K. Furthermore each step emulates the definition of a k-tree and hence as $K(G')$ is a k-tree, so too is K_{fin} the result of Algorithm 11.6 applied to $K(G')$. By induction upon the algorithm, we also see that all of T''_x is added together with all possible edges that can exist between vertices in $T_x = T'_x \cup T''_x$. Thus K_{fin} contains G and has the desired properties.

• Conversely, suppose that G is a partial k-tree. Since it suffices to consider k-trees, we will in fact suppose that G is a k-tree. We prove by induction on $|V(G)|$ that G will have a tree decomposition of width k. We use the stronger hypothesis that if G' is a k-tree with $< |V(G)|$ vertices, then G' has a tree decomposition $\{T_x : x \in \mathcal{T}\}$ where $|T_x| \leq k+1$ for all $x \in \mathcal{T}$, and each k-clique of G' occurs in some T_x. The result is clear if $G = K_{k+1}$. So suppose that G has more than $k+1$ nodes and let v be the last node added in the formation of G. (That is, v is a *simplicial* node.) Let $N(v)$ be the neighbours of v in G. Let G' be the result of the removal of v and the k edges vu with $u \in N(v)$. Then we can apply the induction hypothesis

to G' and obtain a tree decomposition $\{T'_x : x \in \mathcal{T}'\}$ of G' with the desired properties. Now $N(v)$ is a k-clique and hence, by hypothesis, there is some $y \in T'$ with $N(v) \subseteq T'_y$. Create a new tree $\mathcal{T}$ from $\mathcal{T}'$ by attaching a new leaf $\ell = \ell(v)$ at y. Let $T_\ell = N(v) \cup \{v\}$. By hypothesis, $\mathcal{T}$ will be the relevant tree decomposition of G. □

Definition 11.7 (Bounded Treewidth)**.** We say that a class $\mathcal{C}$ of graphs has *bounded treewidth* if there is some k such that for all $G \in \mathcal{C}$, G has treewidth $\leq k$.

Historically, authors often tried to get around classical intractability by looking at the problems at hand on special classes of graphs. Authors often discovered that intractable problems became tractable if the problems were restricted to say, "outerplanar" graphs. Such restriction is not purely an academic exercise since, in many practical situations, the graphs that arise do not in fact demonstrate the full pathology of the class of all graphs.

11.1 Exercises 6

1 Prove Lemma 11.4.

2 (W. Merkle) An ordering $I = \langle I_1, ..., I_n \rangle$ of a set of subsets of V is called an *acyclic hypergraph* iff for all $j \leq n$, there is a $q < j$ such that $I_j \cap (\cup_{i<j} I_i) \subseteq I_q$. (This is also called the *running intersection property*, and we refer to Lauritzen and Spiegelhalter [31].) Given a finite graph, we say that an acyclic hypergraph I is an acyclic hypergraph decomposition of G iff $V = V(G)$ and for each edge e of G there is some i such the vertices of e are in I_i. Prove that G has a tree decomposition of width w iff G has an acyclic hypergraph decomposition I such that for all $i \leq n$, $|I_i| \leq w$. (Hint: Prove that one can construct the tree decomposition from I to have the I_i as the actual bags, and conversely.)

3 Apply the algorithm implicit in Theorem 23 to embed the graph described by the tree decomposition of Figure 23 into a 3-tree.

4 Prove that if a graph G has a size k vertex cover, then it must also have treewidth $\leq k$. (A Vertex Cover of a graph G is a collection

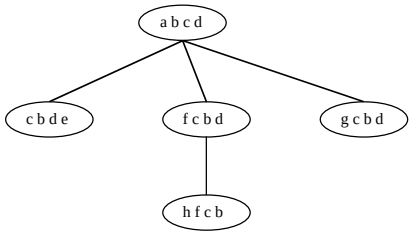


Figure 23: Example of Tree Decomposition of Width 3

Q of vertices of G such that if xy is an edge of G then either x is in Q or y is in Q.)

5 A *Halin graph* is constructed as follows: take a tree without vertices of degree 2. Then, embed in in the plane, and add a cycle through its leaves, in order (i.e., the result remains planar). Prove that a Halin graph must have treewidth 3.

6 A graph is called *outerplanar* iff there exists a plane embedding G of the graph such that G has a cycle C with the property that every edge of G is either a member of C or a chord of C. Prove that if G is outerplanar then it has treewidth 2.

12 Algorithms for graphs of bounded treewidth

We sketch how to run algorithms on graphs of bounded treewidth. This can be viewed as *dynamic programming* a very important algorithmic technique. Many classical problems are known to be algorithmically infeasible on general graphs (in that they take exponential time) but become polynomial time when restricted to some bounded treewidth class. A good introduction to this technique is Bodlaender [2], and a good introduction to other algorithmic aspects of treewidth is Bodlaender [3].

We describe the technique for the INDEPENDENT SET problem we first met in the Ramsey Theory' section. Recall that this asks for a set of vertices I in G with no edges between them in G. The problem of finding the largest independent set in a graph G is one of the known "NP-complete" problems

where basically we don't know how to better than complete search for general graphs. Hence the current best running time is more or less the same as looking at all subsets of $V(G)$ and trying them one by one. This takes essentially $2^{|V(G)|}$ many steps more or less.

In the case of graphs of bounded treewidth, we can give a linear time algorithm. So suppose that we have a tree decomposition of G. Consider the one below

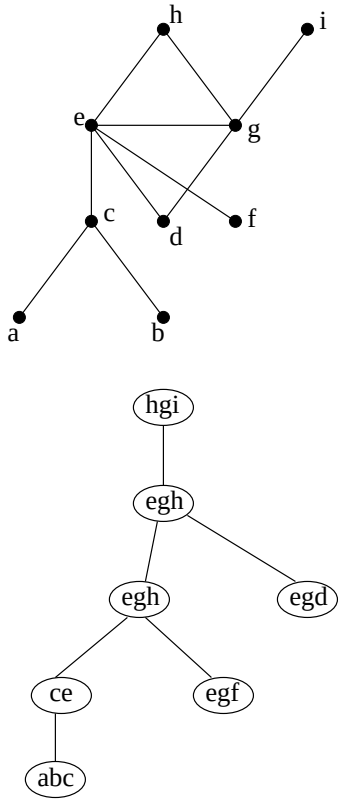


Figure 24: A tree decomposition

Now what we will do is to use *tables* to grow the independent set up the tree starting at the leaves of the decomposition. Notice that once a vertex leaves the bags it will never come back, and hence we don't really need to keep track of its effect. Thus we can work with tables corresponding to all the subsets of the current bag and need only consider independent sets I relative to the current bag. That is, we would consider all the subsets of the

bag and see how the size of independent sets relative to in the information flow across the boundary.

Thus at the leaf corresponding to the triangle abc we could have the table below. Here, for instance ab denotes the set $\{a, b\}$, meaning that the independent set should contain both a and b, and would have cardinality 2, and bc corresponds to $\{b, c\}$ and this entry has a line since $\{b, c\}$ is *not* an independent set.‘

$\emptyset$	a	b	c	ab	ac	bc	abc
0	1	1	1	2	-	-	-

The table for the next box would have only 4 columns since there are only 4 subsets of $\{c, d\}$ and we consider maximal independent sets I containing the specified subset

$\emptyset$	c	e	ce
2	1	3	-

The first entry has value 2 since this means neither of c or d in included in this I and hence we take the maximal one below, namely 2.

The second entry says that I must contain c, and this corresponds to the entry from below containing c, which is 1. The third one corresponds to the independent set containing e *and not* c, and hence we get 3, by choosing the largest one with this property from below (namely the entry for ab contains no c, and this together with e gives $1 + 2 = 3$.)

Finally, the last entry is 0 since I would need to contain both c and d and no independent set has this property.

The rest of the table if filled in similarly. With pointers you can also keep track of the relevant independent set.

The next table is a join node

$\emptyset$	e	g	h	eg	eh	gh	egh
0	1	1	1	2	-	-	-

This table corresponds to $I \cap \{e, g, h\}$ being the specified set. The first entry corresponds to I having nothing from this set. It has value 3 since we get that from the 2 on the left and 1 on the right. (We must add here.)

The second entry corresponds to the intersection being $\{e\}$, meaning that it must have e and cannot have h or g. Note e is present in both branches and hence we get 3. Next is g with e and h not present. There are 2 from the left (corresponding to the $\emptyset$), and g is present on the right so we take its entry giving 3. the next is h and neither e or g, giving 2 from the left branch, 2 from the right (using f since h is not present) and h itself; giving the total of 4. etc

The final table looks like:

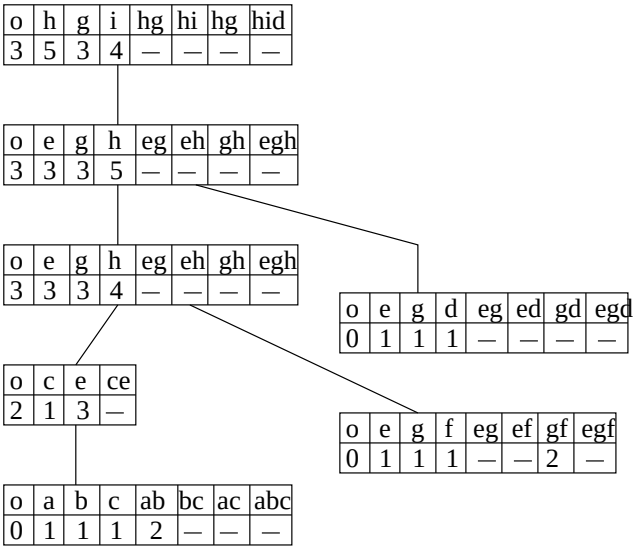


Figure 25: Dynamic programming

We remark that a sly feature we have not mentioned here is that to run this method, we need a tree decomposition for the graph G. It turns out that for a fixed t there is a *linear time algorithm* determining if G has width

t, and then finds the tree decomposition of G should the graph actually have one. However, the algorithm has really terrible constants and there is no actually feasible one for this problem. For more on this we refer the reader to Bodlaender [2], and his web site. John Fouhy [17] in his MSc Thesis looked at computational experiments for treewidth heuristics. (See www.mcs.vuw.ac.nz/∼downey/students.html.) There is a lot of work to be done here.

Actually, this whole process can be implemented by automata acting on trees. We refer the reader to Downey and Fellows [13].

Additionally in some circumstances the whole process can be "automated" using methods from logic. One example of this is *Courcelle's Theorem*

Courcelle's Theorem is a high level machine for establishing that various graph theoretical properties are finite state for bounded treewidth. Similar results were obtained independently by Borie, Parker and Tovey [5]. The high level language is the fragment of the second order theory of graphs called the *Monadic Second Order Theory of Graphs.*

The syntax of the second-order monadic logic of graphs includes the logical connectives $\wedge, \vee, \neg$, variables for vertices, edges, sets of vertices and sets of edges, the quantifiers $\forall, \exists$ that can be applied to these variables, and the five binary relations:
(1) $u \in U$ where u is a vertex variable and U is a vertex set variable
(2) $d \in D$ where d is an edge variable and D is an edge set variable
(3) $inc(d, u)$ where d is an edge variable, u is a vertex variable, and the interpretation is that the edge d is incident on the vertex u
(4) $adj(u, v)$ where u and v are vertex variables and the interpretation is that u and v are adjacent vertices
(5) Equality for vertices, edges, sets of vertices and sets of edges. We will use lower case letters for vertices or edge variables and upper case letters for variables representing sets of edges or sets of vertices.

One famous hard problem is called HAMILTONICITY. Here we ask in a graph G whether there is a cycle passing through each vertex exactly once. This problem is expressible in monadic second order logic as follows. A graph G is Hamiltonian iff the edges of G can be partitioned into two sets red and

blue such that
(i) each vertex has exactly two incident red edges, and
(ii) the subgraph induced by the red edges spans G. In the example below we will represent the set of red edges by R and the set of blue edges by B.

Hamiltonian $\equiv$

$$\exists R \exists B \forall u \forall v [part(R, B) \wedge deg(u, R) = 2 \wedge span(u, v, R)], \text{ where}$$

span, deg and *part* are described below.

$$part(R, B) \equiv \forall e[(e \in R \vee e \in B) \wedge \neg(e \in R \wedge e \in B)].$$

$deg(u, R) = 2 \equiv$
$[\exists e_1, e_2(e_1 \neq e_2 \wedge inc(e_1, u) \wedge inc(e_2, u) \wedge e_1 \in R \wedge e_2 \in R)] \wedge$
$\neg[\exists e_1, e_2, e_3(e_1 \neq e_2 \neq e_3 \wedge (inc(e_i, u) \wedge e_i \in R \text{ for } i \in \{1, 2, 3\}))]$.

$span(u, v, R) \equiv$
$\forall V, W(part(V, W) \wedge u \in V \wedge v \in W) \rightarrow (\exists e, x, y(inc(e, x) \wedge inc(e, y) \wedge x \in V \wedge y \in W \wedge e \in R))]$.

We state the following powerful result.

Theorem 12.1 (Courcelle's MS_2 Theorem [8], [7]). *If F is a family of graphs described by a sentence in second-order monadic logic, then for a fixed treewidth t, there is a linear time algorithm which decides if a graph of treewidth t is in the family F.*

For a proof we refer the reader to Downey and Fellows [13], or Flum and Grohe [18]. For more on "parametric algorithms" we refer to [13], [18] and Niedermeier [35].

12.1 Projects

(i) How does this relate to automata theory and the theory of formal languages. What are tree automata? What is the Myhill-Nerode Theorem and how can it be used to build automata. How can the the action of these algorithms be related to *parsing* graphs.

(ii) There are other well-behaved classes of graphs with parsing languages to allow for interative algorithmics. Investigate, for instance *slim* graphs, graphs of bounded clique-width, d-inductive graphs,

(iii) How are Courcelle's and other "'metatheorem's" used (and proven)? What about the work by Grohe and his co-authors on *local* treewidth. What is the relationship between *definability* and algorithmic efficiency (see Immermann's book).

