Ingrid Gerdes
Frank Klawonn
Rudolf Kruse

Evolutionäre Algorithmen

- Genetische Algorithmen
- Strategien und Optimierungsverfahren
- Beispielanwendungen

Mit Online-Service zum Buch

vieweg

Ingrid Gerdes
Frank Klawonn
Rudolf Kruse

Evolutionäre Algorithmen

Computational Intelligence

Herausgegeben von Wolfgang Bibel, Rudolf Kruse und Bernhard Nebel

Aus den Kinderschuhen der „Künstlichen Intelligenz" entwachsen bietet die Reihe breitgefächertes Wissen von den Grundlagen bis in die Anwendung, herausgeben von namhaften Vertretern Ihres Faches.

Computational Intelligence hat das weitgesteckte Ziel, das Verständnis und die Realisierung intelligenten Verhaltens voranzutreiben. Die Bücher der Reihe behandeln Themen aus den Gebieten wie z. B. Künstliche Intelligenz, Softcomputing, Robotik, Neuro- und Kognitionswissenschaften. Es geht sowohl um die Grundlagen (in Verbindung mit Mathematik, Informatik, Ingenieurs- und Wirtschaftswissenschaften, Biologie und Psychologie) wie auch um Anwendungen (z. B. Hardware, Software, Webtechnologie, Marketing, Vertrieb, Entscheidungsfindung). Hierzu bietet die Reihe Lehrbücher, Handbücher und solche Werke, die maßgebliche Themengebiete kompetent, umfassend und aktuell repräsentieren.

Unter anderem sind erschienen:

Sehen und die Verarbeitung visueller Information
von Hanspeter A. Mallot

Information Mining
von Thomas A. Runkler

Methoden wissensbasierter Systeme
von Christoph Beierle und Gabriele Kern-Isberner

Neuro-Fuzzy-Systeme
von Detlef Nauck, Christian Borgelt, Frank Klawonn und Rudolf Kruse

Evolutionäre Algorithmen
von Ingrid Gerdes, Frank Klawonn und Rudolf Kruse

www.vieweg-it.de

Ingrid Gerdes
Frank Klawonn
Rudolf Kruse

Evolutionäre Algorithmen

Genetische Algorithmen – Strategien und Optimierungsverfahren – Beispielanwendungen

vieweg

Bibliografische Information Der Deutschen Bibliothek
Die Deutsche Bibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über <http://dnb.ddb.de> abrufbar.

Die Wiedergabe von Gebrauchsnamen, Handelsnamen, Warenbezeichnungen usw. in diesem Werk berechtigt auch ohne besondere Kennzeichnung nicht zu der Annahme, dass solche Namen im Sinne von Warenzeichen- und Markenschutz-Gesetzgebung als frei zu betrachten wären und daher von jedermann benutzt werden dürfen.

Höchste inhaltliche und technische Qualität unserer Produkte ist unser Ziel. Bei der Produktion und Auslieferung unserer Bücher wollen wir die Umwelt schonen: Dieses Buch ist auf säurefreiem und chlorfrei gebleichtem Papier gedruckt. Die Einschweißfolie besteht aus Polyäthylen und damit aus organischen Grundstoffen, die weder bei der Herstellung noch bei der Verbrennung Schadstoffe freisetzen.

1. Auflage Juli 2004

Alle Rechte vorbehalten
© Friedr. Vieweg & Sohn Verlag/GWV Fachverlage GmbH, Wiesbaden 2004

Der Vieweg Verlag ist ein Unternehmen von Springer Science+Business Media.
www.vieweg.de

Das Werk einschließlich aller seiner Teile ist urheberrechtlich geschützt. Jede Verwertung außerhalb der engen Grenzen des Urheberrechtsgesetzes ist ohne Zustimmung des Verlags unzulässig und strafbar. Das gilt insbesondere für Vervielfältigungen, Übersetzungen, Mikroverfilmungen und die Einspeicherung und Verarbeitung in elektronischen Systemen.

Umschlaggestaltung: Ulrike Weigel, www.CorporateDesignGroup.de

Gedruckt auf säurefreiem und chlorfrei gebleichtem Papier.

ISBN-13: 978-3-528-05570-7 e-ISBN-13: 978-3-322-86839-8
DOI: 10.1007/978-3-322-86839-8

Vorwort

Evolutionäre Algorithmen sind eine Klasse von Verfahren, mit denen – angelehnt an das Vorbild der biologischen Evolution – Problemlösungen automatisch generiert werden. Die zugrunde liegende Idee ist faszinierend einfach. Man definiert eine geeignete Kodierung für die generelle Struktur einer Problemlösung sowie eine Zielfunktion, die bewertet, wie gut oder schlecht eine Lösung ist. Den Rest überlässt man dem Computer, der eine Startpopulation von zufälligen Lösungen erzeugt. Mit Techniken, die der biologischen Evolution zum Teil sehr abstrakt nachempfunden sind, werden Nachkommen erzeugt. Zu diesen Techniken zählen unter anderen Mutation, die bestehende Lösungen zufällig verändert, und Rekombination, bei der aus zwei Lösungen eine kombinierte Lösung erstellt wird. Die vorher definierte Zielfunktion bewertet die Lösungen und nach dem Prinzip „Survival of the fittest" – die Besten haben also die größten Überlebens- und Vermehrungschancen. Im nächsten Schritt werden die besten Lösungen selektiert, um wiederum Nachkommen zu erzeugen. Auf diese Weise entwickeln sich aus den im Allgemeinen sehr schlechten Zufallslösungen der Startpopulation über Generationen hinweg immer bessere Lösungen bis hin zur optimalen oder zumindest nahezu optimalen Lösung.

