Integration Evolutionärer Algorithmen in das MFOS

Knut Willems

Diplomarbeit

4. Januar 2007

Thema gestellt von

Prof. Dr. W.-M. Lippe

Institut für Informatik

Fachbereich Mathematik und Informatik

Westfälische Wilhelms-Universität Münster

Erklärung

Hiermit versichere ich, dass ich die Arbeit selbständig und unter ausschließlicher Zuhilfenahme der angegebenen Literatur verfasst habe.

Knut Willems

Münster, 4. Januar 2007

Inhaltsverzeichnis

1 Einleitung

Ein Teilbereich der Informatik wird als *Softcomputing* bezeichnet. Unter diesen Begriff ordnet man die Forschungsgebiete Künstliche Neuronale Netze, Fuzzy-Logik, Evolutionäre Algorithmen und Chaostheorie ein. Ihnen ist gemeinsam, dass sie als Vorbild ihrer Arbeitstechniken, die natürliche, biologische Informationsverarbeitung haben.

Softcomputing unterscheidet sich wesentlich vom klassischen Hardcomputing. Entsprechend ihrem natürlichen Vorbild suchen Softcomputingsysteme keine für ein Problem bestmögliche, exakte Lösung, sondern eine approximative Lösung, deren Güte für die Problemstellung ausreicht. Methoden des Softcomputing werden erfolgreich in Bereichen wie Mustererkennung, Data Mining, Analysesysteme, Optimierung und Prozesssteuerung eingesetzt. Softcomputingsysteme werden nicht im herkömmlichen Sinne programmiert, d.h. es braucht kein expliziter Lösungsalgorithmus entwickelt werden, welcher anschliessend in einer bestimmten Programmiersprache codiert werden müsste.

Jedes Gebiet des Softcomputing hat seine spezifischen Stärken und Schwächen. Um die Stärken zu nutzen und die Schwächen zu kompensieren entwickelt man kombinierte Systeme.

Am Institut für Informatik der Westfälischen Wilhelms-Universität Münster wurde ein System zur Optimierung von Fuzzy-Controllern mittels Künstlicher Neuronaler Netze entwickelt, das Münsteraner Fuzzy-Optimierungs-System (MFOS) [Lippe06].

In dieser Arbeit wird untersucht, ob sich das bestehende MFOS durch die Integration von Evolutionären Algorithmen verbessern lässt. Hierzu werden Möglichkeiten Fuzzy-Controller mit evolutionären Methoden zu optimieren betrachtet und Experimente durchgeführt.

Diese Arbeit hat den folgenden Aufbau: Kapitel zwei und drei führen in für diese Arbeit wichtige theoretische Grundlagen von Fuzzy Logik und Evolutionären Algorithmen ein. Im vierten Kapitel werden Ansätze zur Optimierung von Fuzzy-Controllern mit evolutionären Algorithmen betrachtet. In Kapitel fünf wird die Erweiterung der MFOS-M-Implementierung um die Möglichkeit der Optimierung mit genetischen Algorithmen beschrieben. Das sechste Kapitel beschreibt die Durchführung und die Ergebnisse der Experimente zur Optimierung von Fuzzy-Controllern. In Kapitel sieben findet sich das Fazit zu dieser Arbeit.

2 Fuzzy-Logik

Dieses Kapitel behandelt grundlegende Aspekte des Forschungsgebiets Fuzzy-Logik. Die Darstellung stützt sich wesentlich auf [Lippe06] und [Nauck96]. Der interessierte Leser findet dort detaillierte, weiterführende Informationen.

Die mathematische Theorie der Fuzzy-Mengen (unscharfe Mengen) wurde von Lotfi Zadeh entwickelt [Zadeh65]. Er entwarf das erste Fuzzy-Steuerungssystem [Zadeh72]. Mamdani und Assilian konstruierten den ersten Fuzzy-Controller, der reelle Ein- und Ausgabedaten verarbeiten konnte, den sog. Mamdani-Controller [Mamdani75].

Im folgenden werden zunächst die zur Beschreibung eines Fuzzy-Controllers nötigen Grundlagen der Fuzzy-Mengen und der Fuzzy-Logik vorgestellt. Anschließend wird die prinzipielle Arbeitsweise von Fuzzy-Controllern erläutert. Dann werden Vor- und Nachteile von Fuzzy-Controllern betrachtet und anhand des Beispiels des inversen Pendels die Arbeitsweise eines Mamdani-Controllers dargestellt.

2.1 Von klassischen Mengen zu Fuzzy-Mengen

Fuzzy-Mengen stellen eine Erweiterung der klassischen Mengen dar. In der klassischen Mengenlehre ist die Zugehörigkeit eines Elementes zu einer Menge exakt definiert, entweder es gehört zur Menge oder nicht. Dieses Konzept ist für die Modellierung der Ausdrucksweise von Menschen nicht ausreichend. Der Mensch verwendet zur Beschreibung von Sachverhalten häufig unscharfe, vage Begriffe, die auf seiner subjektiven Einschätzung basieren. Beispielsweise könnten bei einer gemeinsamen Autofahrt die eine Person die momentane Geschwindigkeit als ziemlich schnell und die andere als eher langsam bewerten. In der klassischen Mengenlehre müsste eine scharfe Grenze, z.B. bei 80 km/h gezogen werden, die schnelle Geschwindigkeiten von langsamen Geschwindigkeiten trennt. Somit würde die Geschwindigkeit 79,99 km/h eine langsame- und 80,01 km/h eine schnelle Geschwindigkeit darstellen. Dies modelliert den Sachverhalt, dass die Werte beinahe gleich sind, völlig unzureichend.

In den nächsten beiden Abschnitten wird die Erweiterung der klassischen Mengen zu den Fuzzy-Mengen betrachtet.

2.1.1 Klassische Mengen

Im folgenden sei stets X eine Grundmenge, z.B. $X = \mathbb{R}$ oder $X = \{1, \ldots, 100\}$, und A eine Teilmenge von X ($A \subset X$). Dann gilt für jedes Element $x \in X$, dass x entweder in A enthalten ist ($x \in A$), oder dass x nicht in A enthalten ist ($x \notin A$).

Man kann endliche klassische Mengen in der Aufzählungsform notieren, etwa

$$X = \{1, \ldots, 10\}.$$

Unendliche Mengen können durch ein Prädikat spezifiziert werden, z.B.

$$Y = \{y \in \mathbb{N} \mid y \geq 10\}.$$

Für die Erweiterung zu den Fuzzy-Mengen ist die Beschreibung einer klassischen Menge durch eine sogenannte *charakteristische Funktion* wesentlich.

Definition 2.1 (Charakteristische Funktion)
Eine Funktion

$$\chi_A : X \to \{0, 1\}$$

mit

$$\chi_A(x) = \begin{cases} 1 & \text{falls } x \in A \\ 0 & \text{sonst} \end{cases}$$

heisst eine charakteristische Funktion der Menge A.

Es gilt:

$$A = \{x \in X \mid \chi_A(x) = 1\}.$$

Wichtige Operationen auf klassischen Mengen sind die Operationen Vereinigung, Durchschnitt und Komplementbildung.

Definition 2.2 (Standard-Operationen auf klassischen Mengen)
Seien A und B Teilmengen einer Grundmenge X.

$$\begin{aligned} A \cup B &= \{x \in X \mid x \in A \text{ oder } x \in B\} && \text{(Vereinigung von A und B)} \\ A \cap B &= \{x \in X \mid x \in A \text{ und } x \in B\} && \text{(Durchschnitt von A und B)} \\ A^c &= \{x \in X \mid x \notin A\} && \text{(Komplement von A)} \end{aligned}$$

die zugehörigen charakteristischen Funktionen sind:

$$\begin{aligned} \chi_{A \cup B}(x) &= \max\{\chi_A(x), \chi_B(x)\} \\ \chi_{A \cap B}(x) &= \min\{\chi_A(x), \chi_B(x)\} \\ \chi_{A^c}(x) &= 1 - \chi_A(x) \end{aligned}$$

2.1.2 Fuzzy-Mengen

Die charakteristische Funktion χ der klassischen Mengenlehre wird für Fuzzy-Mengen zur Zugehörigkeitsgradfunktion (Mitgliedsgradfunktion) μ erweitert, indem der Wertebereich auf das kompakte Intervall $[0, 1]$ ausgedehnt wird.

Definition 2.3 (Zugehörigkeitsgradfunktion)
Eine Funktion

$$\mu_A : X \to [0, 1]$$

heisst eine Zugehörigkeitsgradfunktion der Menge A. Die Funktion μ beschreibt die graduelle Zugehörigkeit eines Elements $x \in X$ zu A, anders ausgedrückt: x ist zum Grade $\mu_A(x)$ in A enthalten.

Analog der Beschreibung klassischer Mengen über die charakteristische Funktion lassen sich Fuzzy-Mengen vollständig durch ihre Zugehörigkeitsgradfunktion beschreiben.

Definition 2.4 (Fuzzy-Menge)
Eine Fuzzy-Menge $\tilde{A}$ über einer Grundmenge X ist gegeben durch:

$$\tilde{A} = \{(x, \mu_{\tilde{A}}(x)) \mid x \in X\}$$

Bemerkung 2.1
Klassische Mengen sind spezielle Fuzzy-Mengen, denn einer klassischen Menge A über einer Grundmenge X entspricht die Fuzzy-Menge

$$\tilde{A} = \{(x, \mu_{\tilde{A}}(x)) \mid x \in X,\ \mu_{\tilde{A}}(x) = 1\}.$$

Wichtige Fuzzy-Mengen für Theorie und Praxis sind die folgenden Fuzzy-Mengen:

Definition 2.5 (Dreiecksmenge)
Eine Dreiecksmenge $\tilde{D}(l, m, r)$ ist gegeben durch ihre linke Grenze l, ihre Mitte m, und ihre rechte Grenze r und hat die Zugehörigkeitsgradfunktion

$$\mu_{\tilde{D}}(x) = \begin{cases} 0 & : x \leq l \\ \frac{x-l}{m-l} & : l < x < m \\ 1 & : x = m \\ \frac{m-x}{r-m} + 1 & : m < x < r \\ 0 & : x \geq r \end{cases}$$

Definition 2.6 (Trapezmenge)
Eine Trapezmenge $\tilde{T}(l, m, r)$ ist gegeben durch ihre linke Grenze l, ihre „linke Mitte“ m_1,

ihre „rechte Mitte“ m_2 und ihre rechte Grenze r und hat die Zugehörigkeitsgradfunktion

$$\mu_{\tilde{T}}(x) = \begin{cases} 0 & : x \leq l \\ \frac{x-1}{m_1-l} & : l < x < m_1 \\ 1 & : m_1 \leq x \leq m_2 \\ \frac{m_2-x}{r-m_2} + 1 & : m < x < r \\ 0 & : x \geq r \end{cases}$$

Definition 2.7 (Gaußmenge)
Eine Gaußmenge $\tilde{G}(l, m, r)$ ist gegeben durch ihre Mitte m und einen Weitenparameter w und hat die Zugehörigkeitsgradfunktion

$$\mu_{\tilde{G}}(x) = exp\left(-\left(\frac{x-m}{w}\right)^2\right)$$

Abbildung 2.1 zeigt jeweils ein Beispiel für eine Dreiecksmenge, eine Trapezmenge und eine Gaussmenge.

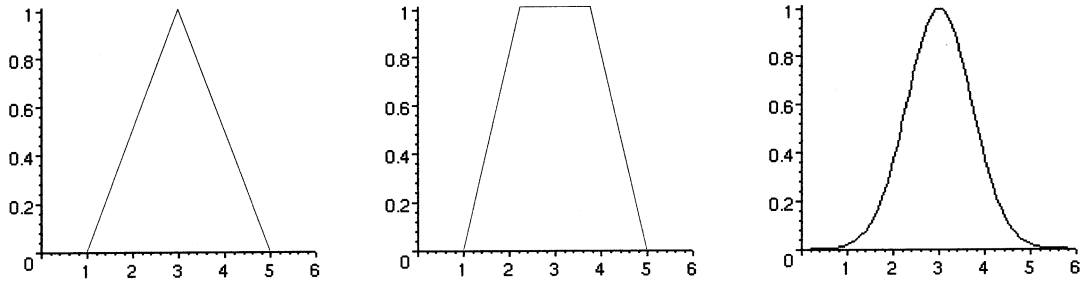


Abbildung 2.1: Fuzzy-Mengen

2.1.3 Operationen auf Fuzzy-Mengen

Die Standard-Operationen auf Fuzzy-Mengen werden in Analogie zu den Standard-Operationen auf klassischen Mengen definiert.

Definition 2.8 (Standard-Operationen auf Fuzzy-Mengen)
Seien $\tilde{A}$ und $\tilde{B}$ Fuzzy-Mengen über einer Grundmenge X mit Zugehörigkeitsgradfunktionen $\mu_{\tilde{A}}(x)$ und $\mu_{\tilde{B}}(x)$.

Vereinigung von $\tilde{A}$ und $\tilde{B}$

$$\mu_{\tilde{A} \cup \tilde{B}} = \max\{\mu_{\tilde{A}}(x), \mu_{\tilde{B}}(x)\}$$

Durchschnitt von $\tilde{A}$ und $\tilde{B}$

$$\mu_{\tilde{A}\cap\tilde{B}} = \min\{\mu_{\tilde{A}}(x), \mu_{\tilde{B}}(x)\}$$

Komplement von $\tilde{A}$

$$\mu_{\tilde{A}^c} = 1 - \mu_{\tilde{A}}(x)$$

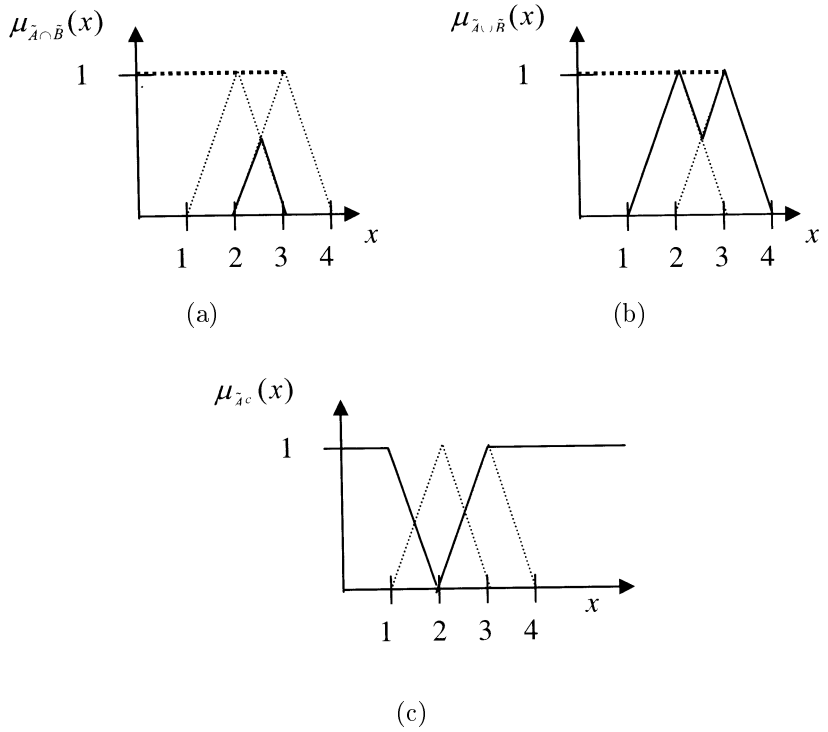


Abbildung 2.2: Standard-Operationen auf Fuzzy-Mengen

Abbildung 2.2 stellt die Standard-Operationen Durchschnitt (a), Vereinigung (b) und Komplement (c) für Dreiecksmengen grafisch dar.

In der Fuzzy-Logik ist das Ergebnis der Verknüpfung zweier Mengen nicht eindeutig festgelegt. Die folgenden Definitionen stellen die Mindestanforderungen für die Operatoren zusammen.

Für die *Fuzzy-Vereinigung* definiert man eine *s-Norm*.

Definition 2.9 (s-Norm)
Eine Funktion $s : [0,1] \times [0,1] \to [0,1]$ *heisst* s-Norm, *wenn sie die folgenden Eigenschaften besitzt:* $\forall a, b, c, d \in [0,1]$ *muss gelten:*

1. $s(0,0) = 0, \quad s(a,0) = s(0,a) = a$
   (Randbedingungen)

2. $s(a,b) \leq s(c,d), \quad falls \quad a \leq c \quad und \quad b \leq d$
   (Monotonie)
3. $s(a,b) = s(b,a)$
   (Kommutativität)
4. $s(a,s(b,c)) = s(s(a,b),c)$
   (Assoziativität)

zusätzlich werden in der Praxis oft folgende Bedingungen gestellt:

1. *s sei stetig.*
2. $s(s(a)) = a$
   (Idempotenz)

Für den *Fuzzy-Durchschnitt* definiert man analog eine *t-Norm.*

Definition 2.10 (t-Norm)
Eine Funktion $t : [0,1] \times [0,1] \rightarrow [0,1]$ *heisst* t-Norm*, wenn sie die folgenden Eigenschaften besitzt:* $\forall a, b, c, d \in [0,1]$ *muss gelten:*

1. $t(0,0) = 0, \quad t(a,1) = t(1,a) = a$
   (Randbedingungen)
2. *Es müssen die Bedingungen 2- 4 für die s-Norm in Definition 2.9 erfüllt sein.*

Die zusätzlichen Bedingungen, die in der Praxis häufig an eine s-Norm gestellt werden, werden auch an eine t-Norm gestellt.

Die Anforderungen an eine *Fuzzy-Komplementfunktion* zeigt die folgende Definition.

Definition 2.11 (Fuzzy-Komplementfunktion)
Eine Funktion $c : [0,1] \rightarrow [0,1]$ *heisst* Fuzzy-Komplementfunktion*, wenn sie die folgenden Eigenschaften besitzt:* $\forall a, b \in [0,1]$ *muss gelten:*

1. $c(0) = 1, \quad c(1) = 0$
   (Randbedingungen)
2. $a \leq b \Rightarrow c(a) \geq c(b))$.
   (monoton fallend)

Aufgrund der allgemeinen Anforderungen für Fuzzy-Durchschnitt, Fuzzy-Vereinigung und Fuzzy-Komplement lassen sich diese Operatoren auf vielfältige Weise definieren. Allgemein kann der Minimum-Operator als der „optimistischste“ Durchschnittsoperator-

und der Maximum-Operator als der „pessimistischste“ Vereinigungsoperator für Fuzzy-Mengen bezeichnet werden, denn für alle t-Normen t gilt

$$\min(a,b) \geq t(a,b)$$

und für alle s-Normen s gilt

$$\max(a,b) \ \leq s(a,b).$$

2.2 Fuzzy-Logik

Die Fuzzy-Logik (unscharfe Logik) erweitert das Zweiwertigkeitsprinzip der klassischen Logik, die sich auf die Wahrheitswerte 1 (*wahr*) und 0 (*falsch*) beschränkt, auf das Intervall [0,1]. Somit ist es möglich zu modellieren zu welchem Grad eine Aussage als wahr eingeschätzt wird.

Die Fuzzy-Logik ist die Grundlage für das sogenannte *unscharfe Schließen*. Unscharfes Schließen ist ein mathematischer Formalismus, der unscharfe Schlussfolgerungen aus unscharfen Prämissen nach formalen Rechenvorschriften ermöglicht. Die Technik des unscharfen Schließens ist wesentlicher Bestandteil der in Kapitel 2.3 dargestellten Arbeitsweise von Fuzzy-Controllern. Im folgenden wird das Prinzip des unscharfen Schließens näher erörtert.

Unscharfes Schließen

In Fuzzy-Controllern wird unscharfes Wissen in Form von Wenn-Dann-Regeln (IF-THEN-Regeln) gespeichert.

Definition 2.12 (IF-THEN-Regel)
Eine IF-THEN-Regel besteht aus einer Prämisse und einer Konklusion. Die Prämisse enthält eine Menge von Bedingungen und die Konklusion eine Menge von Folgerungen.

$$\underbrace{\textit{IF } X_1 \textit{ IS } \tilde{A}_1 \textit{ AND } X_2 \textit{ IS } \tilde{A}_2 \textit{ AND } \ldots \textit{ AND } X_n \textit{ IS } \tilde{A}_n}_{\textit{Prämisse}} \textit{ THEN } \underbrace{Y \textit{ IS } \tilde{B}}_{\textit{Konklusion}}$$

Die in den IF-THEN-Regeln auftretenden Variablen X_i, $i = 1,..,n$ werden *linguistische Variablen* genannt. Sie modellieren eine bei der Regelung betrachtete Grösse, wie z.B. eine Temperatur. Die Werte, die linguistische Variablen annehmen können, heissen *linguistische Terme*. Diese werden verbal erklärt und entsprechen meistens Adjektiven, wie kalt oder warm. Linguistische Terme werden mit Fuzzy-Mengen $\tilde{A}_j$, $j = 1,..,n$ modelliert.

Formal definiert man:

Definition 2.13 (Linguistische Variable)
Eine linguistische Variable ist ein Tupel (X, A_X, G, M_X) mit

X	: *Der symbolische Name der Variablen*
A_X	: *Menge der linguistischen Terme, linguistischen Werte*
G	: *Grundmenge über welcher die Variable definiert ist*
M_X	: *Funktion die jedem linguistischen Term eine Fuzzy-Menge über G zuordnet*

Beispiel 2.1 führt das Temperatur-Beispiel aus.

Beispiel 2.1 (Linguistische Variable Temperatur)
Eine linguistische Variable Temperatur (in $°C$) könnte folgendermaßen definiert werden:

$$(Temperatur, A_{Temperatur}, \mathbb{R}, M_{Temperatur})$$

mit

$$A_{Temperatur} = \{kalt, \quad warm, \quad heiss\}$$

und

$$\begin{aligned} M_{Temperatur}(kalt) &= \tilde{D}_{kalt}(-15, 0, 20) \\ M_{Temperatur}(warm) &= \tilde{D}_{warm}(15, 25, 35) \\ M_{Temperatur}(heiss) &= \tilde{D}_{heiss}(28, 40, 60) \end{aligned}$$

definieren. Hierbei wurden für die definierenden Fuzzy-Mengen Dreiecksmengen verwendet.

Mit dieser Spezifikation der Temperatur berechnet man über die Zugehörigkeitsgradfunktion der jeweiligen Dreiecksmenge, dass eine Temperatur von $31°C$ mit einer Gewichtung von 40 % als warm und 25 % als heiss eingeschätzt wird.

IF-THEN-Regeln werden durch *Fuzzy-Relationen* modelliert.

Definition 2.14 (Fuzzy-Relation)
Eine Fuzzy-Relation ist eine Verallgemeinerung der aus der klassischen Mengenlehre bekannten Relation.

Seien $X_1, .., X_n$ klassische Grundmengen und $X = X_1 \times \cdots \times X_n$ ihr kartesisches Produkt, dann definiert man die Fuzzy-Relation $\tilde{R}$ folgendermaßen:

$$\tilde{R} = \{((x_1, .., x_n), \mu_{\tilde{R}}(x_1, .., x_n)) \mid x_i \in X_i,\ i = 1, ..n\}$$

Der Zugehörigkeitsgrad $\mu_{\tilde{R}}(x_1, .., x_n)$ kann als Stärke der Fuzzy-Relation zwischen den Elementen des Tupels $(x_1, .., x_n)$ interpretiert werden.

Um die konkreten Werte für die Zugehörigkeitsgradfunktion $\mu_{\tilde{R}}$ der eine IF-THEN-Regel

$$\text{IF } X \text{ IS } \tilde{A} \text{ THEN } Y \text{ IS } \tilde{B}$$

modellierenden Fuzzy-Relation $\tilde{R}$ zu berechnen, verwendet man eine *Fuzzy-Implikationsfunktion*. Gebräuchlich sind z.B. die Fuzzy-Implikation nach Zadeh, die auch Zadehsche Maximumsregel genannt wird:

Definition 2.15 (Fuzzy-Implikation nach Zadeh)

$$\mu_{\tilde{R}}(x, y) = \max\left(\min(\mu_{\tilde{A}}(x), \mu_{\tilde{B}}(y)), 1 - \mu_{\tilde{A}}(x)\right)$$

und die Fuzzy-Implikation nach Mamdani:

Definition 2.16 (Fuzzy-Implikation nach Mamdani)

$$\mu_{\tilde{R}}(x, y) = \min(\mu_{\tilde{A}}(x), \mu_{\tilde{B}}(y)).$$

Das Berechnen eines unscharfen Schlusses (einer Fuzzy-Menge) aus einer unscharfen Prämisse und einer zugehörigen IF-THEN-Regel erfolgt nach dem Prinzip des *generalisierten Modus Ponens*. Dieser stellt eine Verallgemeinerung des aus der klassischen Logik bekannten Modus Ponens auf die Fuzzy-Logik dar.

Definition 2.17 (Generalisierter Modus Ponens)
Seien $\tilde{A}$ und $\tilde{A}'$ Fuzzy-Mengen über einer Grundmenge G_1 und $\tilde{B}$, $\tilde{B}'$ Fuzzy-Mengen über einer Grundmenge G_2. Dann nennt man die folgende Schlussfigur

$$\frac{X \text{ IS } \tilde{A}', \quad \text{IF } X \text{ IS } \tilde{A} \text{ THEN } Y \text{ IS } \tilde{B}}{Y \text{ IS } \tilde{B}'}$$

den verallgemeinerten Modus Ponens.

Die Zugehörigkeitsgradfunktion der Ergebnis-Fuzzy-Menge $\tilde{B}'$ berechnet sich nach folgender Vorschrift:

$$\mu_{\tilde{B}'}(y) = \sup_x \left(\mu_{\tilde{A}'}(x) \circ \mu_{\tilde{A}}(x) \circ \mu_{\tilde{B}}(y)\right).$$

Hierbei bezeichnet $\circ$ eine t-Norm wie sie in Definition 2.10 definiert wurde.

Je ähnlicher die Fuzzy-Menge $\tilde{A}'$ (die Beobachtung) der Prämisse der IF-THEN-Regel $\tilde{A}$ ist, desto ähnlicher ist die Ergebnis-Fuzzy-Menge $\tilde{B}'$ (die Schlussfolgerung) der Konklusion $\tilde{B}$ der Regel.

2.3 Fuzzy-Controller

Zur Steuerung dynamischer technischer Systeme werden technische Steuereinheiten eingesetzt, sogenannte Controller. Eine Möglichkeit einen Controller zu entwickeln, ist ein mathematisches Modell des zu steuernden Prozesses aufzustellen. Diese Vorgehensweise erfordert Experten, welche ein ausreichendes Wissen über das Problemfeld besitzen und dieses in ein mathematisches Modell umsetzen können. Oft ist das Modell idealisiert, da die Komplexität des Prozesses für eine genaue mathematische Modellierung zu hoch ist. Das Ergebnis ist häufig ein komplexes Differentialgleichungssystem, dessen Auswertung in der Regel sehr rechenintensiv ist.

Eine alternative Herangehensweise gründet sich auf die Tatsache, dass Menschen in der Lage sind komplizierte technische Systeme zu steuern, ohne das zugrunde liegende formale Modell zu kennen. Beispiele sind das Fahren eines Autos oder die Bedienung einer Waschmaschine. Bei der Entwickelung von Fuzzy-Controllern (Fuzzy-Reglern) modelliert man das Verhalten eines Menschen bei der Steuerung eines technischen Systems. Die nötigen Steuerungsregeln werden in der Form von IF-THEN-Regeln (s. Definition 2.12) erfasst. Dies kann durch Beobachtung und Befragung des Experten, der das System steuert erfolgen. Voraussetzung hierfür ist, das der Experte sein Wissen reflektieren und geeignet formulieren kann. Wird der Prozess jedoch nur intuitiv beherrscht, dann ist das Ermitteln der Regeln schwierig bis unmöglich. In Kapitel 4 werden Möglichkeiten dargestellt die Entwickelung eines Fuzzy-Controllers durch Einsatz Evolutionärer Algorithmen zu unterstützen.

Der nächste Abschnitt stellt die Entwurfsentscheidungen dar, die bei der Konstruktion eines Fuzzy-Controllers zu treffen sind. In der Praxis findet häufig der Mamdani-Controller Anwendung [Mamdani75], der im darauffolgenden Kapitel vorgestellt wird. Im Abschnitt 2.3.3 wird eine wichtige Variante des Mamdani-Controllers vorgestellt, der Sugeno-Controller [Sugeno85]. Daran schließt sich eine Darstellung der Vor- und Nachteile von Fuzzy-Controllern an. Den Abschluss dieses Kapitels bildet ein vollständiges Beispiel für einen Mamdani-Controller zur Steuerung des inversen Pendels.

2.3.1 Entwurfsphase

In der Entwurfsphase sind die folgenden Schritte zu machen:

- Aufstellen der Wissensbasis des Controllers, diese besteht aus Regelbasis und Datenbasis.
  - Aufstellen der Datenbasis des Controllers:
    Dazu sind die Eingabe- und Ausgaberäume der Daten des betrachteten Prozesses zu ermitteln. Für jeden Ein- und Ausgaberaum ist eine linguistische

Variable zu wählen. Für jede linguistische Variable sind geeignete linguistische Terme zu definieren. Jedem linguistischen Term ist eine beschreibende Fuzzymenge zuzuordnen. Die Fuzzy-Mengen einer linguistischen Variable sollen den Definitionsbereich für diese Grösse komplett abdecken, dies nennt man eine Partitionierung des Eingabe- bzw. Ausgaberaums.

  - Aufstellen der Regelbasis des Controllers:
    Es müssen geeignete Regeln aufgestellt werden, die eine ausreichende Steuerung des Systems garantieren.