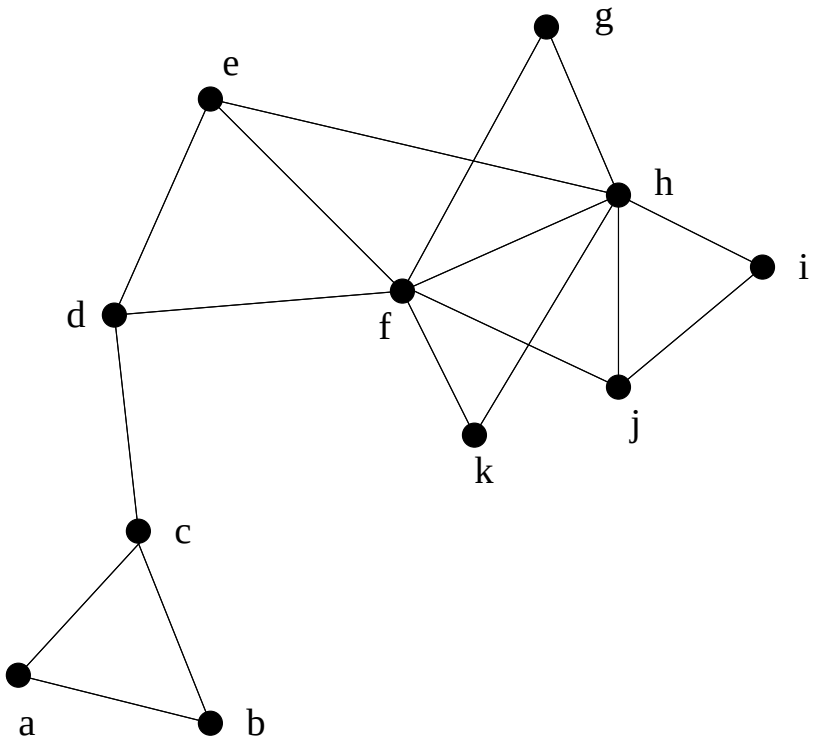
(iv) What is NP completeness? What are the major coping strategies? How can parameterization be used to cope. What about randomization?

(v) How can the notion of treewidth be used as an analog in other areas? Investigate the work on matroids of bounded treewidth by Whittle, Geelen, Hlineny and other, and particularly the MACEK suite of algorithms.

(vi) What are the major algorithms for generating tree decompositions for graphs of width k. What is the most practical if any? What are the heuristics which are used, and how should they be tested?

13 Exercises 7

1 Construct a tree decomposition for the graph of treewidth 2 below.



2 Using dynamic programming on the graph of question 1, compute

(i) A maximum independent set.

(ii) A minimum dominating set. (A dominating set is a set Q or vertices such that, if $x \in V(G)$ then either $x \in Q$ or some $y \in Q$, with $xy \in E$.)

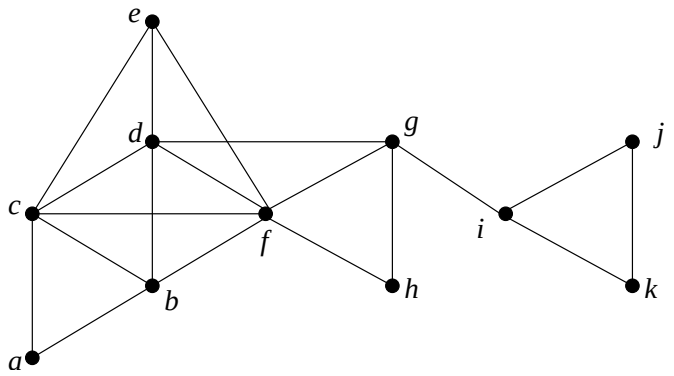
(iii) A minumum vertex cover. (A vertex cover is a collection of vertices such that if xy is an edge then one of x or y (or both) is in the vertex cover.)

2 A graph G is called d-inductive iff there is an ordering of the vertices $v_1, v_2, \ldots, v_n$ such that if for all i, v_i is connected to *at most* d vertices v_j for $j > i$.

(i) Prove that a graph of treewidth k is $k+1$-inductive.

(ii) * Prove that a planar graph is 5-inductive. (You are reminded that every planar graph has a vertex of degree 5.)

3 Construct a tree decomposition for the graph of treewidth 3 below.



4 As for 2 above.

5 Prove that for any fixed t, "G has a vertex cover of size t" is expressible in monadic second order arithmetic. (Actually this is first order.) Prove that 'G has a minimum vertex cover of size t" is expressible in monadic second order arithmetic.

14 Ramsey's Theorem

When you look into the night sky you will see patterns in the stars. In some sense Ramsey's Theorem says that this is to be expected and within that apparent chaos, there must be some order! This theorem has remarkable consequences in many areas of mathematics such as logic, computer science, combinatorics, and even in economics. Recall that K_k denotes the complete graph with k vertices. Recall that an independent set I in a graph G is a set of vertices of G which are all not connected.

14.1 The basic theorem

Theorem 14.1 (Ramsey's Theorem -first version)**.** *For any integer k, there is a number m so large that any graph with m or more edges either has an a copy of K_k, or a copy of I_k, the independent set of k vertices.*

One way to think of this is the "party principle". To wit, in any party of at least m people there is either a clique of k people every pair of which are friends, or there is a set of k people no pair of which are friends. We abstract this to numbers. We let $[n]^2$ denote the 2-element (unordered) subsets of $\{0,\ldots,n-1\}$ (and similarly $[n]^d$) so that $[4]^2$ would be $\{0,1\}$,$\{0,2\}$,$\{0,3\}$,$\{1,2\}$, $\{2,3\}$, and $\{1,3\}$. Think of colouring χ the d-sets from $[n]^d$ into, say, 2 colours. Then a set of numbers $H \subset \{0,\ldots,n\}$ is called *homogeneous* for the colouring χ if $[H]^d]$ all have the same colour. (Be careful, what this says is that the subset of all the d-tuples from $[n]^d$ given by H have the same colour. For example, if $n = 6$ and $d = 3$ $H = \{1,2,4,5\}$, then to say that H is homogeneous for some colouring would be to say that all of the 3-sets from H, namely $\{1,2,4\},\{1,2,5\},\{1,4,5\},\{2,4,5\}$ are monochromatic.

Theorem 14.2 (Ramsey's Theorem -second version)**.** *(i) For all k there is a m so large such that if $n \geq m$, and we colour $[n]^2$ into 2 colours, red and blue. Then there is a homogeneous set of size k.*

(ii) Indeed, for all k, d and p, there is an $m = m(k,d,p)$ so large that if $n \geq m$ and I p colour $[n]^d$, then there is a homogeneous set of size k.

We will write $n \to (\ell_1,\ldots,\ell_r)$ if for any r-colouring of $[n]^2$ there is an i and homogeneous set H with $|H| = \ell_i$ and $[H]^2$ coloured colour i.

For example does $4 \to (4,3)$? Well colour $[4]^2$ with $\{0,1\},\{0,2\},\{0,3\}$ red, and the other three 2-sets blue. Then there is no H of size 3 blue, nor and H of size 4 red.

We prove Theorem 14.2 (i) above.

Proof. We prove that for all ℓ_1,ℓ_2 there is an n so that $n \to (\ell_1,\ell_2$. We denote the least such n as $R(\ell_1,\ell_2)$ the *Ramsey number*. The proof uses induction of ℓ_1 and ℓ_2 Note that $R(\ell_1,2)$ and $R(2,\ell_2)$ exist for all ℓ_i[1]. Now assume that $R(\ell_1,\ell_2-1)$ and $R(\ell_1-1,\ell_2)$ exist. We claim

$$n = R(\ell_1,\ell_2-1) + R(\ell_1-1,\ell_2) \to (\ell_1,\ell_2).$$

[1]This is because either all the pairs are coloured red, or there is some pair coloured blue!

Fix a colouring χ of $[n]^2$. Now take $z \in \{0, \ldots, n-1\}$ and define

$$I_z = \{y \in \{0, \ldots, n-1\} : \chi(\{z, y\}) \text{ red}\}.$$

$$\text{Define } II_z = \{y \in \{0, \ldots, n-1\} : \chi(\{z, y\}) \text{ blue}\}.$$

Now we note that $I_z = [n]^2 - II_z - \{z\}$. Now since $|I_z| + |II_z| = n - 1$, it follows by the pigeonhole principle that either

(i) $|I_z| \geq R(\ell_1 - 1, \ell_2)$ or

(ii) $|II_z| \geq R(\ell_1, \ell_2 - 1)$.

If $|I_z| \geq R(\ell_1 - 1, \ell_2)$, then by definition of $R(\ell_1 - 1, \ell_2)$, there is homogeneous set $H \subset I_z$ with $|H| = \ell_2$ and $[H]^2$ coloured blue, or $|H| = \ell_1 - 1$ and $[H]^2$ coloured red. In the former case we are golden. In the latter case, we claim that $H \cup \{z\}$ is the desired homogeneous set. This follows, since by definition of I_z, for all $y \in I_z$, $\{y, z\}$ is red. Clearly II_z is symmetric. □

With more effort and using arguments of similar types, Ramsey showed that for all $\{\ell_1, \ldots, \ell_r\}$ and all colouring of the d-sets we have the existence of an n, with

$$n \to (\ell_1 \ldots, \ell_r)^d.$$

If $\ell_1 = \ell_2 = \ldots \ell_r = \ell$ then we write compactly

$$n \to (\ell)^d_r.$$

14.2 Exercises 8

1 Let k be given. Show that if n is sufficiently large, then if I have a sequence of n distinct numbers $m_1, \ldots, m_n$, then it must have a subsequence $m_{i_1}, m_{i_2}, \ldots, m_{i_k}$ of size k which is monotonic, in the sense that it is decreasing, or increasing.

2* (Dilworth's Theorem) Show that for 1 $k^2 + 1$ is big enough. (This does not use Ramsey's Theorem.) Prove that this bound is sharp.

3 (Erdos-Szekeres) A convex polygon is one where no internal angles are bigger than 180 degrees. First show that for any 5 points on the plane *in general position* (meaning that no 3 are collinear, there is a convex 4-gon. Show that, for all k, there is an n so large that if I take n points in general position in the plane, there is a convex k-gon. (Hint: think about how to colour i, j, k if $i > j < k$ moving clockwise or anticlockwise.)

14.3 Induced colouring*

There is a much slicker proof of Ramsey's Theorem using the induced colouring method. No doubt I may not have time to run through this but here it is for completeness. I am using the one from Graham, Rothschold, and Spencer [22].

Theorem 14.3 (Ramsey's Theorem)**.** *For all $\ell_1, \ldots, \ell_r, k$ there exists an n_0 such thaat for all $n \geq n_0$,*

$$n \to (\ell_1, \ldots, \ell_r)^k.$$

Proof. Use induction on k. If $k = 1$ then this is just the pigeonhole principle. We assuem the result holds for $k - 1$. It will suffice to show the existence of n with $n \to (\ell)^k_r$, by taking the maximum ℓ_i for ℓ. Let n be sufficiently large, and fix an r-colouring χ of $[n]^k$. Let $t = R_{k-1}(\ell, r)$ given by induction. Define

$$S_{k-2} = [n] - \{a_1, \ldots, a_{k-2}\}.$$

Now we choose a_i, S_i as follows.

(i) S_i having been defined, choose $a_{i+1} \in S_i$ arbitarily.

(ii) Split $S_i - \{a_{i+1}\}$ into equivalence classes by declaring that $x \cong y$ iff for every $T \subset \{a_1, \ldots, a_{i+1}\}$ with $|T| = k - 1$,

$$\chi(T \cup \{x\}) = \chi(T \cup \{y\}).$$

This equivalence class is determined by $\binom{i+1}{k-1}$ many sets and hence there are $r^{\binom{i+1}{k-1}}$ many equivalence classes. Let S_{i+1} be the largest such class, so that $S_{i+1} \subseteq S_i - \{a_{i+1}\}$. We have

$$|S_{i+1}| \geq (|S_i| - 1)r^{-\binom{i+1}{k-1}}.$$

If we choose n large enough then we can continue this process until a_t is defined.

Now consider the sequence $a_1, \ldots, a_t$. Suppose that $1 \leq i_1 < i_2 \cdots < i_{k-1} < s \leq t$. Then $a_s \in S_{s-1} \subseteq S_{i_{k-1}+1}$. The the colour $\chi(\{a_1, \ldots, a_{i_{k-1}}, a_s\}$ remians the same if x is replaced by any x in $S_{i_{k-1}+1}$ (by the definition of the equivalence classes), and in particular any $x = a_r$ with $k - 1 < r < t$. (This is sometimes called a *pre-homogeneous set.*) Now we can define a colour χ^* on the $k-1$ subsets of $\{a_1, \ldots, a_t\}$ by

$$\chi^*(\{a_1, \ldots, a_{i_{k-1}}) = \chi(\{a_1, \ldots, a_{i_{k-1}}, a_s\})$$

for all $i_{k-1} < s \leq t$. By definition of t as the Ramsey number, there is a subsequence $b_1, \ldots, b_\ell$ of $a_1, \ldots, a_t$ monochromatic inder χ^*. Suppose that all subsets are red. Then for any $1 \leq j_1 < \cdots < j_{k-1} < j_k \leq \ell$,

$$\chi(\{b_{j_1}, \ldots, b_{j_{k-1}}, b_{j_k}\}) = \chi^*(\{b_{j_1}, \ldots, b_{j_{k-1}}\}) = \text{red}.$$

Hence $\{b_1, \ldots, b_\ell\}$ is the desired set. □

There is also an infinite form of Ramsey's Theorem which can be proven along the lines above.

Theorem 14.4 (Ramsey's Theorem-infinite form)**.** *Suppose that we ℓ colour the k-subsets of $\mathbb{N}$. There there is an infinite homogeneous set.*

Proof. We sketch the proof. We follow Simpson [42]. This is done by induction on k. For $k = 1$ this is again the pigeonhole principle. If we ℓ colour the integers, then there must be infinitely many elements of some colour class. To prove $RT(k)$ assuming $RT(k+1)$, let $\chi[\mathbb{N}]^{k+1} \to \{0, \ldots, \ell - 1\}$ be the ℓ-colouring.