Auch wenn dieses Prinzip zunächst einfach und einleuchtend erscheint, so erfordert eine gezielte und effiziente Anwendung evolutionärer Algorithmen ein etwas tiefer gehendes Verständnis der Techniken und ihrer Eigenschaften. Es wäre auch etwas zu viel verlangt, wenn es bei einem so universellen Werkzeug wie den evolutionären Algorithmen ausreichend wäre, seine Problembeschreibung hineinzugeben und nach etwas Rechenzeit die optimale Lösung herauszubekommen. Ein gewisses Grundverständnis der Verfahren ist erforderlich, um sie nutzbringend einsetzen zu können. Allerdings geht es hierbei primär um das konzeptionelle Verständnis, erst in zweiter Linie um vertiefte mathematische Modellierung. Dieser Trend zu Werkzeugen,

- die relativ universell einsetzbar sind,
- für die keine aufwendige mathematische Modellierung erforderlich ist,
- die mit Parametern oder Strategien operieren, die relativ einfach verständlich und steuerbar sind, und
- die eventuell einen etwas höheren Rechenaufwand als spezielle problemoptimierte Verfahren benötigen,

zeigt sich ganz allgemein im Bereich der Computational Intelligence[1] Methoden, zu denen auch evolutionäre Algorithmen zählen. Aus diesem Grund haben wir uns auch dazu entschlossen, in diesem Buch neben einer Einführung in die evolutionären Algorithmen den Schwerpunkt bei den ausführlicheren Beispielanwendungen auf die Fuzzy-Systeme

[1] Unter dem Begriff Computational Intelligence werden Methoden subsumiert, die – im Gegensatz zur klassischen künstlichen Intelligenz – nur auf elementares, nicht unbedingt strikt formalisierbares Wissen zurückgreifen und sich dafür mehr auf Daten und Rechenmethoden stützen.

zu legen, die ebenfalls ein Teilgebiet der Computational Intelligence sind. Ein weiterer interessanter Aspekt bei den Fuzzy-Systemen besteht darin, dass sie nicht nur mit Hilfe evolutionärer Algorithmen optimiert, sondern dass sie auch zur Steuerung evolutionärer Algorithmen eingesetzt werden können, um schneller bessere Lösungen zu erhalten.

Es gibt verschiedene Richtungen im Bereich der evolutionären Algorithmen, die lange Zeit und zum Teil auch heute noch Konkurrenzkämpfe darüber ausfechten, wer die besseren Methoden besitzt. In diesem Buch versuchen wir, die unterschiedlichen Konzepte dieser Schulen relativ undogmatisch wiederzugeben und plädieren dafür, sich je nach Problem die am besten geeigneten Techniken herauszusuchen und nicht davor zurückzuschrecken, verschiedene Strategien miteinander zu kombinieren, wenn es erfolgversprechender erscheint. Das Anliegen dieses Buches besteht darin, einen Überblick über das Gebiet der evolutionären Algorithmen zu geben und den Leser in die Lage zu versetzen, die richtigen Techniken für seine spezifischen Anwendungen auszuwählen und gegebenenfalls entsprechend anzupassen.

Es gibt inzwischen eine so große Menge an frei verfügbarer Software, in der die verschiedensten Ansätze aus dem Bereich der evolutionären Algorithmen implementiert sind, angefangen bei sehr allgemeinen bis hin zu äußerst speziellen Techniken und Anwendungsbeispielen, so dass wir darauf verzichtet haben, diesem Buch eigene Software beizufügen. Statt dessen haben wir unter

http://fuzzy.cs.uni-magdeburg.de/books/evoalg/

eine Homepage eingerichtet, die dieses Buch begleitet. Auf dieser Homepage befinden sich unter anderem:

- Verweise auf umfangreiche Sammlungen von Software-Paketen zu evolutionären Algorithmen,
- eine Reihe von Übungsaufgaben zu evolutionären Algorithmen,
- die in diesem Buch enthaltenen Abbildungen, die zu Anschauungszwecken in Lehrveranstaltungen verwendet werden können, sowie
- weitere Zusatzinformationen und Links zu diesem Buch und zum Thema des Buches.

Wir möchten an dieser Stelle zumindest einige Personen erwähnen, ohne die dieses Buch nie – oder zumindest nicht in dieser Form – entstanden wäre. Auch wenn für das Verständnis der evolutionären Algorithmen äußerst elementare Kenntnisse der biologischen Evolution hinreichend sind, wollten wir für den interessierten Leser wenigstens einen kurzen Überblick über Grundbegriffe der biologischen Evolution in dieses Buch mit aufnehmen. Ein ganz besonderer Dank gilt Herrn Marco Temme, der den Anhang A zu diesem Themenkomplex verfasst und uns mit freundlicher Genehmigung zur Verfügung gestellt hat.

Herr Christian Haul hat die große Mühe auf sich genommen, unser handschriftliches Vorlesungsmanuskript zum Thema evolutionäre Algorithmen in elektronische Form umzusetzen. Insbesondere viele von ihm per Computer erstellte Bilder finden sich in diesem Buch wieder. Schließlich haben viele unserer Doktoranden und Diplomanden mit

ihren Arbeiten zur Entstehung dieses Buches beigetragen. Besonders erwähnen möchten wir unter ihnen Gerriet Backer, Bettina Fischer, Jörn Hopf, Jens Kinzel, Maic Masuch, Markus Schröder, Annette Temme und Michael Voigt.