- Wahl der Fuzzifizierungs- und Defuzzifizierungs-Strategie:
  Ein Fuzzy-Controller arbeitet mit Fuzzy-Mengen, die Daten des Prozesses sind jedoch reelle Werte. Daher müssen die Transformation der Eingabewerte in Fuzzy-Mengen (Fuzzifizierung) und die Transformation der berechneten Fuzzy-Mengen in reelle Werte (Defuzzifizierung) festgelegt werden.
- Wahl der in der Entscheidungslogik des Fuzzy-Controllers zu verwendenden Fuzzy-Implikationsfunktion (s. Definitionen 2.16 und 2.15).

2.3.2 Der Mamdani-Controller

Ein Mamdani-Controller ist eine Steuereinheit die reelle Eingabedaten in reelle Ausgabedaten transformiert. Er leistet somit eine Abbildung vom $\mathbb{R}^n$ in den $\mathbb{R}^m$, wobei $n, m \in \mathbb{N}$ sind.

Ein Mamdani-Controller besteht aus den folgenden Komponenten:

- Fuzzifizierer
- Wissensbasis
- Entscheidungslogik
- Defuzzifizierer

Abbildung 2.3 zeigt eine schematische Darstellung des Aufbaus eines Fuzzy-Controllers. Im folgenden werden die einzelnen Komponenten näher erläutert.

Fuzzifizierer

Der Fuzzifizierer wandelt reelle Eingaben in Fuzzy-Mengen um. Möglich sind z.B. die Umwandlung in eine Dreiecks-, Trapez-, oder Gaussmenge. Über die Fuzzifizierung können z.B. Messungenauigkeiten der Daten modelliert werden. In der Praxis wird häufig die *Singleton-Fuzzifizierung* angewendet.

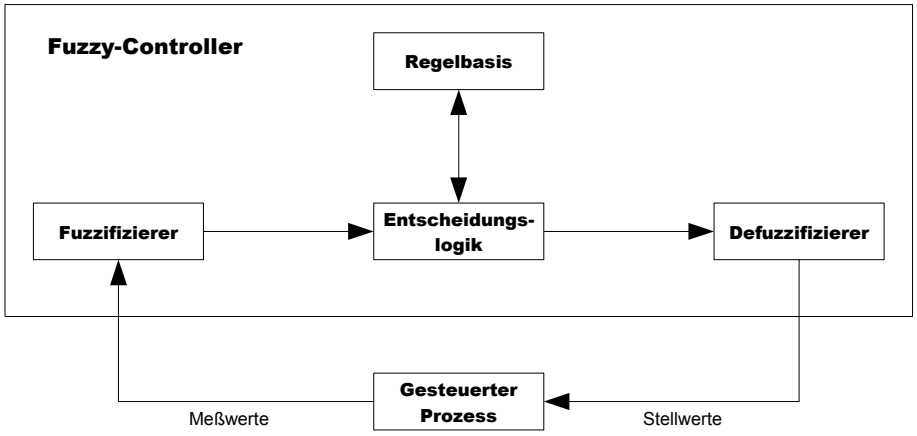


Abbildung 2.3: Aufbau eines Fuzzy-Controllers

Definition 2.18 (Singleton-Fuzzifizierung)
Sei X eine Grundmenge. Die Singleton-Fuzzy-Menge $SF(x')$ zu einem Element $x' \in X$ ist durch die Zugehörigkeitsgradfunktion

$$\mu_{SF(x')}(x) = \begin{cases} 1 & \textit{falls } x = x' \\ 0 & \textit{sonst} \end{cases}$$

gegeben.

Wissensbasis

Die Wissensbasis besteht aus Regelbasis und Datenbasis.
In der Regelbasis sind endlich viele IF-THEN-Regeln enthalten. In den meisten Implementierungen ist die übliche Form dieser IF-THEN-Regeln dergestalt, dass die Teilprämissen einer Prämisse ausschließlich über Konjunktionen verknüpft werden. Außerdem betrifft die Konklusion jeder Regel genau eine Ausgabedimension Die Konjunktionen der Teilprämissen werden über eine t-Norm (s. Definition 2.10) realisiert.
In der Datenbasis sind die linguistischen Bezeichner zusammen mit den sie definierenden Fuzzy-Mengen gespeichert.

Entscheidungslogik

Die Entscheidungslogik berechnet unter Verwendung der Regeln aus der Regelbasis aus den fuzzifizierten Eingabedaten nach dem Prinzip des verallgemeinerten Modus Ponens

(s. Definition 2.17) Ausgabe-Fuzzy-Mengen. Für jeden Ausgaberaum wird eine Ausgabe-Fuzzy-Menge erzeugt, wobei alle Regeln, die diesen Ausgaberaum in ihrer Konklusion haben, berücksichtigt werden.

Da im MFOS ein Singleton-Fuzzifizierer zum Einsatz kommt, wird die Erzeugung einer Ausgabe-Fuzzy-Menge für einen Controller mit Singleton-Fuzzifizierung erläutert:

1. Für jede Teilprämisse wird der Zugehörigkeitsgrad des zugehörigen Eingabewertes bestimmt. Die Verknüpfung dieser Werte mit einer t-Norm stellt den Erfüllungsgrad der Regel dar. Dies ist ein Maß dafür, wie gut die Regel für die durch die Eingabedaten beschriebene Situation passt.
2. Der Erfüllungsgrad der Prämisse wird mittels einer Fuzzy-Implikation (s. Definition 2.16) mit der Konklusion der Regel verknüpft. Da Mamdani hierfür den Minimum-Operator gewählt hat, entsteht die Ergebnis-Fuzzy-Menge durch „Abschneiden“ der Fuzzy-Menge der Konklusion in Höhe des Erfüllungsgrades der Prämisse der Regel.
3. Schlussendlich wird die Gesamt-Ausgabe-Fuzzymenge für einen Ausgaberaum durch die Vereinigung der einzelnen Ausgabemengen aller Regeln, die diesen Ausgaberaum in ihrer Konklusion haben, gebildet. Hierzu nimmt man eine s-Norm (s. Definition 2.9). Im Mamdani-Controller wird hierfür der Maximum-Operator verwendet.

Defuzzifizierer

Der Defuzzifizierer berechnet aus den Ausgabe-Fuzzy-Mengen reelle Ausgabewerte. Diese Umrechnung bedeutet einen erheblichen Informationsverlust und hat entscheidenden Einfluss auf das Verhalten des Controllers. Drei wichtige Verfahren sind die Maximum-Methode (MAX), die Mittelwert-der-Maxima-Methode (MOM) und die Schwerpunkt-Methode(COG).

MAX Es wird ein Wert des Definitionsbereiches gewählt für den die Zugehörigkeitsgradfunktion maximal ist. Problematisch ist, dass dies für mehr als einen Wert zutreffen kann. Um Nichtdeterminismus des Reglers zu vermeiden, muss eindeutig festgelegt werden, wie in einer solchen Situation vorzugehen ist. Eine Möglichkeit ist, dass immer der am weitesten links liegende solche Wert ausgewählt wird.

MOM Bei der Mittelwert-der-Maxima-Methode wird der Mittelwert der Maximalstellen als Ausgabewert genommen.

COG Bei der Schwerpunktmethode nimmt man den x-Wert, der zum Schwerpunkt der Fläche, die von Zugehörigkeitsgradfunktion und x-Achse begrenzt wird, gehört. Die Schwerpunktmethode ist die in der Praxis gebräuchlichste Methode und führt meist, aber nicht garantiert, zu guten Ergebnissen.

Abbildung 2.4 verdeutlicht die verschiedenen Defuzzifizierungsmethoden.

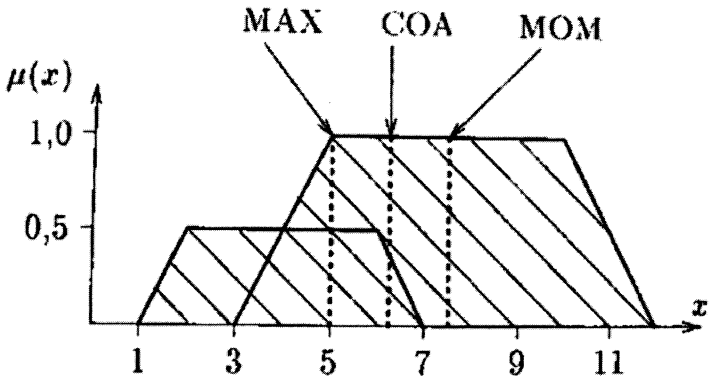


Abbildung 2.4: Verschiedene Defuzzifizierungsmethoden

Eine Variante des Mamdani-Controllers ist der Sugeno-Controller, der ebenfalls häufig eingesetzt wird.

2.3.3 Der Sugeno-Controller

Die einzige Änderung gegenüber dem Mamdani-Controller ist, dass keine Defuzzifizierung notwendig ist. Der Sugeno-Controller berechnet direkt einen reellen Ausgabewert. Hierzu werden IF-THEN-Regeln verwendet, die in der Konklusion keine Fuzzy-Mengen verwenden, sondern reelle Funktionen der Eingabewerte.

Für n Eingaberäume und 1 Ausgaberaum hat man Fuzzy-Regeln der Form:

$$\text{IF } X_1 \text{ IS } \tilde{A}_1 \text{ AND } X_2 \text{ IS } \tilde{A}_2 \text{ AND } \ldots \text{ AND } X_n \text{ IS } \tilde{A}_n \text{ THEN } y = f(x_1, .., x_n)$$

Meistens handelt es sich bei den verwendeten reellen Ausgabefunktionen um lineare Kombinationen der Eingabewerte:

$$f(x_1, .., x_n) = \sum_{i=1}^{n} a_i * x_i + a_0 \quad a_i \in \mathbb{R}, i = 1, .., n.$$

Es werden die Erfüllungsgrade exakt wie beim Mamdani-Controller berechnet. Der Gesamtausgabewert y_j für einen Ausgaberaum Y_j ergibt sich wie folgt: Für jede Regel R_i, die in ihrer Konklusion den Ausgaberaum Y_j enthält, wird der Erfüllungsgrad e_i und der

Ausgabewert $A_i = f_i(x_1, .., x_n)$ berechnet. Dann wird y_j als das mit den Erfüllungsgraden gewichtete Mittel der Ausgabewerte berechnet:

$$y_j = \frac{\sum_i e_i * A_i}{\sum_i e_i}$$

Vorteil des Sugeno-Controllers ist die Umgehung von möglichen Problemen bei der Defuzzifizierung. Nachteilig ist das die reellen Ausgabefunktionen schwierig zu ermitteln sein können und für jede Änderung oder Neuentwicklung eines Controllers erneut aufgestellt werden müssen.

2.3.4 Eigenschaften von Fuzzy-Controllern

Vorteile

- Fuzzy-Controller sind interpretierbare Systeme, d.h. ihr funktionales Verhalten kann relativ einfach nachvollzogen werden. Vorhandenes Wissen über den zu steuernden Prozess kann einfach in den Konstruktionsprozess eingebracht werden. Das Anpassen des Controllers bei Änderung des zu steuernden Systems ist meist einfach möglich.
- Fuzzy-Controller sind i.A. robuster gegen Störungen in den Daten als klassische Controller.
- Die Entwickelung eines Fuzzy-Controllers ist häufig vergleichsweise einfach. Es ist kein mathematisches Prozessmodell nötig.
- Die Realisierung des Fuzzy-Controllers in Hardware ist aufgrund der einfachen Berechnungen des Controllers schnell und günstig.

Nachteile

- Für manche Systeme kann das Aufstellen einer vollständigen Regelbasis schwierig sein, da der Bediener des Systems sein Wissen nicht vollständig formulieren kann. Für eine Generierung der Regeln aus Beispieldaten sind möglicherweise nicht hinreichend viele Beispieldaten vorhanden, um den gesamten Definitionsbereich der Eingaberäume geeignet zu repräsentieren.
- Fuzzy-Controller sind nicht-adaptive Systeme. Sie besitzen keine Fähigkeiten sich selbstständig an Veränderungen des zu steuernden Systems anzupassen. Auch jede kleinste Änderung muss „von Hand“ von einem Experten durchgeführt werden.

2.4 Fuzzy-Steuerung des inversen Pendels

Im folgenden wird ein Beispiel für einen Mamdani-Controller zur Steuerung des inversen Pendels vorgestellt. Die Darstellung beruht auf [Lippe06] und [Nauck96].

Die Steuerung des inversen Pendels, auch Stabbalancier-Problem genannt, besteht in der Balancierung eines auf dem Kopf stehenden Pendels. Ziel ist es das Pendel möglichst senkrecht zu halten. Das Pendel ist auf einem Wagen montiert. Wagen und Pendel können sich in nur einer- und derselben Ebene bewegen. Die Kraft F, die für die Balancierbewegungen des Wagens nötig ist, hängt von dem Winkel θ und und der Winkelgeschwindigkeit $\dot{\theta}$ ab (vgl. Abbildung 2.5).

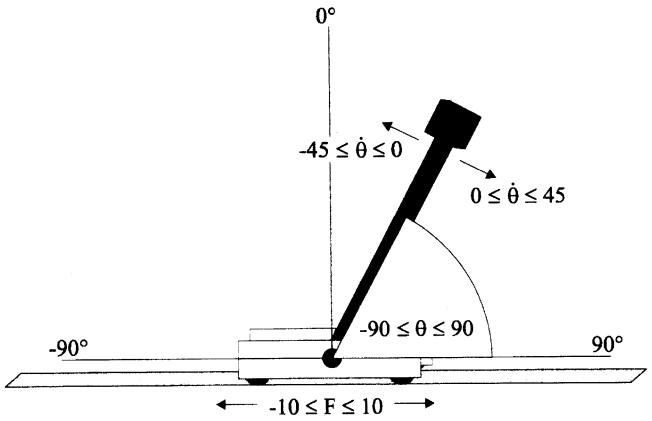


Abbildung 2.5: Inverses Pendel

Beim inversen Pendel hat man die Eingaberäume $X_1 = [-90, 90]$ für den Winkel in Grad und $X_2 = [-45, 45]$ für die Winkelgeschwindigkeit pro Sekunde. Für die Stellgrösse Kraft in Newton definieren wir den Ausgaberaum $Y = [-10, 10]$. Ist $F < 0$ so wird der Wagen nach links bewegt, ist $F > 0$, so wird er nach rechts bewegt. Analog hierzu ist die Interpretation von negativen und positiven Werten für die Variablen Winkel und Winkelgeschwindigkeit.

Es werden die folgenden linguistischen Variablen festgelegt:

1. $X_1 \sim$ Winkel
2. $X_2 \sim$ Winkelgeschwindigkeit
3. $Y \sim$ Kraft

Weitere Einflussgrössen wie Reibungswiderstand oder eine Begrenzung des dem Wagen

zur Verfügung stehenden Raums zum nach links oder rechts fahren werden zur Vereinfachung nicht beachtet.

Die Eingaberäume und der Ausgaberaum müssen geeignet partitioniert werden. Eine mögliche Partitionierung für die Messgröße Winkel zeigt Abbildung 2.6. Die Räume

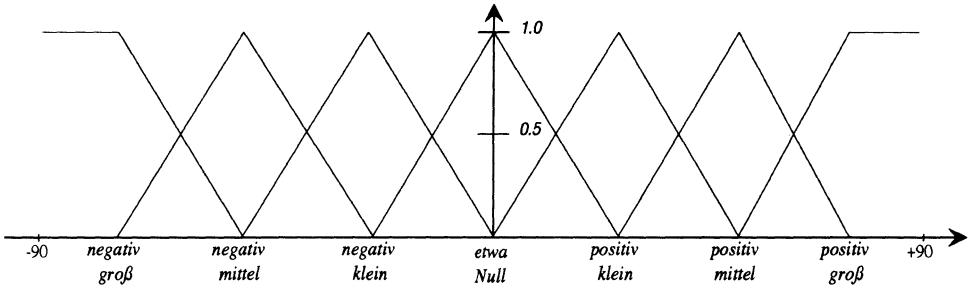


Abbildung 2.6: Partitionierung für die Grösse Winkel

für die Winkelgeschwindigkeit und die Kraft werden nach dem gleichen Schema, wie in Abbildung 2.6 partitioniert.

Eine auf diesen Partitionierungen basierende Regelbasis zeigt Abbildung 2.7.

$\dot{\theta}$ \ θ	ng	nm	nk	en	pk	pm	pg
ng			pk	pg			
nm				pm			
nk	nm		nk	pk			
en	ng	nm	nk	en	pk	pm	pg
pk				nk	pk		pm
pm				nm			
pg				ng	nk		

Abbildung 2.7: Regelbasis für das Inverse Pendel

Dabei gelten die Bezeichnungen:

1) ng = negativ gross
2) nm = negativ mittel
3) nk = negativ klein
4) en = etwa Null
5) pk = positiv klein
6) pm = positiv mittel
7) pg = positiv gross

Die Tabelle muss nicht vollständig ausgefüllt sein, um für alle nötigen Situationen ein hinreichendes Controllerverhalten zu gewährleisten, da sich die Bereiche, in denen die Regeln anwendbar sind, teilweise überlappen.

In Abbildung 2.8 wird die Auswertung zweier Fuzzy-Regeln grafisch dargestellt. Hierbei wird ein Singleton-Fuzzifizier (s. Definition 2.18) verwendet.

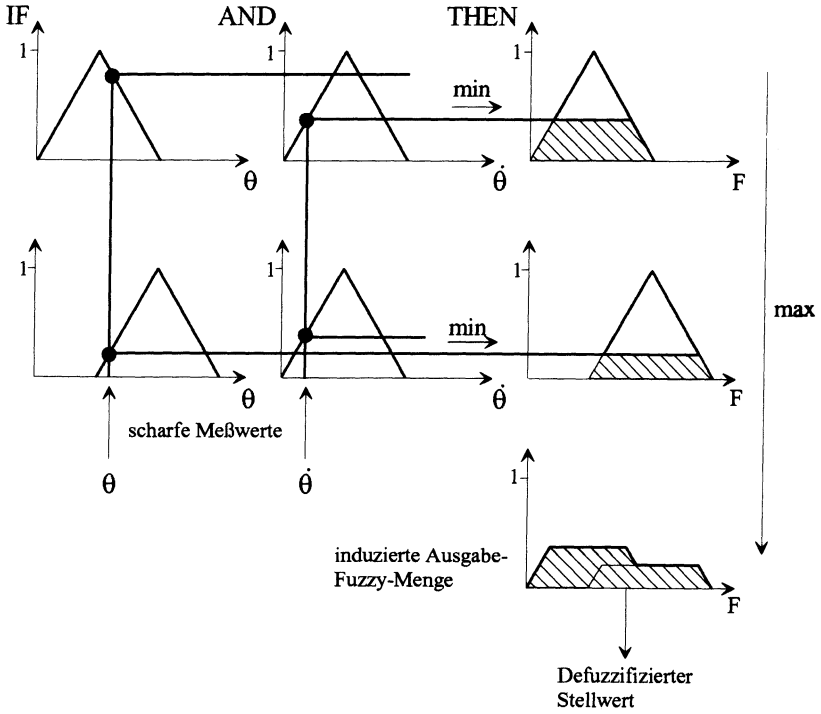


Abbildung 2.8: Auswertung Zweier Fuzzy Regeln

3 Evolutionäre Algorithmen

Dieses Kapitel führt in das Gebiet der Evolutionären Algorithmen ein. Als Einstieg in die Thematik war [Goldberg89] geeignet. Die Darstellung stützt sich wesentlich auf [Lippe06] und [Weicker02]. Ergänzend wurden [Gerdes04] und [Pohlheim99] verwendet.

3.1 Motivation

Evolutionäre Algorithmen sind eine Technik Optimierungsprobleme zu bearbeiten. Sie ahmen den von Charles Darwin erkannten Evolutionsprozess der Natur nach. Dieser Prozess beruht auf der Kombination von einerseits Variation der Individuen durch Modifizierung ihres Erbgutes und andererseits Selektion nach ihrer Überlebensfähigkeit. Der Evolutionsprozess ist somit eine Suche im Raum der möglichen Erbanlagen.

Der folgende Abschnitt behandelt in einer vereinfachten Sichtweise die biologischen Grundlagen, die hilfreich für das Verständnis der nachfolgenden Erörterungen über Evolutionäre Algorithmen sind. Er soll die Motivation ihrer Verwendung und ihrer Terminologie verdeutlichen. Im darauffolgenden Abschnitt wird eine Übersicht über die verschiedenen Varianten innerhalb der Evolutionären Algorithmen gegeben. Daran schließen sich eine Betrachtung der Prinzipien Evolutionärer Algorithmen- und der für diese Arbeit wichtigen Genetischen Algorithmen an. Im Anschluss daran werden einige Eigenschaften von Evolutionären Algorithmen zusammengestellt.

3.2 Biologisches Vorbild

Der natürliche Evolutionsprozess basiert auf der Modifikation der Erbinformation. Diese ist im Zellkern (Nucleus) auf den Chromosomen gespeichert. Sie enthält die „Bauvorschrift“ für den Organismus und bestimmt seine Eigenschaften. Die Chromosome bestehen aus Nukleinsäuren und Proteinen. Die wichtigste Nukelinsäure ist die Desoxyribonukleinsäure (DNA). Diese besteht aus vier verschiedenen Basen: Adenin (A), Guanin (G), Cytosin (C) und Thymin (T). Die genetische Information wird durch Sequenzen dieser Basen codiert.

Ein Gen ist ein Abschnitt auf einem Chromosom. Es stellt die kleinste Einheit der genetischen Information dar. Die Gesamtheit aller Gene eines Organismus wird als Genom oder als Genotyp bezeichnet. Der Genotyp eines Organismus bestimmt sein Erscheinungsbild, seinen Phänotyp. Ein einzelnes Gen kann eine Anzahl verschiedener Ausprägungen (Werte) annehmen. Zum Beispiel kann beim Menschen ein für die Haarfarbe verantwortliches Gen eine Ausprägung besitzen, die für schwarze Haare und eine andere, die für blonde Haare sorgt. Der Wert eines Gens wird als Allel bezeichnet.

Ein Organismus wächst durch Zellteilung. Hierbei kommt es zur *Rekombination* von vorhandenen Chromosomen zu neuen Chromosomen durch *Crossover* (Kreuzung). Dabei entsteht keine neue genetische Information, sondern es wird die genetische Information der Eltern-Chromosomen in den Kinder-Chromosomen vermischt. Neue genetische Information entsteht durch *Mutation* des Erbgutes. Aus der Population der Individuen wählt die Evolution die Individuen mit höherer Überlebensfähigkeit für die Erzeugung von Nachkommen aus, dies wird als *Selektion* bezeichnet.

Das Zusammenwirken der drei Grundprinzipien Selektion, Mutation und Rekombination schafft die enorme Leistungsfähigkeit der Evolution. Der Suchmechanismus der Evolution ist hochgradig parallel, da mit einer Menge von meist sehr vielen Individuen gearbeitet wird. Seine Effektivität entsteht durch die Kombination von gerichtetem und ungerichtetem Vorgehen. Die Selektion ist ein streng gerichteter Prozess. Bessere Individuen werden bevorzugt. Die Mutation hingegen ist ungerichtet. Sie erzeugt zufällige Alternativen nach dem Prinzip try-and-error. Dies wirkt der Stagnation des Optimierungsprozesses in einem suboptimalen Zustand entgegen. Die Rekombination enthält gerichtete und ungerichtete Anteile. Einerseits werden die Stellen an denen der Crossover stattfindet zufällig bestimmt, andererseits gelten für die Rekombination statistische Gesetzmäßigkeiten, wie z.B. die Mendelschen Gesetze.

3.3 Varianten Evolutionärer Algorithmen

Bei der Umsetzung von Prinzipien der natürlichen Evolution in technische Systeme entstanden vier wichtige Forschungsrichtungen:

1. Genetische Algorithmen [Holland75]
2. Evolutionsstrategien [Rechenberg73]
3. Evolutionäre Programmierung [Fogel65]
4. Genetische Programmierung [Koza92]

Die verschiedenen Varianten simulieren alle den natürlichen Evolutionsprozess. Sie unterscheiden sich hinsichtlich der Art wie Informationen codiert werden und der Gewichtung

des Einsatzes von Rekombination und Mutation. Dies rührt daher, dass die Varianten in verschiedenen Anwendungsbereichen entwickelt wurden. Die Genetische Programmierung beschäftigt sich mit der automatischen Generierung von Computerprogrammen. Die Evolutionsstrategien werden vorwiegend eingesetzt, um Optimierungsprobleme bei technischen Konstruktionsprozessen zu unterstützen. Der Bereich der Genetischen Algorithmen entstand bei der Modellierung von adaptiven Systemen. Evolutionäre Programmierung wurde zur Vorhersage von Zeitreihen verwendet.

3.4 Grundschema Evolutionärer Algorithmen

Die Terminologie der Elemente eines Evolutionären Algorithmus ist dem biologischen Vorbild entlehnt: Die von Optimierungsproblemen bekannte Zielfunktion wird Fitnessfunktion genannt. Eine potentielle Lösung wird als Individuum bezeichnet. Eine Population ist eine Menge von Individuen. Als Generation bezeichnet man eine Population zu einem bestimmten Zeitpunkt. Die in der Phase der Reproduktion aktiven Individuen nennt man Eltern, die durch genetische Operatoren (Mutation, Crossover) erzeugten Individuen heissen Nachkommen oder Kinder. Die mit der Fitnessfunktion berechnete Fitness eines Individuums entspricht im biologischen Vorbild der Überlebensfähigkeit des Individuums.

Für die Konstruktion eines Evolutionären Algorithmus zur Bearbeitung eines gegebenen Optimierungsproblems sind die folgenden Schritte zu machen:

3.4.1 Wahl einer geeigneten Codierung der Individuen

Es muss eine geeignete Codierung der Individuen gefunden werden. Eine Möglichkeit ist die Codierung durch Zeichenketten (Strings). Im Bereich der Genetischen Algorithmen wurden ursprünglich vorwiegend Strings aus binären Zahlen verwendet, bei den Evolutionsstrategien Strings aus reellen Zahlen. In diesen Fällen besteht ein Individuum (Chromosom) aus der Aneinanderreihung von codierten Parametern des Problems. In der Genetischen Programmierung werden z.B. Baumstrukturen (Syntaxbäume) verwendet. Im Bereich der Evolutionären Algorithmen besteht eine mögliche Codierung in der Verwendung von endlichen Automaten.

3.4.2 Aufstellen einer geeigneten Fitnessfunktion

Die Fitnessfunktion ist in der Regel eine Funktion aller codierten Parameter des Optimierungsproblems. Wenn die Fitness eines Individuums nicht durch Angabe einer Funktion berechnet werden kann, verwendet man Simulationen zur Auswertung der Fitness.

3.4.3 Definition einer Startpopulation und eines Abbruchkriteriums

Es ist eine geeignete Startpopulation zu generieren. Diese wird häufig zufällig bestimmt. Existiert Vorwissen über das Optimierungsproblem lässt es sich in die Generierung des Startpopulation einbringen. Es ist jedoch darauf zu achten die Verschiedenheit (Diversität) der Startpopulation nicht zu gering zu wählen, da sonst eine vorzeitige Konvergenz zu einem suboptimalen Zustand wahrscheinlich ist. Vor dem Starten des Evolutionären Algorithmus ist es nötig ein geeignetes Abbruchkriterium zu definieren. Denn ein evolutionärer Algorithmus ist ein fortwährender Kreislauf, daher spricht man auch vom evolutionären Zyklus. Häufig eingesetzte Abbruchkriterien sind eine bestimmte erreichte Güte des besten Individuums, eine maximale Anzahl von zu entwickelnden Generationen, der Ablauf einer festgelegten Zeitspanne oder das Stagnieren der Gesamtfitness über eine bestimmte Anzahl von Generationen.

3.4.4 Auswahl der Selektionsmethode und der genetischen Operatoren

Die Selektion eines Individuums basiert ausschliesslich auf seinem Fitnesswert. Dieser zeigt die Güte des Individuums im Verhältnis zu allen anderen Individuen der Population. Die selektierten Individuen werden mittels der genetischen Operatoren modifiziert. Diese müssen einerseits zu der gewählten Codierung passen, z.B. ist darauf zu achten, dass keine „ungültigen“ Individuen erzeugt werden, und sie müssen andererseits problemangemessen definiert werden.

Abbildung 3.1 stellt das Grundschema eines Evolutionären Algorithmus graphisch dar. Der folgende Abschnitt erläutert die für diese Arbeit maßgebende Technik der Genetischen Algorithmen näher.

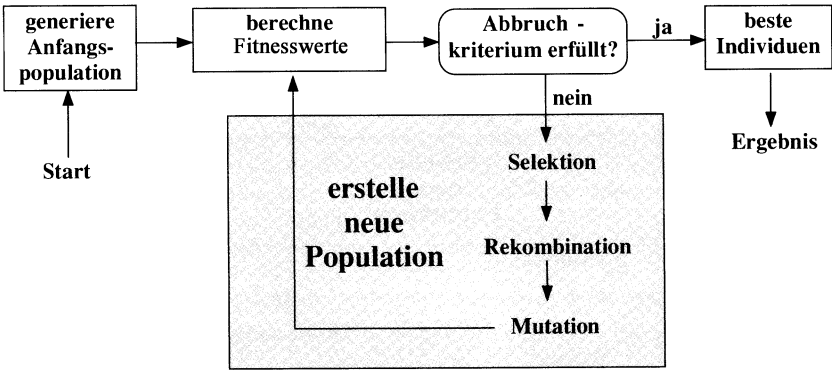


Abbildung 3.1: Grundschema Evolutionärer Algorithmen

3.5 Genetische Algorithmen

Die Genetischen Algorithmen wurden von John Holland [Holland75] begründet. Er verwendete sie, um ein allgemeines Modell für adaptive Systeme zu entwickeln.