Now we define a tree $T \subset \mathbb{N}^{<\mathbb{N}}$ by putting a string of integers t in T iff for all $n < |t|$, $t(n)$ is the least j such that

(i) $t(m) < j$ for all $m < n$, and

(ii) $\chi(\{t(m_1), \ldots, t(m_k), j\}) = \chi(\{t(m_1), \ldots, t(m_k), t(m)\}$ for all $m_1 < m_2 < \cdots < m_k < m \leq n$.

(This is a direct analog of the construction in the finite inductive proof.) Then T is a tree and is finitely branching since, given t of length n, there are at most ℓ^{n^k} distinct j with $t^\frown\langle j\rangle \in T$.

(We remark that T is called the Erdös-Rado Tree.)

We claim that for each j there is an $s \in T$ such that $s(n) = j$ for some $n < |s|$. To see this, fix j and let t be maximal such that $t(m) < j$ for all $m < |t|$, and $\chi(\{t(m_1), \ldots, t(m_k), j\}) = \chi(\{t(m_1), \ldots, t(m_k), t(m)\}$ for all $m_1 < m_2 < \cdots < m_k < m \leq |t|$. (There is clearly one such t, namely t the empty string.) Then clearly $t^\frown\langle j\rangle \in T$. This proves the claim.

Therefore T is infinite, and hence by König's Lemma (Every finitely branching infinite tree has an infinite path), T has a path g, say. From (i) we have that $m < n$ implies $g(m) < g(n)$. Now we define $\chi^*(\{m_1, \ldots, m_k\}) = \chi(\{g(m_1), \ldots, g(m_k), g(m)\})$ where $m_1 < m_2 < \cdots < m_k < m$, which is in the previous proof, by (ii) does not depend on the choice of m. Now using the induction hypothesis, $RT(k)$, take $i \in [\ell]$ and homogeneous set X' monochromatic for χ^*, and this will be homogeneous for χ. □

Actually, the infinite form of Ramsey's Theorem follows from the finite form plus a "compactness" argument in the same way that the 4 colour theorem for infinite planar graphs follows from the 4 colour theorem for finite graphs. The reader can find out about this if they do a basic course in mathematical logic.

14.4 Other Ramsey-type theorems

There are a lot of wonderful theorems like Ramsey's Theorem in the literature. Here are some gems.

Theorem 14.5 (Van der Waerden's Theorem [45]). *For all r and k, there is a number $n = W(r,k)$ such that if we r colour the numbers from 1 to n, then there is a monochromatic arithmetical progression of size k.*

Actually, Ramsey's Theorem and van der Waerden's Theorem are both corollaries of another theorem called the Hales-Jewitt Theorem. It would take us a little afar to go through the details and we refer the reader to Graham, Rothschild and Spencer [22] for a lot more on this subject. (And to Downey [12] for more background information.)

A deep generalization of van der Waerden's Theorem is due to Szemeredi and basically says that there will be an arithmetical progression *in the most common colour*. That is, if we say that red (say) in some colouring of $\mathbb{N}$ has *positive upper density* if

$$\lim_{n\to\infty} \frac{\{j : j \leq n \wedge J \text{ red}\}}{n} \to 1.$$

Theorem 14.6 (Szemeredi [43]). *Suppose that $\mathbb{N}$ is coloured and red has positive upper density. Then there are arbitarily long red arithmetical progressions.*

It is worth saying that there may not be *infinite* red arithmetical progressions, only arbitarily long ones. Szemeredi's Theorem was originally proven using a powerful combinatorial lemma called the *Regularity Lemma* which has found applications elsewhere. Furstenberg [19] proved Szemeredi's Theorem using "analytic" (probabilistic) methods from an area of mathematics called *Ergodic Theory*.

van der Waerden's and Szemeredi's beautiful theorems on arithmetical progressions attracted the interest of a number of very fine mathematicians. One reason for this is that the original proof and Furstenberg's) involved amazingly large numbers. The proof of van der waerden gave values for (even) $W(2,k)$ that were not "elementary" (meaning that they dwarfed any function build from "reasonable" towers of 2's, such as $2^{2^{2^{\cdots}}}$ (stacked k high), and were absolutely astronomical. Saharon Shelah [39] gave elementary (but large) bounds for van der Waerden's Theorem. Recent work of Timothy Gower [20, 21] has analysed the towers of 2 needed for Szemeredi's Theorem. Gower won the Field's Medal (a kind of Nobel prize in math) for this work.

Recently, Terrence Tao, a young mathematician, and Ben Green [23] proved a related rather amazing result that there are arbitarily long arithmetical progressions in the primes.

From the last section, we recall that there is an infinite version of Ramsey's Theorem. (Theorem 14.4)

Theorem 14.7 (Ramsey [37])**.** *Suppose that we k-colour the d-sets from* $\mathbb{N}$*. Then there is an infinite homogeneous set.*

The infinite form of Ramsey's Theorem has been a tool for many years in mathematical logic. It can bes used to prove a finite form which aroused a lot of interest when it was proven.

We say that a set A of numbers is *relatively large* if there is some number $n \in A$ such that $|A| > n$. That is, someone in the set things the set is large! The following result can be proven using the infinite form of Ramsey's Theorem:

Theorem 14.8 (Paris and Harrington [36])**.** *For all k there is an so large that for any 2-colouring of* $[n]^2$ *there is a* relatively large *homogeneous set.*

The Paris-Harrington Theorem can be shown in a technical sense to actually *need* the use of infinite sets in its proof! Paris and Harrington gave a proof that there is not proof of this result in the system of finite mathematics called *Peano Arithmetic.* This was the first "mathematical incompleteness" found for PA.

For more on the subject of finite mathematical principles not provable in "normal" systems of mathematics, we refer the reader to Simpson [41].

There is a lot more to say about this material and we refer the reader to Graham, Rothschild and Spencer [22], Landman and Robertson [30] and Nesetril and Rodl [34]. The Landman and Robertson book contains a lot of research problems.

14.5 Projects

(i) What are the other Ramsey type theorems? How would you prove these results using ergodic theory?

(ii) How do you use these results to prove certain combinatorial problems involving rapidly growing Ramsey functions are not provable in systems such as Peano Arithmetic. What is the relationship between growth rate and provability?

(iii) What is the Friedman-Simpson "reverse mathematics" programme?

15 Well quasi-ordering theory

A *quasi-ordering* on a set S is a reflexive transitive relation on S. We will usually represent a quasi-ordering as $\leq_S$ or simply $\leq$ when the underlying set S is clear. Let $\langle S, \leq \rangle$ be a quasi-ordered set. We will write $x < y$ if $x \leq y$ and $y \not\leq x$, $x \equiv y$ if $x \leq y$ and $y \leq x$ and finally $x|y$ if $x \not\leq y$ and $y \not\leq x$. Note that if $\langle S, \leq \rangle$ is a quasi-ordered set then $S/\equiv$ is partially ordered by the quasi-order induced by $\leq$. Recall th at a *partial ordering* is a quasi-ordering that is also antisymmetric.

Definition 15.1 (Ideal and Filter)**.** Let $\langle S, \leq \rangle$ be a quasi-ordered set. Let S' be a subset of S.

(i) We say that S' is a *filter* if it is closed under $\leq$ upwards. That is, if $x \in S'$ and $y \geq x$ then $y \in S'$. The *filter generated by* S' is the set $F(S') = \{y \in S : \exists x \in S'(x \leq y)\}$.[2]

(ii) We say that S' is a *(lower) ideal* if S' is $\leq$ closed downwards. That is if $x \in S'$ and $y \leq x$ then $y \in S'$. The *ideal generated by* S' is the set $I(S') = \{y \in S : \exists x \in S'(x \geq y)\}$.

(iii) Finally, if S' is a filter (an ideal) that can be generated by a finite subset of S' then we say that S' is *finitely generated*.

[2]Sometimes, filters are called *upper ideals*.

We will need some distinguished types of sequences of elements.

Definition 15.2. Let $\langle S, \leq \rangle$ be a quasi-ordered set. Let $A = \{a_0, a_1, ...\}$ be a sequence of elements of S. Then we say the following.

(i) A is *good* if there is some $i < j$ with $a_i \leq a_j$.

(ii) A is *bad* if it is not good.

(iii) A is an ascending *chain* if for all $i < j$, $a_i \leq a_j$.

(iv) A is an *antichain* if for all $i \neq j$ $a_i | a_j$.

(v) $\langle S, \leq \rangle$ is *Noetherian* if S contains no infinite (strictly) descending sequences. (i.e. there is no sequence $b_0 > b_1 > b_2....$)

(vi) $\langle S, \leq \rangle$ has the *finite basis property* if for all subsets $S' \subseteq S$, $F(S')$ is finitely generated.

Theorem 15.3 (Folklore, after Higman [24])**.** *Let $\langle S, \leq \rangle$ be a quasi-ordered set. The following are equivalent.*

(i) $\langle S, \leq \rangle$ has no bad sequences.

(ii) Every infinite sequence in S contains an infinite chain.

(iii) $\langle S, \leq \rangle$ is Noetherian and S contains no infinite antichain.

(iv) $\langle S, \leq \rangle$ has the finite basis property.

Proof. (i) $\rightarrow$ (ii) and (ii) $\rightarrow$ (iii) are easy. (See Exercise 16.)

(iii) $\rightarrow$ (iv). Suppose that $\langle S, \leq \rangle$ is Noetherian and S contains no infinite antichain. Let F be a filter of S with no finite basis. We generate a sequence from F in stages as follows. Let f_0 be any element from F. As F is not finitely generated there is some $f_1 \in F$ with $f_1 \notin F(\{f_0\})$. For step $i+1$, having chosen $\{f_0, ..., f_i\}$, we find some $f_{i+1} \in F$ with $f_{i+1} \notin F(\{f_0, ...f_i\})$. Since F is not finitely generated, this process will not terminate. Notice that for all $i < j$ it cannot be that $f_i \leq f_j$ since, in particular, $f_j \notin F(\{f_i\})$. To see that (ii) implies (iii), for $i < j$, colour the pairs f_i, f_j *red* if $f_i > f_j$

and colour the pair *blue* if $f_i | f_j$. (Every pair is coloured either red or blue by the construction of the sequence.) By Ramsey's Theorem, there is an infinite homogeneous set: That is there is a sequence $f_{i_0} f_{i_1} ...$ and a colour $\chi \in \{red, blue\}$, such that for all $i_k < i_t$, the pair f_{i_k}, f_{i_t} is coloured χ. If χ is red then we have an infinite descending sequence. If χ is blue we have an infinite antichain. In either case we contradict (iii), and hence we get (iv).

(iv) $\rightarrow$ (i) Suppose that $\langle S, \leq \rangle$ has the finite basis property. Let $B = \{b_0, b_1, ...\}$ be a sequence of elements of S. Let $B' \subseteq B$ be a finite basis for $F(B)$. Let $m = \max\{i : b_i \in B'\}$. Choose $b_j \in B - B'$ with $j > m$. Then for some $b_i \in B'$ we must have $b_i \leq b_j$ since $b_j \in F(B')$, and $i < j$ by construction. Hence B is good. □

This leads us to...

Definition 15.4 (Well Quasi-Ordering)**.** Let $\langle S, \leq \rangle$ be a quasi-ordering. If $\langle S, \leq \rangle$ satisfies any of the characterizations of Theorem 15.3, then we say that $\langle S, \leq \rangle$ is a *well quasi-ordering* (WQO).

The reader might well wonder what any of this abstract pure mathematics has to do with algorithmic considerations. The key is provided by the finite basis characterizations of a WQO. Suppose that $\leq$ is the relevant quasi-ordering and for a fixed x the question "Q_y: Is $x \leq y$?" is polynomial time. Then for a WQO, if F is a filter, then F has a finite basis $\{b_1, ..., b_n\}$. Then to decide if $y \in F$ we need only ask "$\exists i \leq n(b_i \leq_S y)$?" That is, membership of each filter is polynomial time (even though we don't know the finite basis!).

Often the argument is phrased in terms of *obstruction sets*. Let $\langle S, \leq \rangle$ be a quasi-ordering. Let I be an ideal of $\langle S, \leq \rangle$. We say that a set $O \subseteq S$ forms an *obstruction set* for I if

$$x \in I \text{ iff } \forall y \in O(y \not\leq x).$$

That is O is an obstruction set for I if I is the complement of $F(O)$. The WQO principle says all ideals have finite obstruction sets.

So here is our new engine for demonstrating that problems are in P. Prove that the problem is characterized by a WQO with a finite obstruction set in a quasi-ordering with Q_y in P.

The best known example of an obstruction set is provided by topological ordering.

Definition 15.5 (Topological Ordering)**.** A *homeomorphic or topological embedding* of a graph $G_1 = (V_1, E_1)$ in a graph $G_2 = (V_2, E_2)$ is an injection from vertices V_1 to V_2 with the property that the edges E_1 are mapped to disjoint paths of G_2. (These disjoint paths in G_2 represent possible *subdivisions* of the edges of G_1.) The set of homeomorphic embeddings between graphs gives a partial order, called the *topological order*. We write $G_1 \leq_{top} G_2$.

While topological ordering is not a WQO, there are a number of important finite basis results. The most famous is the following (which was independently discovered by Pontryagin).

Theorem 15.6 (Kuratowski's Theorem, Kuratowski [28])**.** *The graphs $K_{3,3}$ and K_5 of Figure 26 form an obstruction set for the ideal of planar graphs in the topological ordering.*

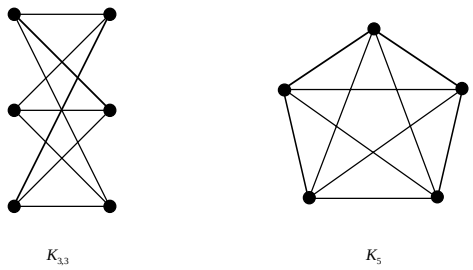


Figure 26: Obstructions for planarity

For a proof of Kuratowski's Theorem we refer the reader to Dirac and Schuster [11]. We remark that it is an important open question whether $x \leq_{top} y$ is polynomial time for a fixed x. (In the sense that it is $O(|y|^c)$ for a fixed c. There *is* a polynomial time algorithm but the running time has the exponent depending on $|x|$ also.) A generalization of topological ordering is much more amenable. An equivalent formulation of $\leq_{top}$ is the following. $G \leq_{top} H$ iff G can be obtained from H by a sequence of the following two operations.