Braunschweig, im Juni 2004

Ingrid Gerdes
Frank Klawonn
Rudolf Kruse

Inhaltsverzeichnis

Vorwort **V**

1 Einleitung **1**

2 Optimierungsprobleme **5**
2.1 Beispiele 5
2.2 Der Suchraum 6
2.3 Die Zielfunktion 7
2.4 Die Struktur des Suchraums und der Zielfunktion 10
2.5 Was ist Optimierung? 12

3 Optimierungsverfahren **15**
3.1 Analytische Lösung des Optimierungsproblems 15
3.2 Gradientenverfahren 16
3.3 Newton-Verfahren 17
3.4 Optimierung mit Nebenbedingungen 19
3.5 Tabu-Suche 21
3.6 Greedy-Heuristiken 23
3.7 Hillclimbing 25
3.8 Simulated Annealing 26
3.9 Threshold Accepting 28
3.10 Sintflut-Algorithmus 28
3.11 Ameisenkolonieoptimierung 29
3.12 Grundsätzliche Elemente von Optimierungsstrategien 31

4 Genetische Algorithmen und Optimierung **33**
4.1 Biologische Evolution 33
4.2 Kanonische Genetische Algorithmen: Struktur und Operatoren . . . 36

5 Theoretischer Hintergrund **47**
5.1 Schema-Theorem und Building-Block-Hypothese 50
5.2 Konvergenzbetrachtungen 56
5.3 Vorzeitige Konvergenz 61

6 Problemangepasste Operatoren und Verfahren **63**
6.1 Anwendungsbereich 63
6.2 Kodierung und Startpopulation 64
6.3 Fitnessfunktion 68
6.4 Selektion 79
6.5 Rekombination und Reparaturmechanismen 87

6.6 Mutation 98
6.7 Weitere Aspekte 100
6.8 Evolutionäre Algorithmen zur Optimierung von Flugrouten 110

7 Klassifizierung evolutionärer Algorithmen 115
7.1 Evolutionsstrategien 115
7.2 Evolutionäre Programmierung 125
7.3 Genetische Programmierung 127
7.4 Weitere evolutionäre Algorithmen 143
7.5 Parallele und hybride Ansätze 154
7.6 Lernende Classifier Systeme 158

8 Testumgebungen 163
8.1 Aufbau einer Testumgebung 163
8.2 Funktionen 165
8.3 Kombinatorische Optimierungsprobleme 174
8.4 Strategieentwicklung 175

9 Fuzzy-Systeme 179
9.1 Grundprinzipien 179
9.2 Fuzzy-Mengen 180
9.3 Fuzzy-Regler 183
9.4 Fuzzy-Klassifikatoren 188
9.5 Fuzzy-Clusteranalyse 189

10 Kombinationen evolutionärer Algorithmen mit Fuzzy-Systemen 194
10.1 Fuzzy-Regler-Optimierung: Ein ausführliches Beispiel 194
10.2 Optimierung von Fuzzy-Systemen mit evolutionären Algorithmen . . 208
10.3 Fuzzy-Clustering mit evolutionären Algorithmen 209
10.4 Steuerung evolutionärer Algorithmen mit Fuzzy-Regeln 213

A Anhang: Biologische Evolution und evolutionäre Algorithmen 217
A.1 Die biologische Evolution 217
A.2 Terminolgie der evolutionären Algorithmen 226

B Anhang: NP-Vollständigkeit 227
B.1 Die Klasse $\mathcal{P}$ 228
B.2 Die Klasse $\mathcal{NP}$ 228

Literaturverzeichnis 230

Index 249

1 Einleitung

Lebewesen sind vollendete Problemlöser. In der Vielzahl der Aufgaben, die sie bewältigen, übertreffen sie die besten Computerprogramme bei weitem – zur besonderen Frustration der Programmierer, die Monate oder gar Jahre harter geistiger Arbeit für einen Algorithmus aufwenden, während Organismen ihre Fähigkeiten durch den scheinbar ziellosen Mechanismus der Evolution erwerben.

J.H. Holland (Spektrum der Wissenschaft 9/1992)

Dieses Zitat von John Holland zielt weniger auf die biologische Evolution selbst, sondern viel mehr auf die Idee, Prinzipien der Evolution zu nutzen, um Probleme mit dem Computer zu lösen. Die klassische Strategie zur Lösung von Problemen mit dem Computer besteht darin, auf der Grundlage möglichst detaillierten Wissens über die Struktur und die Eigenschaften des Problems ein effizientes Verfahren zu entwickeln, das die gewünschte Lösung konstruiert. Voraussetzung für diese Vorgehensweise ist zum einen, dass die benötigten strukturellen Informationen über das Problem genau bekannt sind, und zum anderen, dass Experten und Programmierer die Zeit und den Aufwand investieren, an einem entsprechenden Modell und seiner Implementierung zu arbeiten. Die erste Voraussetzung ist nicht immer gegeben, die zweite bedeutet meist erhebliche Kosten. Experten müssen zunächst ein Modell erstellen, auf der Basis dieses Modells geeignete Problemlösungsstrategien entwickeln und diese Verfahren schließlich implementieren.

Mit der zunehmenden Leistungsfähigkeit von Rechnern hat in den letzten Dekaden eine Entwicklung begonnen, die einen alternativen Weg einschlägt. Man verwendet zur Problemlösung relativ universelle Strategien, in die sich strukturelles Wissen über das Problem ohne großen Formalisierungs- oder Modellierungsaufwand integrieren lässt. Der Preis besteht häufig in einem deutlich größeren Rechenaufwand, der aber durch die Geschwindigkeit heutiger Computer vielfach kompensiert werden kann und zudem nahezu kostenlos im Vergleich zur Bezahlung von Experten ist. Techniken, die auf dieser Philosophie basieren, werden zum Bereich der *Computational Intelligence* gezählt. Der Begriff Computational Intelligence wurde Anfang der neunziger Jahre geprägt [Bez92, Bez94, ZMIR94].