Der sogenannte kanonische Genetische Algorithmus, ist in Pseudo-Code in Abbildung 3.2 dargestellt.

```
Programm kGA:
    begin
      generiere StartPopulation P;
      berechne Fitness der Individuen in P;
      while (Abbruchkriterium nicht erfüllt) do
        S := selektiere Individuen aus P anhand ihrer Fitness;
        T := Kinder der Rekombination der Individuen aus S;
        T := mutiere Individuen in T;
        P := P + T;
        berechne Fitness der Individuen von P;
      end while
    end.
```

Abbildung 3.2: Pseudo-Code des kanonischen genetischen Algorithmus (kgA)

Das Folgende stellt weitere Ausprägungen der Elemente eines Genetischen Algorithmus vor.

3.5.1 Codierung

Genetische Algorithmen verwenden meist eine Codierung der Individuen durch Strings. Diese wurden ursprünglich über dem binären Alphabet definiert, waren also Zeichenketten aus Nullen und Einsen. Eine andere vermehrt eingesetzte Möglichkeit ist eine reelle Codierung zu verwenden. Es gibt theoretische Untersuchungen, die zeigen, dass bei reeller Codierung bessere Ergebnisse zu erwarten sind (vgl. [Gerdes04], S. 64). Weiterhin ist die Fitnessoberfläche durch Verwendung einer reellen Codierung bei Problemen mit reellen Parametern besser an das Problem angepasst. Die Verwendung einer reellen Codierung erfordert darauf zugeschnittene genetische Operatoren. Beispiele finden sich in den Abschnitten 3.5.4 und 3.5.5.

Gene werden durch Abschnitte auf dem Chromosom dargestellt. Die Semantik eines Gens kann positionsabhängig oder positionsunabhängig sein. Bei der positionsabhängigen Semantik ist der Ort des Gens auf dem Chromosom und die Länge des Teilstrings, der das Gen codiert, festgelegt. Die Dekodierung des Genotyps in den Phänotyp basiert auf dieser festen Struktur.

Beispiel 3.1 (Positionsabhängige Codierung)
Die Individuen einer Population seien durch binäre Strings, die aus 5 Bit bestehen, codiert. Dabei codiere jedes Bit ein Gen. Ein mögliches Individuum x könnte z.B. die folgende Form

$$x = 10011$$

haben. Die Dekodierung in den zugehörigen Phänotyp verwendet die Position jedes Bit und dessen Wert zur Bestimmung der phänotypischen Eigenschaften des Individuums.

Bei einer positionsunabhängigen Semantik wird jedem Gen ein Symbol zugeordnet, welches es eindeutig klassifiziert. Eine Form sind die *Messy Genetischen Algorithmen* (s. Abschnitt 3.6). Bei diesen ist auch die Vorgabe einer festen Chromosomlänge aufgehoben.

3.5.2 Fitnessfunktion

Die Fitnessfunktion fit stellt die Verbindung zum Optimierungsproblem dar und muss für jedes Problem spezifisch entwickelt werden. Die Fitness $\text{fit}(x)$ eines Individuums beschreibt die Güte der durch dieses Individuum repräsentierten Lösung im Verhältnis zu allen anderen Individuen der Population. Die Fitness basiert auf dem Zielfunktionswert $f(x)$ des Individuums.

Als Fitnessfunktion wird z.B. die sogenannte proportionale Fitness-Funktion

$$\text{fit}(x) = \frac{f(x)}{\sum_{x \in P} f(x)}$$

verwendet, wobei P die Population der Individuen bezeichnet.

Nachteile der proportionalen Fitnessbewertung ist eine zu starke Bevorzugung weniger sehr fitter Individuen einerseits und eine geringe Differenzierung bei etwa gleich fitten Individuen andererseits. Diese Nachteile werden durch die sogenannte reihenfolgebasierte Fitnessbewertung vermieden. Hierbei basiert die Fitness eines Individuums nicht direkt auf dem Zielfunktionswert für dieses Individuum. Stattdessen wird eine Reihenfolge der Individuen gemäß ihres Zielfunktionswertes hergestellt und der Fitnesswert eines Individuums basiert auf der Position des Individuums in dieser Reihenfolge. Beispiele für reihenfolgebasierte Fitnessfunktionen finden sich in [Weicker02].

3.5.3 Selektion

Der Begriff Selektion bezeichnet die Auswahlvorgänge beim Genetischen Algorithmus. Damit der Genetische Algorithmus eine gute Effektivität zeigt, muss die Selektion einen geeigneten Kompromiss zwischen der Erforschung neuer Gebiete des Suchraums und der Nutzung bereits entdeckter guter Gebiete darstellen. Je nach konkreter Ausprägung des Genetischen Algorithmus findet Selektion z.B. bei der Auswahl von

- Individuen zur Erzeugung von Nachkommen für eine neue Population.
- Individuen, die mutiert werden sollen.
- Individuen, die in eine Population eingefügt werden sollen.

statt.

Im folgenden werden einige bekannte Selektionsverfahren vorgestellt. Details finden sich in [Lippe06] und [Pohlheim99].

Fitnessproportionale Selektionsverfahren

Fitnessproportionale Selektionsverfahren wählen Individuen mit einer Wahrscheinlichkeit aus, die proportional zu ihrem Fitnesswert ist.

Roulette-Selektion

Bei der Roulette-Selektion teilt man eine Rouletteschеibe in soviele Segmente auf, wie Individuen vorhanden sind. Die Fläche eines Segments ist proportional zur Fitness des zugehörigen Individuums.

Jetzt lässt man die Roulettekugel rollen, lässt also den Zufall entscheiden, welches Individuum ausgewählt wird. Dies wiederholt man so oft, wie man Individuen auswählen möchte. Nachteilig bei diesem Verfahren ist, dass die Häufigkeit des Auftretens eines Individuums in der Auswahl stark von der durch die Fitness des Individuums erwarteten Häufigkeit abweichen kann. Dies bezeichnet man als *spread*. Auch besteht die Möglichkeit, daß nur realtiv schlechte Individuen ausgewählt werden, auch wenn diese Möglichkeit relativ unwahrscheinlich ist.

Stochastic Universal Sampling (SUS)

Eine Variante der Roulette-Selektion ist das *Stochastic Universal Sampling (SUS)*.

SUS löst das Problem des hohen spread bei der Roulette-Selektion. Der Unterschied von SUS zur Roulette-Selektion ist, dass die Rouletteschеibe nur einmal gedreht wird. Um die Scheibe herum befinden sich soviele Zeiger in regelmässigem Abstand, wie Individuen ausgewählt werden sollen. Die Anzahl der Zeiger, die auf den zu einem Individuum gehörenden Bereich zeigen, bestimmt wie häufig es ausgewählt wird.

Reihenfolgebasierte Selektionsverfahren

Bei reihenfolgebasierter Selektion werden die Individuen zunächst entsprechend ihren Fitnesswerten absteigend angeordnet. Diese Reihenfolge wird auch als Rangliste bezeichnet. Die Selektionsverfahren arbeiten dann auf dieser Rangliste und nicht direkt mit den absoluten Fitnesswerten der Individuen.

Exponentielle Selektion

Die exponentielle Selektion hat einen Parameter p, welcher die Wahrscheinlichkeit für die Auswahl eines Individuums bestimmt.

Jetzt wird die Rangliste sukzessive abgeschritten. Dabei bekommt das aktuelle Individuum die Chance mit Wahrscheinlichkeit p ausgewählt zu werden. Wird es nicht ausge-

wählt, kommt das nächste Individuum daran.

Die Wahrscheinlichkeit für die Auswahl des k-besten Individuums aus einer Population von n Individuen ist durch

$$p\,(1-p)^{k-1} \quad 1 \leq k \leq n$$

gegeben.

Turnier-Selektion

Die Turnier-Selektion hat einen Parameter k, der die Anzahl der Individuen, die an einem Turnier teilnehmen festlegt.

Sollen q Individuen ausgewählt werden, so werden q Turniere veranstaltet. Für jedes Turnier werden k Individuen gleichverteilt aus der Population ausgewählt.

Das fitteste Individuum eines Turniers gewinnt das Turnier, wird also ausgewählt.

Exponentielle Turnier-Selektion

Die exponentielle Turnier-Selektion kombiniert die exponentielle Selektion mit der Turnier-Selektion. Es wird analog zur Turnier-Selektion verfahren, aber der Gewinner eines Turniers wird per exponentieller Selektion bestimmt.

Truncation-Selektion

Die Truncation-Selektion hat einen Parameter s, der einen Prozentsatz darstellt. Es werden die s Prozent besten Individuen ausgewählt. Die Population schrumpft in jeder Generation um 100 - s Prozent.

3.5.4 Crossover

Beim Crossover entstehen aus vorhandenen Chromosomen durch Austausch von Genen neue Chromosome. Die Art eines Crossover-Verfahrens hängt direkt von der verwendeten Codierungsform ab.

Grundlegend ist der 1-Punkt-Crossover. Bei diesem wird eine Trennstelle per Zufallszahl ermittelt. Die Teile hinter der Trennstelle werden ausgetauscht. Es entstehen also aus zwei Eltern-Chromosomen zwei neue Kinder-Chromosome.

Definition 3.1 (1-Punkt-Crossover, binäre Codierung)

$$\begin{array}{ccc} \underset{Elter_1}{1111 \mid \underline{0000}} & & \underset{Kind_1}{1111 \mid \underline{1111}} \\ & \longrightarrow & \\ \underset{Elter_2}{0000 \mid \underline{1111}} & & \underset{Kind_2}{0000 \mid \underline{0000}} \end{array}$$

Weiter existiert der 2-Punkt-Crossover. Bei diesem werden zwei Trennstellen per Zufallszahl bestimmt und der dazwischenliegende Teil wird ausgetauscht. Der allgemeine n-Punkt-Crossover funktioniert so, dass bei Durchnummerieren der entstehenden Teile, alle Teile mit gerader Nummer ausgetauscht werden.

Bei reeller Codierung wird ebenfalls n-Punkt-Crossover verwendet. Sie ermöglicht jedoch eine Vielzahl weiterer möglicher Crossover-Methoden. Eine davon ist der sogenannte *Intermediäre Crossover* (vgl. [Lippe06], Kapitel 4.5.3).

Definition 3.2 (Intermediärer Crossover für reelle Codierung)
Seien die folgenden Eltern-Chromosomen gegeben:

$$C_1 = (x_{11}, \ldots, x_{1n}) \text{ und } C_2 = (x_{21}, \ldots, x_{2n})$$

dann berechnet sich das Kind-Chromosom daraus so:

$$C_3 = (x_{31}, \ldots, x_{3n}) \text{ mit } x_{3k} = \frac{1}{2} \cdot x_{1k} + \frac{1}{2} \cdot x_{2k} \quad k = 1, \ldots, n$$

Weitere Crossoververfahren finden sich z.B. in [Lippe06].

Bemerkung 3.1
Bei einem Optimierungsproblem sind häufig starke Korrelationen zwischen den Paramtern des Problems vorhanden. Auf das codierende Chromosom übertragen bedeutet das, dass verschiedene Gene stark voneinander abhängig sind. Im Allgemeinen wird die Gesamtfitness durch das Trennen von stark voneinander abhängigen Genen reduziert. Es ist daher vorteilhaft den Crossoveroperator so zu gestalten, daß die Wahrscheinlichkeit des Trennens solcher stark korrelierter Gene gering ist (vgl. [Gerdes04], S.196 f).

3.5.5 Mutation

Die Häufigkeit von Mutationen ist im Vergleich zur Rekombination bei genetischen Algorithmen sehr selten. Sie garantiert die Erreichbarkeit aller Punkte des Suchraums.

Mutationen können jedoch gute Allelfolgen zerstören und so eine bereits erreichte gute Fitness reduzieren. Ebenso wie die Crossover-Verfahren hängen auch die Mutations-Verfahren direkt von der verwendeten Codierung ab. Bei binärer Codierung kann z.B. so vorgegangen werden: es wird eine Stelle auf dem Chromosom stochastisch bestimmt und sein Wert invertiert.

Beispiel 3.2 (Mutation bei binärer Codierung)

$$11\underline{0}1 \longrightarrow 11\underline{1}1$$

Für eine reelle Codierung hat man z.B. folgenden Mutationsoperator:

Beispiel 3.3 (Random Mutation bei reeller Codierung)
Sei das zu mutierende Chromosom durch $C = (x_{11}, \ldots, x_{1n})$ gegeben.

Dann wird zunächst $i \in \{1, \ldots, n\}$ zufällig gewählt. Daraufhin wird x'_{1i} zufällig aus dem Definitionsbereich von x_{1i} gewählt und x_{1i} durch x'_{1i} ersetzt.

Weitere Möglichkeiten der Mutation finden sich z.B. in [Lippe06].

3.6 Messy Genetische Algorithmen

Messy Genetische Algorithmen verwenden eine positionsunabhängige Semantik. Die Gene können in einer beliebigen Permutation auf dem Chromosom liegen. Auch die Länge der Strings, die die Chromosomen codieren, kann verschieden sein. Dazu ordnet man jedem Gen ein eindeutiges Symbol zu, welches es eindeutig klassifiziert. Gene werden jetzt durch (Symbol, Wert)-Paare codiert (vgl. Beispiel 3.4).

Beispiel 3.4 (Messy GA Codierung)
Die Symbole werden aus der Menge der natürlichen Zahlen gewählt. Eine messy-Codierung eines Chromosoms in binärer Codierung

$$C = 1101$$

wäre z.B.:

$$C = (\,(1,1),(2,1),(4,0),(3,1)\,)$$

Durch den erhöhten Freiheitsgrad bei den Messy Genetischen Algorithmen können Chromosome entstehen die überspezifiziert oder unterspezifiziert sind.

Das Problem der Überspezifizierung besteht darin, dass ein Chromosom für ein Gen mehrere verschiedene Werte spezifiziert. Daraus folgt, dass das Chromosom nicht eindeutig dekodiert werden kann.

Beispiel 3.5 (Überspezifizierung bei Messy Genetischen Algorithmen)
Das folgende Chromosom ist überspezifiziert, da es 3 verschiedene Werte für das Gen mit dem Symbol 1 enthält:

$$C = (\,(1,0)(1,1)(1,2)(2,1)(3,0)(4,1)\,)$$

Eine mögliche Lösung des Überspezifizierungsproblems besteht darin, das am weitesten links liegende von allen ein Gen codierenden (Symbol, Wert)-Paaren auszuwählen. Im Beispiel würde so das Paar (1,0) ausgewählt.

Das Problem der Unterspezifizierung liegt vor, wenn Gene auf dem Chromosom fehlen.

Beispiel 3.6 (Unterspezifizierung bei Messy Genetischen Algorithmen)
Das Chromosom

$$C = (\,(1,0)(2,0)\,)$$

ist unterspezifiziert, da es die Gene 3 und 4 nicht definiert.

Lösungsmöglichkeiten für das Unterspezifizierungsproblem sind z.B. das Chromosom zu löschen oder die fehlenden Gene mit Zufallswerten aufzufüllen.

Crossover bei Messy Genetischen Algorithmen ist in Abbildung 3.3 dargestellt und funktioniert nach der Methode *cut-and-splice* (Schneiden und Zusammenfügen). Aus zwei

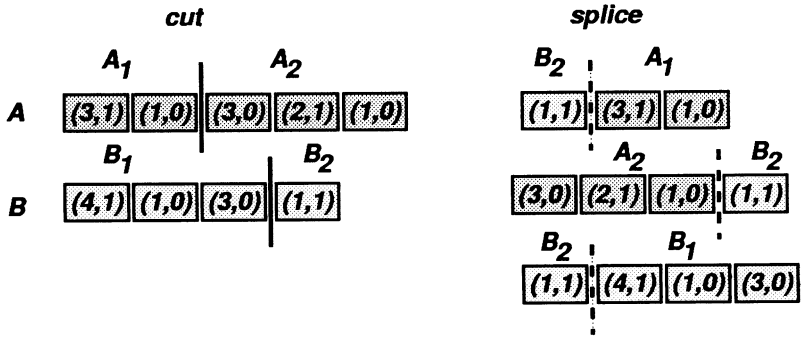


Abbildung 3.3: Crossover bei Messy GA, Cut and Splice Methode

Eltern-Chromosomen A und B entstehen durch zufällige Auswahl von zwei Trennstellen,

die für die beiden Chromosomen nicht gleich sein müssen, vier Teilstrings A_1, A_2, B_1, B_2. Diese können in beliebiger Reihenfolge zusammengesetzt werden.

Zur Mutation ist anzumerken, dass die Mutationsoperatoren auf die Allele angewendet werden, nicht auf die Symbole.

3.7 Eigenschaften von Evolutionären Algorithmen

Evolutionäre Algorithmen stellen einen robusten und effizienten Suchmechanismus dar. Indem sie auf einer Population von Individuen arbeiten, ist die Suche hochgradig parallel. Dies verhilft ihnen bei stark zerklüfteten Fitnesslandschaften nicht in einem lokalen Minimum hängen zu bleiben. Sie brauchen keine zusätzlichen Informationen über die Problemstellung, wie etwa einen Gradienten der Zielfunktion. Nur der Zielfunktionswert wird für die Steuerung der Suche benötigt. Gerade dies macht sie in Fällen in denen der Suchraum des Problems schlecht verstanden ist attraktiv. Weiterhin können Evolutionäre Algorithmen eine Vielzahl von ausreichenden Lösungen finden, aus denen der Anwender die für ihn geeignetste auswählen kann.

Der Erfolg eines Lösungsversuches mit Genetischen Algorithmen kann nicht garantiert werden. Seine Konfiguration kann viel Zeit kosten, da er über eine Vielzahl von Parametern konfiguriert wird. Zur optimalen Wahl dieser Parameter existieren keine bewiesenen Regeln, sondern nur durch Experimente erworbene Heuristiken. Jedes neue Problem erfordert in der Regel eine neue Wahl der Parameter. Weiterhin hängen die Parameter stark voneinander ab, so dass sie nicht als unabhängige Regler zu verwenden sind.

Theoretische Resultate, wie das *Schema Theorem* und die *Building Block Hypothese* beziehen sich auf einfache Modelle, wie den kanonischen Genetischen Algorithmus, und ihre Gültigkeit ist umstritten (vgl. [Gerdes04] und [Weicker02]).

Die Berechnungen bei der Zielfunktionsauswertung können so komplex sein, dass eine akzeptable Rechendauer überschritten wird. Aufgrund der parallelen Natur Evolutionärer Algorithmen kann durch eine Implementierung auf paralleler Architektur eine potentielle Performanzsteigerung erreicht werden.

4 Optimierung von Fuzzy-Controllern mit Evolutionären Algorithmen

Dieses Kapitel behandelt Möglichkeiten der Optimierung von Fuzzy-Controllern mit evolutionären Algorithmen. Die Darstellung stützt sich wesentlich auf [Cordon01].

Betrachtet wird die Optimierung der Wissensbasis eines Fuzzy-Controllers. Wie in dem Kapitel 2.3 über Fuzzy-Controller dargestellt, besteht die Wissensbasis aus den zwei verschiedenen Komponenten Datenbasis und Regelbasis.

Die Optimierung der Datenbasis wird *Genetic Tuning* genannt. Dabei geht man von einem bereits vorhandenen funktionierenden Fuzzy-Controller aus. Von diesem wird die Datenbasis verbessert. Die Regelbasis wird entweder unverändert gelassen oder nur der Konklusionsteil der Regeln wird geringfügig modifiziert.

Das Erlernen und Optimieren der Regelbasis wird *Genetic Learning* genannt. Dabei geht man von einer vorgegebenen Datenbasis aus. Die Partitionierung der Ein- und Ausgaberäume wird dabei entweder automatisch, z.B. durch gleichmäßige Überdeckung des Definitionsbereichs mit gleichförmigen Dreiecksmengen, oder durch einen Experten vorgenommen. Wird die Regelbasis von Grund auf neu erlernt, so startet man mit einer zufällig generierten Regelbasis, in welche a-priori Wissen über die erwünschte Regelbasis integriert werden kann.

4.1 Optimieren der Datenbasis (Genetic Tuning)

Ziel des Genetic Tuning Prozesses ist die Datenbasis besser an die gegebene Regelbasis anzupassen. Dabei existieren im wesentlichen die folgenden Möglichkeiten:

1. Modifikation von skalierenden Funktionen
2. Modifikation der Mitgliedsgradfunktionen

3. Modifikation des Konklusionsteils von Regeln

Diese sind in obiger Liste absteigend dem Effekt den eine Modifikation hat aufgeführt. Eine skalierende Funktion bildet normalisierte Definitionsbereiche von Fuzzy-Mengen auf anwendungsspezifische Definitionsbereiche ab. Sie entscheidet somit wesentlich über den semantischen Kontext in welchem der Fuzzy-Controller arbeitet. Für diese Arbeit spielen skalierende Funktionen keine Rolle und werden daher nicht weiter betrachtet. Näheres über skalierende Funktionen findet sich in [Cordon01].

Ebenso wie die Modifikation von skalierenden Funktionen verändert auch die Modifikation von Zugehörigkeitsgradfunktionen den Kontext in dem der Fuzzy-Controller arbeitet. Die Kontextänderung darf nicht zu groß sein, da sonst die gegebene Regelbasis unbrauchbar werden kann.

Beim Genetic Tuning wird in der Regel der mittlere quadratische Fehler (MSE) zwischen Fuzzy-Controller-Ausgabe und einer gegebenen Trainingsmenge als Fitnessfunktion verwendet.

4.1.1 Genetic Tuning von Mamdani-Controllern

Die Parameter der Zugehörigkeitsgradfunktionen werden codiert. Für eine Eingabedimension werden gemäß der Reihenfolge ihres Auftretens von links nach rechts die charakteristischen Punkte der Fuzzy-Mengen in den String aufgenommen. So werden etwa für Dreiecksmengen $\tilde{D}_1$, $\tilde{D}_2$ und $\tilde{D}_3$ die definierenden Punkte (links,mitte,rechts) reell in einen String codiert:

$$\begin{array}{l} \tilde{D}_1(1,2,3) \\ \tilde{D}_2(2.5,3.5,4.5) \\ \tilde{D}_3(3.75,7,9) \end{array} \quad \longrightarrow \quad (1,2,3,2.5,3.5,4.5,3.75,7,9)$$

Für die Rekombination können Standardverfahren für reelle Rekombination angewendet werden. Der verwendete Rekombinationsoperator muss sicherstellen, dass sich wieder valide Fuzzy-Mengen für das Problem ergeben. Bei n-Punkt-Crossover können Nachfahren entstehen die den Definitionsbereich nicht mehr vollständig abdecken. Soll eine bestimmte Form, wie etwa gleichschenklige Dreiecke, erhalten bleiben, so muss darauf geachtet werden die Schnittstellen so zu wählen, dass keine Mengen zerschnitten werden. Eine andere Lösungsmöglichkeit ist, die Kindwerte aus den Mittelwerten der Werte der beiden Eltern zu bilden (Intermediärer Crossover, s. Definition 3.2) :

$$\begin{array}{l} (1,3,5 \mid 4,6,8 \mid 7,9,11) \\ \qquad\quad \underset{\text{Elter}_1}{} \\ \\ (1,2,3 \mid 2,5,8 \mid 7,9,11) \\ \qquad\quad \underset{\text{Elter}_2}{} \end{array} \quad \longrightarrow \quad \underset{\text{Kind}}{(1,2.5,4 \mid 3,5.5,8 \mid 7,9,11)}$$

Dadurch bleibt die Form der Dreiecksmengen und meist eine gute Überdeckung des Definitionsbereichs erhalten.

Für die Mutation können Standard-Verfahren für reell-kodierte Chromosome verwendet werden (s. Abschnitt 3.5.5). Auch bei der Mutation muss darauf geachtet werden, dass für das Problem valide Fuzzy-Mengen entstehen.

Die Modifikation der Regelkonklusion kann z.B. im Austauschen derselben durch eine benachbarte Fuzzy-Menge in der gemeinsamen Fuzzy-Partition bestehen.

4.1.2 Genetic Tuning von Sugeno-Controllern

Der Sugeno-Controller unterscheidet sich vom Mamdani-Controller durch die Verwendung von reellen Ausgabefunktionen in den Regel-Konklusionen. Bei der Datenbasisoptimierung codiert man die Partitionierung der Eingaberäume wie beim Mamdani-Controller. Die Parameter der Ausgabefunktionen werden der Reihe nach in einen String codiert.

Beispiel 4.1 (Codierung der Parameter der Ausgabefunktionen beim Sugeno-Controller) Seinen m Regeln der Form

$R_i =$ IF X_1 IS $\tilde{A_{i1}}$ AND ... AND X_n IS $\tilde{A_{in}}$ THEN $Y = p_{i1}X_1 + p_{i2}X_2 + \cdots + p_{in}X_n + p_{i0}$

gegeben.

Die Parameter p_{ij} der Ausgabefunktionen werden in dem Chromosom

$$C = (C_1, C_2, \ldots, C_m)$$

gespeichert, wobei die Parameter jeder Regel durch

$$C_i = p_{i0}, p_{i1}, \ldots, p_{in} \quad i = 1, \ldots, m$$

codiert werden.

Es können die bekannten reellen genetischen Operatoren angewendet werden, wobei für die Parameter festgelegt werden muss, in welchen Definitionsbereichen sie variieren dürfen.

4.2 Lernen und Optimieren der Regelbasis (Genetic Learning)

Beim Genetic Learning wird von einer vordefinierten Datenbasis ausgehend anhand einer Trainingsmenge, bestehend aus (Eingabe, Ausgabe)-Paaren, ein vollständiger Controller entwickelt. Wenn die Regelbasis eine gewisse Güte erreicht hat verwendet man Genetic Tuning zur weiteren Steigerung der Performanz des Controllers.

Die drei wichtigsten Ansätze Fuzzy-Controller mit Evolutionären Algorithmen zu optimieren sind der *Pittsburgh*-Ansatz, der *Michigan*-Ansatz und der *Iterative-Regellern*-Ansatz (IRL). Jeder dieser Ansätze löst ein grundsätzliches Problem, welches bei der Optimierung von Fuzzy-Controllern mit Evolutionären Algorithmen auftritt, das im Abschnitt 4.2.1 beschriebene *cooperation versus competition Problem (CCP)*. Daran schließt sich in Abschnitt 4.2.2 eine Darstellung von Kriterien an, die bei dem Erlernen einer Regelbasis zu beachten sind. Im Anschluss daran werden Michigan-, Pittsburgh- und der IRL-Ansatz vorgestellt. Hierbei werden vom Michigan- und dem IRL-Ansatz die grundlegenden Prinzipien behandelt. Der Pittsburgh-Ansatz wird ausführlich dargestellt, da er Grundlage der zu dieser Arbeit gehörenden Implementierung ist (s. Kapitel 5).

4.2.1 Das cooperation versus competition Problem (CCP)

Bei der Optimierung der Regelbasis eines Fuzzy-Controllers tritt folgendes Problem auf:

Auf der einen Seite ist es ein Prinzip des Fuzzy-Controllers, dass die Regeln der Regelbasis möglichst gut kooperieren sollen, um eine optimale Performanz des Controllers zu erreichen.

Auf der anderen Seite ist es ein Prinzip der Evolutionären Algorithmen durch Wettkampf unter den Individuen einer Population bessere Lösungen zu finden.

Wählt man die Strategie, dass jedes Individuum eine Regel repräsentiert, dann muß der Konflikt zwischen diesen beiden Prinzipien bewältigt werden.

4.2.2 Kriterien beim Erlernen einer Regelbasis

Das Lernen der Regelbasis sollte sich an den folgenden Kriterien orientieren, da sie die Approximationsgenauigkeit eines Fuzzy-Controllers erhöhen:

1. **Vollständigkeit**
   Zu jeder möglichen Eingabe $x \in U$ muss eine gültige Ausgabe, also eine nichtleere Ausgabemenge B, erzeugt werden. Also muss für jede mögliche Eingabe mindestens eine Regel aktiv werden. Dazu definiert man die σ-Vollständigkeit:

   $$\forall x \in U \quad \text{height}(B) \geq \sigma > 0$$

   wobei height(B) die Höhe der Fuzzy-Menge B bezeichnet.

2. **Konsistenz**
   Eine Regelbasis heisst konsistent, wenn sie keine widersprüchlichen Regeln enthält. In der klassischen Logik ist das Feststellen von Widersprüchlichkeiten einfach. In der Fuzzy-Logik kann jedoch das Vorhandensein von verschiedenen Konsequenzen zu gleicher Prämisse nicht als eindeutiger Klassifikator für die Inkonsistenz der Regel verwendet werden. Eine allgemeine Lösung dieses Problems ist nicht gegeben. Ein Ansatz ist eine Regel als inkonsistent zu beurteilen, wenn sie gleiche Prämissen aber disjunkte Fuzzy-Mengen als Konsequenzen hat. Ein anderer Ansatz ist die Definition einer k-Konsistenz. Dabei werden für eine Regel positive und negative Testbeispiele bestimmt und ein $k \in [0, 1]$ gewählt. Eine Regel ist genau dann k-konsistent, wenn die Anzahl der negativen Beispiele kleiner als 100*k Prozent der positiven Beispiele ist.

3. **Geringe Komplexität**
   Mit steigender Anzahl von Regeln in der Regelbasis nimmt die Komplexität derselben zu. Dies ist nachteilig, wenn die Interpretierbarkeit des Systems relevant ist. Weiterhin steigt der Berechnungsaufwand des Controllers mit wachsender Regelanzahl. Andererseits erreicht der Controller durch mehr Regeln im Allgemeinen eine höhere Genauigkeit bei der Ausgabe.

4. **Geringe Redundanz**
   Redundante Regeln sind Regeln deren Löschen aus der Regelbasis auf die Güte des Controllers keinen wesentlichen Effekt hat. Da sie trotzdem ausgewertet werden, erhöhen sie unnötig den Berechnungsaufwand.