(i) (deletions) Deleting vertices or edges.

(ii) (degree 2 contractions) The *contraction* of an edge xy in a graph W is obtained by indentifying x with y. A contraction has degree 2 iff one of x or y has degree 2.

The *minor ordering* is a generalization of $\leq_{top}$ is obtained by relaxing the degree 2 requirement in (ii).

Definition 15.7 (Minor Ordering)**.** We say that G is a minor of H if G can be obtained from H by a sequence of deletions and contractions. We write $G \leq_{minor} H$.

An example of the minor ordering is given in Figure 27

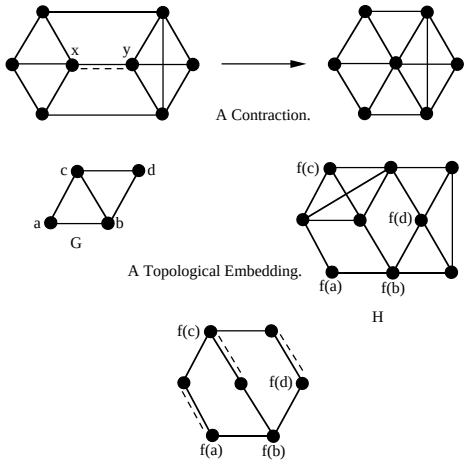


Figure 27: A contraction and a Topological Embedding

The proof of Kuratowski's Theorem also gives the following.

Theorem 15.8 (Kuratowski's Theorem (II))**.** *$K_{3,3}$ and K_5 are an obstruction set for the ideal of planar graphs under $\leq_{minor}$.*

The way to think of $\leq_{minor}$ is to think of $G \leq_{minor} H$ as taking $|G|$ many collections C_i of connected vertices of H, and coalescing each collection C_i to a single vertex, then H being topologically embeddable into the new coalesced graph, so that the edges of G become disjoint paths from coalesced vertices.

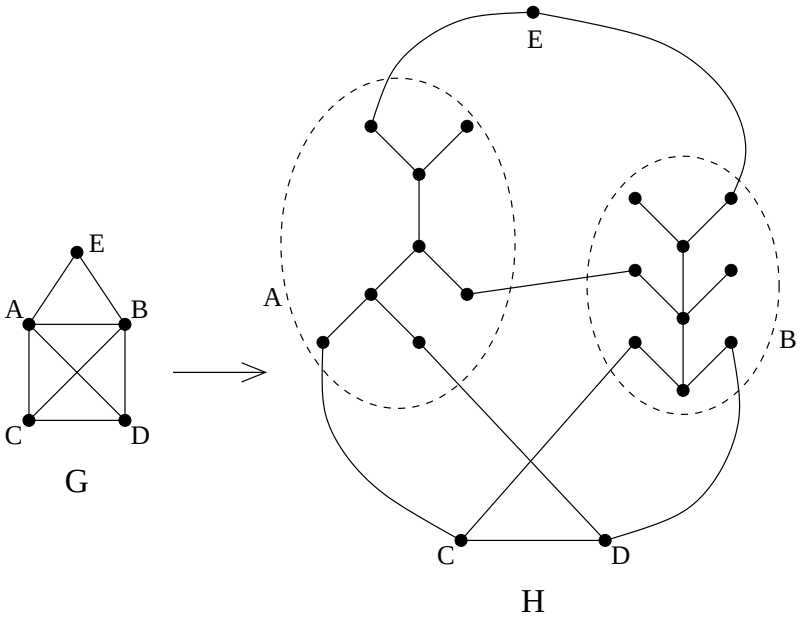


Figure 28: The folio definition

Sometimes this is called the "folio" definition of the minor ordering. Figure 28 below demonstrates this idea.

Notice that H has no vertices of degree 4 and hence there can be no tolological embedding of G into H.

Wagner [46] conjectured the following.

Wagner's Conjecture:*Finite graphs are well quasi-ordered by the minor ordering.*

Notice that graphs of genus $\leq g$ for a fixed g form an ideal in the minor ordering. Hence a consequence of Wagner's conjecture is a Kuratowski Theorem for surfaces. *Graphs of genus $\leq g$ have a finite obstruction set.* One of the triumphs of 20th century mathematics is the following great theorem of Neil Robertson and Paul Seymour.

Theorem 15.9 (Graph Minor Theorem, Robertson and Seymour)**.** *Wagner's Conjecture holds: Finite graphs are well quasi-ordered by the minor ordering.*

Remarkably, Robertson and Seymour showed that $x \leq_{minor} y$ is $O(|y|^3)$ for a fixed x. Thus *every minor ideal has a cubic time recognition algorithm!.*

15.1 Kruskal's Theorem

A proof of the Graph Minor Theorem is currently beyond the scope of these notes, but we will sketch the ideas. The material began with a Theorem of Kruskal.

Theorem 15.10 (Kruskal [29], Kruskal's Theorem)**.** *The class of rooted finite trees are well quasi-ordered by $\leq_{top}$ and by $\leq_{minor}$. (The reader should note that for rooted trees T_1, T_2, if $T_1 \leq_{top} T_2$ via the topological embedding f and v and w are vertices of branches of T_1 meeting at z, then it can only bethat this happens iff $f(v)$ and $f(w)$ are vertices of branches of T_2 meeting at $f(z)$ in T_2.)*

Proof. We work with $\leq_{top}$, since $\leq_{minor}$ is coarser. We can consider trees as 1-trees. Trees can be parsed by the operators $\oplus$ and 1, where 1 is the operator which adds a vertex makes it a boundary vertex and joins an edge from the old boundary vertex to the new one.

Figure 29 below shows what I mean here.

The following proof is along the lines of the ones first articulated by Nash-Williams [33]. Suppose that $T_0, T_1, ...$ is a minimal bad sequence of trees. That is we suppose that we have chosen T_0 the be the smallest tree (by cardinality) to begin a bad sequence, and from the bad sequences of trees beginning with T_0, we have chosen T_1 to be smallest, etc. Let P_i be a corresponding parse tree for T_i. Either there are infinitely many P_i with root $\oplus$ or almost all P_i have root 1. Both cases are easy. Suppose that $P_{i_0}, P_{i_1}, ...$ all have root $\oplus$. Now for a parse tree P, let $T(P)$ denote the tree obtained from P. Let Q_i, R_i denote the parse trees obtained by deleting the roots from P_i, and consider the collection

$$\mathcal{C} = \cup_j (T(Q_{i_j}) \cup T(R_{i_j})).$$

We claim that $\langle \mathcal{C}, \leq_{top} \rangle$ is a WQO. If this is not the case, then there is a bad sequence $B_0, B_1, ...$ from $\mathcal{C}$. For some least k, $B_0 \in \{Q_k, R_k\}$. Then

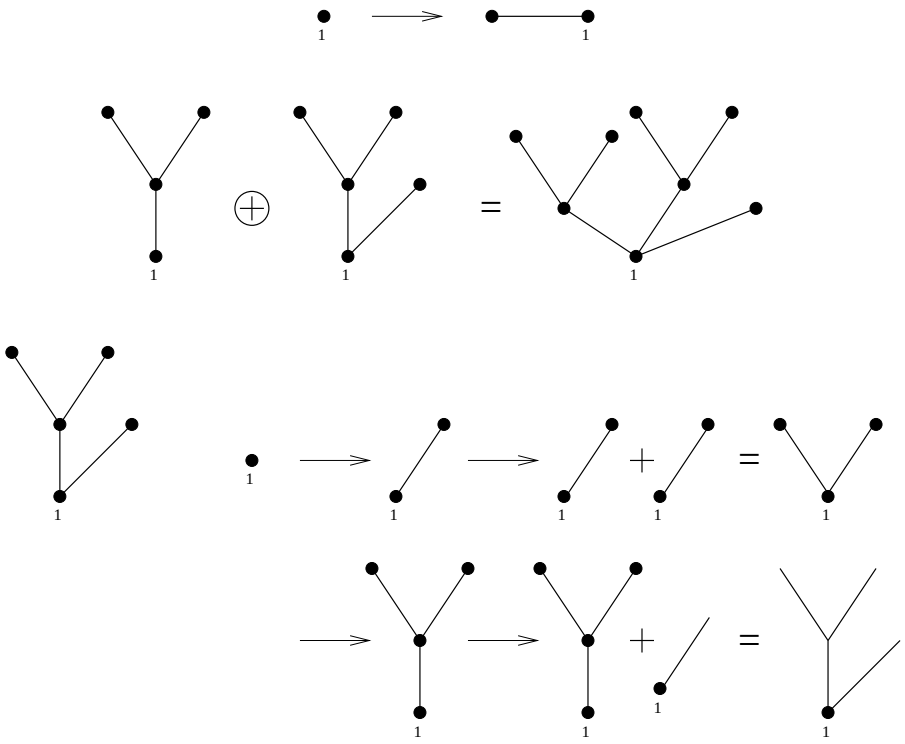


Figure 29: Parsing a tree

the sequence $T_0, T_1, ..., T_{k-1}, B_0, B_1, ...$ is bad, contradicting the minimality of $T_0, T_1, ...$ Hence $\langle \mathcal{C}, \leq_{top} \rangle$ is a WQO. It is easy to prove that $\mathcal{C} \times \mathcal{C}$ is well quasi-ordered by the product ordering. (i.e. $(W_1, V_1) \leq_{top} (W_2, V_2)$ iff $W_1 \leq_{top} W_2 \wedge V_1 \leq_{top} V_2$.) But then there are $m < n$ with

$$T(Q_{i_m}) \leq_{top} T(Q_{i_n}) \wedge T(R_{i_m}) \leq_{top} T(R_{i_n}).$$

Clearly this implies that $T_{i_m} \leq_{top} T_{i_n}$. The 1 case is similar but easier. □

Let $\langle Q, \leq_Q \rangle$ be a quasi-ordered set Extend the definition of topological ordering to labeled topological ordering by saying that a Q-labeled tree T_1 topologically embeds into a labeled tree T_2 via f by asking that f takes edges to disjoint paths, and furthermore if v has label q and $f(v)$ has label q' then $q \leq_Q q'$. The proof above easily extends to the following.

Corollary 15.11. Kruskal's Theorem for Labelled Trees, Kruskal [29] *If $\langle Q, \leq \rangle$ is a WQO then the collection of Q-labeled trees is a WQO under labeled topological embedding.*

The main modification in the proof is that one argues from an infinite sequence of roots $\{q_{i_0}, q_{i_1}, ...\}$ with the property $q_{i_0} \leq_Q q_{i_1} \leq_Q ....$ which are guaranteed to exist by the fact that Q is a WQO. We remark that the standard proofs of Kruskal's Theorem don't use parse trees. Rather, they use the following lemma which is of importance in its own right. (Also see Exercise 16.)

Let $\langle S, \leq \rangle$ be any quasi-ordered set. One can define the induced product ordering on $\mathcal{S}$, the set of finite sequences of elements of S via $\langle s_1, ..., s_n \rangle \leq \langle s'_1, ..., s'_m \rangle$ iff there is a strictly increasing map $j : \{1, ..., n\} \to \{1, ..., m\}$ such that for all $i \leq n$,

$$s_i \leq s'_{j(i)}.$$

For instance, $\langle 1, 2, 1, 4 \rangle \leq \langle 0, 2, 2, 0, 1, 6, 1 \rangle$, but $\langle 1, 2, 1 \rangle \not\leq \langle 0, 1, 1, 2, 0, 0 \rangle$.

Lemma 15.12 (Higman's Lemma, Higman [24])**.** *Suppose that $\langle S, \leq \rangle$ is a WQO. Then the induced ordering is a well quasi-ordering of $\mathcal{S}$.*

We leave the proof of Higman's Lemma to the exercises. It can be deduced from the Labelled version of Kruskal's Theorem.

15.2 Treewidth again

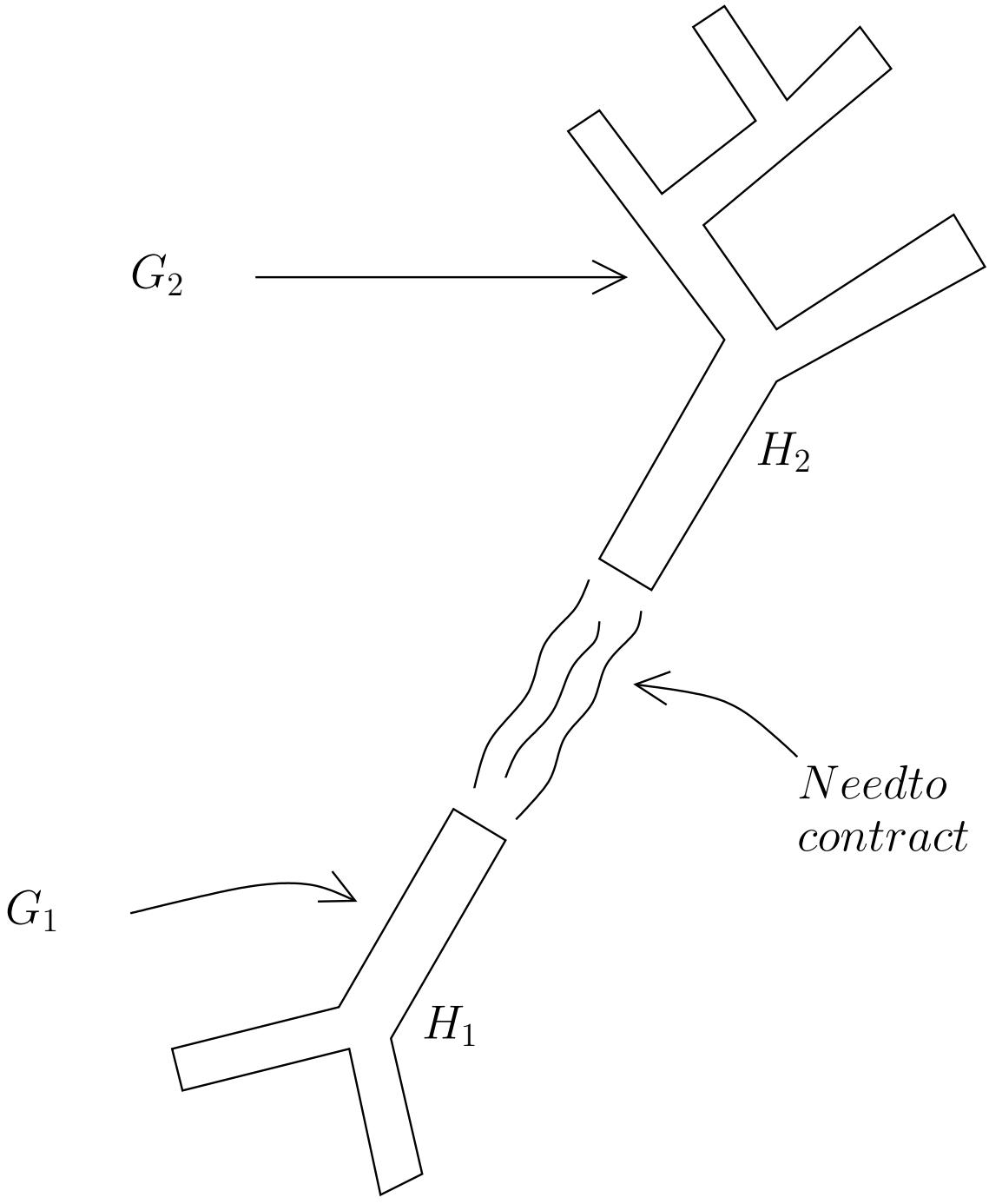
Knowing that finite trees are WQO by $\leq_{top}$, the next part of the strategy is to show that graphs of bounded treewidth are WQO by $\leq_{minor}$.