Evolutionäre Algorithmen[1] – neben Fuzzy-Systemen und neuronalen Netzen eines der wichtigsten Gebiete im Bereich Computational Intelligence – bilden eine Klasse von Verfahren, die sich die Prinzipien der biologischen Evolution zum Vorbild nehmen, um primär Optimierungsprobleme zu lösen. Wie in vielen Bereichen der Technik, sei es im Flugzeugbau oder bei den (künstlichen) neuronalen Netzen, geht es nicht darum, das natürliche Vorbild möglichst exakt nachzubilden. Die Flügelform von Flugzeugen ist zwar denen von Vögeln ähnlich, die Antriebstechniken sind jedoch gänzlich verschieden.

[1] Im Englischen verwendet man neben dem Begriff „evolutionary algorithms" auch „evolutionary computation".

Auch die neuronalen Netze, die in der Informatik verwendet werden, haben sich sehr weit von realistischen Modellen natürlicher Neuronennetzwerke entfernt. Genauso verhält es sich bei den evolutionären Algorithmen. Die Mechanismen der biologischen Evolution sind enorm komplex und bis heute noch nicht vollständig verstanden. Selbst wenn ein sehr exaktes Modell der biologischen Evolution bekannt wäre, sind die Populationsgrößen und die Zeiträume, die dabei eine Rolle spielen, für technische Anwendungen und Realisierungen auf dem Computer indiskutabel.

Die ersten Ansätze zur Ausnutzung von Konzepten der biologischen Evolution wurde bereits vor fast fünfzig Jahren publiziert [Box57, Fra57, Fri58, And53, Bre62]. Später entwickelten sich neben anderen Ansätzen die folgenden Hauptrichtungen, die auch heute noch von großer Bedeutung sind:

Evolutionsstrategien wurden in den sechziger Jahren von Rechenberg und Schwefel für Optimierungsprobleme aus dem technisch-physikalischen Bereich entwickelt und wurden zuerst in [Rec73, Sch75] publiziert. Evolutionsstrategien verwenden kontinuierliche Parameter und Mechanismen zur Selbstadaption ihrer Steuerungsparameter. Sie sind vor allem auf Optimierungsprobleme mit kontinuierlichen Parametern zugeschnitten.

Genetische Algorithmen gehen zum einen auf Holland, der sie eher aus der Sicht lernender oder adaptiver Systeme betrachtet hat [Hol75], und zum anderen auf De Jong zurück, der sie erstmals für Optimierungsprobleme eingesetzt hat [DJ75]. Genetische Algorithmen verwenden in ihrer ursprünglichen Form eine binäre Kodierung, so dass sie sich vor allem für kombinatorische Optimierung eignen.

Genetische Programmierung wurde – wie sie in der heutigen Form verwendet wird – vor allem durch Koza entwickelt [Koz90] und durch seine ersten beiden Bücher [Koz92, Koz94] populär. Im Gegensatz zu den Evolutionsstrategien und den genetischen Algorithmen, die vor allem auf Parameteroptimierung ausgerichtet sind, hat die genetische Programmierung die Entwicklung von optimalen Strategien in Form von einfachen Computerprogrammen zum Ziel.

Von historischer Bedeutung ist auch die *evolutionäre Programmierung*, die auf L.J. Fogel [FOW66] und D.B. Fogel [Fog92] zurückgeht, aufgrund einiger einschränkender Annahmen heute aber eine geringere Rolle als die anderen Richtungen spielt.

Lange Zeit haben vor allem Evolutionsstrategien, genetische Algorithmen und evolutionäre Programmierung als konkurrierende Modelle nebeneinander existiert. Einzelne Vertreter einer Richtung haben versucht, die Überlegenheit ihres favorisierten Ansatzes gegenüber den anderen Ansätzen theoretisch oder empirisch zu belegen. Die Oberbegriffe „evolutionäre Algorithmen" und international vor allem „evolutionary computation" haben sich erst später etabliert, als bereits alle Einzelrichtungen Erfolge in den Anwendungen vorweisen konnten.

Das gemeinsame Grundgerüst aller evolutionären Algorithmen sieht folgendermaßen aus. Zunächst muss für ein Problem definiert werden, wie eine Lösung – egal, ob sie gut oder schlecht ist – prinzipiell strukturiert ist. Dies definiert den Suchraum. Des weiteren ist eine Zielfunktion erforderlich, die die Elemente des Suchraums bewertet und

ausdrückt, wie gut oder schlecht eine Lösung ist. Evolutionäre Algorithmen erzeugen dann eine Startpopulation von meist zufälligen Lösungen. Ausgehend von der Startpopulation werden schrittweise neue Generationen von Lösungen erzeugt, indem zunächst genetische Operatoren angewendet werden. Hier sind vor allem die Mutation, die eine meist kleine zufällige Änderung an einer Lösung vornimmt, und das Crossover zu nennen, das zwei gegebene Lösungen auf geeignete Art und Weise kreuzt, d.h. kombiniert. In die nächste Generation werden in der Regel nur die besten Lösungen übernommen, was dem Selektionsprinzip „survival of the fittest" entspricht. Auf diese Weise werden im Laufe der Generationen immer bessere Lösungen erzeugt, bis der Prozess abgebrochen wird, weil beispielsweise eine genügend gute Lösung gefunden wurde oder die Population konvergiert ist, d.h. die Folgegenerationen unterscheiden sich kaum noch von den Vorgängergenerationen.

Genetischen Algorithmen, Evolutionsstrategien, genetische Programmierung und alle andere evolutionären Algorithmen basieren auf diesem Grundprinzip, das im Detail unterschiedlich realisiert wird. Das Ziel dieses Buches besteht vor allem in der Vermittlung der zugrunde liegenden Konzepte evolutionärer Algorithmen und deren Bedeutung. Es sollte keine Richtung dogmatisch verfolgt werden, sondern bei einem konkreten Problem die Techniken ausgewählt und kombiniert werden, die für eine schnelle, effiziente und gute Lösung am viel versprechendsten erscheinen.