4.2.3 Der Michigan-Ansatz

Der Michigan-Ansatz wurde 1975 von John Holland an der Michigan-Universität entwickelt [Holland75]. Er beschreibt *Classifier Systems*, welche mit einfachen Regeln, den Classifiern, arbeiten. Classifier Systeme interagieren mit ihrer Umgebung und besitzen die Fähigkeit sich kontinuierlich an Veränderungen der Umgebung anzupassen. Hierbei wird ein Gütebewertungsmechanismus für die Classifier (Credit Assignment System) mit der Erzeugung neuer optimierter Classifier durch einen genetischen Algorithmus kombiniert (Classifier Discovery System). Der zugrundeliegende Aufbau ist schematisch in Abbildung 4.1 dargestellt. Die Fuzzifizierung des Michigan-Ansatzes führt zu *Fuzzy Clas-*

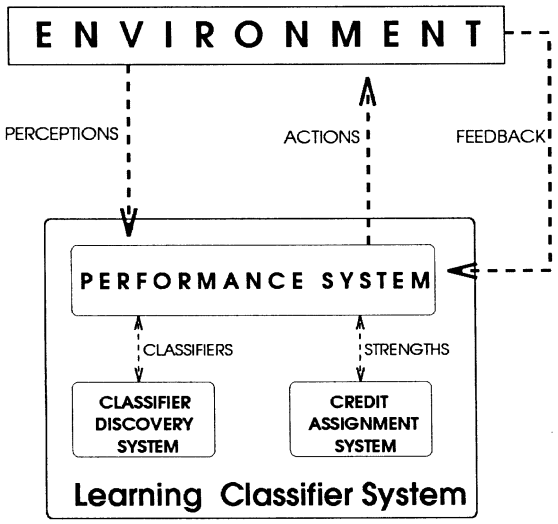


Abbildung 4.1: Classifier System

sifier Systems. Abbildung 4.2 zeigt die Erweiterung von Abbildung 4.1 zu einem Fuzzy Classifier System. Jede Fuzzy-Regel entspricht dabei einem Classifier. Im Genetischen Algorithmus in der *Rule Generation Mechanism*-Komponente (vgl. Abbildung 4.2) entspricht jede Regel einem Individuum der Population. Daher stellt die Lösung des CCP eine äußerst schwierige Aufgabe in solchen Systemen dar: Da der evolutionäre Prozess mit den einzelnen Regeln als Individuen arbeitet, ist eine gute Kooperation zwischen den miteinander in Wettbewerb stehenden Regeln sehr schwer zu erreichen. Es ist nötig eine Fitnessfunktion aufzustellen, welche sowohl die Güte einer Regel als auch die Kooperation aller Regeln miteinander, bewertet. Eine solche Fitnessfunktion aufzustellen stellt eine sehr schwierige Aufgabe dar.

4.2.4 Der Pittsburgh-Ansatz

Der Pittsburgh-Ansatz wurde 1980 von Smith an der Pittsburgh-Universität entwickelt [Smith80].

Während der Michigan-Ansatz für den Einsatz in Echtzeitsystemen (online-Learning) geeignet ist, liegt der Schwerpunkt des Pittsburgh-Ansatzes auf dem Trainieren auf einer vorgegebenen Trainingsmenge (offline-Learning). Der Michigan-Ansatz basiert auf dem Prinzip, dass eine Regelbasis sich durch Interaktion mit ihrer Umgebung selbstständig an diese anpasst. Der Pittsburgh-Ansatz hingegen verwendet eine Population miteinander in Konkurrenz stehender Regelbasen und wählt die am besten angepasste aus.

Das CCP wird durch den Pittsburgh-Ansatz hervorragend gelöst, da jedes Individuum

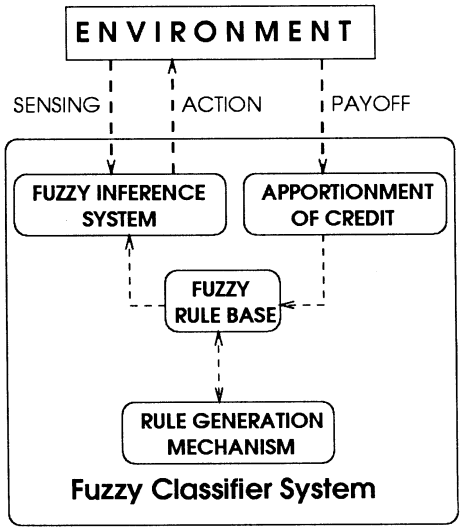


Abbildung 4.2: Fuzzy Classifier System

eine gesamte Regelbasis repräsentiert und deswegen die Fitnessfunktion des Genetischen Algorithmus die Kooperation der Regeln direkt mitbewertet. Komplexe und aufwändige Bewertungs- und Konfliktbewältigungsmechanismen wie sie beim Michigan-Ansatz nötig sind, entfallen beim Pittsburgh-Ansatz. Die nötige Auswertung aller Regelbasen einer Population verursacht einen deutlich höheren Rechenaufwand als beim Michigan-Ansatz. Im Pittsburgh-Ansatz muss der Genetische Algorithmus einen wesentlich umfangreicheren Suchraum durchforsten, was die Entdeckung einer guten Lösung im Allgemeinen erschwert.

Abbildung 4.3 zeigt den Aufbau eines auf dem Pittsburgh-Ansatz basierenden Systems analog der Abbildung 4.1 für das Michigan-System. In Abbildung 4.4 ist die Anwendung des Pittsburgh-Ansatzes auf die Optimierung von Fuzzy-Controllern schematisch dargestellt.

Repräsentation- und Codierung der Regelbasis

Im Pittsburgh-Ansatz entspricht jede Regelbasis einem Chromosom und jede Regel entspricht einem Gen.

Im folgenden werden drei mögliche Darstellungsformen der Regelbasis vorgestellt:

1. Entscheidungstabelle
2. Relationsmatrix

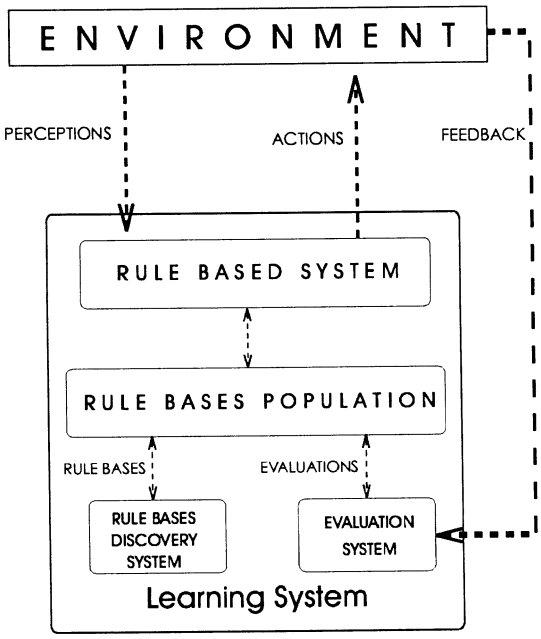


Abbildung 4.3: Adaptives System nach dem Pittsburgh-Ansatz

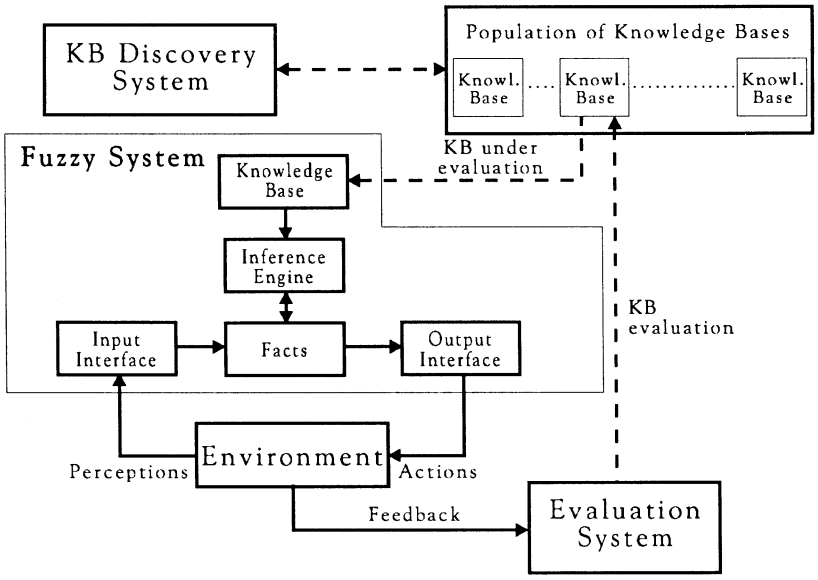


Abbildung 4.4: Genetisches Fuzzy System basierend auf dem Pittsburgh-Ansatz

3. Liste von Regeln

Die Entscheidungstabelle und die Relationsmatrix eigenen sich nur für Regelbasen, die eine feste Struktur besitzen. Der Code hat eine feste Länge und eine positionsabhängige Semantik:

- Die Länge des Codes entspricht der Anzahl der Elemente der Entscheidungstabelle bzw. Relationsmatrix.
- Jedem Ort eines Gens auf dem Chromosom entspricht genau ein Element in der Entscheidungstabelle bzw. Relationsmatrix und der Wert des Gens entspricht dem Wert des Elements.

Die Repräsentation durch eine Liste von Regeln eignet sich sowohl für Regelbasen mit einer festen Anzahl von Regeln als auch für Regelbasen mit einer variablen Anzahl von Regeln. Weiterhin kann die Codelänge der Regeln variieren. Die Semantik des Codes ist positionsunabhängig, da die Reihenfolge der Regeln in der Liste für den Inferenzmechanismus des Fuzzy-Controllers irrelevant ist.

Wenn mit Regeln fester Länge gearbeitet wird ist die Semantik des Codes für jede Regel positionsabhängig, die Semantik des Codes für die Regelbasis jedoch positionsunabhängig.

Repräsentation der Regelbasis durch eine Entscheidungstabelle

Eine Entscheidungstabelle ist eine Funktion die jedem Element eines endlichen mehrdimensionalen Raumes ein Element eines endlichen eindimensionalen Raumes zuordnet. Diese Codierungsform eignet sich für Regelbasen mit mehreren Eingabe-Variablen und einer Ausgabe-Variable. Jedem Element des kartesischen Produkts der Fuzzy-Mengen der Partitionierungen der Eingaberäume wird entweder eine Fuzzy-Menge der Ausgaberaum-Partitionierung zugeordnet oder die leere Menge.

Die folgende Entscheidungstabelle stellt eine Regelbasis mit zwei Eingabevariablen X_1 und X_2 und einer Ausgabe-Variable Y dar. Die Eingabe-Variablen sind jeweils durch die Fuzzy-Mengen A_{11}, A_{12}, A_{13} und A_{21}, A_{22}, A_{23} restringiert. Die Ausgabe-Variable ist durch die Fuzzy-Mengen B_1, B_2, B_3, B_4 restringiert.

	A_{21}	A_{22}	A_{23}
A_{11}		B_1	B_2
A_{12}	B_1	B_2	B_3
A_{13}	B_1	B_3	B_4

Der Eintrag in der ersten Spalte und zweiten Zeile codiert die Regel

$$\text{IF } X_1 \text{ IS } A_{12} \text{ AND } X_2 \text{ IS } A_{21} \text{ THEN } Y \text{ IS } B_1$$

Der leere Eintrag in der ersten Spalte und der ersten Zeile bedeutet, dass für diese Kombination von Fuzzy-Mengen keine Regel existiert.

Die Repräsentation der Regelbasis durch eine Entscheidungstabelle erfüllt das Kriterium der Vollständigkeit, wenn die verwendeten Eingabepartitionierungen die Eingaberäume vollständig überdecken. Allerdings ist dieses Verfahren nur für ein bis drei Eingabe-Variablen praktikabel, da die Anzahl der Einträge in der Tabelle exponentiell mit der Anzahl der Eingabe-Variablen wächst.

Codierung

Für jede Partition jedes Eingaberaums wird eine feste Reihenfolge der restringierenden Fuzzy-Mengen definiert. Dann wird jeder Fuzzy-Menge der Partition des Ausgaberaums ein eindeutiger Bezeichner zugeordnet. Möglich ist z.B. sie durchzunummerieren. Die Darstellung der Entscheidungstabelle, die auf dieser festen Reihenfolge basiert, wird nun durch sukzessive Verkettung in einen String transformiert.

Obiges Beispiel würde, unter Verwendung des Wertes 0 für leere Tabelleneinträge, durch den String

$$(0, 1, 2, 1, 2, 3, 1, 3, 4)$$

codiert.

Crossover

Aufgrund der positionsabhängigen Semantik können Standardoperatoren für Crossover (s. Abschnitt 3.5.4) angewendet werden.

Bei oben angeführter Codierungsweise in eine lineare Struktur werden Nachbarschaftsbeziehungen nicht erhalten. Daher wird die Entscheidungstabelle als planare Struktur (Matrix) aufgefasst, da so bestehende Nachbarschaftsbeziehungen erhalten werden können (vgl. [Gerdes04], S. 197 f).

Planarer 1-Punkt-Crossover Es wird per Zufallszahl ein Gen, d.h. eine Zelle der Entscheidungstabelle, als Crossoverpunkt ausgewählt. Es werden alle Gene ausgetauscht, deren „Koordinaten“ mindestens so groß sind, wie die des Crossoverpunktes. Abbildung 4.5 verdeutlicht diese Vorgehensweise. Das eingerahmte Gen ist der Crossoverpunkt.

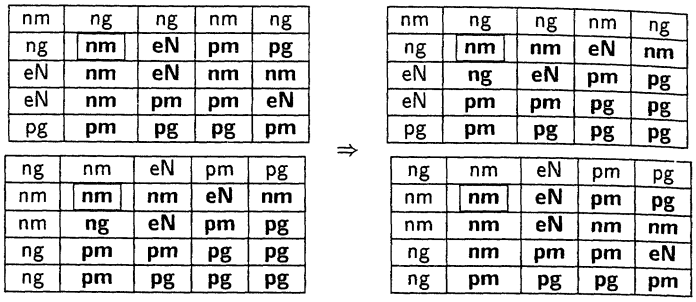


Abbildung 4.5: Funktionaler 1-Punkt-Crossover für planare Chromosomen

Planarer 2-Punkt-Crossover Beim 2-Punkt-Crossover wählt man zwei Crossoverpunkte aus und tauscht alle Gene aus, deren „Koordinaten" in dem durch die zwei Crossoverpunkte begrenzten Rechteck liegen. Abbildung 4.6 stellt diesen Vorgang dar. Die eingerahmten Gene sind die Crossoverpunkte.

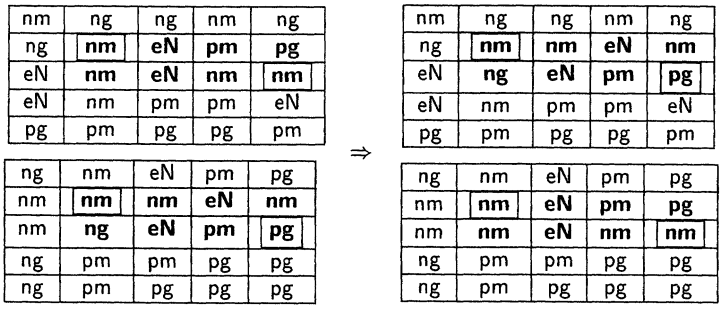


Abbildung 4.6: Funktionaler 2-Punkt-Crossover für planare Chromosomen

Erweiterung zum funktionalen Crossover Der planare 1- und 2-Punkt-Crossover für zwei Eingabedimensionen wird kanonisch zum *funktionalen Crossover* für mehr als zwei Eingabedimensionen fortgesetzt. So werden etwa beim funktionalen 2-Punkt-Crossover im dreidimensionalen Fall alle Gene, die in dem durch die Koordinaten der beiden Crossoverpunkte definierten Würfel liegen, ausgetauscht.

Mutation

Nachbarschafts-Mutation Repräsentiert das Allel des zu mutierenden Gens den Fall, das für diese Kombination von Eingabe-Fuzzy-Mengen keine Regel vorliegt, so wird diesem ein zufälliger Wert aus der Menge der Bezeichner für die Fuzzy-Mengen der Ausgabepartition zugewiesen. Ansonsten wird der Wert des Gens gemäß der vorher definierten

Reihenfolge der Fuzzy-Mengen in der Ausgabepartition durch einen Bezeichner für eine benachbarte Fuzzy-Menge ersetzt.

Gleichverteilte Mutation Dem zu mutierenden Gen wird ein zufälliger Wert aus der Menge der Bezeichner für die Fuzzy-Mengen der Ausgabepartition zugewiesen.

Repräsentation der Regelbasis durch eine Relationsmatrix

Für Fuzzy-Controller mit einer Eingabe-Variable und einer Ausgabe-Variable werden oft Relationsmatrizen verwendet. Die Relationsmatrix ordnet jedem Element des kartesischen Produkts der Fuzzy-Mengen des Eingaberaums mit den Fuzzy-Mengen des Ausgaberaums einen Zugehörigkeitsgrad zu.

Die folgende Relationsmatrix codiert eine Regelbasis mit einer Eingabe-Variable X und einer Ausgabe-Variable Y. Diese sind jeweils durch die Fuzzy-Mengen A_1, A_2, A_3 und B_1, B_2, B_3 restringiert.

	B_1	B_2	B_3
A_1	0.4	0.3	0.1
A_2	0.2	0.8	0.2
A_3	0.1	1.0	0.4

Wie bei der Entscheidungstabelle kann die Relationsmatrix durch zeilenweise Verkettung in eine Stringdarstellung überführt werden.

Repräsentation der Regelbasis durch eine Liste von Regeln

Wenn mehr als drei Eingabevariablen vorliegen ist die Codierung durch eine Entscheidungstabelle nicht mehr praktikabel. In diesem Fall eignet sich eine Codierung der Regelbasis durch eine Liste von Regeln.

Regeln fester Länge

Eine Regel, die in ihrer Prämisse jede Eingabevariable genau einmal enthält, in ihrer Konklusion genau eine Ausgabevariable enthält und den `OR`-Operator nicht verwendet, wird als *elementare Regel* bezeichnet.

Durch die Verwendung der *disjunktiven Normalform*(DNF) ist es möglich mehrere elementare Regeln zu einer Regel in DNF zusammenzufassen und so die Grösse der Regelbasis zu reduzieren.

Nachteilig bei der Codierung durch eine Liste von Regeln ist, dass bei dieser Codierungsform die Regelbasis unvollständig und inkonsistent sein kann. Diesem Problem kann durch Verwendung geeigneter Vollständigkeits- und Konsistenzkriterien in der Fitnessfunktion begegnet werden.

Beispiel 4.2 (Elementare Regeln durch DNF zusammenfassen)
(1) und (2) zeigen elementare Regeln, die in (3) durch DNF zu einer Regel zusammengefasst werden.

$$\text{IF } X_1 \text{ IS } A_{11} \text{ AND } X_2 \text{ IS } A_{21} \text{ THEN } Y \text{ IS } B_1 \quad (1)$$
$$\text{IF } X_1 \text{ IS } A_{12} \text{ AND } X_2 \text{ IS } A_{21} \text{ THEN } Y \text{ IS } B_1 \quad (2)$$
$$\text{IF } X_1 \text{ IS } \{A_{11} \text{ OR } A_{12}\} \text{ AND } X_2 \text{ IS } A_{21} \text{ THEN } Y \text{ IS } B_1 \quad (3)$$

Verwendet man Regeln fester Länge, also einen positionsabhängigen Code, so können ausschliesslich elementare Regeln codiert werden. Diese Form der Codierung erhöht die Länge des benötigten Codes gegenüber der Entscheidungstabelle, wenn die Regelbasis vollständig ist:

Beispiel 4.3
Die Regel

$$\text{IF } X_1 \text{ IS } A_{1\underline{1}} \text{ AND } X_2 \text{ IS } A_{2\underline{1}} \text{ THEN } Y \text{ IS } B_{\underline{1}}$$

wird durch den String

$$(1, 1, 1)$$

codiert. Im Vergleich zur Entscheidungstabelle werden also drei statt ein Symbol verwendet.

Diese Codierung bringt erst Vorteile, wenn die Regelbasis unvollständig ist. Unvollständigkeit ist aber keine wünschenswerte Eigenschaft einer Regelbasis. Außerdem können aufgrund der Einschränkung auf elementare Regeln keine Fusionen zu DNF-Regeln durchgeführt werden.

Verwendung von DNF-Regeln Die folgende Codierungsform ermöglicht die Repräsentation von DNF-Regeln: Für jede linguistische Variable werden die Fuzzy-Mengen ihrer Partitionierung geordnet. Für jede Eingabevariable X_i wird ein Bitstring gebildet, der die Länge N_i besitzt, welche der Anzahl der Fuzzy-Mengen in der zu X_i gehörigen Partitionierung $(A_{i1}, \ldots, A_{iN_i})$ entspricht. Der Code entsteht dann durch Verkettung der Bitstrings in der Reihenfolge wie die Variablen in den Regeln verwendet werden.

Jeder Position eines Bits in dem resultierenden Code für eine Regel ist also eindeutig eine Fuzzy-Menge zugeordnet. Eine 1 bedeutet, dass die Fuzzy-Menge in dieser Regel vorkommt, eine 0 bedeutet, dass die Fuzzy-Menge nicht in dieser Regel vorkommt.

Beispiel 4.4
Für die Partitionierungen gelte:

$$\begin{aligned} X_1 &= A_{11}, A_{12}, \ldots, A_{15} \\ X_2 &= A_{21}, A_{22}, A_{23} \\ X_3 &= A_{31}, A_{32}, \ldots, A_{35} \\ Y &= B_{11}, B_{12}, \ldots, B_{17} \end{aligned}$$

Dann codiert der String

$$(01100\,001\,00011\,0000011)$$

die Fuzzy-Regel

IF X_1 IS $\{A_{12}$ OR $A_{13}\}$ AND X_2 IS A_{23} AND X_3 IS $\{A_{34}$ OR $A_{35}\}$ THEN Y IS B_{16} OR B_{17}

Genetische Operatoren

Standard-Crossoververfahren arbeiten auf Codierungen mit positionsabhängiger Semantik. Da diese durch die beliebige Anordnung der Regeln im die Regelbasis codierenden Chromosom nicht gegeben ist, sind die Standard-Crossoververfahren untauglich. Denn bei ihrem Einsatz können die entstehenden Regelbasen unvollständig und inkonsistent werden.

Es gibt zwei Lösungsansätze für dieses Problem:

1. **Umordnen**
   Der Umordnungsmechanismus wählt per Zufallszahl eine Stelle im Chromosom aus und vertauscht die beiden entstehenden Teile. Dadurch wird die Diversität der Population erhöht.
2. **Ausrichten** Hierbei werden die an einem Crossover beteiligten beiden Chromosome so ausgerichtet, dass die Distanz der Loci von Genen, die Regeln mit ähnlicher Prämisse repräsentieren minimiert wird. Hierdurch wird die Wahrscheinlichkeit unvollständiger und inkonsistenter Nachfahren reduziert. Allerdings ist dieses Verfahren mit hohem Rechenaufwand verbunden.

Ein funktionsorientierter Crossoveroperator Die Operationen Umordnen und Ausrichten versuchen den postionsabhängigen Crossover besser an die postionsunabhängige Semantik anzupassen, dadurch wird kein echter funktionsorientierter Crossover erreicht.

Durch die Zuordnung der Regeln zu „virtuellen Loci“ wird ein funktionsorientierter Crossover ermöglicht. Dies erreicht man durch Anordnen der Regeln der Liste wie sie in einer Entscheidungstabelle angeordnet wären. Statt eine Schnittstelle per Zufallszahl zu bestimmen, bestimmt man jetzt eine Schnittfläche. Die beiden am Crossover beteiligten Regelbasen werden entlang der Schnittfläche zerteilt und jeweils zwei zugehörige Fragmente ausgetauscht (vgl. Abbildung 4.7). Damit diese Methode praktikabel wird, muss

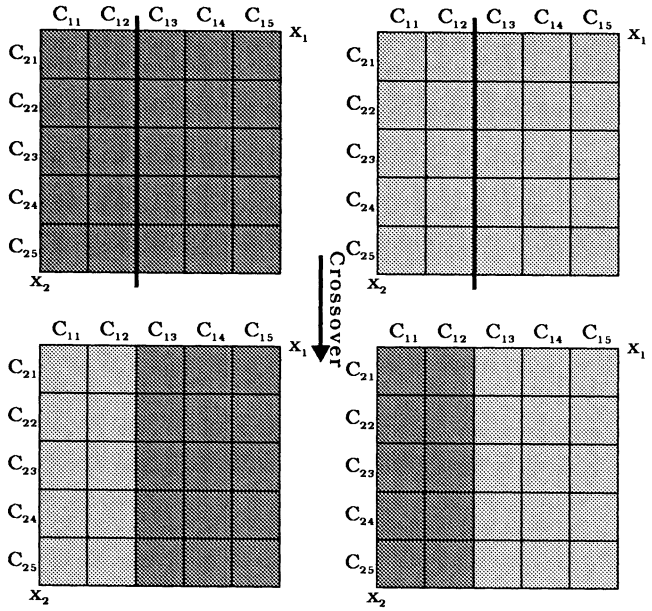


Abbildung 4.7: Crossover von Entscheidungstabellen durch Schnittfläche

eine Möglichkeit gefunden werden, ohne den Zwischenschritt des Überführens in eine Entscheidungstabelle auszukommen. Die Lösung stellen sogenannte Schnittregeln dar. Jede Schnittregel definiert ein Fragment der Entscheidungstabelle. Schnittregeln haben den folgenden Aufbau:

IF	die Prämisse enthält eine bestimmte Kombination von Fuzzy-Mengen
THEN	die Regel gehört zu einem bestimmten Fragment

Die Prämissen der Schnittregeln werden mit demselben Verfahren codiert, mit welchem die Regeln der Regelbasis codiert werden. Die Schnittfläche in Abbildung 4.7 wird durch die beiden Schnittregeln

IF	die Prämisse enthält X_1 IS C_{11} oder X_1 IS C_{12}	(SR1)
THEN	die Regel gehört zu Fragment 1	
IF	die Prämisse enthält X_1 IS C_{13} oder X_1 IS C_{14} oder X_1 IS C_{15}	(SR2)
THEN	die Regel gehört zu Fragment 2	

erreicht.

Die Codierung ist:

$$\text{SR1} = (11000\,00000) \quad \text{und} \quad \text{SR2} = (00111\,00000)$$

Der Crossover wird realisiert, indem jede Schnittregel mit jeder Fuzzy-Regel durch bitweises `UND` verknüpft wird. Die Anzahl der Einsen im Ergebnis-Bitstring bestimmt den Zugehörigkeitsgrad der Regel zu dem durch die Schnittregel spezifizierten Fragment.

Beispiel 4.5
Gegeben seien die DNF-Fuzzy-Regeln:

$$\text{IF } X_1 \text{ IS } \{C_{11} \text{ OR } C_{12} \text{ OR } C_{13}\} \text{ AND } X_2 \text{ IS } \{C_{21} \text{ OR } C_{22} \text{ OR } C_{23}\} \text{ THEN } Y \text{ IS } D_5 \tag{R1}$$

$$\text{IF } X_1 \text{ IS } \{C_{13} \text{ OR } C_{14} \text{ OR } C_{15}\} \text{ AND } X_2 \text{ IS } \{C_{23} \text{ OR } C_{24} \text{ OR } C_{25}\} \text{ THEN } Y \text{ IS } D_4 \tag{R2}$$

Für die Partitionierungen der Variablen X_1, X_2 und Y gelte:

$$\begin{aligned} X_1 &= A_{11}, A_{12}, \ldots, A_{15} \\ X_2 &= A_{21}, A_{22}, \ldots, A_{25} \\ Y &= B_{11}, B_{12}, \ldots, B_{15} \end{aligned}$$

Die Codierung ist:

$$\begin{aligned} \text{R1} &= (11100\,11100\,00001) \\ \text{R2} &= (00111\,00111\,00010) \end{aligned}$$

Anwendung der Schnittregeln ergibt:

$$\begin{aligned} \text{SR1 } \texttt{UND} \text{ R1} &= 11000\,00000 \\ \text{SR2 } \texttt{UND} \text{ R1} &= 00100\,00000 \\ \\ \text{SR1 } \texttt{UND} \text{ R2} &= 00000\,00000 \\ \text{SR2 } \texttt{UND} \text{ R2} &= 00111\,00000 \end{aligned}$$

Damit wird Regel R2 eindeutig dem zweiten Fragment zugeordnet, während Regel R1 nicht eindeutig zugeordnet werden kann.

Wenn die Schnittfläche Regeln zerschneidet, wie im Fall bei Regel R1, dann existieren im Wesentlichen drei Möglichkeiten der Zuordnung zu einem Fragment:

1. Die Regel wird in zwei Regeln aufgeteilt, die sich eindeutig zuordnen lassen. Bei häufiger Anwendung dieses Verfahrens bleiben hauptsächlich elementare Regeln übrig und der Vorteil der reduzierten Regelbasis durch DNF ist dahin.
2. Die Regel wird stochastisch zugeordnet. Die Auswahlwahrscheinlichkeit für ein Fragment ergibt sich aus dem Ergebnis-String folgendermaßen: Die Anzahl der Einsen für das Fragment wird durch die Gesamtanzahl der Einsen dividiert. In obigem Beispiel wird also das erste Fragment mit Wahrscheinlichkeit 2/3 gegenüber dem zweiten Fragment ausgewählt.

3. Die Regel wird deterministisch zugeordnet. Es wird das Fragment ausgewählt das mehr Einsen im Ergebnis-String besitzt. In obigem Beispiel wird R1 dem ersten Fragment zugeordnet, da diesem zwei Einsen zugeordnet sind und dem zweiten Fragment nur eine Eins.

Die Schnittfläche kann auch entlang mehrerer Variablen definiert werden.

Regeln variabler Länge

Bei der Codierung der Regeln mit positionsabhängiger Semantik wächst die Länge des Codes für eine Regel proportional zur Anzahl der Variablen und bei DNF-Einsatz auch zur Anzahl der restringierenden Fuzzy-Mengen.