In this section we will sketch how to do this. Think about the argument we used for Kruskal's Theorem. The main idea is that of a minimal bad sequence. Here this will be akin to Higman's lemma on trees.

In place now of paths being glued together to make trees, now we have k-paths being grued together along boundaries to make k-trees.

The key problem with the argument is that suppose that we have, say $G_1 \oplus G_1$ two k-trees glued along some boundary of size $k+1$, say. and we know that in some host graph further long the bad sequence we have P_1 embedded as a minor into H_1 and similarly for P_2 into H_2. How do we contract the embeddings along the H_1, H_2 boundaries to make $G_1 \oplus G_2$ embed into $H_1 \oplus H_2$?

The folowing diagram illustrates this idea.


G_2
H_2
$Needto$
$contract$
G_1
H_1

What is needed is a *Menger Type Theorem* for k-trees. This was provided by Thomas with the following notion.

Definition 15.13. *We say that a tree decomposition $\mathcal{T} = \{V_t : t \in T\}$ of a graph G is a* lean *decomposition if for all $t_1, t_2 \in T$, and $s \in \mathbb{N}$, either there is a $t \in T$ between t_1 and t_2 and with $|V_t| < s$, or G has s disjoint paths taking members of V_{t_1} to members of V_{t_2}.*

In any tree decomposition we know that a bag is a vertex separation of the graph. In a lean one the separations are as small as possible. Thus, if, for example, a branch has bags of high cardinality, then that would correspond to a highly commented part of the graph. The key to the application of this notion is the following.

Theorem 15.14 (Thomas [44])**.** *Every graph of treewidth k has a lean decomposition of treewidth k.*

Proof. (sketch) Define the *fatness* of a tree decomposition to be the n-tuple $(a_1, \ldots, a_n)$ where a_i denotes the number of bags with exactly $n - i$ vertices. Then let $\mathcal{T} = \{v_t : t \in t\}$ be the decomposition with lexicographically least fatness. Then $\mathcal{T}$ is lean!

The main idea in the proof of this fact (first discovered by Thomas) is to assume not and then use surgery on the lex-least tree decomposition to make it less fat (assuming it is not lean). The proof of this result uses a method of tree decomposition amalgamations. If G is a graph and $X \subseteq V(G)$ a separating set of vertices, we want to amalgamate the tree decompositions of $H = G[C \cup X]$ (where C ranges over the components of $G - X$) into a tree decomposition of G. Let $\mathcal{T} = \{V_t : t \in T\}$ be a tree decomposition of G and $s \in T$. For each $x \in X$ pick a home node $t_x \in T$ with $x \in V_{t_x}$. Define

$$W_t = (V_t \cap V(H)) \cup \{x \in X : t \text{ between } t_x \text{ and } s \text{ in } T\}.$$

Then this is a new tree decomposition incorporating X.

The proof of Thomas theorem is to suppose that the result fails, and choose (t_1, t_2, Z_1, Z_2) in the decomposition where the leanness axiom fails, and one for which the distance between t_1 and t_2 is least in T. Then the

main idea is to apply the amalgamation property above to rearrange the tree decomposition into a new one which is less fat. The agrument is not difficult and can be found in Diestel's home page. (Bellenbaum and Diestel [4]) □

The proof of the well quasi-ordering of graphs of bounded treewidth is then to take a minimal bad sequence of graphs of treewidth t. Consider them as being presented by lean decompositions. Then within them there is some least size bag size s that occurs infinitely often. Cracking the decompositions on that bag, we can apply a Higman lemma-type argument, arguing that there are only so many ways that this separation can "Menger" to the rest of the tree decomposition, and some way will occur infinitely often. On this we can then use the minimal bad sequence argument as we did with the $\oplus$ case above.

15.3 The Robertson-Seymour Theorem

Now how does the proof of the well-quasi-ordering of finite graphs work? The argument works as follows. First you prove a general "excluded minor" theorem. For instance, it can be shown that the collection of graphs $\mathcal{G}_H$ such that $H \not\leq_{minor} G$ for a fixed *grid* H will have uniformly bounded treewidth. Thus if you had a bad sequence of graphs containing a grid, then the bad sequence would need to be of bounded treewidth, and this is impossible.

In general, Robertson and Seymour define a notion roughly like "surfacewidth" and show that if an arbitary graph M is excluded as a minor, then the class of such graphs excluding M has bounded "surfacewidth" and is well-quasi-ordered by methods like those used for bounded treewidth. Unfortunately, in this more general cases the Menger-like theorems etc and the notions themselves are much more complicated and at present no simple proof of this material exists. But the idea is analogous.

15.4 Projects

(i) What other well-quasi ordering results can you find? How have the methods been applied in the verification of *infinite state* systems?

(ii) What is a better-quasi-ordering? How is Laver's Theorem proven? What is the new material by Montalbán on signed trees?

(ii) What are the proof-theoretical strength of these WQO results?

(iii) How does the material move from bounded treewidth graphs to more general graphs.

(iv) How is the methodology applied in the thoery of matroids by Geelen, Gerards and Whittle and others?

16 Exercises 9

1 Prove that (i) implies (ii) and (ii) implies (iii) in Theorem 15.3.

2 Prove Higman's Lemma.

3 Suppose that $\mathcal{T} = \{V_t : t \in T\}$ is a tree decomposition of a graph G, and t_1t_2 is an edge of T. Let T_1 and $T - 2$ denote the components of $T - \{t_1t_2\}$. Then $\cup_{\{t \in T_1\}} V_t$ and $\cup_{\{t \in T_2\}} V_t$ are separated by $V_{t_1} \cap V_{t_2}$.

4 Prove Kruskal's Theorem using Higman's Lemma. (Higman's lemma can be proven without Kruskal's theorem by a minimal bad sequence argument. Can you see it?)

17 Online algorithms and pathwidth

Parameterizing problems by looking at problems which are treelike or pathlike is not only useful in general algorithms by using dynamic programming. As we will soon see, they can be used for problems where dynamic programming is impossible, but the knowledge of the shape of the input is known.

17.1 Online problems

Suppose that you are packing three cars with boxes. Nobody tells you the shape of the boxes or the size, and you are given them one by one. You are not allowed to take a box out once it is in. You are in an *online* situation. This is quite different from the *offline* case where you get to see all the boxes *laid out* and then can decide how to pack.

Online problems abound in many diverse areas of computer science. There are many situations where it is necessary to make decisions without access to future inputs, or where only partial information about the input data is available. Examples include robot motion planning, or robot reconfiguration, in an uncharted environment; visual searching and mapping; resource management for multi-user, or multi-processor, computing systems; and network configuration problems. In short, any programmable system that operates in, or responds to, a dynamic environment can be considered as being "online". Due to advances in hardware design, complex online systems are rapidly becoming feasible.

An online algorithm is basically a response strategy for a sequence of inputs. Formally, we can think of an online problem as being one where the input data is supplied as a sequence of small units, one unit at a time. An online algorithm to solve the problem must respond by producing a sequence of outputs: after seeing t units of input, it must produce the tth unit of output.

For our purposes an online graph would be $G_0 \subset G_1 \subset \ldots$ where I give you the graph one vertex at a time (and its relationship with all the vertices seen so far) and you need to decide what to do with it. That is, you bust build a function $f : G_s \to \hat{G}_s$ where $\hat{G}_s$ is some kind of *expansion* of G_s before seeing G_{s+1}. The expansion would be a colouring (of G_s), scheduling, packing etc. (There are of course other variations, sometimes, as with a scheduler in a computer vertices come and go as people log off and on etc.)

In early work, probabilistic analysis was used to gauge the difficulty of online problems and to measure the performance of online algorithms. The probabilistic approach attempts to analyze the performance of online algorithms on "typical" inputs. It suffers from the criticism that we seldom have

a probability distribution to model a typical input.

During the last decade or so this approach has been largely superceded by *competitive analysis*, introduced by Sleator and Tarjan [40], and also Manasse, McGeoch, and Sleator [32]. In the competitive model the input sequence for an online problem is generated by an adversary, and the performance of an online algorithm is compared to the performance of an optimal offline algorithm, that is allowed to know the entire sequence of inputs in advance. Usually we do this by looking at the ratio $\frac{\text{Online}(G)}{\text{Offline}(G)}$ of the best online performance and the best offline one. Competitive analysis gives us the ability to make strong theoretical statements about the performance of online algorithms without making probabilistic assumptions about the input. Nevertheless, practitioners voice reservations about the model, citing its inability, in some important cases, to discern between online algorithms whose performances differ markedly in practice.

17.2 Colouring

A nice example is cououring graphs. We all know the famous 4-colour theorem that plan graphs can be coloured with 4 colours, and trees can be coloured with 2. Colouring these objects can be done efficiently in terms of the number of colours because we can "see" the whole graph.

Now suppose that I am giving you the graph one vertex at a time and all I tell you is the relationship ot the given vertex with the previously given ones. I will even tell you it is a tree. can we 2 colour it *online*.

So suppose that I give you a single vertex v_1. You colour it red say. Then I give you one more. Call this one v_2 and I tell you v_2v_1 is an edge. You colour it blue. Now I give you a new vertex v_3 and I tell you that it is not connected to v_1 or v_2. You must colour it red or blue to keep within the 2 colours. If you say red then I add a v_4 and connect it to v_3 (which is red) and v_1 (which is blue). You then must choose a 3rd colour for v_4.

In general, there is a tree with n vertices needing around $\log_2 n$ many colours; and trees can be coloured with $\log n$ many colours. Graphs of

treewidth k can be coloured with around $k \log n$ many colours and this is tight (e.g. Downey and McCartin [14]) The *performance ratio* compares the offline and the online performances is $O(\log n)$.

17.3 Online colouring graphs of bounded pathwidth

We have the following result of Kierstead and Trotter.

Theorem 17.1 (Kierstead and Trotter [25]). *There is an online algorithm that will colour a graph of pathwidth k with $3k-2$ many colours.*

To prove this result we will restate an equivalent formulation in the domain of partial orderings. A partial ordering $(P, \leq)$ is called *an interval ordering* if P is isomorphic to $(I, \leq)$ where I is a set of intervals of the real line and $x \leq y$ iff the right point of x is left of the left point of y.

There is a related notion of interval graph where the vertices are real intervals and xy is an edge implies that $I_x \cap I_y \neq \emptyset$. The following lemma relates pathwidth to interval orderings.

Lemma 17.2 (Folklore). *A graph G has pathwidth k iff G is a subgraph of an interval graph G' of maximal clique size $k+1$.*

Interval orderings can be characterized by the following theorem.

Theorem 17.3 (Fishburn [16]). *(i) Let P be a poset. Then the following are equivalent.*

- *(a) P is an interval ordering.*
- *(b) P has no subordering isomorphic to $\mathbf{2}+\mathbf{2}$ which is the ordering of four elements with $\{a,b,c,d\}$ with $a<b$, $c<d$ and no other relationships holding.*

(ii) (folklore) G is an interval graph iff G G is the incomparibility graph of some interval order P. That is, we join x to y if $x|y$ in P.

The proofs of the above are not difficult, but we omit for space considerations. Figure 17.3 below is an illustration of (ii).

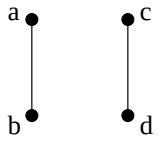
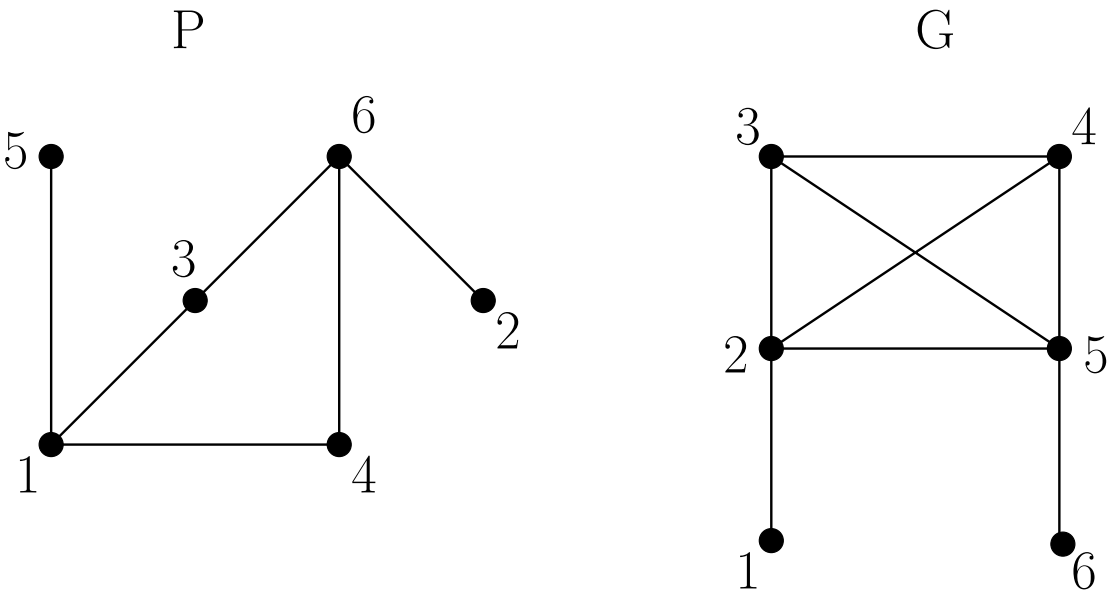


Figure 30: The canonical blockage



$$I_1 = [1,2) I_3 = [3,4) I_5 = [2,5)$$

$$I_2 = [0,4) I_4 = [3,4) I_6 = [4,6)$$

We first prove the following.