Für das bessere Verständnis der Prinzipien, auf denen evolutionäre Algorithmen basieren, ist eine gewisse Vertrautheit mit den generellen Strukturen von Optimierungsproblemen und Lösungsverfahren erforderlich. Aus diesem Grund wird in Kapitel 2 zunächst untersucht, durch welche Eigenschaften Optimierungsprobleme charakterisiert werden und auf welche Weisen sie sich formulieren und klassifizieren lassen.

Das 3. Kapitel enthält einen Überblick über eine Reihe klassischer Optimierungstechniken wie das Gradienten- oder Newton-Verfahren, aber ebenso heuristische, neuere Methoden. Auch wenn die dort vorgestellten Verfahren nicht zu den evolutionären Algorithmen zählen, so lassen sich aus ihrer Kenntnis grundsätzliche Strategien für die Optimierung ableiten, die am Ende des Kapitels zusammengefasst werden. Die Berücksichtigung dieser Aspekte erlaubt eine genauere Einordnung und Einschätzung der evolutionären Algorithmen. Darüber hinaus erlaubt eine grundlegende Kenntnis klassischer Methoden eine bessere Bewertung der Nützlichkeit des Einsatzes evolutionärer Verfahren im Vergleich zu den sonstigen Techniken.

Nach einer kurzen Erläuterung der aus dem Bereich der biologischen Evolution entlehnten Begriffe führt Kapitel 4 den kanonischen genetischen Algorithmus als erstes Beispiel eines evolutionären Algorithmus ein, an dem auch die Umsetzung der biologischen Terminologie in Algorithmen und Datenstrukturen erklärt wird.

Kapitel 5 zeigt anhand des kanonischen genetischen Algorithmus exemplarisch auf, mit welchen Techniken evolutionäre Algorithmen theoretisch untersucht werden können, um zu verstehen, wie sie eine Lösung konstruieren. Resultate wie das so genannte Schema-Theorem sowie die daraus abgeleitete, allerdings in ihrer Bedeutung für genetische Algorithmen eher fragwürdige Building-Block-Hypothese werden vorgestellt. Auch Konvergenzeigenschaften, ob ein Algorithmus die beste Lösung finden kann und wie stark seine Neigung ist, in einem unerwünschten lokalen Optimum zu konvergieren, werden analysiert.

Erweiterte Techniken für evolutionäre Algorithmen, die zum Teil auch aus der biologischen Evolution entliehen sind, bilden das Thema des 6. Kapitels. Am Ende des Kapitels stellen wir eine Anwendung aus dem Bereich der Luftfahrt vor, die veranschaulicht, wie bei komplexeren Problemen Modifikationen an einfach strukturierten evolutionären Algorithmen vorgenommen werden können, um bessere Lösungen zu erhalten.

Kapitel 7 beinhaltet einen Überblick über die bereits am Anfang erwähnten Klassen evolutionärer Algorithmen wie Evolutionsstrategien, genetische Programmierung und evolutionäre Programmierung sowie eine Reihe weiterer Techniken, unter Anderem parallele Ansätze und lernende Classifier Systeme.

Neben den theoretischen Analysen, die bereits in Kapitel 5 ausgeführt und zum Teil im Kapitel 7 noch einmal für die entsprechenden Verfahren angesprochen wurden, spielen empirische Untersuchungen anhand von speziell entworfenen Tests eine wichtige Rolle bei der Bewertung evolutionärer Algorithmen. Kapitel 8 ist dieser Thematik gewidmet.

Kapitel 9 gibt eine kurze Einführung in das Gebiet der Fuzzy-Systeme. Aufbauend auf diese Einführung werden im 10. Kapitel Kombinationen von evolutionären Algorithmen mit Fuzzy-Systemen vorgestellt. Zum einen dient die Optimierung der Fuzzy-Systeme als ausführliches Beispiel dazu um aufzuzeigen, wie evolutionäre Algorithmen für konkrete Probleme eingesetzt und speziell an sie angepasst werden können. Zum anderen lassen sich Fuzzy-Systeme verwenden, um heuristisches Wissen zur Steuerung von evolutionären Algorithmen einzusetzen.

Auch wenn die Erläuterung der evolutionären Algorithmen nur ein sehr rudimentäres Verständnis der Begriffe, Konzepte und Prozesse der biologischen Evolution voraussetzt, so wollten wir dennoch nicht darauf verzichten, eine etwas stärker biologisch orientierte Sicht auf diese Dinge einzubeziehen. Aus diesem Grund haben wir den Anhang A, der von Marco Temme verfasst wurde, in das Buch mit aufgenommen.

Sowohl aus theoretischer als auch aus praktischer Sicht spielt die komplexitätstheoretische Betrachtung von Optimierungsalgorithmen und -problemen eine wichtige Rolle, insbesondere im Bereich der kombinatorischen Optimierung, wo häufig NP-vollständige Probleme eine wichtige Rolle spielen. Anhang B beinhaltet eine kurze, eher informelle Einführung in diese Begrifflichkeiten und Fragestellungen.

Es wurde bewusst darauf verzichtet, diesem Buch ein Software-Paket wie in [GH03] beizulegen. Die bereits im Vorwort erwähnten Internet-Adressen bieten umfangreiche Verweise auf frei verfügbare Programme zu nahezu allen Verfahren, die in diesem Buch angesprochen werden.

2 Optimierungsprobleme

Das Ziel dieses Kapitels besteht in einer Definition und Klassifikation von Optimierungsproblemen. Zunächst sollen aber einige Beispiele für Optimierungsprobleme erläutert werden, die zur Einführung der notwendigen Begriffe geeignet sind und die Optimierungsproblemen zugrunde liegenden Strukturen veranschaulichen.