Bei einer positionsunabhängigen Semantik wird diese Abhängigkeit aufgehoben. Eine Möglichkeit ist die Codierung analog der bei den Messy Genetischen Algorithmen.

Für die Codierung sind die folgenden Schritte zu machen: Jeder linguistischen Variablen muss ein eindeutiger Bezeichner zugewiesen werden, z.B. durch durchnummerieren. Jeder Fuzzy-Menge einer linguistischen Variable muss ein, innerhalb der zugehörigen Partition, eindeutiger Bezeichner zugewiesen werden, z.B. durch durchnummerieren.

Beispiel 4.6
Mögliche Codierung von Fuzzy-Regeln mit Eingabevariablen X_1 und X_2 und Ausgabevariable Y durch durchnummerieren:

$$\begin{array}{lcllcl} X_1 & = & \{A_{11}, A_{12}, A_{13}\} & \longrightarrow & 1 & = & \{1,2,3\} \\ X_2 & = & \{A_{21}, A_{22}, A_{23}\} & \longrightarrow & 2 & = & \{1,2,3\} \\ Y & = & \{B_1, B_2\} & \longrightarrow & 3 & = & \{1,2\} \end{array}$$

Die elementaren Bausteine für die Codierung sind Paare der Form (Variable-Bezeichner, Fuzzy-Menge-Bezeichner), die eine Klausel der Art „Variable IS Fuzzy-Menge“ codieren. Durch die Verkettung dieser Paare lässt sich die gesamte Regel codieren. Hat man eine DNF-Fuzzy-Regel, welche Klauseln wie

$$\text{IF } X_1 \text{ IS } \{A_{11} \text{ OR } A_{12}\}$$

enthält, so werden diese in Einzelklauseln überführt:

$$\text{IF } X_1 \text{ IS } A_{11} \text{ OR } X_1 \text{ IS } A_{12}$$

und diese codiert.

Wenn in einer Fuzzy-Regel bestimmte Variablen nicht verwendet werden, so tauchen sie im Code für diese Fuzzy-Regel nicht auf. Es werden ausschliesslich genau die Variablen

und Fuzzy-Mengen codiert, die auch verwendet werden. Damit ist die Länge des Regelcodes nicht mehr direkt abhängig von der Gesamtanzahl der linguistischen Variablen und der Anzahl der restringierenden Fuzzy-Mengen des Systems. Dies kann zu einer deutlichen Reduzierung der Grösse der Regelbasis führen.

Zur Dekodierung einer so kodierten Regel fasst man alle Tupel mit der gleichen Variable zu einer Disjunktion zusammen. Dann verknüpft man alle erhaltenen Disjunktionen konjunktiv und erhält wieder eine DNF-Fuzzy-Regel.

Beispiel 4.7
Unter Bezug auf die zuletzt gezeigten Beispiele, wird die DNF-Fuzzy-Regel

$$\text{IF } X_1 \text{ IS } \{A_{12} \text{ OR } A_{13}\} \text{ AND } X_2 \text{ IS } \{A_{21} \text{ OR } A_{23}\} \text{ THEN } Y \text{ IS } \{B_1 \text{ OR } B_2\}$$

durch

$$< (1,2)(1,3)(2,1)(2,3)(3,1)(3,2) >$$

codiert. Hierbei ist zu beachten, dass jede Permutation der Paare ebenfalls eine gültige Codierung darstellt.

Genetische Operatoren

Crossover und Mutation folgen dem Schema bei den Messy Genetischen Algorithmen (s. Abschnitt 3.6). Für den Crossover kommt die cut-and-splice-Technik zum Einsatz. Der verwendete Crossover-Operator muss sicherstellen, dass keine ungültigen Regeln entstehen. Weiterhin treten die bei den Messy Genetischen Algorithmen bekannten Probleme Überspezifizierung und Unterspezifizierung auf.

Anpassung der Codierungsformen für Sugeno-Controller

Wird ein Sugeno-Controller verwendet, so besteht die Änderung darin, daß statt einem Bezeichner für eine Ausgabemenge ein Tupel bestehend aus den Parametern der zu der betreffenden Regel gehörigen reellen Ausgabefunktion verwendet wird.

4.2.5 Der Iterative Regellern-Ansatz

Der Iterative Regellern-Ansatz wurde 1993 von Venturini vorgestellt [Venturini93]. Er versucht zur Lösung des CCP die Stärken des Michigan-Ansatzes und des Pittsburgh-Ansatzes zu kombinieren. Dazu wird das Lernen der Regelbasis in zwei Schritte unterteilt:

1. **Regel-Generierungsprozess**
   Im Regel-Generierungsprozess konkurrieren die Fuzzy-Regeln wie im Michigan-Ansatz miteinander. Ziel ist es eine bestmögliche Menge von Fuzzy-Regeln zu erhalten. Dazu wird der Generierungsprozess mehrfach iteriert. In jeder Iteration wird nur die Regel ausgewählt, die auf der aktuellen Trainingsmenge am besten abschneidet.
2. **Nachbearbeitungsprozess**
   Der Regel-Generierungsprozess hat den Kooperationsaspekt zwischen den Fuzzy-Regeln völlig vernachlässigt. Mögliche Folge ist eine unnötig komplizierte und umfangreiche Regelbasis (over-training). Ziel des Nachbearbeitungsprozesses ist die Kooperation zwischen den Fuzzy-Regeln zu erhöhen. Dazu werden redundante und inkonsistente Regeln entfernt. Dann findet ein Genetic Tuning Prozess zur verbesserten Anpassung der Datenbasis an die gefundene Regelbasis statt.

Die Bestimmung der besten Regel erfolgt nach verschiedenen Kriterien. Wichtig sind Regelstärke (Anzahl der abgedeckten Trainingsbeispiele), Regelkonsistenz und Einfachheit einer Regel.

Vorteilhaft ist, dass der Suchraum des Genetischen Algorithmus durch die Einschränkung der Suche auf nur eine beste Regel spürbar verkleinert wird. Dadurch sinkt der benötigte Suchaufwand. Nachteilig ist, dass die Aufgabe, die beste Regel auszuwählen, schwierig ist und aufwändige Mechanismen erfordert. Das iterative Ablaufschema des Iterativen Regellern-Ansatzes lässt sich wie in Abbildung 4.8 darstellen:

1. Benutze einen GA zur Erzeugung einer Regel (Generierungsphase).
2. Füge diese Regel der finalen Regelbasis hinzu.
3. Bestrafe diese Regel, z.B. durch Löschen von ihr abgedeckter Trainingsbeispiele (damit sie nicht noch einmal ausgewählt werden kann)
4. Ist die Güte der finalen Regelbasis ausreichend, dann gehe zu 5., sonst gehe zurück zu 1.
5. Optimiere die Regelbasis (Redundanzverminderung, Konsistenzförderung)
6. Verbessere die Datenbasis (Genetic Tuning)
7. Ende.

Abbildung 4.8: Ablaufschema des Iterativen Regellern-Ansatzes

5 Implementierung

Der Zweck dieser Arbeit ist die Untersuchung, ob sich die Integration von Evolutionären Algorithmen in das MFOS lohnt und diese im positiven Fall durchzuführen.

Hierzu sollen Experimente durchgeführt werden. Diese bestehen darin denselben suboptimalen Fuzzy-Controller einerseits mit den bestehenden MFOS-Implementierungen und andererseits mit Evolutionären Algorithmen zu optimieren.

Um sichere und aussagekräftige Ergebnisse bei den Experimenten zu erhalten, ist eine valide Versuchsumgebung nötig. Das heisst die verwendete Software muss hinreichend fehlerfrei sein. Die zur Verfügung gestellten Implementierungen der MFOS-Systeme erfüllen diese Anforderung nicht. Die Begründung hierfür wird in Abschnitt 5.1.2 gegeben.

Abschnitt 5.1.1 beschreibt kurz das Prinzip der MFOS-Systeme, wobei die technischen Details der zum Einsatz kommenden Neuronalen Netze nicht behandelt werden, hierzu sei auf [Lippe06] und [Niendieck03] verwiesen. Der darauf folgende Abschnitt beschreibt die in den Implementierungen der MFOS-Systeme entdeckten Fehler und die durchgeführten Reparaturen. Im Anschluss daran wird die vorgenommene Implementierung der Optimierung von Fuzzy-Controllern mit Genetischen Algorithmen beschrieben. Die mit der Implementierung durchgeführten Experimente und die dabei erzielten Resultate werden in Kapitel 6 dargestellt.

5.1 Vorhandene Software

5.1.1 MFOS

Die Münsteraner Fuzzy Optimierungssysteme (MFOS) wurden ab Ende der neunziger Jahre an der Universität Münster von Lippe, Niendieck und Tenhagen entwickelt ([Lippe06], [Niendieck03]). Die folgende Darstellung fasst das für diese Arbeit relevante zusammen.

Es existieren zwei Varianten: das MFOS-M für die Optimierung von Mamdani-Controllern

und das MFOS-S für die Optimierung von Sugeno-Controllern.

Das Grundschema für beide Systeme ist dasselbe und wird in Abbildung 5.1 dargestellt. Ein gegebener Fuzzy-Controller C_1 nach Mamdani oder nach Sugeno wird in ein funktional äquivalentes Neuronales Netz NN_1 transformiert. Das Neuronale Netz kann jetzt mittels verschiedener Lernverfahren trainiert werden. Hierdurch werden die verschiedenen Bestandteile des repräsentierten Fuzzy-Controllers modifiziert. Ist eine geeignete Performanz erreicht, wird das trainierte Neuronale Netz NN_2 in einen funktional äquivalenten Fuzzy-Controller C_2 transformiert. Die prinzipiellen Möglichkeiten

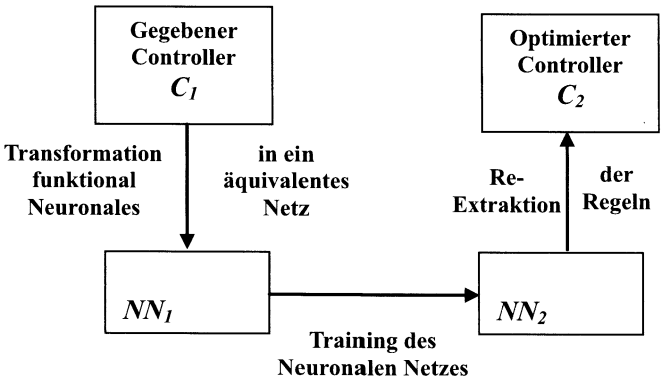


Abbildung 5.1: Arbeitsweise der MFOS-Systeme

einen Fuzzy-Controller zu optimieren sind

- Veränderung bestehender Regeln
- Löschen bestehender Regeln
- Erstellung neuer Regeln
- Veränderung von Fuzzy-Mengen
- Löschen von Fuzzy-Mengen
- Erstellung von neuen Fuzzy-Mengen

Die MFOS-Systeme unterstützen alle diese Verfahren.

Alle Lernverfahren basieren auf der Repräsentation von geeigneten Trainingsbeispielen.

Der Nutzer agiert interaktiv mit MFOS, er entscheidet welche Lernverfahren- und in welcher Reihenfolge diese angewendet werden.

Die Wahl einer geeigneten Reihenfolge der Anwendung der Lernverfahren ist vom jeweiligen Fuzzy-Controller, der zu optimieren ist abhängig und entscheidet maßgeblich

über Erfolg oder Misserfolg der Optimierung. In [Lippe06] werden Empfehlungen zur Vorgehensweise angegeben. Das Finden einer zum Erfolg führenden Reihenfolge der Anwendung der Lernverfahren ist eine potentiell schwierige Aufgabe.

Anforderungen an den zu optimierenden Fuzzy-Controller:
In den MFOS-Systemen stehen Dreiecks-, Gauss-, und Trapezmengen für die Partitionierung von Ein- und Ausgaberäumen zur Verfügung. Für die Fuzzifizierung wird ein Singleton-Fuzzifizierer verwendet. Verschiedene Defuzzifizierungsverfahren wie Schwerpunkt-Methode und Maximumsmethode sind vorhanden. Die Klauseln der Prämisse jeder Regel werden mit `AND` verknüpft. Jede Regel bezieht sich in ihrer Konklusion auf genau eine Ausgabedimension.

5.1.2 Fehler- und Verbesserung der MFOS-Implementierungen

Die für diese Arbeit zur Verfügung gestellten Implementierungen der MFOS-Systeme sind die MFOS-M-Implementierung von Hendrik Brinkschulte [Brinkschulte02] und die MFOS-S-Implementierung von Wolfram Urich [Urich04]. Beide Implementierungen verwenden die Programmiersprache Java von der Firma Sun Microsystems Inc. und können daher auf allen gängigen Betriebssystemen verwendet werden. Beide Varianten können über eine graphische Benutzeroberfläche (GUI) bedient werden.

Die MFOS-M-Implementierung wurde als erstes System näher betrachtet. Bei der Arbeit mit der MFOS-M-Implementierung zeigte sich das die Lernverfahren „Neue Regeln erzeugen“ und „Regeln entfernen“ nicht funktionieren. Weiterhin traten Fehler in der Benutzeroberfläche und den zugrundeliegenden Datenstrukturen auf. Somit ist die MFOS-M-Implementierung zur Durchführung von Experimenten ungeeignet.

Nun wurde die MFOS-S-Implementierung untersucht. Durch die Erfahrungen mit der fehlerhaften Implementierung des MFOS-M sensibilisiert wurde die MFOS-S-Implementierung systematisch und ausführlich auf Fehler geprüft. Da die mitgelieferten Beispiele der MFOS-S-Implementierung nur einfache Situationen darstellen, wurden weitere Testbeispiele entworfen. Es wurden Fehler in den Lernverfahren und den zugrundeliegenden Datenstrukturen festgestellt. Somit ist auch die MFOS-S-Implementierung zur Durchführung von Experimenten ungeeignet.

Nach der Analyse der Quellcodes beider Implementierungen sowie der Testergebnisse wurde entschieden die MFOS-S-Implementierung zu reparieren, da die Reparatur für diese erfolgversprechender erschien.

Es wurde ein intensiver und zeitaufwändiger Reparaturversuch vorgenommen, der schlussendlich abgebrochen wurde, als sich zeigte, daß wesentliche Strukturen des Programms

sich auf ein unzureichendes Konzept bzgl. der Nachbarschaftsbeziehung von Partitionsmengen und der Speicherung der Partitionsmengen stützen und daher bei Reparatur dieser Fehler neu implementiert werden müssten.

Daraufhin wurde ein intensiver und zeitaufwändiger Reparaturversuch an der MFOS-M-Implementierung vorgenommen.

Es konnte eine Version der MFOS-M-Implementierung hergestellt werden, die eine hinreichende Basis für die Implementierung der Erweiterung um die Möglichkeit der Optimierung mit Genetischen Algorithmen darstellt.

Es folgt eine genauere Beschreibung der entdeckten Fehler und durchgeführten Reparaturen.

Fehler in der MFOS-M-Implementierung

Bei der Erprobung der Lernverfahren zeigte sich, dass das Lernverfahren „Neue Regeln erzeugen“ nicht funktioniert. Die Konsolenausgabe meldet „`Array Index Out of Bounds Exception`“. Ein Beispiel zur Reproduzierung des Fehlers ist auf der zu dieser Arbeit gehörigen CD-ROM enthalten. Der Pfad des Beispiels ist \mfos_m\Mfos_m_Fehler_Beispiele\ Neue Regeln erzeugen .

Auch das Verfahren zum „Entfernen von Regeln“ funktioniert nicht: Erzeugt man einen Controller mit einer nicht-redundanten Regelbasis und generiert hierzu mittels der Funktion „Trainingsdaten Erzeugen“ Trainingsdaten und wendet dann „Entfernen von Regeln an“, so werden im Beispiel auf der CDROM von den 23 nötigen Regeln, 22 gelöscht. Es dürfte aber keine einzige gelöscht werden. Der Pfad des Beispiels ist \mfos_m\Mfos_m_Fehler_Beispiele\Regeln entfernen .

Die Lernverfahren für das MFOS-M sind in der Klasse `Training.java` implementiert. Das Lernverfahren „Neue Regeln erzeugen“ ist in der Methode `public static int regelErgaenzung` implementiert. Bei der Analyse des Fehlers zeigte sich, daß die Logik des Algorithmus für „Neue Regeln erzeugen“ völlig falsch implementiert wurde. Das Lernverfahren „Entfernen von Regeln“ ist in der Methode `public static int regelLoeschen` implementiert.

Eine detaillierte Schilderung aller gefundenen Fehler, die in der bestehenden Implementierung für diese Lernverfahren und allgemein in der Implementierung des MFOS-M auftreten, erscheint mir hier nicht sinnvoll. Der fachkundige Leser kann den Vergleich der in ([Lippe06], Kapitel 5.2.4) detailliert beschriebenen Algorithmen und die fehlerhafte Umsetzung anhand der Dateien auf der CD-ROM selbst ersehen. Der Pfad des ursprünglichen MFOS-M auf der CD ist \mfos_m\Mfos_m_alt . Das Programm kann dort durch

Aufruf von mfos_m_alt_start.bat gestartet werden.

Reparatur und Verbesserung der MFOS-M-Implementierung

Das Lernverfahren „Neue Regeln erzeugen“ war so fehlerhaft implementiert, dass eine Reparatur nicht sinnvoll war. Daher wurde der Algorithmus nach [Lippe06] erfolgreich neu implementiert. Hierbei wurde darauf geachtet, daß die Implementierung den zugrundeliegenden Algorithmus gut erkennen läßt. Die verbesserte Version der MFOS-M-Implementierung befindet sich auf der CDROM im Verzeichnis \mfos_m\Mfos_m_verbessert . Das Programm kann von dort durch Aufruf von mfos_m_verbessert_start.bat gestartet werden.

Während der Reparatur traten auch in den zugrundeliegenden Datenstrukturen und den darauf operierenden Verwaltungsfunktionen des MFOS-M Fehler zutage, meistens der `Array Index Out Of Bounds`-Fehler und die `Null Pointer Exception`. So etwa in der Klasse `DreieckPart.java` in der Methode `public void Einfuegen` und beim Löschen der letzten Regel einer Regelbasis (`private void createRegelbasisPanel()`). Die Reparaturen dieser beiden Fehler sind mit dem Tag `//FIXED` im Sourcecode markiert.

Die Erstellung von neuen Datensätzen ist fehlerhaft implementiert. Erzeugt man einen neuen Datensatz, so kann man diesen den Trainingsdaten nicht hinzufügen, da der Menüpunkt nicht anwählbar ist. Nachdem dies behoben war, funktioniert das Hinzufügen weiterhin nicht.

Das Löschen des letzten Datensatzes führte zur `Array Index Out of Bounds Exception`.

Es war nicht möglich einen Controller mit mehr als einer Ausgabedimension zu verwenden, da nach dem Speichern von Eingabepartitionierung, Ausgabepartitionierung und Regelbasis das Laden dieser Komponenten nicht möglich ist. Beim Versuch die gespeicherte Regelbasis zu laden erschien die Meldung, dass die Regelbasis gelöscht wurde, da das Modell nicht konsistent gewesen sei.

Fehler in- und Reparaturen an der MFOS-S-Implementierung

Die zur Verfügung gestellte Version der MFOS-S-Implementierung ist auf der CD-ROM im Verzeichnis \mfos_s\mfos_s_alt\JarFileGUIDeutsch gespeichert. Das Programm kann von dort durch Aufruf von mfos_s_start.bat gestartet werden. Der zugehörige Sourcecode ist im Verzeichnis \mfos_s\mfos_s_alt\SourceCodeGUI gespeichert. Sie enthält Fehler, von denen manche als „nur“ die Benutzung des Programms erschwerend andere

aber als schwerwiegend, d.h. die Untauglichkeit der Implementierung für diese Arbeit verursachend, erkannt wurden.

Die aktuelle verbesserte Version des MFOS-S ist auf der CD-ROM im Verzeichnis \mfos_s\mfos_s_verbessert gespeichert. Das Programm kann von dort durch Aufruf von mfos_s_verbessert.bat gestartet werden.

In ihr wurden einige Fehler erfolgreich behoben. Im folgenden werden die Details zu den Fehlern und den Reparaturen aufgeführt.

Es wurde eine einheitliche Kennzeichnungsweise und Klassifizierung für gefundene Fehler (Bugs) entwickelt: Reparaturen sind im Quelltext durch:

```
//FIXED : BUG n : Relevanz
//Beschreibung des Fehlers
```

gekennzeichnet.

Hierbei gilt:

$n \in \mathbb{N}$
und
Relevanz $\in$ {NK, PK, K!}

wobei die folgenden Bezeichnungen gelten:

NK	für nicht kritische Fehler, z.B. ineffiziente Benutzbarkeit des Programms, aufgrund eines GUI-Fehlers
PK	für potentiell kritische Fehler es konnten während der Tests keine Funktionsstörungen festgestellt werden, aber aufgrund von möglichen Seiteneffekten können diese auch nicht ausgeschlossen werden.
K!	für kritische Fehler eindeutig zu beobachtende Fehlfunktionen des Programms, wie z.B. Fehler in Algorithmen

Nicht kritische Fehler

BUG 0 Die Regelbasisview wurde nicht angezeigt, wenn sie angezeigt hätte werden müssen.

BUG 3 und BUG 5 Die DatensätzeView wurde nicht aktualisiert.

Kritische Fehler

BUG 1 Es kann nur 1 Ausgabedimension verwendet werden, da die Werte welche für eine weitere Dimension eingestellt werden nicht gespeichert werden.

BUG 4 und BUG 7 Betrifft die Klasse EntferneReg.java.
Mehrfaches Löschen von Regeln (falsche IF-Bedingungen), mehrfache Anzeige alter und modifizierter Regeln behoben. Ein Beispiel hierfür ist RegelEntfernen, welches im Verzeichnis \mfos_s\mfos_s_Fehler_Beispiele auf der CD-ROM gespeichert ist. Der Mechanismus ist weiterhin fehlerhaft.Es treten Unregelmäßigkeiten beim Löschen von Regeln auf, sowohl was die Anzahl von tatsächlich gelöschten Regeln, als auch die Meldungen des Programms was angeblich gelöscht wurde, angeht.

BUG 6 Die Implementierung des Lernverfahrens „Partitionsmengen zusammenfügen“ ist fehlerhaft realisiert. Dies belegen die Beispiele
PartitionsmengenZusammenfassen_2_Mengen_zusammenfassbar
und
PartitionsmengenZusammenfassen_4_Mengen_zusammenfassbar
die sich auf der CD-ROM im Verzeichnis
\mfos_s\mfos_s_Fehler_Beispiele befinden.

Es werden stets alle Partitionsmengen zusammengefaßt und nicht ausschließlich diejenigen, die nach dem Algorithmus in [Lippe06] zusammengefaßt werden dürfen.

In der verbesserten Version des MFOS-S wurde dieser Fehler behoben. Dies belegen die oben genannten Beispiele, bei welchen das alte MFOS-S versagt.

Die bestehende Implementierung des Lernverfahrens, welches in der Klasse Zusammen.java implementiert ist, wurde nach intensiver Analyse, als nicht-reparierbar erkannt. Dies gründet sich auf die Feststellung, dass die Struktur des Quelltextes den zugrundeliegenden Algorithmus schwer erkennen läßt. Anhand des in [Lippe06] beschriebenen Algorithmus wurde das Lernverfahren erfolgreich neu implementiert.

Dann wurde jedoch erkannt, daß die alte MFOS-S-Implementierung die Nachbarschaft von zwei Partitionsmengen dann und nur dann als gegeben ansieht, wenn der Benutzer diese unmittelbar hintereinander erzeugt hat. Zur Korrektur dieses Fehlers wurde eine Methode `ordnePartitionsmengenNachDirekterNachbarschaftAn` entwickelt, die die Nachbarschaft von Partitionsmengen gemäß [Lippe06] umsetzt. Die Methode befindet sich in der Implementierung des MFOS-S in der Klasse Dreieckpart.java. Es stellte sich dann jedoch heraus, daß der Fehler nicht nur lokaler, sondern globaler Natur ist. Denn wesentliche Komponenten des Programms, wie die Anpassung von Regeln nach dem Zusammenfügen von Partitionsmengen oder das Einfügen von Partitionsmengen, basieren auf der unzureichenden Nachbarschaftsimplementierung. Daher ist die Methode, ebenso wie ihre Ausführungen, auskommentiert, da die verbesserte MFOS-S-Implementierung sonst nicht zur Verifikation der vorher behobenen Fehler verwendet werden könnte.

In Anbetracht des übermäßigen Aufwandes wurde die Reparatur der MFOS-S-Implementierung abgebrochen.

Konsequenzen der Fehler in den MFOS-Implementierungen

Diese Arbeit besteht darin die Nützlichkeit von Evolutionären Algorithmen für die Optimierung von Fuzzy-Controllern zu erforschen, die Effektivität des bestehenden MFOS festzustellen und ggf. eine Integration der Evolutionären Algorithmen in die bestehende MFOS-Implementierung vorzunehmen.

Dazu sollten Experimente mit den bestehenden MFOS-Implementierungen durchgeführt werden, um die Effektivität von MFOS beurteilen zu können und Hinweise für die Integration von Evolutionären Algorithmen zu erhalten.

Aufgrund der festgestellten Fehler in den bestehenden MFOS-Implementierungen konnten diese Experimente nicht durchgeführt werden. Es wurde versucht mindestens eine der beiden Implementierungen vollständig zu reparieren, um die Experimente durchführen zu können. Dieses Ziel konnte aufgrund des zeitlichen Rahmens dieser Diplomarbeit nicht erreicht werden.

Es wurde eine Version der MFOS-M-Implementierung erschaffen, welche als Basis für die Implementierung der Erweiterung um die Möglichkeit der Optimierung mit Genetischen Algorithmen hinreichend ist. Dies gilt für die Einschränkung der Partitionsmengen auf Dreiecksmengen.

Die Reparaturversuche waren sehr zeitaufwändig. Es war notwendig die Lernverfahren des MFOS genau zu studieren. Die verbesserten Implementierungen der MFOS-Implementierungen wurden um erläuternde Kommentare angereichert und können als Basis für eine vollständige Reparatur dienen.

Diese Arbeit besteht wesentlich in der Erarbeitung der Theorie der Optimierung von Fuzzy-Controllern mit Evolutionären Algorithmen und einer anschließenden Implementierung. Mit dieser Implementierung sollen Experimente durchgeführt werden, um die Effektivität der Verfahren zu erkennen.

Die noch verbleibende Zeit für die Bearbeitung der Diplomarbeit wurde diesen Aufgaben gewidmet.

5.1.3 JGAP

Für die Implementierung des Genetischen Algorithmus wird JGAP (Java Genetic Algorithm Package) in der Version 3.0 Final verwendet. JGAP ist ein in JAVA programmiertes Framework zur Unterstützung der Programmierung von Genetischen Algorithmen

und Genetischer Programmierung in JAVA. Die wesentlichen Klassen für Genetische Algorithmen zeigt Abbildung 5.2.

JGAP ist unter http://jgap.sourceforge.net beziehbar. Es handelt sich um freie Software gemäß der GNU Lesser Public License.

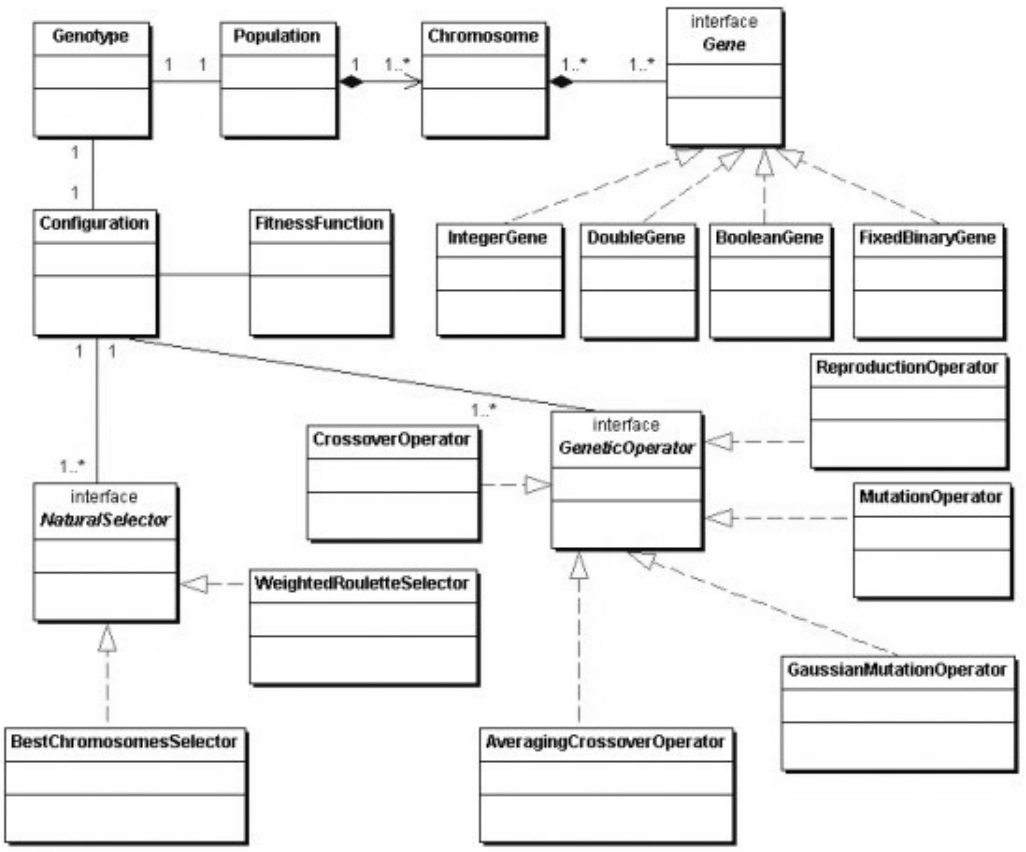


Abbildung 5.2: JGAP-Klassendiagramm

5.1.4 JFreeChart

Für die Realisierung der graphischen Anzeige des MSE-Verlaufs in der Ergebnisanzeige für die Optimierung mit Genetischen Algorithmen wird JFreeChart in der Version 1.0.2 verwendet. JFreeChart ist ein in JAVA programmiertes Framework zur Generierung von Plots und Diagrammen in JAVA.