Theorem 17.4 (Kierstead and Trotter [25]). *Suppose that $(P, \leq)$ is a online interval ordering of width k. Then P can be online covered by $3k-2$ many chains.*

We need the following lemma. For a poset P, and subsets S, T, we can

define $S \leq T$ iff for each $x \in S$ there is some $y \in T$ with $x \leq y$. (similarly $S|T$ etc.)

Lemma 17.5. *If P is an interval order and $S, T \subset P$ are maximal antichains the either $S \leq t$ or $T \leq S$.*

Proof. The argument is interesting and instructive. It uses induction on k. If $k = 1$ then P is a chain, and there is nothing to prove. Suppose the result for k, and consider $k = 1$. We define B inductively by

$$B = \{p \in P : \text{width}(B^P \cup \{p\}) \leq k\}.$$

Here B^p denotes the amount of B constructed by step p of the online algorithm. Then B is a maximal subordering of P or width k. By the inductive hypothesis the algorithm will have covered B by $3k-2$ chains. Let $A = P-B$. Now it will suffice to show that A can be covered by 3 chains.

To see this it is enough to show that every elements of A is incomparible with at most two other elements of A. Then the relevant algorithm is the greedy algorithm, which will cover A, as we see elements *not* in B.

Lemma 17.6. *The width of A is at most 2.*

Proof. To see this, consider 3 elements $q, r, s \in A$. Then there are antichains Q, R, S in P of width k with $q|Q$, $r|R$ and $s|S$. Applying Lemma 17.5, we might as well suppose $Q \leq R \leq S$. Suppose that $r|q$ and $r|s$. Then we prove that $q < s$. Since $q|r$ and $\text{width}(P) \leq k + 1$, there is some $r' \in R$ with q and r' comparible. Since $q|Q$, $r' \notin Q$. Since the width of B is $\leq k$, there is some $q' \in Q$ q' and r' comparible. Since $Q \leq R$, there is some $r_0 \in R$ with $q' \leq r_0$. Sinc eR is an antichain, $q' \leq r'$. Since $q|q'$, $q \leq r'$. Similarly, there exists $r'' \in R$ with $r'' \leq s$. Since P does not have any ordering isomorphic to $\mathbf{2} + \mathbf{2}$, we can choose $r' = r''$, and hence $q < s$. □

Now we suppose that r, q, s, t are distinct elements of A with $q|\{r, s, t\}$. Then without loss of generality $r < s < t$ since the width of A is at most 2. Since $s \in A$ there is an antichain $S \subset B$ of length k with $s|S$. Since $s|q$, and $\text{width}(P) \leq k + 1$, q is comparible with some element $s' \in S$. If $s' < q$, then $s'|r$ and hence the suborder $\{s', q, r, s\}$ is isomorphic to $\mathbf{2}+\mathbf{2}$. Similarly,

$q < s'$ implies $s'|t$ and then the subordering $\{q, s', s, t\}$ is isomorphic to $\mathbf{2}+\mathbf{2}$. Thus there cannot be 4 elements r, q, s, t of A with $q|\{r, s, t\}$. Hence A can be covered by 3 chains. □

Since the algorithm uses only the information about comparibility of various elements, and not their ordering, we have the following corollary.

Corollary 17.7 (Kierstead and Trotter [25]). *Suppose that G is a online interval graph of width k. Then G can be online coloured by $3k - 2$ many colours.*

It has been shown in [25] that the bound above is *sharp*. The algorithm resembles the simplest of all online algorithms: *first fit*. A first fit algorithm for the problem above would be "colour with the fist available colour." In fact for pathwidth 2 or less, the algorithm above *is* first fit.

Question: Should there be any first fit algorithm for colouring interval graphs online? The answer is yes.

Theorem 17.8 (Kierstead [26]). *First fit online colours interval graphs of width k with at most $40k$ colours.*

Kierstead and Qin [27] improved the constant to 25.72. It is unknown what the the correct constant is, but the lower bound is 4.4 as shown by Chrobak and Slusarek [6] leaving a rather large gap.

17.4 Projects

(i) What about online algorithms for other problems? Bin packing and scheduling are good candidates here.

(ii) Is the notion of performance ratio the best measure? Why not average case algorithms?

(iii) What about other online versions of Dilworth's decomposition theorem. What's the best version here?

(iv) How well do these algorithms perform *in practice*? Fouhy found that first fit colours randomly generated bounded pathwidth graphs very well. What about online performance of chain decompositions. What parameterizations could be used here?

18 Exercises 10

1 Prove Lemma 17.2.

2 Prove Lemma 17.5.

3 Construct an interval representation of the following graph G of width 4.

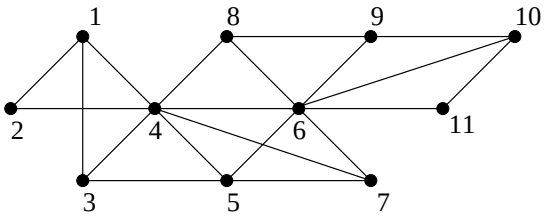


Figure 31: An interval graph

4 Prove Fishburn's theorem.

19 Selected solutions

19.1 Exercises 1

1 (b) is not a flow as $f(A, E) = 4 > C(A, E) = 3$, for instance

(b) Flow, value 8.

(d) Flow value 14.

2 Find the capacity of each cut in the transportation network (a).

There are 32 cuts, with minimum value 14.

For example, if $X = \{s, a\}$, and so $\overline{X} = \{b, c, d, e, t\}$, then $C(X, \overline{X}) = 14$.

Below is a table of the cuts and their capacities.

Cut	Capacity
$\{s\}$	16
$\{s, a\}$	14
$\{s, b\}$	20
$\{s, c\}$	19
$\{s, d\}$	25
$\{s, e\}$	22
$\{s, a, b\}$	15
$\{s, a, c\}$	17
$\{s, a, d\}$	23
$\{s, a, e\}$	17
$\{s, b, c\}$	23
$\{s, b, d\}$	29
$\{s, b, e\}$	22
$\{s, c, d\}$	22
$\{s, c, e\}$	23
$\{s, d, e\}$	29
$\{s, a, b, c\}$	18
$\{s, a, b, d\}$	24
$\{s, a, b, e\}$	14
$\{s, a, c, d\}$	20
$\{s, a, c, e\}$	18
$\{s, a, d, e\}$	29
$\{s, b, c, d\}$	26
$\{s, b, c, e\}$	23
$\{s, b, d, e\}$	29
$\{s, c, d, e\}$	24
$\{s, a, b, c, d\}$	21
$\{s, a, b, c, e\}$	15
$\{s, a, b, d, e\}$	21
$\{s, a, c, d, e\}$	19
$\{s, b, c, d, e\}$	24
$\{s, a, b, c, d, e\}$	16

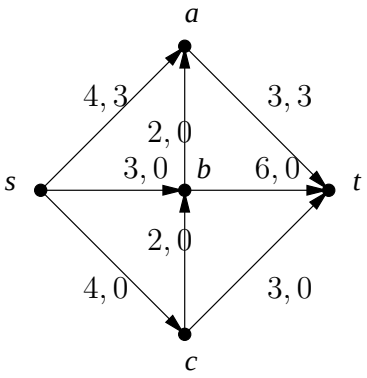
3 Suppose that $(X, \overline{X})$ and $(Y, \overline{Y})$ are minimum cuts in a transportation network N. Suppose that f is a maximum flow. Let $(a, b) \in (X \cup Y, \overline{X \cup Y}) = (X \cup Y, \overline{X} \cap \overline{Y})$ Then $(a, b) \in (X, \overline{X})$, or $(a, b) \in (Y, \overline{Y})$. (This is beciause a in in either X or Y, and b is in $\overline{X} \cap \overline{Y}$, and hence

both $\overline{X}$ *and* $\overline{Y}$.) In either case $f(a,b) = \gamma(a,b)$ by max-flow/ min cut. Hence the cut $(X \cup Y, \overline{X \cup Y})$ is a minumum cut since all edges are at capacity or zero. The case $(X \cap Y, \overline{X \cap Y})$ is the same.

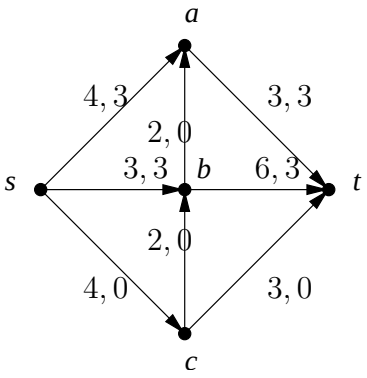
19.2 Exercises 2

1 (a)

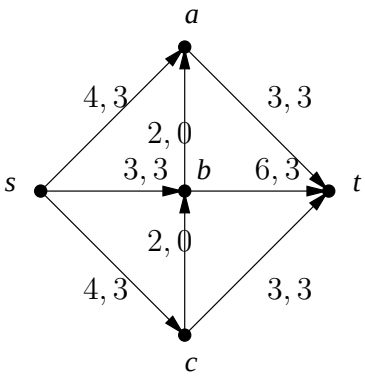
vertex processed	vertex labeled	label
	s	$[s+, \infty]$
s	a	$[s+, 4]$
	b	$[s+, 3]$
	c	$[s+, 4]$
a	t	$[a+, 3]$



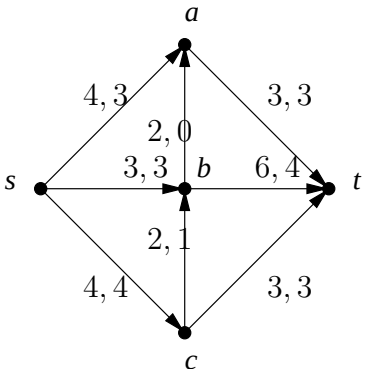
vertex processed	vertex labeled	label
	s	$[s+, \infty]$
s	a	$[s+, 1]$
	b	$[s+, 3]$
	c	$[s+, 4]$
a	—	
b	t	$[b+, 3]$



vertex processed	vertex labeled	label
	s	$[s+, \infty]$
s	a	$[s+, 1]$
	c	$[s+, 4]$
a	—	
c	b	$[c+, 2]$
	t	$[c+, 3]$



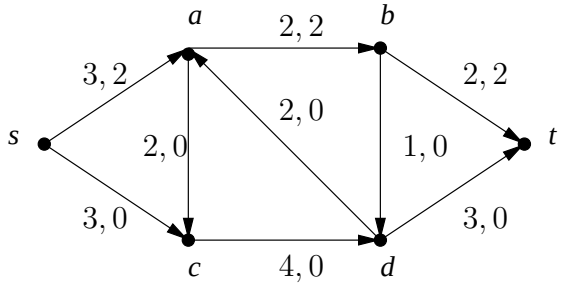
vertex processed	vertex labeled	label
	s	$[s+, \infty]$
s	a	$[s+, 1]$
	c	$[s+, 1]$
a	—	
c	b	$[c+, 1]$
b	t	$[b+, 1]$



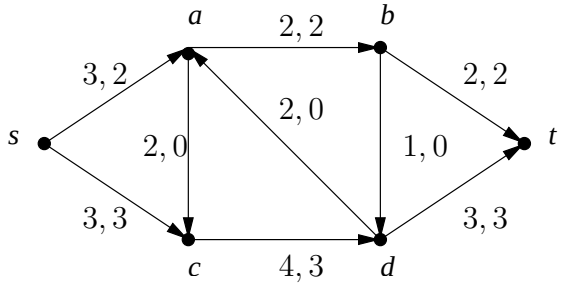
(b)

vertex processed	vertex labeled	label
	s	$[s+, \infty]$
s	a	$[s+, 1]$
a	—	

vertex processed	vertex labeled	label
	s	$[s+, \infty]$
s	a	$[s+, 3]$
	c	$[s+, 3]$
a	b	$[a+, 2]$
b	d	$[b+, 1]$
	t	$[b+, 2]$



vertex processed	vertex labeled	label
	s	$[s+, \infty]$
s	a	$[s+, 1]$
	c	$[s+, 3]$
a	—	
c	d	$[c+, 3]$
d	t	$[d+, 3]$

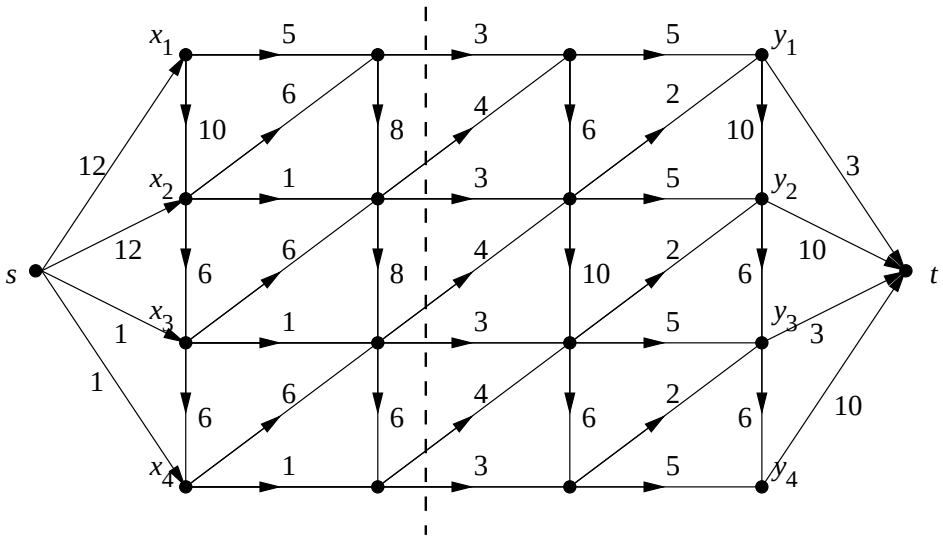


vertex processed	vertex labeled	label
	s	$[s+, \infty]$
s	a	$[s+, 1]$
a	c	$[a+, 1]$
c	d	$[c+, 1]$
d	—	

19.3 Exercises 3

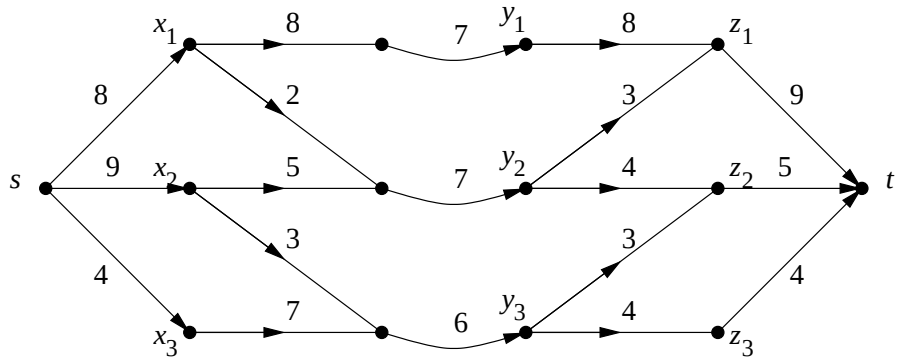
1 To determine if supply can meet demand, we attach extra edges, as shown below, and then determine if a maximum flow would saturate the demand edges (which connect to t).