2.1 Beispiele

Beispiel 2.1 (Das Problem des Handlungsreisenden (TSP)) wird im Englischen als *Travelling Salesman Problem (TSP)* bezeichnet. Ein Handlungsreisender möchte auf einer Rundreise n Städte $\{s_1, \ldots, s_n\}$ besuchen und will seine Reise so planen, dass die Rundtour möglichst kurz ist. Man geht dabei davon aus, dass die jeweilige Reisedauer zwischen zwei Städten bekannt und konstant ist, d.h. nicht davon abhängt, wann die Reise durchgeführt wird.

Das TSP findet sich in vielen anderen Anwendungen wieder. Es muss sich nicht um einen Handlungsreisenden handeln, der auf einer Rundreise im Allgemeinen nur eine sehr kleine Anzahl von Städten besuchen wird. Es kann sich auch um einen Paketdienst handeln, der eine größere Anzahl von Paketen an verschiedenen Orten (innerhalb einer Stadt) ausliefern muss. Wenn ein Roboter unter Umständen tausende von Löchern auf einer Platine bohren soll, muss ebenfalls ein TSP gelöst werden; denn je kürzer die zurückgelegten Wege für den Roboter sind, desto schneller können die Platinen produziert werden.

Das TSP kann auch in wesentlich komplexerer Form gestellt werden. Beispielsweise könnte der Paketdienst bei seiner Planung berücksichtigen, dass die Reisedauer – d.h. die Fahrt – zwischen zwei Zielen von der Tageszeit abhängt, je nachdem, wie stark die entsprechende Strecke zu der Tageszeit befahren ist. Eine andere Verallgemeinerung des TSP besteht darin, dass der Paketdienst einen Fuhrpark von mehreren Fahrzeugen verwaltet und neben den Routen der der Fahrzeuge auch die Verteilung der Pakete auf die Fahrzeuge planen muss.

Beispiel 2.2 (Regression) Die einfachste Form der Regression besteht in der Bestimmung einer Ausgleichsgeraden. Man misst n Wertepaare $(x_1, y_1), \ldots, (x_n, y_n)$ zweier Größen x und y und vermutet einen – eventuell leicht verrauschten – linearen Zusammenhang zwischen x und y, d.h. $y \approx ax + b$ mit noch zu bestimmenden Parametern a und b. Beispielsweise könnte die Größe x die Zeit darstellen und y die zurückgelegte Strecke eines Fahrzeugs, das sich mit annähernd konstanter Geschwindigkeit bewegt. Geht man davon aus, dass das Fahrzeug sich nicht mit konstanter Geschwindigkeit bewegt, sondern eine annähernd konstante Beschleunigung erfährt, würde man anstelle der Ausgleichs- oder Regressionsgeraden $y = ax + b$ eine quadratische Regressionsfunktion ansetzen: $y \approx ax^2 + bx + c$.

Allgemein versteht man unter *Regression* die Vorhersage einer reellen Größe aus einer oder mehreren anderen reellen Größen. Ein anderes Regressionsproblem besteht z.B. in der Vorhersage des Wachstums einer mit ausreichend Nährstoffen versorgten Bakterienkultur. Hier würde man den Zusammenhang $y \approx ae^{bx}$ erwarten, wenn die Größe x wiederum für die Zeit steht und y die Masse oder Anzahl der Bakterien angibt.

Die vorherzusagende Größe y eines Regressionsproblems kann auch von mehreren Größen abhängen. Möchte man beispielsweise den Stromverbrauch y eines neu geplanten Wohngebietes aus der Kenntnis des Stromverbrauchs bereits bestehender Wohngebiete vorhersagen, wäre wahrscheinlich $y \approx a_0 + a_1x_1 + a_2x_2 + a_3x_3 + a_4x_4$ ein sinnvoller Ansatz, wobei x_1 für die erwartete Anzahl der Einpersonen-, x_2 der Zweipersonen-, x_3 der Dreipersonen- und x_4 der Vier- und Mehrpersonenhaushalte steht.

Beispiel 2.3 (Regelung) Im Leerlauf soll die Benzinzufuhr für den Motor eines Autos automatisch bestimmt werden in Abhängigkeit von der aktuellen Drehzahl und der aktuellen Änderung der Drehzahl. Eine derartige Regelung ist erforderlich, da der Motor auch im Stand belastet werden kann, beispielsweise durch das Einschalten der Klimaanlage oder die Benutzung der Servolenkung.

2.2 Der Suchraum

Bei der Formulierung eines Optimierungsproblems muss zunächst ein *Suchraum* $\mathcal{S}$ spezifiziert werden, der die Menge aller möglichen Lösungen enthält.

Bei der Bestimmung einer Regressionsgeraden, aber auch bei der Bestimmung der Regressionsfunktion $y = ae^{bx}$ wäre $\mathcal{S} = \mathbb{R}^2$ zu wählen, da eine Lösung der Optimierungsaufgabe aus der Bestimmung der beiden Parameter a und b besteht. Entsprechend würde man für die in Beispiel 2.2 angegebene Funktion Regressionsfunktion für den Stromverbrauch als Suchraum $\mathcal{S} = \mathbb{R}^5$ definieren.

Beim TSP besteht der Suchraum

$$\mathcal{S} = \{(c_1, \ldots, c_n) \mid c_1, \ldots, c_n \in \{s_1, \ldots, s_n\} \text{ und } c_i \neq c_j \text{ für } i \neq j\}$$

aus der Menge aller Permutationen der Städte $s_1, \ldots, s_n$. Im Gegensatz zu den Suchräumen bei den Regressionsproblemen hat der Suchraum hier eine komplexere Struktur. Während bei der Regression prinzipiell alle Kombinationen der einzelnen Werte zugelassen sind, darf beim TSP keine Stadt doppelt besucht werden, so dass nicht alle Elemente des größeren Suchraums

$$\widehat{\mathcal{S}} = \{s_1, \ldots, s_n\}^n$$

korrekte Rundreisen repräsentieren.