JFreeChart ist unter http://www.jfree.org/jfreechart beziehbar. Es handelt sich um freie Software gemäß der GNU Lesser Public License.

5.2 Erweiterung der MFOS-M-Implementierung um die Möglichkeit der Optimierung mit Genetischen Algorithmen

Es werden die wesentlichen Zusammenhänge und Strukturen der Erweiterung der MFOS-M-Implementierung um die Möglichkeit der Optimierung mit Genetischen Algorithmen dargestellt.

Die interne MFOS-M-Hilfe wurde um die Anleitung zur Optimierung mit Genetischen Algorithmen in MFOS-M erweitert. Es wurde eine Dokumentation des Progamm-Codes mittels JavaDoc erzeugt. Beide Dokumente befinden sich auf der CD-ROM in Unterverzeichnissen des Verzeichnis mfos_m\MFOS_M_GA.

5.2.1 Implementierung des Genetischen Algorithmus

Für die Implementierung des Genetischen Algorithmus für die Regelbasisoptimierung eines Fuzzy-Controllers wird der Pittsburgh-Ansatz verwendet (s. Abschnitt 4.2.4).

Die implementierten Selektionsverfahren werden in Abschnitt 3.5.3 beschrieben. Die Beschreibungen der implementierten Ansätze für die Codierung und die genetischen Operatoren finden sich in Abschnitt 4.2.4.

- **Codierung**
  Die Codierung basiert auf der Repräsentation der Regelbasis durch eine Entscheidungstabelle. Es wird mit ganzzahliger Codierung gearbeitet.
- **Genetische Operatoren**
  - **Selektion**
    Für die Selektion stehen die Verfahren Roulette-Selektion, Stochastic-Universal-Sampling, Turnierselektion, exponentielle Selektion und exponentielle Turnierselektion zur Verfügung.
  - **Crossover**
    Für den Crossover wurden Standard-1-Punkt- und 2-Punkt-Crossover sowie funktionaler Crossover realisiert.
  - **Mutation**
    Für die Mutation wurden Nachbarschafts-Mutation und gleichverteilte Mutation implementiert.
- **Abbruchkriterien**
  Es wurden die folgenden Abbruchkriterien implementiert:

- maximale Anzahl von Generationen
- maximale Zeitdauer
- maximale Verbesserung gegenüber dem Start-Controller
- minimaler MSE
- maximale Anzahl direkt aufeinanderfolgender Generationen ohne Fitnessverbesserung

Der implementierte Genetische Algorithmus orientiert sich am kanonischen Genetischen Algorithmus (s. Abbildung 3.2) und ist in Abbildung 5.3 in Pseudo-Code dargestellt.

```
1   begin
2     generiere StartPopulation P;
3     while (Abbruchkriterium nicht erfüllt) do
4       P := P + crossover(P, crossoverRate);
5       P := P + mutation(P, mutationsRate);
6       berechne Fitness der Individuen von P;
7       P := selektion(P, maxPopulationsgroesse);
8     end while
9   end.
```

Abbildung 5.3: Pseudo-Code des implementierten Genetischen Algorithmus

Erklärungen zum Pseudo-Code in Abbildung 5.3

Vor dem Start des Programms müssen die Parameter crossoverRate, mutationsRate und maxPopulationsgroesse definiert werden. Hierbei muß gelten:

$$\begin{array}{ll} \text{crossoverRate} \in \mathbb{N}, & \text{crossoverRate} \geq 1 \\ \text{mutationsRate} \in \mathbb{N}, & \text{mutationsRate} \geq 0 \\ \text{maxPopulationsgroesse} \in \mathbb{N}, & \text{maxPopulationsgroesse} \geq 1 \end{array}$$

In Zeile 2 wird die Startpopulation des GA per Zufall erzeugt. Ist im MFOS-M eine Regelbasis geladen, so wird diese in die Startpopulation übernommen.

In Zeile 4 werden die durch Crossover erzeugten Kinder der bestehenden Population P hinzugefügt. Es werden k-mal zwei Individuen zufällig gleichverteilt aus der Population ausgewählt und gekreuzt, wobei $k = \lceil \frac{\text{populationsgroesse}}{\text{crossoverRate}} \rceil$ ist.

In Zeile 5 wird auf alle Chromosome der Population der Mutationsoperator angewendet. Ein Gen eines Chromosoms wird mit Wahrscheinlichkeit $\frac{1}{mutationsRate}$ mutiert. Es werden die mutierten Chromosome der bestehenden Population P hinzugefügt.

In Zeile 7 werden die besten maxPopulationsgroesse-viele Chromosome von P für die nächste Generation selektiert.

5.2.2 Dokumentation des Programm-Codes

Der Programm-Code der Erweiterung befindet sich auf der CD-ROM im Verzeichnis mfos_m\MFOS_M_GA\src. Als Basis für die Erweiterung wurde die verbesserte Version der MFOS-M-Implementierung verwendet. Während der Entwicklung wurden weitere Fehler korrigiert und Verbesserungen gemacht. Die Verbesserungen beinhalten Bereiche wie das Abfangen von ungültigen Benutzereingaben, Restrukturierung, Vereinfachung von Codeteilen und Erweiterungen von Funktionalität der Klassen.

Die zu Beginn der Erweiterung vorhandenen Klassen der MFOS-M-Implementierung sowie neu erstellte Klassen, die nicht direkt für die Optimierung mit Genetischen Algorithmen dienen, wurden in das Paket mfos_m zusammengefasst. Die neu erstellten Klassen für die GA-Erweiterung befinden sich im Paket mfos_m.ga.

Es wurde bei der Programmierung darauf geachtet die Java-Syntax-Konventionen einzuhalten. Für die Verständlichkeit des Codes für zukünftige Erweiterungen und Wartungsarbeiten ist durch „selbsterklärende“ Bezeichner und erläuternde Kommentare gesorgt.

Einen Überblick über die wesentlichen Assoziationen zwischen den Objekten der Erweiterung zeigt das Klassendiagramm in Abbildung 5.4.

Paket mfos_m.ga

Schnittstellen:

- **DBTransformator**
  Transformator für Datenbasen
- **GAFuerDieDBOptimierungInMFOS**
  Schnittstelle für Genetische Algorithmen die eine Datenbasis des MFOS optimieren sollen.
- **GAFuerDieRBOptimierungInMFOS**
  Schnittstelle für Genetische Algorithmen die eine Regelbasis des MFOS optimieren sollen.
- **RBTransformator**
  Transformator für Regelbasen

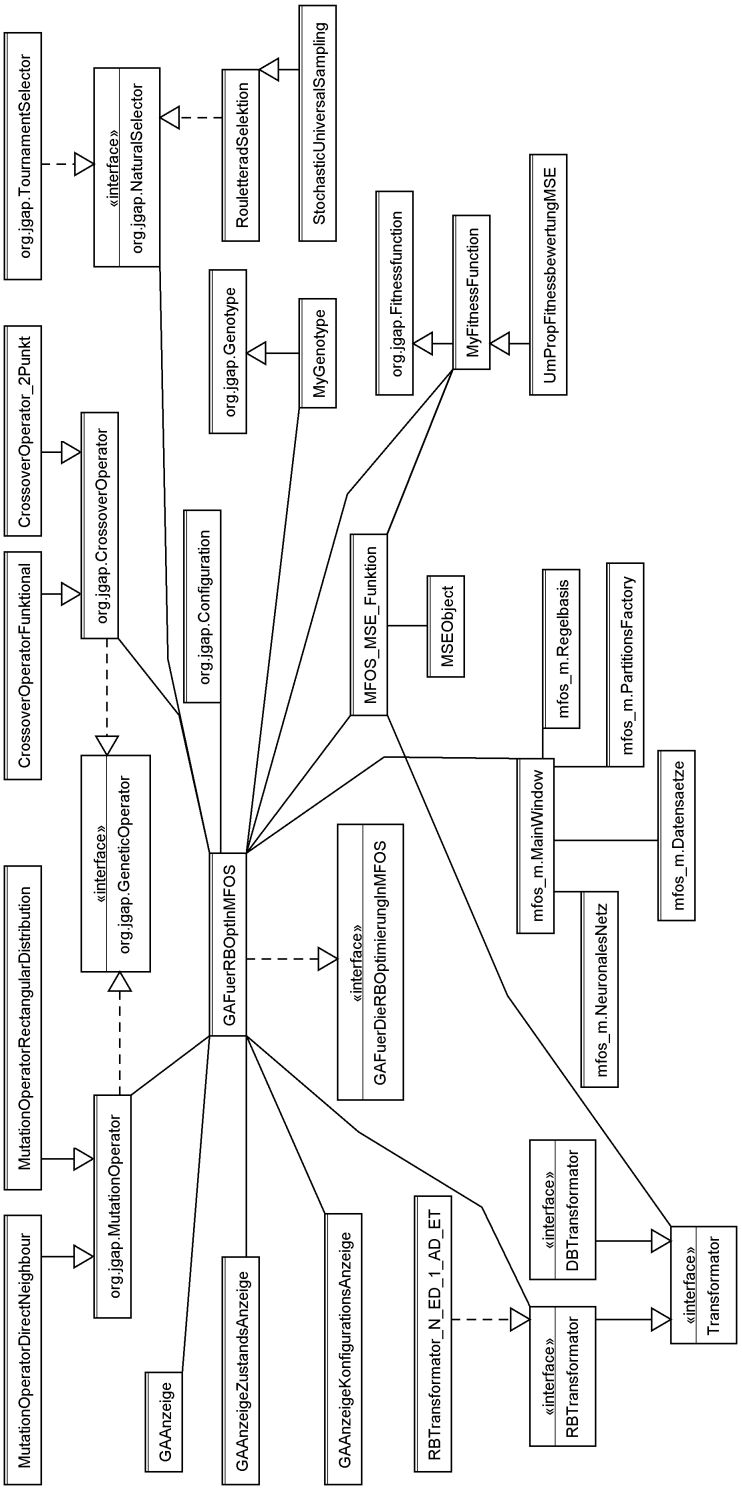


Abbildung 5.4: Klassendiagramm der wesentlichen Klassen der MFOS-M-Erweiterung

- **Transformator**
  Ein Transformator transformiert eine Komponente des MFOS in ein Chromosom und zurück.

Klassen:

- **CrossoverOperator_2Punkt**
  Standard 2 Punkt-Crossover
- **CrossoverOperatorFunktional**
  Funktionaler 1- und 2-Punkt-Crossover gemäß Abschnitt 4.2.4 für beliebig viele Eingabedimensionen.
- **Datenbasis**
  Die im MFOS vorhandenen Eingabe- und Ausgabepartitionen werden in einen Vector gesammelt: erst alle Eingabedimensionen, dann alle Ausgabedimensionen.
- **GAAnzeige**
  Genetischer Algorithmus Ergebnisdatenanzeige.
- **GAAnzeigeAusDimAuswahlDialog**
  Abfrage der Ausgabedimensionen, die der GA optimieren soll.
- **GAAnzeigeContainerPanel**
  Panel für die Anzeige der Ergebnisse eines Genetischen Algorithmus in einem Tab im MFOS.
- **GAAnzeigeKonfigurationsAnzeige**
  Die Konfigurationsdatenverwaltung und -Anzeige für den Genetischen Algorithmus.
- **GAAnzeigeZustandsAnzeige**
  Zustandsanzeigefenster während eines Laufs eines Genetischen Algorithmus.
- **GAFuerRBOptInMFOS**
  Genetischer Algorithmus für die Regelbasis-Optimierung in MFOS.
- **MFOS_MSE_Funktion**
  Zielfunktion, die mit der aktuellen Konfiguration der Inferenzlogik des MFOS den MSE berechnet und diesen der Fitnessfunktion übergibt.
- **MSEObject**
  Datenobjekt für ein Chromosom, welches den zu diesem Chromosom zugehörigen MSE speichert.
- **MutationOperatorDirectNeighbour**
  Mutationsoperator, welcher bei vorliegen eines NIL-Wertes, also für diese Eingabekombination von Eingabepartitionsmengen ex. keine Regel, eine AusgabeFuzzy-Menge gleichverteilt aus den verfügbaren Ausgabepartitionsmengen wählt und der sonst die direkte Nachbarpartitionsmenge auswählt.

- **MutationOperatorRectangularDistribution**
  Mutationsoperator der gleichverteilte Mutation über alle möglichen Ausgabepartitionsmengen und den NIL-Wert realisiert (NIL-Wert repräsentiert die Situation, dass keine Regel für diese Kombination von Eingabepartitionsmengen vorhanden ist).
- **MyFitnessFunction**
  Anpassung der JGAP-Fitnessfunction-Klasse.
- **MyGenotype**
  Anpassung der JGAP-Genotype-Klasse, insbes. evolve() für MFOS-Optimierung angepasst.
- **RBTransformator _N_ED_1 _AD_ET**
  Die Regelbasis des MFOS wird in eine vollständige Entscheidungstabelle überführt und diese dann in ein Chromosom (Gen-Array) geschrieben.
- **RouletteradSelektion**
  Roulette-Rad-Selektionsverfahren
- **StochasticUniversalSampling**
  StochasticUniversalSampling (SUS) - Selektionsverfahren
- **UmPropFitnessbewertungMSE**
  Fitnessfunktion, die einem Chromosom eine Fitness umgekehrt proportional zum Zielfunktionswert des Chromosoms (= MSE) zuweist.

Ausnahmen (Exceptions):

- **InvaliderZustandException**
  Exception, die bei vorliegen eines invaliden Zustandes geworfen wird.

- **MeldungException**
  Exception die ausgelöst wird, wenn ein Informationsfenster angezeigt werden soll.

- **UngueltigeEingabeException**
  Exception, die bei ungültiger Eingabe geworfen wird.

Paket mfos_m

Von den Erweiterungen der bestehenden Klassen sind folgende besonders wichtig:

Klasse: MainWindow

- **Seperater Thread für den Genetischen Algorithmus**
  Der Genetische Algorithmus wird in einem seperaten Thread ausgeführt. Dies

bringt erheblich höhere Performanz. Der notwendige Synchronisationsmechanismus zwischen dem Haupt-Thread und dem GA-Thread beruht im wesentlichen auf dem Zusammenspiel der folgenden Methoden:

- rbOptInitialisieren()
- rbOptStarten()
- rbOptFertig()

und zugehöriger globaler Attribute:

- private Thread gaThread
- private boolean gaWurdeAbgebrochen
- private boolean konfigDialogAbgebrochen

- **Optimierung von Controllern mit mehr als einer Ausgabevariablen**
  Sollen mehr als eine Ausgabevariable optimiert werden, so werden aus dem bestehenden Controller soviele *Sub-Controller* erzeugt wie Ausgabevariablen für die Optimierung ausgewählt wurden.

  Ein Sub-Controller ist die Einschränkung des Gesamt-Controllers auf eine bestimmte Ausgabedimension. Er besitzt also diegleichen Partitionierungen und Datensätze wie der Gesamt-Controller; die Regelbasis beinhaltet jedoch ausschließlich genau die Regeln der Gesamt-Controller-Regelbasis, die in ihrer Konklusion die betreffende Ausgabedimension haben.

  Nach der Generierung der Sub-Controller werden diese sukzessive durch Anwendung des Genetischen Algorithmus optimiert. Nach der Optimierung des letzen Sub-Controllers werden die optimierten Sub-Controller-Regelbasen zu einer optimierten Gesamt-Regelbasis vereinigt.

  Die für diesen Mechanismus entscheidenden Methoden sind:

  - erzeugeSubController(boolean[])
    - erzeugeSubControllerRB(int)
    - erzeugeSubControllerDS(int)
    - erzeugeSubControllerAP(int)
  - aktviereSubController(int)
    - aktiviereEingabepartition(PartitionsFactory)
    - aktiviereAusgabepartition(PartitionsFactory)
    - aktiviereRegelbasis(Regelbasis)
    - aktiviereDatensaetze(Datensaetze)

  und zugehöriger globaler Attribute:

  - private boolean letzteSubControllerOptimierung
  - private int anzahlZuOptimierenderSubController
  - private boolean esGibtLeereSubContrRB
  - private int zuOptimierenderSubController
  - private Vector<Regelbasis> subControllerRB

- private Vector<Regelbasis> subControllerRBOriginalRegeln
- private Vector<PartitionsFactory> subControllerAP
- private Vector<Datensaetze> subControllerDS
- private Vector<Integer> aktOptAusDim
- private PartitionsFactory gaGesamtControllerAP
- private Regelbasis gaGesamtControllerRB
- private Datensaetze gaGesamtControllerDS

Klasse: Datensaetze

- **Berechnung und Anzeige des mittleren quadratischen Fehlers (MSE)**
  Die für diesen Mechanismus entscheidenden Methoden sind:
  - getMSE()
  - updateMSE()
  - berechneMSE()
- **Generierung des Kreuzprodukts (kartesischen Produkts)**

1. Die Generierung des kartesischen Produkts spielt bei der Transformation einer Regelbasis des MFOS-M in eine Entscheidungstabelle, welche dann in ein Chromosom codiert wird, eine entscheidende Rolle. Die für diesen Mechanismus entscheidenden Methoden sind:
   - erzeugeKreuzprodukt(int[][], int[])
   - erzeugeKreuzprodukt(int[], int[])
   - erzeugeKreuzprodukt(int[])
2. Die Funktion *Traingsdaten erzeugen*, die im MFOS-M über das Datei-Menü erreichbar ist, wurde wesentlich erweitert. Jetzt können zusätzlich zur Generierung von Trainingsdaten aus den Modalwerten der definierten Partitionsmengen Trainingsdaten aus dem kartesischen Produkt von äquidistant über die Definitionsbereiche der Partitionierungen verteilten Stützstellen generiert werden. Eine Anleitung zur Vorgehensweise wird im internen MFOS-M-Handbuch in Abschnitt 5.1.5 „Trainingsdaten erzeugen“ gegeben.

   Die für diesen Mechanismus entscheidenden Methoden sind:
   - datensaetzeGenerieren(Regelbasis, PartitionsFactory, PartitionsFactory, int[], float[], float[], boolean, boolean)
   - erzeugeKreuzprodukt(float[][], float[])
   - erzeugeKreuzprodukt(float[], float[])
   - erzeugeKreuzprodukt(float[])

5.3 Anleitung zur Optimierung mit Genetischen Algorithmen in MFOS-M

5.3.1 Das Genetische Algorithmen - Menü

Im Genetische Algorithmen-Menü kann ein Genetischer Algorithmus zur Optimierung des Fuzzy-Controllers gestartet werden (s. Abbildung 5.5).

Das Genetische Algorithmen-Menü wird erst anwählbar, wenn die Voraussetzungen für eine Optimierung mit einem Genetischem Algorithmus vorliegen. Dies ist der Fall, wenn Eingabepartition, Ausgabepartition und Trainingsdaten geladen oder erzeugt sind.

Abbildung 5.5: Das Genetische Algorithmen - Menü

5.3.2 Auswahl von zu optimierenden Ausgabevariablen

Nach Auswahl von *Regelbasis optimieren* erscheint bei Existenz von mehr als einer Ausgabedimension ein Dialogfenster, in welchem die zu optimierenden Ausgabedimensionen ausgewählt werden können (s. Abbildung 5.6).

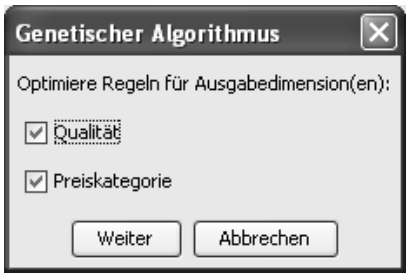


Abbildung 5.6: Auswahl zu optimierender Ausgabedimensionen

5.3.3 Konfiguration des Genetischen Algorithmus

Im Konfigurationsfenster können die benötigten Einstellungen für den Genetischen Algorithmus vorgenommen werden (s. Abbildung 5.7).

Crossover Für den Crossover stehen die folgenden Verfahren zur Verfügung:

- 1-Punkt-Crossover
- 2-Punkt-Crossover
- Funktionaler Crossover gemäß Abschnitt 4.2.4 für beliebig viele Eingabedimensionen.
  - funktionaler 1-Punkt-Crossover
  - funktionaler 2-Punkt-Crossover

Mutation Für die Mutation stehen die folgenden Verfahren zur Verfügung:

- Gleichverteilte Mutation
- Nachbarschafts-Mutation

Selektion Für die Selektion stehen die folgenden Verfahren zur Verfügung:

- Roulette-Rad-Selektion
- Stochastic Universal Sampling (SUS)
- Turnier-Selektion
- Exponentielle Selektion
- Exponentielle Turnier-Selektion

Crossoverrate Für die Crossoverrate ist eine natürliche Zahl zu definieren. Es wird k mal Crossover durchgeführt, wobei k der Wert der Division von Populationsgrösse durch Crossoverrate ist.

Mutationsrate Für die Mutationsrate ist eine natürliche Zahl m zu definieren. Bei jedem Chromosom der Population wird jedes Gen mit Wahrscheinlichkeit $\frac{1}{m}$ mutiert.

Die eingestellten Werte bleiben in der Konfigurationsanzeige gespeichert. Dies ermöglicht die effiziente Wiederholung des gleichen, ggf. abgeänderten, Versuchs.

5.3.4 Zustandsanzeige des Genetischen Algorithmus

Während des Laufs eines Genetischen Algorithmus wird eine Zustandsanzeige angezeigt, welche Auskunft über den aktuellen Zustand der Optimierung mit dem Genetischen Algorithmus gibt (s. Abbildung 5.8).

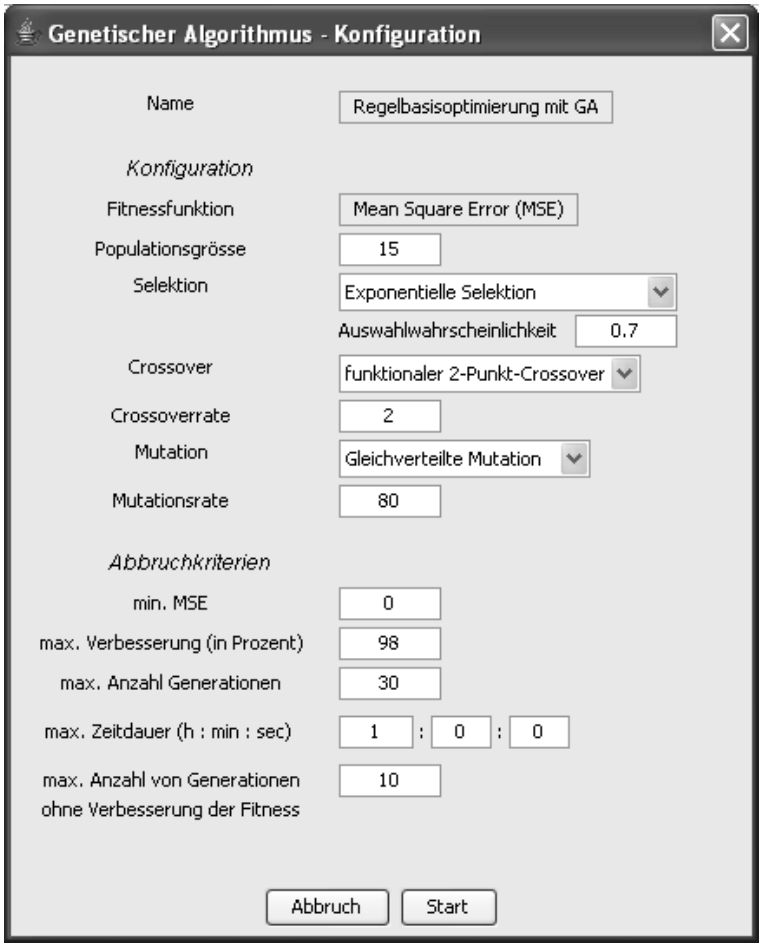


Abbildung 5.7: Konfiguration des Genetischen Algorithmus

Die Ausführung des Genetischen Algorithmus kann jederzeit über den Beenden-Knopf abgebrochen werden. Wird der Genetische Algorithmus über den Beenden-Knopf abgebrochen, so wird der aktuelle Zyklus des Genetischen Algorithmus zuende geführt und dann der Genetische Algorithmus beendet. Das bis dahin erzielte Ergebnis der Optimierung wird in der Ergebnisanzeige des Genetischen Algorithmus angezeigt (s. Abschnitt 5.3.5).

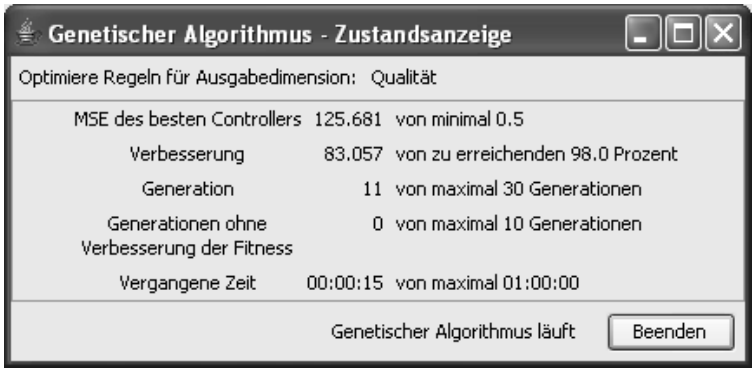


Abbildung 5.8: Zustandsanzeige des Genetischen Algorithmus

5.3.5 Ergebnisanzeige des Genetischen Algorithmus

Das Ergebnis der Optimierung mit dem Genetischen Algorithmus wird in der Registerkarte Genetischer Algorithmus dargestellt (s. Abbildung 5.10).

Das Abbruchkriterium, welches den Abbruch des Genetischen Algorithmus verursacht hat, wird durch Einfärbung in türkis farblich hervorgehoben.

In der Grafik, welche den MSE-Verlauf darstellt, kann ein Bereich mit der Maus ausgewählt werden, welcher dann vergrößert dargestellt wird. Dazu ist so vorzugehen: Man hält die linke Maustaste gedrückt und zieht ein Auswahlrechteck von linker oberer Ecke zu rechter unterer Ecke über dem gewünschten Bereich auf. Durch Bewegen der Maus bei gedrückter linker Maustaste nach links wird die Darstellung wieder verkleinert dargestellt.

Die Koordinaten eines Datenpunkts werden durch Positionieren des Mauszeigers auf diesen nach kurzer Zeit per Popup-Fenster eingeblendet.

Durch Klicken mit der linken Maustaste auf einen Datenpunkt wird ein Fadenkreuz mit dem Datenpunkt als Mittelpunkt eingeblendet.

Durch Klicken mit der rechten Maustaste in einer beliebigen Stelle der Grafik, wird ein Kontextmenü geöffnet. Dieses bietet Funktionen zur Manipulation der Grafik. Auch kann die Grafik abgespeichert werden.

5.3.6 Export der Ergebnisanzeige des Genetischen Algorithmus in HTML

Über die Funktion *Export in HTML-Datei*, die über die Toolbar und über das Datei-Menü erreicht werden kann, können die Ergebnisdaten in HTML exportiert werden (s. Abbildung 5.9).