In this case, there is a cut with capacity 24 (also shown in the figure below) so the total demand of 26 cannot be met. (There is even a cut with capacity 20.)

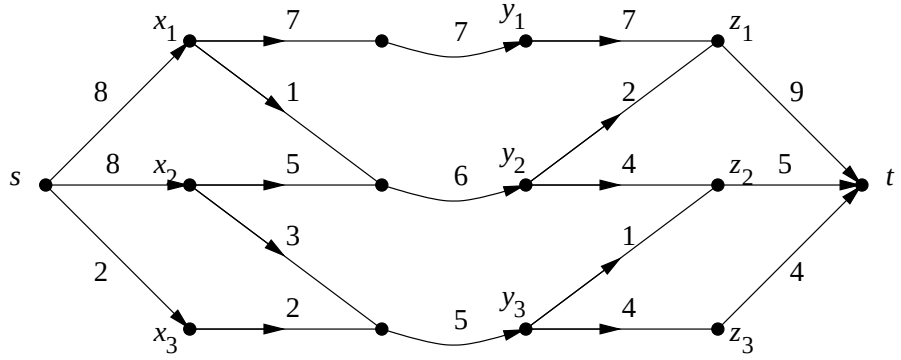


2

- This time we have extra constraints to satisfy. The upper bounds on the amounts that y_1, y_2 and y_3 can process can be added to the graph by replacing the distribution plants with edges as shown below.

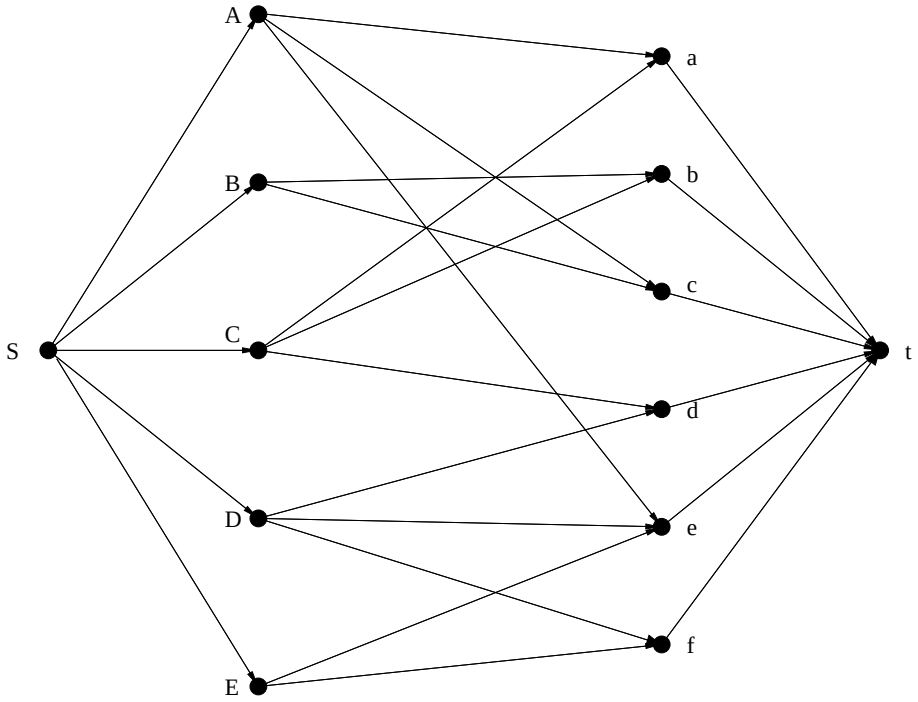


It is harder to fit the lower bounds into our framework, but as luck would have it, they are satisfied by any maximum flow of the graph. Such a flow is shown below. Because the maximum flow saturates the outgoing edges, supply *can* meet demand while satisfying the constraints.



3 (i) The network is below

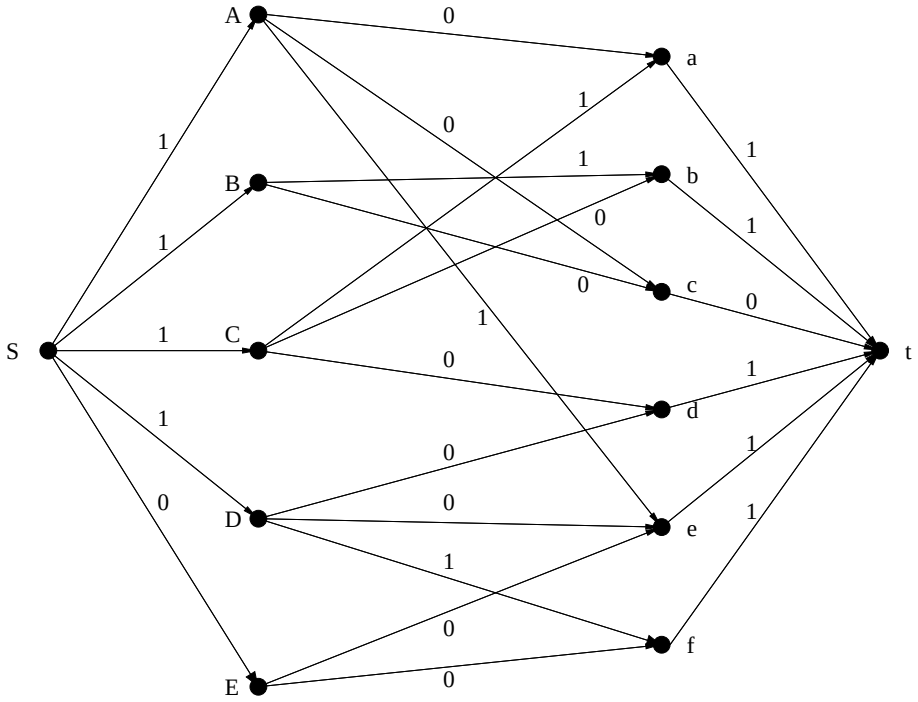
Figure 1



All edges have capacity 1. Possible answer (A,a),(B,b), (C,c), (D,e), (E,f).

(iii)

Figure 2



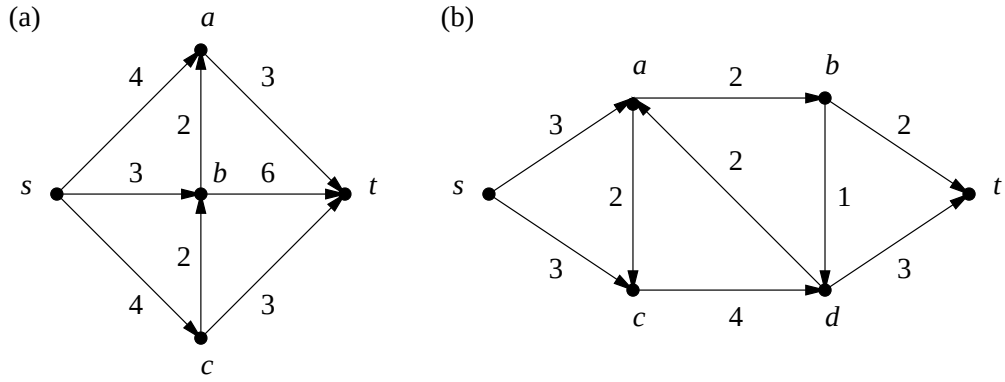
This does depend on the order. There are several possible augmenting paths. For example: (s,E), (E,e), (e,A),(A,c), (c,t).

19.4 Exercises 4

2 2.

19.5 Exercises 5

-
- We run the Goldberg-Tarjan algorithm on the networks shown below.



1 (a) The maximum flow is 10 and a minimum cut is $\{s, a\}$.

	$d(x)$			$f(x,y)$								
step	a	b	c	(s,a)	(s,b)	(s,c)	(a,t)	(b,t)	(c,t)	(b,a)	(c,b)	active
1	1	1	1	4	3	4	0	0	0	0	0	a, b, c
2	1	1	1	4	3	4	**3**	0	0	0	0	a, b, c
3	1	1	1	4	3	4	3	**3**	0	0	0	a, c
4	1	1	1	4	3	4	3	3	**3**	0	0	a, c
5	1	1	**2**	4	3	4	3	3	3	0	0	a, c
6	1	1	2	4	3	4	3	3	3	0	**1**	a, b
7	1	1	2	4	3	4	3	**4**	3	0	1	a
8	**6**	1	2	4	3	4	3	3	3	0	1	a
9	6	1	2	**3**	3	4	3	3	3	0	1	$-$

1 (b) The maximum flow is 5 and a minimum cut is $\{s, a, c, d\}$.

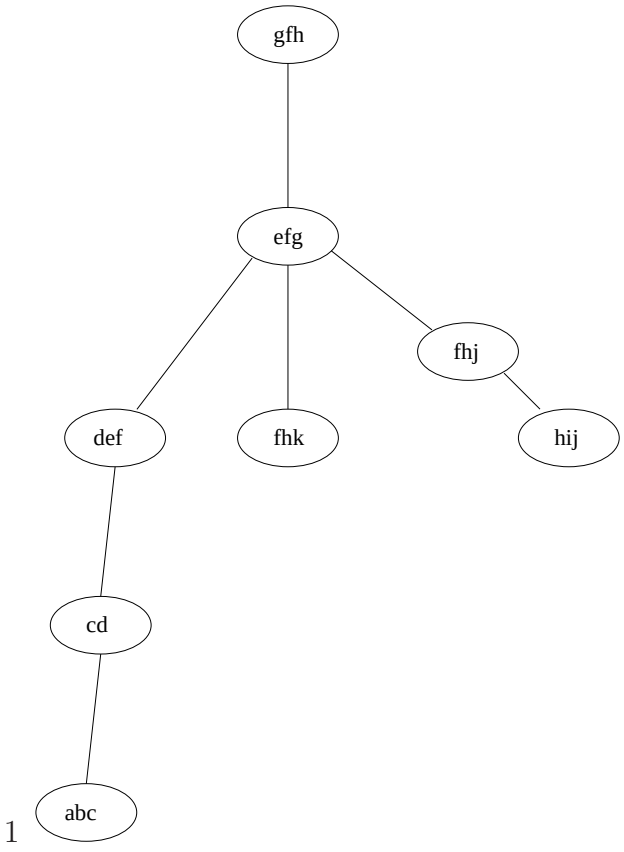
	$d(x)$				$f(x,y)$									
step	a	b	c	d	(s,a)	(s,c)	(a,c)	(a,b)	(c,d)	(b,t)	(b,d)	(d,a)	(d,t)	active
1	1	1	1	1	3	3	0	0	0	0	0	0	0	a,c
2	**2**	1	1	1	3	3	0	0	0	0	0	0	0	a,c
3	2	1	1	1	3	3	0	**2**	0	0	0	0	0	a,b,c
4	2	1	1	1	3	3	**1**	2	0	0	0	0	0	b,c
5	2	1	1	1	3	3	1	2	0	**2**	0	0	0	c
6	2	1	**2**	1	3	3	1	2	0	2	0	0	0	c
7	2	1	2	1	3	3	1	2	**4**	2	0	0	0	d
8	2	1	2	1	3	3	1	2	4	2	0	0	**3**	d
9	2	1	2	**3**	3	3	1	2	4	2	0	0	3	d
10	2	1	2	3	3	3	1	2	4	2	0	**1**	3	a
11	**3**	1	2	3	3	3	1	2	4	2	0	1	3	a
12	3	1	2	3	3	3	**2**	2	4	2	0	1	3	c
13	3	1	**4**	3	3	3	2	2	4	2	0	1	3	c
14	3	1	4	3	3	3	**1**	2	4	2	0	1	3	a
15	**4**	1	4	3	3	3	1	2	4	2	0	1	3	a
16	4	1	4	3	3	3	1	2	4	2	0	**0**	3	d
17	4	1	4	**5**	3	3	1	2	4	2	0	0	3	d
18	4	1	4	5	3	3	1	2	4	2	0	**1**	3	a
19	**5**	1	4	5	3	3	1	2	4	2	0	1	3	a
20	5	1	4	5	3	3	**2**	2	4	2	0	1	3	c
21	5	1	**6**	5	3	3	2	2	4	2	0	1	3	c
22	5	1	6	5	3	3	**1**	2	4	2	0	1	3	a
23	**6**	1	6	5	3	3	1	2	4	2	0	1	3	a
24	6	1	6	5	3	3	1	2	4	2	0	**0**	3	d
25	6	1	6	**7**	3	3	1	2	4	2	0	0	3	d
26	6	1	6	7	3	3	1	2	4	2	0	**1**	3	a
27	**7**	1	6	7	3	3	1	2	4	2	0	1	3	a
28	7	1	6	7	**2**	3	1	2	4	2	0	1	3	–

Note that by step 10 (and certainly by step 18) it was clear that we would end up sending the last unit of flow back to s. It is not really necessary to run the algorithm to its bitter end.