Für viele Optimierungsverfahren ist es günstiger, auf einem einfacher strukturierten Suchraum zu operieren. Deswegen wird häufig ein größerer, einfach strukturierter Suchraum $\widehat{\mathcal{S}}$ anstelle des korrekten Suchraums $\mathcal{S}$ betrachtet. Zur Vermeidung inkorrekter Lösungen gibt es zwei Strategien:

- Das Optimierungsverfahren erlaubt es, Nebenbedingungen zu berücksichtigen. Die Nebenbedingungen sind dann so zu definieren, dass nur korrekte Lösungen die Nebenbedingungen erfüllen.
- Die noch zu definierende Zielfunktion, die angibt, wie gut oder schlecht eine Lösung ist, wird um einen Bestrafungsterm erweitert, der nicht korrekten Lösungen negativ bewertet.

Der Suchraum im Falle der Leerlaufdrehzahlregelung im Beispiel 2.3 ist wesentlich schwieriger festzulegen. Man sucht eine Funktion

$$b : (0, \infty) \times (-\infty, \infty) \to [0, \infty),$$

die jedem gemessenen Wertepaar aus aktueller Drehzahl d und Drehzahländerung Δd eine geeignete Menge $b(d, \Delta d)$ des Benzinflusses zuordnet. Realistischerweise sollte man die Intervalle der Wertebereiche durch geeignete Grenzen beschränken. Man wird in diesem Beispiel kaum die Menge aller derartiger Funktionen als Suchraum wählen. Man wird sich eher auf eine Klasse parametrisierter Funktionen einschränken.

2.3 Die Zielfunktion

Der Suchraum legt nur fest, wie mögliche Lösungen eines Optimierungsproblems grundsätzlich aussehen. Bei der Optimierung geht es darum, nicht irgendeine, sondern eine gute, am besten optimale Lösung zu finden. Dazu muss eine Bewertungs- oder *Zielfunktion* $f : \mathcal{S} \to \mathbb{R}$ definiert werden, die jedem Element des Suchraums – d.h. jeder möglichen Lösung – eine Bewertung zuordnet. Je nachdem, ob diese Bewertung $f(s)$ für $s \in \mathcal{S}$ die Güte der Lösung s angibt oder – etwa in Form einer Fehlerfunktion – ausdrückt, wie fehler- oder mangelhaft s ist, wird ein Maximum bzw. Minimum der Zielfunktion gesucht. Zwischen der Lösung von Minimierungs- und Maximierungsproblemen besteht kein prinzipieller Unterschied. Jedes Minimierungsproblem mit der Zielfunktion f kann in ein äquivalentes Maximierungsproblem mit der Zielfunktion $\hat{f} = -f$ transformiert werden.

2.3.1 Die Wahl der Zielfunktion

Beim TSP ordnet die Zielfunktion einer gegebenen Rundreise ihre Länge zu. Diese offensichtliche und eindeutige Wahl einer Zielfunktion ist bei Optimierungsproblemen häufig nicht gegeben. Selbst im Falle der Bestimmung einer Regressionsgeraden ist die Wahl der Zielfunktion keineswegs eindeutig. In den meisten Fällen wird zwar die Methode der kleinsten Fehlerquadrate verwendet, d.h., die Fehlerfunktion bestimmt die Summe der quadratischen Fehler – oder gleichwertig dazu den mittleren quadratischen Fehler – in y-Richtung. Bei gegebenem, zu approximierendem Datensatz $(x_1, y_1), \ldots, (x_n, y_n)$ ist diese Fehlerfunktion durch

$$f(a,b) = \frac{1}{n}\sum_{i=1}^{n}(ax_i + b - y_i)^2 \tag{2.1}$$

definiert. Die Aufgabe der Zielfunktion besteht darin anzugeben, wie gut – bzw. als Fehlerfunktion wie schlecht – die Gerade $y = ax+b$ den gegebenen Datensatz approximiert. Sicherlich ist der mittlere quadratische Fehler ein sinnvolles Maß dafür. Allerdings kann man sich auch andere Fehlermaße vorstellen, um zu beurteilen wie gut bzw. schlecht die Gerade zu den Daten passt. Beispiele für derartige Fehlermaße sind

- der mittlere absolute Fehler in y-Richtung,
- der maximale absolute Fehler in y-Richtung,
- der mittlere absolute oder quadratische euklidische Abstand der Daten zur Geraden,
- der maximale absolute oder quadratische euklidische Abstand der Daten zur Geraden.

Durch Verwendung quadratischer Fehler werden extreme Ausreißer stärker gewichtet als bei der Berücksichtigung absoluten Fehler. Der maximale Fehler lässt bei der Optimierung im Gegensatz zum mittleren Fehler keine Kompromisse zwischen vielen gut approximierten und wenigen schlecht angenäherten Daten zu. Alle Daten sollten bei Verwendung des maximalen Fehlers möglichst gleich gut angenähert werden. Jede dieser Zielfunktion hat ihre Berechtigung, ihre Vor- und Nachteile.

Im Beispiel der Leerlaufdrehzahlregelung ist die Zielfunktion noch schwieriger zu definieren. Die Regelung sollte gewährleisten, dass der Motor nicht in einen zu niedrigen Drehzahl bereich gerät, da er sonst abgewürgt werden könnte. Andererseits sollte die Regelung für einen möglichst geringen Benzinverbrauch sorgen. Neben der Aufgabe, eine geeignete Zielfunktion für diese beiden Einzelziele zu definieren, muss noch festgelegt werden, was man unter der gleichzeitigen Optimierung beider Kriterien versteht. Diese Schwierigkeit tritt bei vielen Optimierungsproblemen auf und wird als *Mehrkriterienoptimierung* bezeichnet.