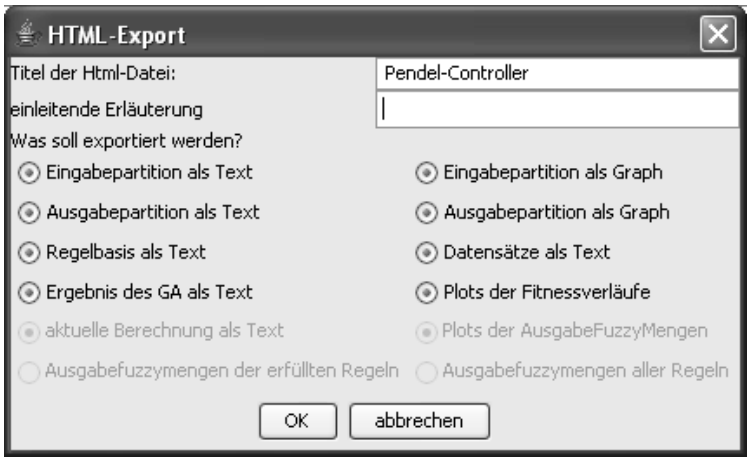


Abbildung 5.9: Export in HTML-Datei

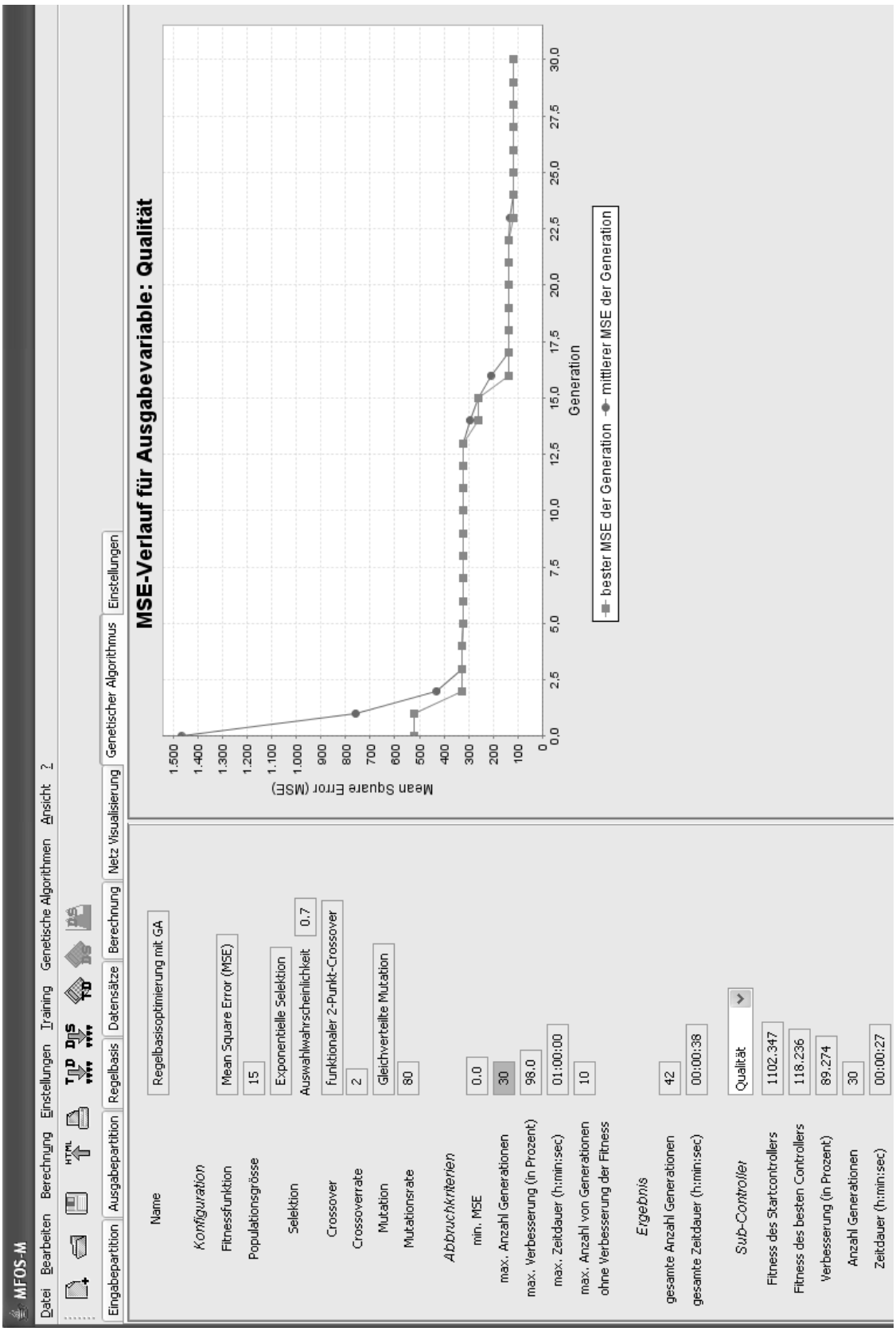


Abbildung 5.10: Ergebnisanzeige des Genetischen Algorithmus

6 Experimente

6.1 Fuzzy-Controller für die Experimente

Es werden die Fuzzy-Controller aufgeführt, die bei den Experimenten zum Einsatz kommen. Hierbei gehören jeweils ein Referenz-Controller mit welchem die Trainingsdaten erzeugt wurden und ein Start-Controller zusammen. Die Referenz-Controller wurden nicht entwickelt, um die zugehörigen Prozesse tatsächlich steuern zu können. Die Partitionierungen wurden willkürlich gesetzt mit der Absicht diese im Anschluss an die Regelbasisoptimierung mittels Datenbasisoptimierung zu erlernen. Dies ist bei der Interpretation der gelernten Regelbasen zu berücksichtigen.

Für die Start-Controller wurden eine unvollständige Regelbasis und eine äquidistante Partitionierung der Ein- und Ausgaberäume verwendet. Es wird jeweils die Suchraumgrösse, die der Anzahl der möglichen Regelbasen entspricht, angegeben.

Die aufgeführten Fuzzy-Controller sind zusammen mit weiteren Controllern auf der zur Diplomarbeit gehörenden CD-ROM im Verzeichnis \Experimente\Controller gespeichert.

6.1.1 Fuzzy-Steuerung des inversen Pendels

Eine Beschreibung des Inversen Pendels befindet sich in Abschnitt 2.4.

1. Version: Grobe Steuerung des Inversen Pendels

Die beiden Eingabevariablen X_1 für den *Winkel* und X_2 für die *Winkelgeschwindigkeit* und die Ausgabevariable Y für die *Kraft* werden jeweils in 5 Fuzzy-Mengen partitioniert. Dabei gelten die Bezeichnungen:

1) ng = negativ gross
2) nk = negativ klein
3) un = ungefähr Null
4) pk = positiv klein
5) pg = positiv gross

Referenz-Controller

Die Partitionierungen in Dreiecksmengen sind:

Winkel

ng	-110.0	-90.0	-85.0
nk	-100.0	-75.0	38.0
un	-90.0	0.0	85.0
pk	-40.0	50.0	95.0
pg	80.0	85.0	110.0

Winkelgeschwindigkeit

ng	-50.0	-40.0	0.0
nk	-30.0	-10.0	0.0
un	-5.0	0.0	5.0
pk	0.0	10.0	30.0
pg	0.0	40.0	50.0

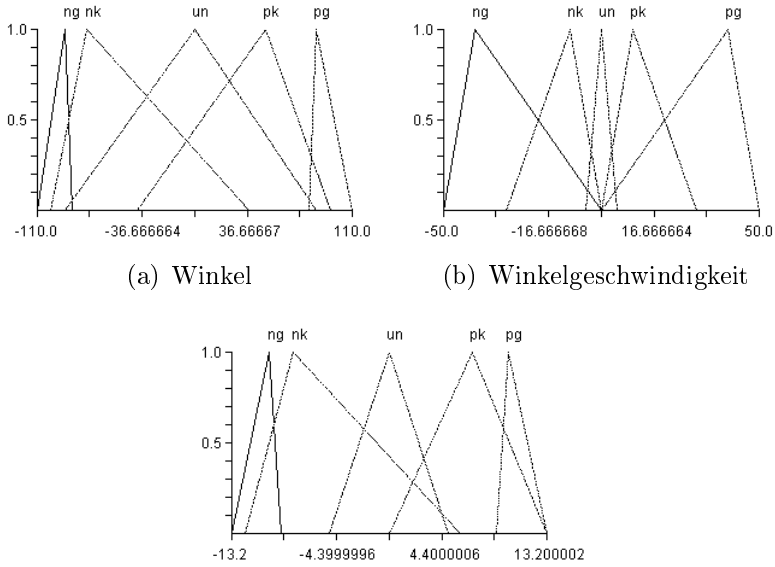
Kraft

ng	-13.2	-10.0	-9.0
nk	-12.0	-8.0	6.0
un	-5.0	0.0	5.0
pk	0.0	7.0	13.2
pg	9.0	10.0	13.2

Diese sind in Abbildung 6.1 dargestellt.

Die Regelbasis umfasst 11 Regeln, die in der folgenden Entscheidungstabelle dargestellt sind:

		X_1				
		ng	nk	un	pk	pg
X_2	ng			ng		
	nk		nk	nk		
	un	ng	nk	un	pk	pg
	pk			pk	pk	
	pg			pg		



(a) Winkel

(b) Winkelgeschwindigkeit

(c) Kraft

Abbildung 6.1: Referenz-Controller-Partitionierungen: Inverses Pendel grobe Steuerung

Start-Controller

Die Partitionierungen in Dreiecksmengen sind:

Winkel

ng	-120.0	-80.0	-40.0
nk	-80.0	-40.0	0.0
un	-40.0	0.0	40.0
pk	0.0	40.0	80.0
pg	40.0	80.0	120.0

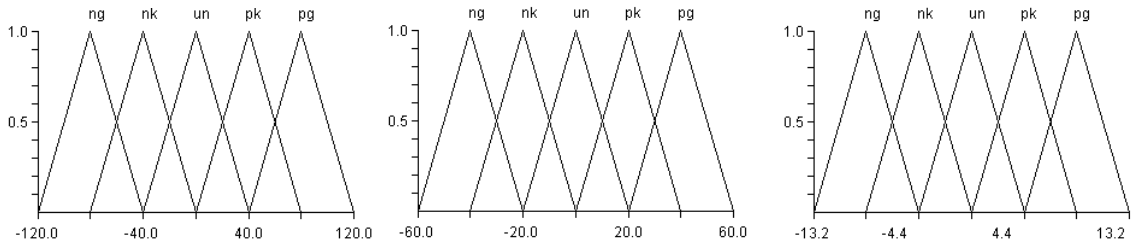
Winkelgeschwindigkeit

ng	-60.0	-40.0	-20.0
nk	-40.0	-20.0	0.0
un	-20.0	0.0	20.0
pk	0.0	20.0	40.0
pg	20.0	40.0	60.0

Kraft

ng	-13.2	-8.8	-4.4
nk	-8.8	-4.4	0.0
un	-4.4	0.0	4.4
pk	0.0	4.4	8.8
pg	4.4	8.8	13.2

Diese sind in Abbildung 6.2 dargestellt.



(a) Winkel (b) Winkelgeschwindigkeit (c) Kraft

Abbildung 6.2: Start-Controller-Partitionierungen: Inverses Pendel grobe Steuerung

Größe des Suchraums für den Genetischen Algorithmus: Es existieren $6^{25} \cong 2,843 \cdot 10^{19}$ mögliche Regelbasen.

2. Version: Feine Steuerung des inversen Pendels

Die beiden Eingabevariablen X_1 für den *Winkel* und X_2 für die *Winkelgeschwindigkeit* und die Ausgabevariable Y für die *Kraft* werden jeweils in 7 Fuzzy-Mengen partitioniert.

Dabei gelten die Bezeichnungen:

1)	ng	=	negativ gross
2)	nm	=	negativ mittel
3)	nk	=	negativ klein
4)	un	=	ungefähr Null
5)	pk	=	positiv klein
6)	pm	=	positiv mittel
7)	pg	=	positiv gross

Referenz-Controller

Die Partitionierungen werden analog der 1.Version gleichmäßig über den Definitionsbereich gemacht. Die Partitionierungen in Dreiecksmengen sind:

Winkel

ng	-120.0	-90.0	-80.0
nm	-100.0	-80.0	-70.0
nk	-120.0	-60.0	0.0
un	-15.0	0.0	120.0
pk	0.0	30.0	60.0
pm	70.0	80.0	100.0
pg	80.0	90.0	120.0

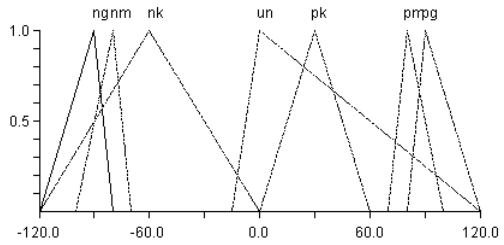
Winkelgeschwindigkeit

ng	-60.0	-55.0	0.0
nm	-45.0	-30.0	-15.0
nk	-10.0	-5.0	0.0
un	-5.0	0.0	5.0
pk	0.0	5.0	10.0
pm	15.0	30.0	45.0
pg	0.0	55.0	60.0

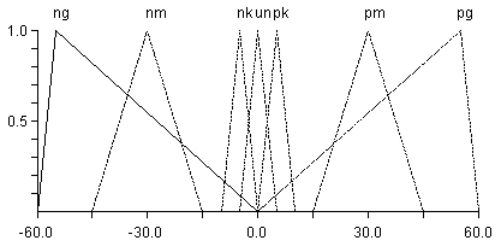
Kraft

ng	-13.2	-9.9	-6.6
nm	-13.2	-6.6	0.0
nk	-6.6	-3.3	0.0
un	-1.0	0.0	1.0
pk	0.0	3.3	6.6
pm	0.0	6.6	13.2
pg	6.6	9.9	13.2

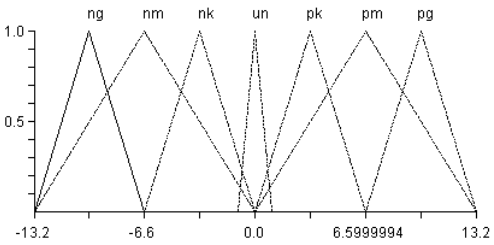
Diese sind in Abbildung 6.3 dargestellt.



(a) Winkel



(b) Winkelgeschwindigkeit



(c) Kraft

Abbildung 6.3: Referenz-Controller-Partitionierungen: Inverses Pendel feine Steuerung

Die Regelbasis umfasst 19 Regeln, die in der folgenden Entscheidungstabelle dargestellt sind:

X_2 \ X_1	ng	nm	nk	un	pk	pm	pg
ng			pk	pg			
nm				pm			
nk	nm		nk	pk			
un	ng	nm	nk	un	pk	pm	pg
pk				nk	pk		pm
pm				nm			
pg				ng	nk		

Start-Controller

Winkel

ng	-120.0	-90.0	-60.0
nm	-90.0	-60.0	-30.0
nk	-60.0	-30.0	0.0
un	-30.0	0.0	30.0
pk	0.0	30.0	60.0
pm	30.0	60.0	90.0
pg	60.0	90.0	120.0

Winkelgeschwindigkeit

ng	-60.0	-45.0	-30.0
nm	-45.0	-30.0	-15.0
nk	-30.0	-15.0	0.0
un	-15.0	0.0	15.0
pk	0.0	15.0	30.0
pm	15.0	30.0	45.0
pg	30.0	45.0	60.0

Kraft

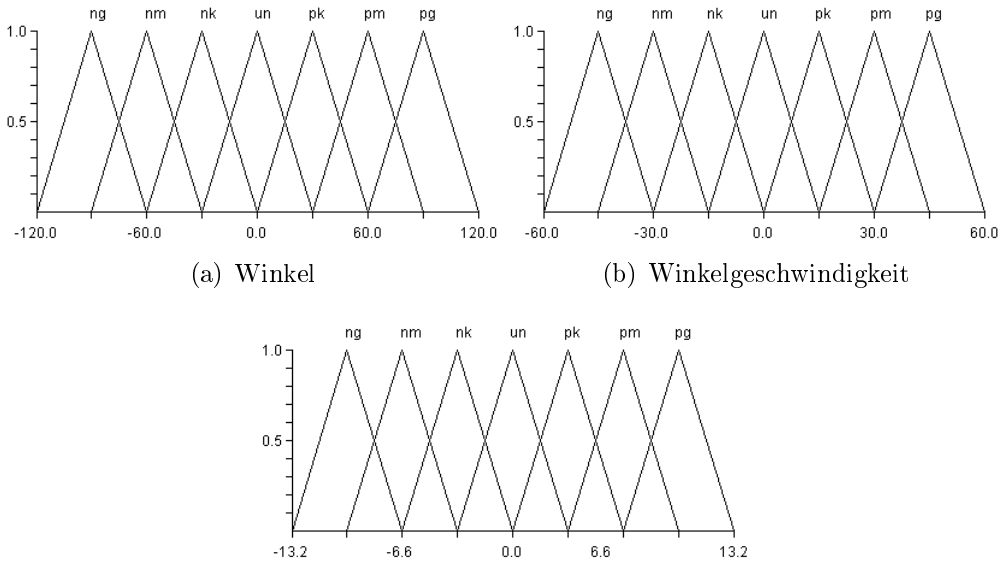
ng	-13.2	-9.9	-6.6
nm	-9.9	-6.6	-3.3
nk	-6.6	-3.3	0.0
un	-3.3	0.0	3.3
pk	0.0	3.3	6.6
pm	3.3	6.6	9.9
pg	6.6	9.9	13.2

Diese sind in Abbildung 6.4 dargestellt.

Größe des Suchraums für den Genetischen Algorithmus: Es existieren $8^{49} \cong 1,784 \cdot 10^{44}$ mögliche Regelbasen.

6.1.2 Fuzzy-Steuerung eines Heizreglers

Das Beispiel verwendet zwei Eingabevariablen *Temperatur* und *Luftfeuchtigkeit* und eine Ausgabevariable *Heizleistung*.



(a) Winkel

(b) Winkelgeschwindigkeit

(c) Kraft

Abbildung 6.4: Start-Controller-Partitionierungen: Inverses Pendel feine Steuerung

Referenz-Controller

Die linguistische Variable X_1 für die Temperatur wird auf dem Intervall [13,23] durch

$$\begin{aligned} \text{kalt} &= (10, 14, 18) \\ \text{warm} &= (14, 18, 22) \\ \text{heiss} &= (18, 24, 30) \end{aligned}$$

definiert.

Die linguistische Variable X_2 für die Luftfeuchtigkeit wird auf dem Intervall [0,100] in Prozent durch

$$\begin{aligned} \text{trocken} &= (0, 15, 30) \\ \text{normal} &= (15, 35, 55) \\ \text{feucht} &= (50, 70, 90) \\ \text{sehr feucht} &= (80, 90, 100) \end{aligned}$$

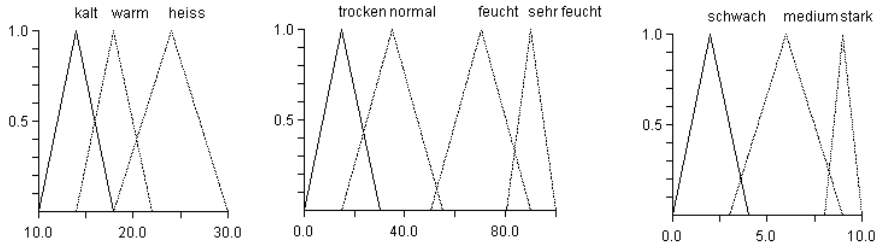
definiert.

Die linguistische Variable Y für die Heizleistung wird auf dem Intervall [0,10] durch

$$\begin{aligned} \text{schwach} &= (0, 2, 4) \\ \text{medium} &= (3, 6, 9) \\ \text{stark} &= (8, 9, 10) \end{aligned}$$

definiert.

Diese sind in Abbildung 6.5 dargestellt.



(a) Temperatur (b) Luftfeuchtigkeit (c) Leistung

Abbildung 6.5: Referenz-Controller-Partitionierungen: Heizregler

Die zugehörige Regelbasis ist:

1)	IF X_1 IS kalt	AND X_2 IS sehr feucht	THEN Y IS stark
2)	IF X_1 IS kalt	AND X_2 IS feucht	THEN Y IS medium
3)	IF X_1 IS warm	AND X_2 IS normal	THEN Y IS medium
4)	IF X_1 IS heiss	AND X_2 IS trocken	THEN Y IS schwach

Start-Controller

Die linguistische Variable X_1 für die Temperatur wird auf dem Intervall [13,23] durch

$$\begin{aligned} \text{kalt} &= (10, 15, 20) \\ \text{warm} &= (15, 20, 25) \\ \text{heiss} &= (20, 25, 30) \end{aligned}$$

definiert.

Die linguistische Variable X_2 für die Luftfeuchtigkeit wird auf dem Intervall [0,100] in Prozent durch

$$\begin{aligned} \text{trocken} &= (0, 20, 40) \\ \text{normal} &= (20, 40, 60) \\ \text{feucht} &= (40, 60, 80) \\ \text{sehr feucht} &= (60, 80, 100) \end{aligned}$$

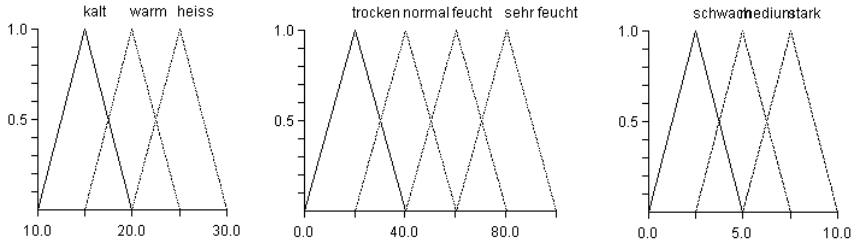
definiert.

Die linguistische Variable Y für die Heizleistung wird auf dem Intervall [0,10] durch

$$\begin{aligned} \text{schwach} &= (0.0, 2.5, 5.0) \\ \text{medium} &= (2.5, 5.0, 7.5) \\ \text{stark} &= (5.0, 7.5, 10.0) \end{aligned}$$

definiert.

Diese sind in Abbildung 6.6 dargestellt.



(a) Temperatur (b) Luftfeuchtigkeit (c) Leistung

Abbildung 6.6: Start-Controller-Partitionierungen: Heizregler

Größe des Suchraums für den Genetischen Algorithmus: Es existieren $4^{12} = 16.777.216$ mögliche Regelbasen.

6.1.3 Fuzzy-Produktbewertung

Es werden drei Eingabevariablen *Größe*, *Gewicht* und *Aussehen*- und zwei Ausgabevariablen *Qualität* und *Preiskategorie* verwendet. Es werden Dreiecksmengen für die Partitionierungen verwendet.

Referenz-Controller

Die linguistische Variable X_1 für die Größe des Produkts wird auf dem Intervall [0,10] durch

$$\begin{aligned} \text{klein} &= (0, 1, 9) \\ \text{groß} &= (1, 9, 10) \end{aligned}$$

definiert.

Die linguistische Variable X_2 für das Gewicht des Produkts wird auf dem Intervall [0,10] durch

$$\begin{aligned} \text{klein} &= (0, 1, 9) \\ \text{groß} &= (1, 9, 10) \end{aligned}$$

definiert.

Die linguistische Variable X_3 für das Aussehen des Produkts wird auf dem Intervall [0,10] durch

$$\begin{aligned} \text{schlecht} &= (0, 1, 9) \\ \text{gut} &= (1, 9, 10) \end{aligned}$$

definiert.

Die linguistische Variable Y_1 für die Qualität des Produkts wird auf dem Intervall [0,100] durch

$$\begin{aligned} \text{schlecht} &= (0, 15, 30) \\ \text{mittel} &= (20, 40, 60) \\ \text{gut} &= (60, 70, 80) \\ \text{sehrgut} &= (70, 85, 100) \end{aligned}$$

definiert.

Die linguistische Variable Y_2 für die Preiskategorie des Produkts wird auf dem Intervall [0,100] durch

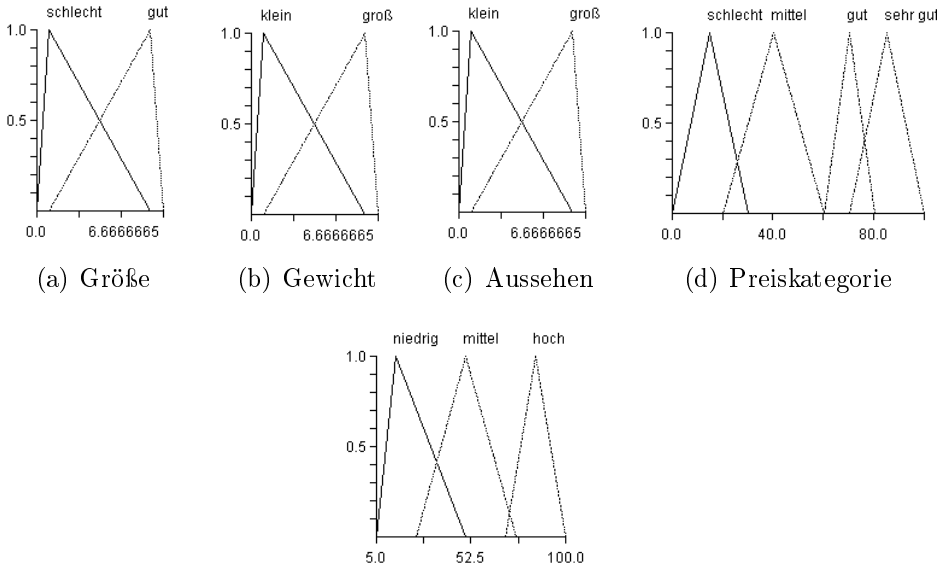
$$\begin{aligned} \text{niedrig} &= (5, 15, 50) \\ \text{mittel} &= (25, 50, 75) \\ \text{hoch} &= (70, 85, 100) \end{aligned}$$

definiert.

Diese sind in Abbildung 6.7 dargestellt.

Die zugehörige Regelbasis ist:

1)	IF X_1 IS klein	AND X_2 IS klein	AND X_3 IS schlecht	THEN Y_1 IS schlecht
2)	IF X_1 IS klein	AND X_2 IS groß	AND X_3 IS schlecht	THEN Y_1 IS mittel
3)	IF X_1 IS klein	AND X_2 IS groß	AND X_3 IS gut	THEN Y_1 IS gut
4)	IF X_1 IS groß	AND X_2 IS klein	AND X_3 IS schlecht	THEN Y_1 IS mittel
5)	IF X_1 IS groß	AND X_2 IS klein	AND X_3 IS gut	THEN Y_1 IS gut
6)	IF X_1 IS groß	AND X_2 IS groß	AND X_3 IS gut	THEN Y_1 IS sehrgut
7)	IF X_1 IS klein	AND X_2 IS klein	AND X_3 IS schlecht	THEN Y_2 IS niedrig
8)	IF X_1 IS klein	AND X_2 IS groß	AND X_3 IS schlecht	THEN Y_2 IS mittel
9)	IF X_1 IS klein	AND X_2 IS groß	AND X_3 IS gut	THEN Y_2 IS mittel
10)	IF X_1 IS groß	AND X_2 IS klein	AND X_3 IS schlecht	THEN Y_2 IS mittel
11)	IF X_1 IS groß	AND X_2 IS klein	AND X_3 IS gut	THEN Y_2 IS mittel
12)	IF X_1 IS groß	AND X_2 IS groß	AND X_3 IS gut	THEN Y_2 IS hoch



(a) Größe (b) Gewicht (c) Aussehen (d) Preiskategorie

(e) Qualität

Abbildung 6.7: Referenz-Controller-Partitionierungen: Produktbewertung

Start-Controller

Die linguistische Variable X_1 für die Größe der Ware wird auf dem Intervall [0,10] durch

$$\begin{aligned} \text{klein} &= (0.0, 3.34, 6.67) \\ \text{groß} &= (3.34, 6.67, 10.0) \end{aligned}$$

definiert.

Die linguistische Variable X_2 für das Gewicht des Produkts wird auf dem Intervall [0,10] durch

$$\begin{aligned} \text{klein} &= (0.0, 3.34, 6.67) \\ \text{groß} &= (3.34, 6.67, 10.0) \end{aligned}$$

definiert.

Die linguistische Variable X_3 für das Aussehen des Produkts wird auf dem Intervall [0,10] durch

$$\begin{aligned} \text{klein} &= (0.0, 3.34, 6.67) \\ \text{groß} &= (3.34, 6.67, 10.0) \end{aligned}$$

definiert.

Die linguistische Variable Y_1 für die Qualität des Produkts wird auf dem Intervall [0,100]

durch

$$\begin{aligned} \text{schlecht} &= (0, 20, 40) \\ \text{mittel} &= (20, 40, 60) \\ \text{gut} &= (40, 60, 80) \\ \text{sehrgut} &= (60, 80, 100) \end{aligned}$$

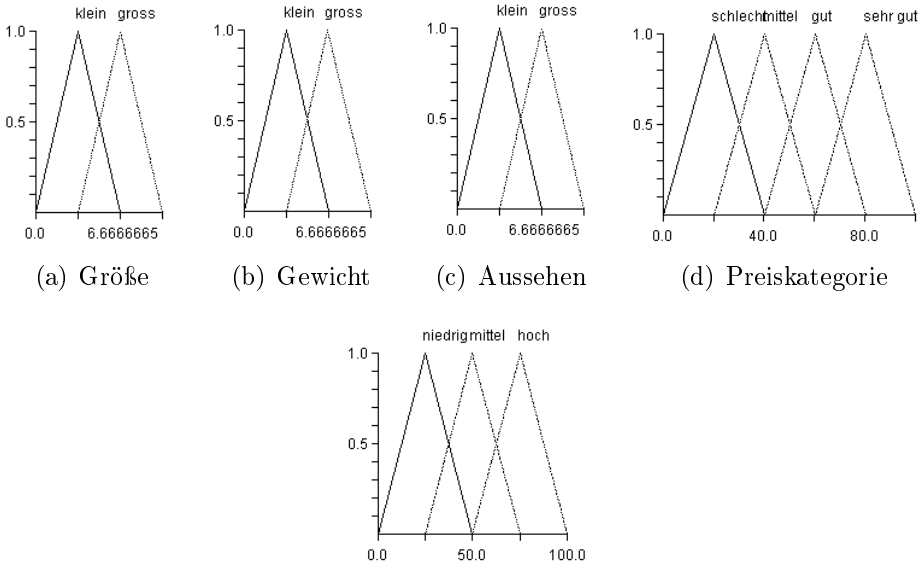
definiert.

Die linguistische Variable Y_2 für die Preiskategorie des Produkts wird auf dem Intervall [0,100] durch

$$\begin{aligned} \text{niedrig} &= (0, 25, 50) \\ \text{mittel} &= (25, 50, 75) \\ \text{hoch} &= (50, 75, 100) \end{aligned}$$

definiert.

Diese sind in Abbildung 6.8 dargestellt.



(a) Größe (b) Gewicht (c) Aussehen (d) Preiskategorie

(e) Qualität

Abbildung 6.8: Start-Controller-Partitionierungen: Produktbewertung

Größe des Suchraums für den Genetischen Algorithmus: Es existieren $4^8 + 3^8 = 72097$ mögliche Regelbasen.

6.2 Anwendung der Optimierung mit Genetischen Algorithmen in MFOS-M

Die in Kapitel 6.1 aufgeführten Fuzzy-Controller werden mit Genetischen Algorithmen in MFOS-M optimiert.

6.2.1 Versuchsumgebung

Der Computer auf dem die Versuche durchgeführt werden hat folgende Kenndaten:

- Mobile AMD Sempron(tm) Prozessor 3100+, 1.58 GHz
- 1 GB RAM
- Microsoft Windows XP Home-Edition [Version 5.1.2600]
- Java(TM) 2 Runtime Environment, Standard Edition Version [1.5.007]

6.2.2 Ergebnisse

Die Ergebnisse der Versuche wurden mithilfe der MFOS-M-Funktion *Export in HTML-Datei* auf der CD-ROM zur Diplomarbeit im Verzeichnis Experimente gespeichert.