19.6 Exercises 6

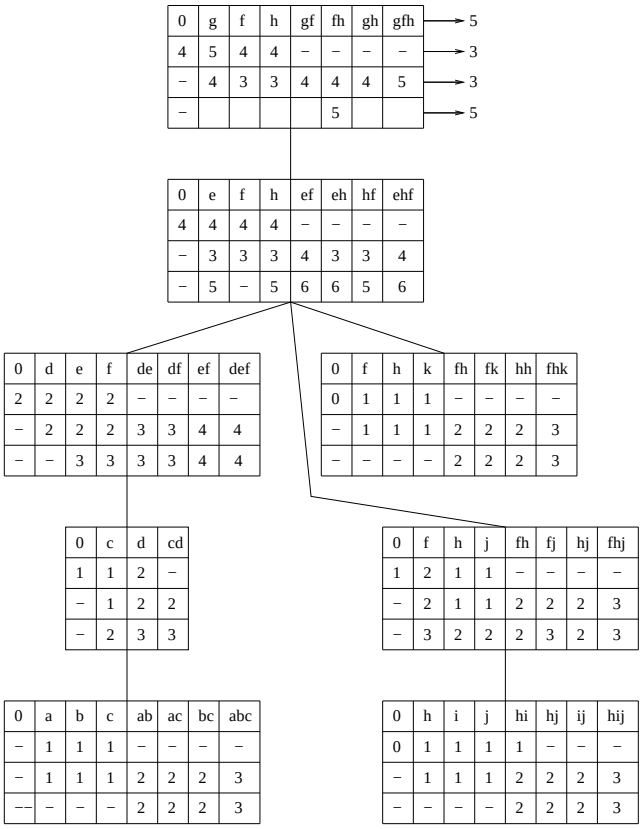
1 (i) This follows by property 2. of Definition 11.3. To see (ii), for each pair of vertices x, y in C there is some T_z with $x, y \in T_z$. By (i) the collection of z with T_z containing x form a subtree $\mathcal{T}_x$ of $\mathcal{T}$. Now let $\{x_1, ..., x_n\}$ list C. Each x_i thus generates a subtree $\mathcal{T}_i$ of $\mathcal{T}$. The intersection $\cap \mathcal{T}_i$ is nonempty by (i) and property 2., and hence there is some z with $C \subseteq \mathcal{T}_z$.

19.7 Exercises 7



1

2 (Modulo mistakes)



3 (i)Consider a tree decomposition $\{T_x : x \in T\}$ for some rooted tree T.

Without loss of generality, we can consider it to be a smooth decomposition. (That is, as we move from one bag to another, at most one vertex is deleted; and children of some bag are identical to that bag.) Now start from the leftmost bag using depth-first ordering. This has some vertex that is lost as we move to a parent. Make this the first vertex. It can only be connected to vertices in the very bag it is in. That's at most k of them. Remove this bag, and continge this process inductively.

(ii) Suppose that G is planar. Then, by Euler's formula, there is some vertex in G of degree 5. call it the first vertex. Remove it and the edges connecting it to G. repeat.

19.8 Exercises 8

1 Colour $\{m_i, m_j\}$ red if $i < j$ and $m_i < m_j$ and blue otherwise. The take $n = R(2, 2)$.

2 This is trickier. A more general result is that if $P = (M, \leq)$ is a partial ordering with $|M| \geq mn + 1$, then P either has a chain of size m or an antichain of size n. This follows by the pigeonhole principle and the more general fact that if P is any partial ordering of with largest antichain of size d (this is called the *width*), then P can be decomposed into d disjoint chains $M_1 \cup \cdots \cup M_d$. The proof briefly is the following. Prove this by induction on $n = |M|$. It is clearly true for $n = 1$, and then P is a point. If we assume for n, and take any poset P of size $n + 1$. From this remove a maximal chain C, and call the result $\hat{P}$. Now we can apply the induction hypothesis to $\hat{P}$. If the width of $\hat{P}$ is $d-1$, then we can decompuse $\hat{P} = \hat{M}_1 \cup \cdots \cup \hat{M}_{d-1}$ and add C to get a width d decomposition of P. On the other hand, $\hat{P}$ might have width d. Then you can decompose $\hat{P}$ into d chains and have a antichain $a_1 \ldots a_d$ within it. Now consider the collection $P^+ = \{p \in P : \exists i(p \geq a_i)\}$, and $P^- = \{p \in P : \exists i(p \leq a_i)\}$. We can apply induction to P^+ and decompose $P^=$ into d chains, and it is easy to show that each has a a_i at the bottom. Similarly for P^-. Then we put these chains together.

3 (Tarsy) Take n with $n \to (k)^3_2$. Now colour $\{i, j, k\}$ if $i < j < k$ red if i to j to k is clockwise, and blue otherwise. take the homogeneous set. If all red then all clockwise and otherwie all anticlockwise. Hence convex.

19.9 Exercises 9

2 We can give an easy proof of this lemma by using the labeled version of Kruskal's Theorem. We can consider a sequence $\langle a_0, a_1, ...a_n\rangle$ as a tree with root labeled a_0, immediate successor $a_1, ...$ To a bad $\mathcal{S}$ sequence would correspond a sequence of trees, bad with respect to labeled topological ordering.

19.10 Exercises 10

1 (sketch) One direction is to take a path decomposition and map the parts of the path with bags comntains a vertex to intervals. The other dierction is similar.

2 Prove Lemma 17.5. We need to prove: *If P is an interval order and $S, T \subset P$ are maximal antichains the either $S \leq t$ or $T \leq S$.*

Suppose $S \not\leq T$. then for some $s \in S$, and every $t \in T$, $s \not\leq t$. But T is a maximal antichain. Thus there is some $t \in T$ comparible with s. Hence $t < s$, for this t. Similarly, there is some $s' \in S$ and $t' \in T$ with $s' < t'$ should $T \not\leq S$. Since S and T are antichains, $s \neq s'$ and $t \neq t'$. Thus $s|s'$ and $t|t'$. But then the subordering $\{s, s', t, t'\}$ is isomorphic to $\mathbf{2} + \mathbf{2}$. This contradicts the fact that P is an interval order.

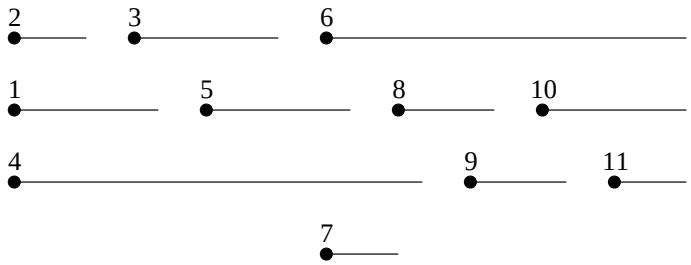


Figure 32: An interval decomposition

3

References

[1] R. Ahuja and J Orlin, *A fast and simple algorithm for the maximum flow problem, Operations Research*, 37 (1989), 748-759.

[2] H. L. Bodlaender,*A Tourist's Guide Through Treewidth*, TR RUU-CS-92-12, Computer Science Department, Univ. Urtech, The Netherlands, March 1992.

[3] H. L. Bodlaender, *Discovering treewidth*, TR RUU-CS-2005-18, Computer Science Department, Univ. Urtech, The Netherlands, 2005.

[4] P. Bellenbaum and R. Diestel, *Two short proofs concerning tree-decompositions, Probability and Computing*, **11** (2002), 1-7.

[5] R. B. Borie, R. G. Parker and C. A. Tovey. *Automatic generation of linear-time algorithms from predicate calculus descriptions of problems on recursively constructed graph families,* (1988 Manuscript) final version appeared in *Algorithmica* 7 (1992), 555–582.

[6] M. Chrobak and M. Slusarek, *On some packing problems related to dynamic storage allogation, RARIO Inform. Theor. Appl.*, Vol. 22 (1988), 487-499.

[7] B. Courcelle, *Recognizability and second-order definability for sets of finite graphs,* Technical Report I-8634, Universite de Bordeaux, 1987.

[8] B. Courcelle, *Graph rewriting: an algebraic and logic approach,* in *Handbook of Theoretical Computer Science,* Chapter 5 ed J. van Leeuwen, North–Holland, Amsterdam Vol. B, (1990)

[9] B. Courcelle, *The monadic second order theory of graphs I : recognisable sets of finite graphs, Information and Computation,* , Vol. 85, (1990) 12-75.

[10] R. Diestel, *Graph Theory*, 2nd Edition, Springer-Verlag, 1999.

[11] G. Dirac and S. Schuster, *A theorem of Kuratowski, Nederl. Akad. Wetensch. Proc. Ser. A*, Vol. 57 (1954) 343-348

[12] R. Downey, *On Ramsey-type theorems and their applications, Singapore Math. Medley*, 17 (1989), 58-78.

[13] R. Downey and M. Fellows, *Parameterized Complexity*, Springer-Verlag, 1999.

[14] R. Downey and C. McCartin, *Online problems, pathwidth, and persistence, Proceedings IWPEC 2004.* Springer-Verlag Lecture Notes in Computer Science 3162, pp 13-24, 2004.

[15] P. Erdös and G. Szekeres, *A combinatorial problem in geometry, Compositio Math.*, 2 (1935), 464-470.

[16] P. Fishburn, *Intransitive indifference in preference theory: A survey, Operations Research*, **18** (1970), 207-228.

[17] J. Fouhy, *Computational Experiments on Graph Width Metrics,* MSC Thesis, Victoria University, Wellington, 2003.

[18] J. Flum and M. Grohe, *Parameterized Complexity Theory*, Springer-Verlag, 2006.

[19] H. Furstenberg, *Ergodic behaviour of diagonal measures and a theorem of Szemeredi on arithmetical progressions, J. Anal. Math.*, 31 (1977), 204-256.

[20] T. Gower, *A new proof of Szemeredi's theorem*, *Geom. Funct. Anal.*, 8 (1998), 529-551.

[21] T. Gower, *Lower bounds of tower type for Szemeredi's uniformity lemma, Geom. Funct. Anal.*, 7 (1997), 322-337.

[22] R. Graham, B. Rothschild, and J. Spencer, *Ramsey Theory*, 2nd Edition, Wiley, 1980.

[23] B. Green and T. Tao, *The primes contain arbitrarily long arithmetic progressions, Annals of Math.* to appear.

[24] G. Higman, *Ordering by divisibility in abstract algebras, Proc. London Math. Soc.*, Vol. 2, (1952), 326-336.

[25] H. A. Kierstead, W. A. Trotter, *An Extremal Problem in Recursive Combinatorics. Congressus Numeratium* 33 , (1981), 143-153.

[26] H. Kierstead, *The linearity of first-fit colouring of interval graphs, SIAM J. on Discrete Math.*, Vol. 1 (1988), 528-530.

[27] H. Kierstead and J. Qin, *Colouring interval graphs with first-fit, Discrete Math.*, Vol. 144 (1995), 47-57.

[28] K. Kuratowski, *Sur le probleme des courbes gauches en topologie, Fund. Math.*, Vol. 15, (1930), 271-283.

[29] J. B. Kruskal, *Well-quasi-ordering, the tree theorem, and Vazsonyi's conjecture, Trans Amer. math. Soc.*, Vol. 95, (1960), 210-225.

[30] B. Landman and A. Robertson, *Ramsey Theory on the Integers,* AMS publ. 2003.

[31] S. L. Lauritzen and D. J. Spiegelhalter, *Local computations with probabilities on graphical structures and their applications to expert systems, J. Roy. Stat. Soc. Ser. B*, Vol. 50 (1988), 157-224.

[32] M. S. Manasse, L. A. McGeoch, D. D. Sleator: *Competitive Algorithms for Online Problems.*

[33] C. St. J. A. Nash-Williams, *Well-quasi-ordering finite trees, Proc. Camb. Philos. Soc.*, Vol. 59, (1963), 291-296.

[34] J. Nesetril and V. Rodl, *Mathematics of Ramsey Theory,* Springer-Verlag, 1990.

[35] R. Niedermeier, *Invitation to Fixed Parameter Algorithms,* Oxford University press, 2006.

[36] J. Paris and L. Harrington, *A mathematical incompleteness in Peano Arithmetic,* in *Handbook of Mathematical Logic,* (J. Barwise ed.) North Holland, 1977), 1133-1142.

[37] F. Ramsey, *A problem in formal logic,proc. London math. Soc.*, 30 (1930), 264-286.

[38] I. Rival, *Graphs and Order*, Reidel, 1984.

[39] S. Shelah, *Primitive recursive bounds for van der Waerden numbers, J. Amer. math. Soc*, Vol. 1 (1988), 638-697.

[40] D. D. Sleator, R. E. Tarjan: *Amortized Efficiency of List Update and Paging Rules.* Comunication of the ACM 28, pp 202-208, 1985.

[41] S. Simpson, *Logic and Combinatorics,* (ed.) Amer. Math. Soc. Publ., Contemporary Math. Vol. 65 (1985), (especially Simpson 259-394.)

[42] S. Simpson, *Subsystems of Second Order Arithmetic,* Springer-Verlag, 1999.

[43] E. Szemeredi, *On sets of integers containing k elements in arithmetic progression, Acta. Arith.*, 27 (1975), 199-245.

[44] R. Thomas, *A Menger-like property of tree-width, the finite case, J. Comb. Theory, B.* Vol. 48 (1990), 67-76.

[45] B. van der Waerden, *Beweis einer Baudetschen vermutung, Nieuw. Arch. Wisk.,* 15 (1927), 212-216.

[46] K. Wagner, *Uber einer eigenschaft der ebener complexe, Math. Ann.*, Vol. 14, (1937), 570-590.