MFOS-M lief bei allen Experimenten stabil und ohne Fehler.

Die durchgeführten Experimente zeigen die gute Leistung der Optimierung mit Genetischen Algorithmen in MFOS-M. Der MSE konnte in allen Versuchen mehr als halbiert werden. Im Mittel wurde eine etwa 75-prozentige Verbesserung gegenüber dem MSE der Startregelbasis erzielt. Auch mit relativ kleinen Populationen von 5 bis 20 Individuen können gute Verbesserungen erreicht werden.

Für die Fitnessfunktion wird der mittlere quadratische Fehler (MSE) verwendet. Die Minimierung des MSE ist einziges Kriterium der Optimierung. Daher sind die gelernten Regelbasen meist beinahe vollständig bis vollständig besetzt.

Die Partitionierungen der Referenz-Controller bestehen ebenso wie die der Startcontroller ausschließlich aus Dreiecksmengen. Die Anzahl der Partitionsmengen stimmt für Start- und Referenz-Controller jeweils überein. Die Partitionsmengen für die Start-Controller sind in gleichmäßigem Abstand angeordnet und gleichförmig gestaltet. Hierin unterscheiden sich die Referenz-Controller von den Start-Controllern. Die Partitionsmengen der

Referenz-Controller sind weder gleichförmig gestaltet, noch in gleichem Abstand angeordnet.

Im folgenden wird jeweils ein typisches Versuchsergebnis für jeden Versuchs-Controller-Typ aufgeführt.

Optimierung der Fuzzy-Steuerung eines Heizreglers

Der Heizregler-Controller wurde mit einer Regelbasis, die aus 50 Prozent richtigen Regeln besteht, und 229 Trainingsdaten optimiert.

Konfiguration des Genetischen Algorithmus:

Fitnessfunktion	Mean Square Error (MSE)
Populationsgrösse	30
Selektion	Turnierselektion (Turniergrösse: 5)
Crossover	funktionaler 2-Punkt-Crossover
Crossoverrate	2
Mutation	Gleichverteilte Mutation
Mutationsrate	10

Ergebnis:

Fitness des Startcontrollers	8.262
Fitness des besten Controllers	2.308
Verbesserung (in Prozent)	72.063
Anzahl Generationen	36
Zeitdauer (h:min:sec)	00:01:44

Start-Regelbasis:

1)	IF X_1 IS warm	AND X_2 IS normal	THEN Y IS medium
2)	IF X_1 IS heiss	AND X_2 IS trocken	THEN Y IS schwach

Referenz-Regelbasis:

1)	IF X_1 IS kalt	AND X_2 IS sehr feucht	THEN Y IS stark
2)	IF X_1 IS kalt	AND X_2 IS feucht	THEN Y IS medium
3)	IF X_1 IS warm	AND X_2 IS normal	THEN Y IS medium
4)	IF X_1 IS heiss	AND X_2 IS trocken	THEN Y IS schwach

Gelernte Regelbasis:

1)	IF X_1 IS kalt	AND X_2 IS sehr feucht	THEN Y IS stark
2)	IF X_1 IS kalt	AND X_2 IS feucht	THEN Y IS medium
3)	IF X_1 IS warm	AND X_2 IS normal	THEN Y IS medium
4)	IF X_1 IS heiss	AND X_2 IS trocken	THEN Y IS schwach
5)	IF X_1 IS heiss	AND X_2 IS normal	THEN Y IS schwach
6)	IF X_1 IS warm	AND X_2 IS sehr feucht	THEN Y IS schwach

Die gelernte Regelbasis enthält alle Regeln des Referenz-Controllers und noch 2 weitere Regeln.

Abbildung 6.9 zeigt den MSE-Verlauf der Optimierung.

Optimierung der Fuzzy-Steuerung des inversen Pendels, grobe Steuerung

Der Fuzzy-Controller zur groben Steuerung des inversen Pendels wurde mit einer Regelbasis, die aus 20 Prozent richtigen Regeln besteht, und 111 Trainingsdaten optimiert.

Konfiguration des Genetischen Algorithmus:

Fitnessfunktion	Mean Square Error (MSE)
Populationsgrösse	40
Selektion	Turnierselektion (Turniergrösse: 8)
Crossover	funktionaler 2-Punkt-Crossover
Crossoverrate	2
Mutation	Gleichverteilte Mutation
Mutationsrate	60

Ergebnis:

Fitness des Startcontrollers	86.081
Fitness des besten Controllers	12.130
Verbesserung (in Prozent)	85.909
Anzahl Generationen	54
Zeitdauer (h:min:sec)	00:06:15

Start-Regelbasis: X_2 / X_1

	ng	nk	un	pk	pg
ng					
nk					
un					
pk				pk	
pg			pg		

Referenz-Regelbasis: X_2 / X_1

	ng	nk	un	pk	pg
ng			ng		
nk		nk	nk		
un	ng	nk	un	pk	pg
pk			pk	pk	
pg			pg		

Gelernte Regelbasis: X_2 / X_1

	ng	nk	un	pk	pg
ng					
nk	ng	ng	ng	ng	ng
un					pk
pk		pk	pg	pg	
pg		pg	pg	pg	

Die gelernte Regelbasis hat einen rund 86 Prozent geringeren MSE als die Start-Regelbasis. Trotzdem ist die gelernte Regelbasis von dem Steuerungsverhalten der Referenz-Regelbasis weit entfernt.

Abbildung 6.10 zeigt den MSE-Verlauf der Optimierung.

Optimierung der Fuzzy-Steuerung des inversen Pendels, feinere Steuerung

Der Fuzzy-Controller zur feineren Steuerung des inversen Pendels wurde mit einer Regelbasis, die aus rund 21 Prozent richtigen Regeln besteht, und 119 Trainingsdaten optimiert.

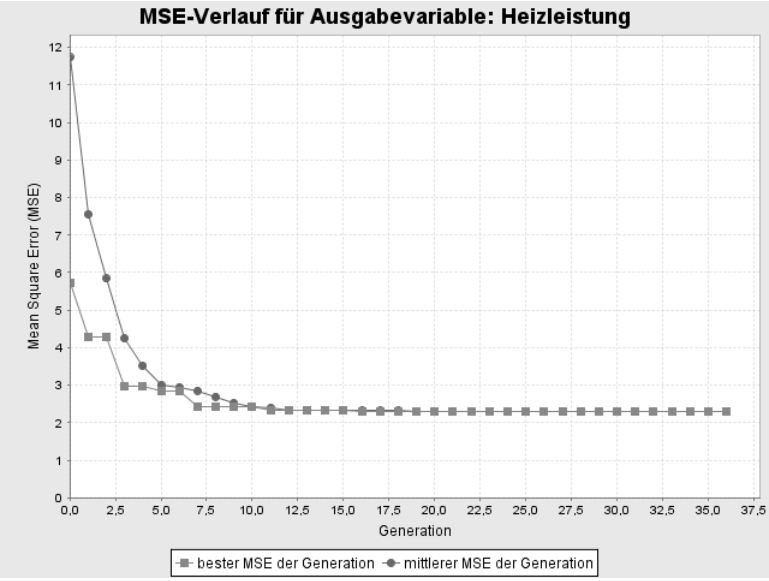


Abbildung 6.9: Optimierung Heizregler

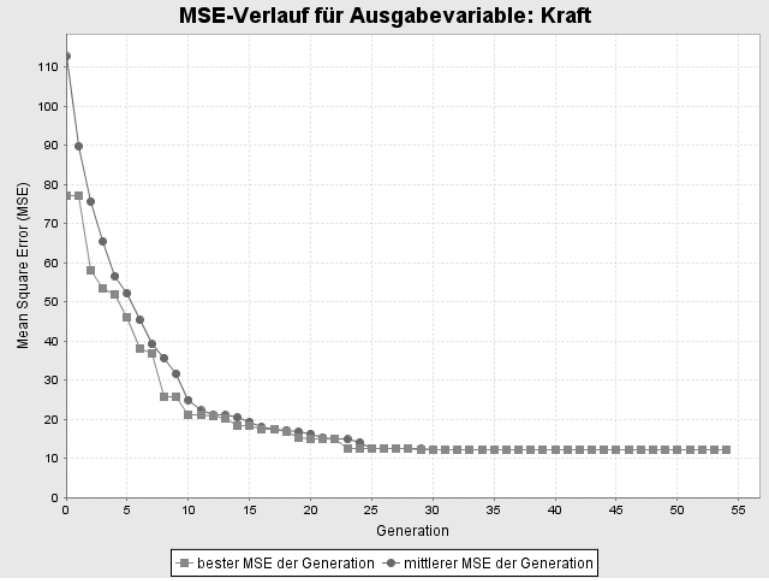


Abbildung 6.10: Optimierung Controller für inverses Pendel, grobe Steuerung

Konfiguration des Genetischen Algorithmus:

Fitnessfunktion	Mean Square Error (MSE)
Populationsgrösse	40
Selektion	Turnierselektion (Turniergrösse: 10)
Crossover	2-Punkt-Crossover
Crossoverrate	2
Mutation	Gleichverteilte Mutation
Mutationsrate	50

Ergebnis:

Fitness des Startcontrollers	162.450
Fitness des besten Controllers	3.954
Verbesserung (in Prozent)	97.566
Anzahl Generationen	117
Zeitdauer (h:min:sec)	00:42:48

Start-Regelbasis:

X_2 \ X_1	ng	nm	nk	un	pk	pm	pg
ng							
nm							
nk							
un						pm	
pk							
pm				nm			
pg				ng	nk		

4 Regeln

Referenz-Regelbasis:

X_2 \ X_1	ng	nm	nk	un	pk	pm	pg
ng			pk	pg			
nm				pm			
nk	nm		nk	pk			
un	ng	nm	nk	un	pk	pm	pg
pk				nk	pk		pm
pm				nm			
pg				ng	nk		

19 Regeln

Gelernte Regelbasis:

X_2 \ X_1	ng	nm	nk	un	pk	pm	pg
ng	pk	pk	pk	pg	pg	pg	pm
nm	pk	pk	pk	pm	pm		
nk	nk	pk	un	pg	pg		pg
un		nm		un	pk	un	pm
pk				ng	ng		ng
pm				nm	nm	ng	
pg				ng	nk	ng	

32 Regeln

Abbildung 6.11 zeigt den MSE-Verlauf der Optimierung.

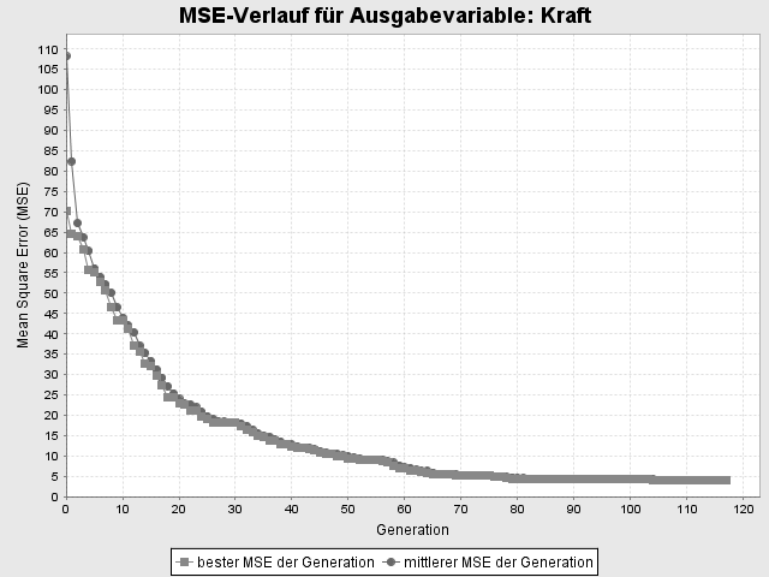


Abbildung 6.11: Optimierung Controller für inverses Pendel, feine Steuerung

Optimierung der Fuzzy-Produktbewertung

Der Fuzzy-Controller für die Produktbewertung wurde mit einer Regelbasis, die aus 25 Prozent richtigen Regeln besteht, und 131 Trainingsdaten optimiert.

Konfiguration des Genetischen Algorithmus:

Fitnessfunktion	Mean Square Error (MSE)
Populationsgrösse	20
Selektion	Turnierselektion (Turniergrösse: 5)
Crossover	2-Punkt-Crossover
Crossoverrate	2
Mutation	Gleichverteilte Mutation
Mutationsrate	30

Ergebnis für Sub-Controller Qualität:

Fitness des Startcontrollers	657.763
Fitness des besten Controllers	148.257
Verbesserung (in Prozent)	77.460
Anzahl Generationen	43
Zeitdauer (h:min:sec)	00:05:46

Ergebnis für Sub-Controller Preiskategorie:

Fitness des Startcontrollers	852.085
Fitness des besten Controllers	210.369
Verbesserung (in Prozent)	75.311
Anzahl Generationen	43
Zeitdauer (h:min:sec)	00:05:23

Start-Regelbasis:

1)	IF X_1 IS groß	AND X_2 IS klein	AND X_3 IS gut	THEN Y_0 IS gut
2)	IF X_1 IS groß	AND X_2 IS klein	AND X_3 IS gut	THEN Y_1 IS mittel
3)	IF X_1 IS groß	AND X_2 IS groß	AND X_3 IS gut	THEN Y_0 IS sehrgut
3)	IF X_1 IS groß	AND X_2 IS groß	AND X_3 IS gut	THEN Y_1 IS hoch

Referenz-Regelbasis:

1)	IF X_1 IS klein	AND X_2 IS klein	AND X_3 IS schlecht	THEN Y_1 IS schlecht
2)	IF X_1 IS klein	AND X_2 IS groß	AND X_3 IS schlecht	THEN Y_1 IS mittel
3)	IF X_1 IS klein	AND X_2 IS groß	AND X_3 IS gut	THEN Y_1 IS gut
4)	IF X_1 IS groß	AND X_2 IS klein	AND X_3 IS schlecht	THEN Y_1 IS mittel
5)	IF X_1 IS groß	AND X_2 IS klein	AND X_3 IS gut	THEN Y_1 IS gut
6)	IF X_1 IS groß	AND X_2 IS groß	AND X_3 IS gut	THEN Y_1 IS sehrgut
7)	IF X_1 IS klein	AND X_2 IS klein	AND X_3 IS schlecht	THEN Y_2 IS niedrig
8)	IF X_1 IS klein	AND X_2 IS groß	AND X_3 IS schlecht	THEN Y_2 IS mittel
9)	IF X_1 IS klein	AND X_2 IS groß	AND X_3 IS gut	THEN Y_2 IS mittel
10)	IF X_1 IS groß	AND X_2 IS klein	AND X_3 IS schlecht	THEN Y_2 IS mittel
11)	IF X_1 IS groß	AND X_2 IS klein	AND X_3 IS gut	THEN Y_2 IS mittel
12)	IF X_1 IS groß	AND X_2 IS groß	AND X_3 IS gut	THEN Y_2 IS hoch

Gelernte Regelbasis:

1)	IF X_1 IS klein	AND X_2 IS klein	AND X_3 IS schlecht	THEN Y_1 IS schlecht
2)	IF X_1 IS klein	AND X_2 IS klein	AND X_3 IS gut	THEN Y_1 IS mittel
3)	IF X_1 IS klein	AND X_2 IS groß	AND X_3 IS schlecht	THEN Y_1 IS mittel
4)	IF X_1 IS klein	AND X_2 IS groß	AND X_3 IS gut	THEN Y_1 IS mittel
5)	IF X_1 IS groß	AND X_2 IS klein	AND X_3 IS gut	THEN Y_1 IS mittel
6)	IF X_1 IS groß	AND X_2 IS klein	AND X_3 IS schlecht	THEN Y_1 IS mittel
7)	IF X_1 IS groß	AND X_2 IS groß	AND X_3 IS gut	THEN Y_1 IS sehrgut
8)	IF X_1 IS klein	AND X_2 IS klein	AND X_3 IS schlecht	THEN Y_2 IS niedrig
9)	IF X_1 IS klein	AND X_2 IS groß	AND X_3 IS schlecht	THEN Y_2 IS mittel
10)	IF X_1 IS klein	AND X_2 IS groß	AND X_3 IS gut	THEN Y_2 IS mittel
11)	IF X_1 IS groß	AND X_2 IS klein	AND X_3 IS schlecht	THEN Y_2 IS mittel
12)	IF X_1 IS groß	AND X_2 IS klein	AND X_3 IS gut	THEN Y_2 IS mittel
13)	IF X_1 IS groß	AND X_2 IS groß	AND X_3 IS gut	THEN Y_2 IS hoch

In der gelernten Regelbasis sind zehn der insgesamt dreizehn Regeln enthalten, die auch in der Referenz-Regelbasis enthalten sind. Die drei nicht in der Referenz-Regelbasis enthaltenen Regeln sind die Regeln 2), 4), und 5).

Die Abbildungen 6.12 und 6.13 zeigen die MSE-Verläufe der Optimierung.

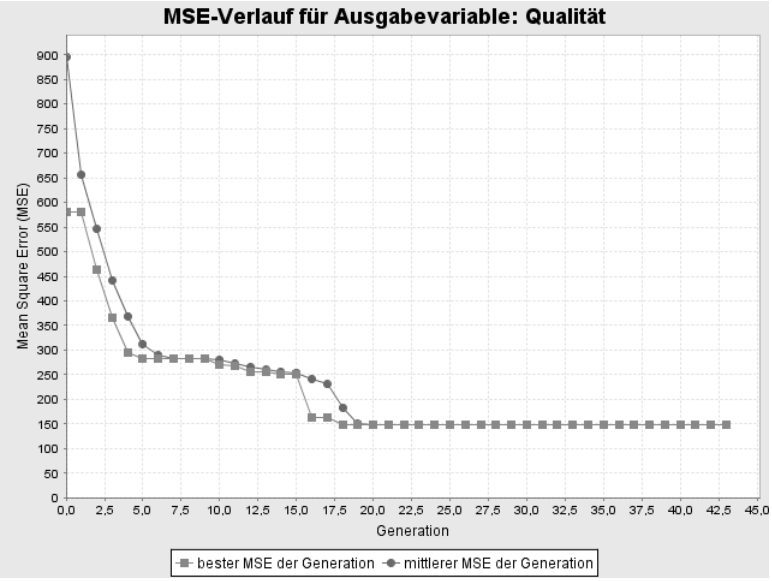


Abbildung 6.12: Optimierung von Controller für Fuzzy-Produktbewertung, 1. Ausgabedimension

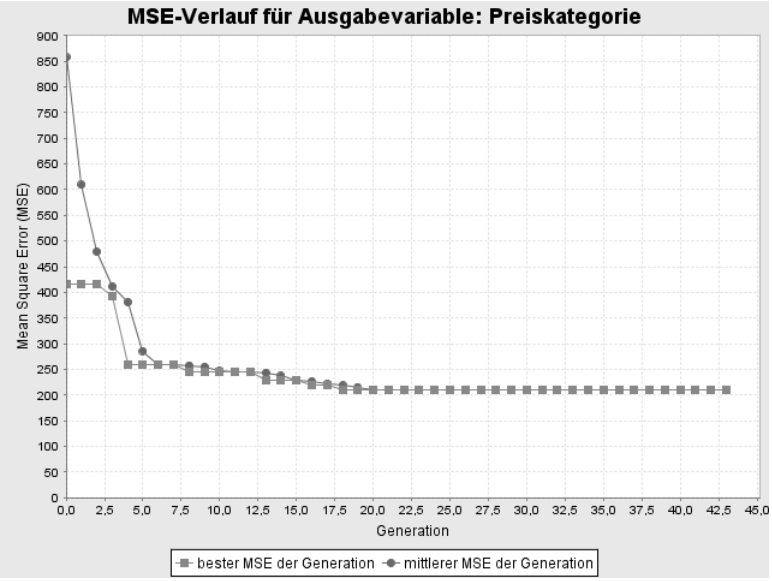


Abbildung 6.13: Optimierung von Controller für Fuzzy-Produktbewertung, 2. Ausgabedimension

7 Fazit

7.1 Zusammenfassung

In dieser Diplomarbeit wurde die zur Verfügung gestellte MFOS-M-Implementierung erfolgreich um die Möglichkeit der Optimierung der Regelbasis eines Fuzzy-Controllers mit Genetischen Algorithmen erweitert. Im ersten Teil dieser Arbeit werden die hierfür wesentlichen theoretischen Grundlagen beschrieben. Der zweite Teil dieser Arbeit dokumentiert die Implementierung und die durchgeführten Experimente. Die Erweiterung gliedert sich nahtlos in das bestehende System ein. Die durchgeführten Experimente zeigen die gute Leistung der Optimierung der Regelbasis mit Genetischen Algorithmen in MFOS-M.

7.2 Ausblick

Für weiterführende Arbeiten empfiehlt sich, die begonnene Reparatur der MFOS-M-Implementierung zuende zu führen. Dadurch wird ein vielversprechendes kombiniertes Arbeiten mit den bestehenden neuronalen- und neuen genetischen Optimierungsmethoden ermöglicht.

Weiterhin könnten Genetische Algorithmen zur Optimierung der Datenbasis eines Fuzzy-Controllers in die MFOS-M-Implementierung integriert werden.

A CD-ROM

Alle im folgenden beschriebenen Dateien und Verzeichnisse befinden sich im Hauptverzeichnis der zu dieser Diplomarbeit gehörigen CD-ROM.

- Diplomarbeit.pdf
- Im Verzeichnis Artikel befinden sich eine Auswahl von für diese Arbeit nützlichen und weiterführenden Artikeln zum Thema Optimierung von Fuzzy-Controllern mit Evolutionären Algorithmen. Die meisten dieser Artikel sind von den Autoren von [Cordon01].
- Im Verzeichnis mfos_m befinden sich
  - die zur Verfügung gestellte MFOS-M-Implementierung im Verzeichnis mfos_m_alt
  - Beispiele zur Reproduktion der gefundenen Fehler
  - die in dieser Arbeit entstandene verbesserte Version der MFOS-M-Implementierung im Verzeichnis mfos_m_verbessert
  - die um die Möglichkeit der Optimierung mit genetischen Algorithmen erweiterte MFOS-M-Implementierung im Verzeichnis MFOS_M_GA.
    Diese kann auf Windows-Betriebssystemen von dort durch Aufruf von mfos_m_ga_start.bat gestartet werden.
- Das Verzeichnis Experimente enthält:
  - Das Unterverzeichnis Controller, in dem die Controller für die Versuche gespeichert sind.
  - Die aus MFOS-M mithilfe der Export in HTML-Datei gespeicherten Versuchsergebnisse.
- Das Verzeichnis jgap30final enthält den JGAP 3.0 Final Quellcode und die JGAP 3.0 Final - JavaDoc
- Das Verzeichnis jfreechart-1.0.2 enthält den JFreeChart 1.0.2 Quellcode und die zugehörige JavaDoc.
- Das Verzeichnis mfos_m_s enthält:
  - den Quellcode der zur Verfügung gestellten Implementierung des MFOS-S im Verzeichnis mfos_s_alt
  - Beispiele zur Reproduktion der Fehler im Verzeichnis mfos_s_Fehler_Beispiele
  - den Quellcode der verbesserten MFOS-S-Version im Verzeichnis mfos_s_verbessert

Abbildungsverzeichnis

Literaturverzeichnis

[Börcsök00] Börcsök, Josef: *Fuzzy Control - Theorie und Industrieeinsatz.* HUSS-MEDIEN Gmbh, Berlin, 1. Auflage, 2000

[Brinkschulte02] Brinkschulte, Hendrik: *Einbindung von Fuzzy-Implikationen und Defuzzifizierungsmethoden in das MFOS.* Diplomarbeit am Institut für Informatik des Fachbereichs Mathematik und Informatik der Westfälischen Wilhelms-Universität Münster, Münster, 2002

[Cordon01] Cordon, Oscar, Herrera Francisco, Hoffmann Frank, Magdalena Luis: *Genetic Fuzzy Systems - Evolutionary Tuning and Learning of Fuzzy Knowledge Bases.* World Scientific Publishing Co. Pte. Ltd., Singapore, 1. Auflage, 2001

[Deimann01] Deimann, Marc: *Visualisierung einer grafischen Benutzeroberfläche für einen Fuzzy-Optimierer.* Diplomarbeit am Institut für Informatik des Fachbereichs für Mathematik und Informatik der Westfälischen Wilhelms-Universität Münster, Münster, 2001

[Esken99] Esken, Petra: *Optimierung eines hybriden Neuro-Fuzzy-Systems.* Diplomarbeit am Institut für Informatik des Fachbereichs Mathematik und Informatik der Westfälischen Wilhelms-Universität Münster, Münster, 1999

[Fogel65] Fogel, L.J. et al.: *Artifical Intelligence through a Simulation of Evolution.* Biophysics and Cybernetic Systems, London, pp.131-155, 1965

[Gerdes04] Gerdes, Ingrid, Frank Klawonn, Rudolf Kruse: *Evolutionäre Algorithmen - Genetische Algorithmen - Strategien und Optimierungsverfahren - Beispielanwendungen.* Vieweg Verlag, Wiesbaden 2004, 1. Aufage, 2004

[Goldberg89] Goldberg, David Edward: *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison Wesley Longman Inc., 22. Auflage, 2001

[Holland75] Holland, John: *Adaption in natural and artificial systems.* The University of Michigan Press, Ann Arbor, 1975

[Koza92] Koza, J.R.: *Genetic Programming: On the Programming of Computers by means of Natural Selection.* MIT Press, Cambridge, 1992

[Küfer04] KÜFER, THORSTEN: *Optimierung von Fuzzy-Controllern mit Hilfe von Genetischen Algorithmen.* Diplomarbeit am Institut für Informatik des Fachbereichs Mathematik und Informatik der Westfälischen Wilhelms-Universität Münster, Münster, 2004

[Lippe06] LIPPE, WOLFRAM-MANFRED: *Soft-Computing.* Springer-Verlag, Berlin Heidelberg, 1. Auflage, 2006

[Mamdani75] MAMDANI, E. H., ASSILIAN, S.: *An experiment in linguistic synthesis with a fuzzy logic controller.* Int. Journal of Man-Machines Studies, Vol. 7, pp. 1-13, 1975

[Mayer93] MAYER, ANDREAS ET.AL.: *Fuzzy Logic - Einführung und Leitfaden zur praktischen Anwendung, mit Fuzzy-Shell in C++.* Addison-Wesley, 1. Auflage, 1993

[Nauck96] NAUCK, DETLEF; KLAWONN, FRANK; KRUSE, RUDOLF: *Neuronale Netze und Fuzzy-Systeme - Grundlagen des Konnektionismus, Neuronaler Fuzzy-Systeme und der Koppelung mit wissensbasierten Methoden.* Vieweg, Braunschweig/Wiesbaden, überarbeitete und erweiterte 2. Auflage, 1996

[Niendieck98] NIENDIECK, STEFFEN: *Eine universelle Repräsentation von Fuzzy-Controllern durch neuronale Netze.* Diplomarbeit am Institut für Informatik des Fachbereichs Mathematik und Informatik der Westfälischen Wilhelms-Universität Münster, Münster, 1998

[Niendieck03] NIENDIECK, STEFFEN: *Optimierung von Fuzzy-Controllern. Von Untersuchungen hybrider Neuro-Fuzzy-Systeme zum Entwurf des universellen konnektionistischen Modells MFOS (Münsteraner-Fuzzy-Optimierungs-System).* Dissertation am Institut für Informatik des Fachbereichs Mathematik und Informatik der Westfälischen Wilhelms-Universität Münster, Münster, 2003

[Pohlheim99] POHLHEIM, HARTMUT: *Evolutionäre Algorithmen - Verfahren, Operatoren und Hinweise für die Praxis.* Springer, Berlin, Heidelberg, 1. Auflage, 1999

[Rechenberg73] RECHENBERG, I.: *Evolutionsstrategie - Optimierung technischer Systeme nach Prinzipien der biologischen Evolution.* Frommann-Holzboog, Stuttgart, 2006

[Smith80] SMITH, S.F.: *A learning system based on genetic adaptive algorithms.* Dissertation an der Pittsburgh-Universität, 1980

[Sugeno85] SUGENO M., TAKAGI, T.: *Fuzzy identification of systems and its applications to modeling and control.* IEEE Transactions on Systems, Man and Cybernetics, Vol. 15, pp. 116-132, 1985

[Urich04] URICH, WOLFRAM: *Portierung des Münsteraner Fuzzy-Optimierungs-Systems für Sugeno-Controller von C++ nach Java und Entwicklung*

einer graphischen Benutzeroberfläche. Diplomarbeit am Institut für Informatik des Fachbereichs Mathematik und Informatik der Westfälischen Wilhelms-Universität Münster, Münster, 2004

[Venturini93] VENTURINI, G.: *SIA: a supervised inductive algorithm with genetic search for learning attribute based concepts.* Proc. European Conference on Machine Learning, pp. 280-296, Viena, 1993

[Weicker02] WEICKER, KARSTEN: *Evolutionäre Algorithmen.* Teubner Verlag, Stuttgart, 1. Auflage, 2002

[Zadeh65] ZADEH, LOTFI A.: *Fuzzy sets.* Information and Control Vol. 8, pp. 338-353, 1965

[Zadeh72] ZADEH, LOTFI: *A rationale for fuzzy control.* Dynamic Systems, Measurement and Control Vol. 94, pp. 3-4, 1972

[Zell94] ZELL, ANDREAS: *Simulation neuronaler Netze.* Oldenbourg Verlag, München, dritter unveränderter Nachdruck, 2000

[Zhuang98] ZHUANG, QIANG, GAYKO, JENS, KREUTZ, MARTIN: *Evolutionäre Optimierung von Fuzzy-Systemen mit variabler Codierung.* Institut für Neuroinformatik, Ruhr-Universität Bochum, Internal Report 98-7