2.3.2 Mehrkriterienoptimierung

In unserem täglichen Leben lösen wir ständig Probleme der Mehrkriterienoptimierung. Bei jedem Kauf einer Ware möchte man möglichst hohe Qualität zu einem möglichst geringen Preis erhalten. Diese beiden sich in dem meisten Fällen widersprechenden Einzelziele sollen gleichzeitig verfolgt werden. Häufig stehen sogar mehr als zwei sich widersprechende Kriterien in Konkurrenz. Beim Autokauf wünscht man sich

- einen niedrigen Preis,
- einen geringen Kraftstoffverbrauch,
- möglichst viel Komfort wie elektrische Fensterheber oder eine Klimaanlage, was üblicherweise einen etwas leistungsfähigeren Motor und damit einen höheren Preis und Benzinverbrauch bedingt

- und viele weitere individuelle Kriterien wie möglichst viel Platz im Innenraum, aber gleichzeitig einen möglich kleinen Wagen, der in jede Parklücke passt.

Allgemein sind bei der Mehrkriterienoptimierung mit k Optimierungskriterien auch k Zielfunktionen $f_i : \mathcal{S} \to \mathbb{R}$ $(i = 1, \ldots, k)$ gegeben. Es gibt zwei prinzipielle Ansätze, Mehrkriterienoptimierungsprobleme zu lösen.

Die erste Möglichkeit besteht darin, die einzelnen Zielfunktionen f_i zu einer geeigneten Gesamtzielfunktion f zusammenzufassen. Eine gängige Technik dafür ist die Bildung der gewichteten Summe über die Einzelkriterien

$$f(s) = \sum_{i=1}^{k} w_i f_i(s).$$

Damit die unterschiedlichen Kriterien korrekt zusammenwirken, müssen zunächst die Vorzeichen der Gewichte korrekt gewählt werden. Soll die Gesamtzielfunktion f maximiert werden, erhalten die Zielfunktionen f_i, die ebenfalls zu maximierenden sind, ein positives Gewicht $w_i > 0$, während für die zu minimierenden Zielfunktionen f_i negative Gewichte $w_i < 0$ festgelegt werden. Soll die Gesamtzielfunktion f minimiert werden, ergeben sich genau die umgekehrten Vorzeichen für die Gewichte.

Mit Hilfe der Absolutwerte der Gewichte kann man die Wichtigkeit der einzelnen Kriterien bei der Optimierung beeinflussen. Ein großer Absolutwert eines Gewichtes bedeutet allerdings nichts zwangsläufig, dass das Kriterium einen großen Einfluss bei der Optimierung hat. Wenn die Werte der Zielfunktion f_i im Verhältnis zu den anderen Zielfunktionen nur in einem sehr kleinen Bereich schwanken, wird das entsprechende Kriterium auch bei einem etwas größeren Gewicht w_i keine wesentlich Rolle bei der Optimierung spielen. Man muss also bei der Wahl der Gewichte neben der Wichtigkeit des Kriterien außerdem die Schwankungsbreite der einzelnen Zielfunktionen berücksichtigen.

Der zweite Ansatz bei der Mehrkriterienoptimierung bildet aus den einzelnen Zielfunktionen keine Gesamtzielfunktion. Man belässt die einzelnen Zielfunktionen so, wie sie sind, und sucht nach so genannten *pareto-optimalen* Lösungen. Eine Lösung $s_0 \in \mathcal{S}$ heißt pareto-optimal bzgl. der Zielfunktionen f_i $(i = 1, \ldots, k)$, wenn es keine Lösung $s \in \mathcal{S}$ gibt, die für alle Zielfunktionen mindestens gleichgute Werte wie s_0 und für mindestens eine Zielfunktion sogar einen besseren Wert als s_0 liefert. Man sagt, eine Lösung s_1 *dominiert* eine Lösung s_2, wenn s_1 bei keiner der Zielfunktionen einen schlechteren Wert erhält als s_2. Wenn s_1 die Lösung s_2 dominiert und in mindestens einem Kriterium besser als s_2 ist, so wird s_2 von s_1 *echt dominiert*. Damit ist eine Lösung genau dann pareto-optimal, wenn sie von keiner anderen Lösung echt dominiert wird.

Nach der Bestimmung pareto-optimaler Lösungen muss dann der Experte oder Anwender noch entscheiden, welche dieser Lösungen ausgewählt werden sollte. Auf Mehrkriterienoptimierung und Pareto-Optimalität im Zusammenhang mit evolutionären Algorithmen werden wir im Abschnitt 6.3.5 genauer eingehen.

2.4 Die Struktur des Suchraums und der Zielfunktion

Optimierungsprobleme werden im wesentlichen durch einen Suchraum $\mathcal{S}$ und eine Zielfunktion $f : \mathcal{S}$ definiert. Es können unter Umständen noch Nebenbedingungen eingeführt werden, die nur bestimmte Elemente des Suchraums als korrekte Lösungen zulassen. Bei der Mehrkriterienoptimierung betrachtet man mehrere Zielfunktionen.

In diesem Abschnitt soll die durch den Suchraum und die Zielfunktion definierte Struktur etwas genauer betrachtet werden. Besteht das Optimierungsproblem in der Optimierung zweier reeller Größen, wäre der Suchraum $\mathcal{S} = \mathbb{R}^2$. Ein typisches Beispiel für einen derartigen Suchraum sind einfache Regressionsprobleme, bei denen die Regressionsfunktion nur zwei Parameter besitzt. Die Zielfunktion beschreibt eine Funktion $f : \mathbb{R}^2 \to \mathbb{R}$, die maximiert bzw. im Falle der Regression minimiert werden sollte.

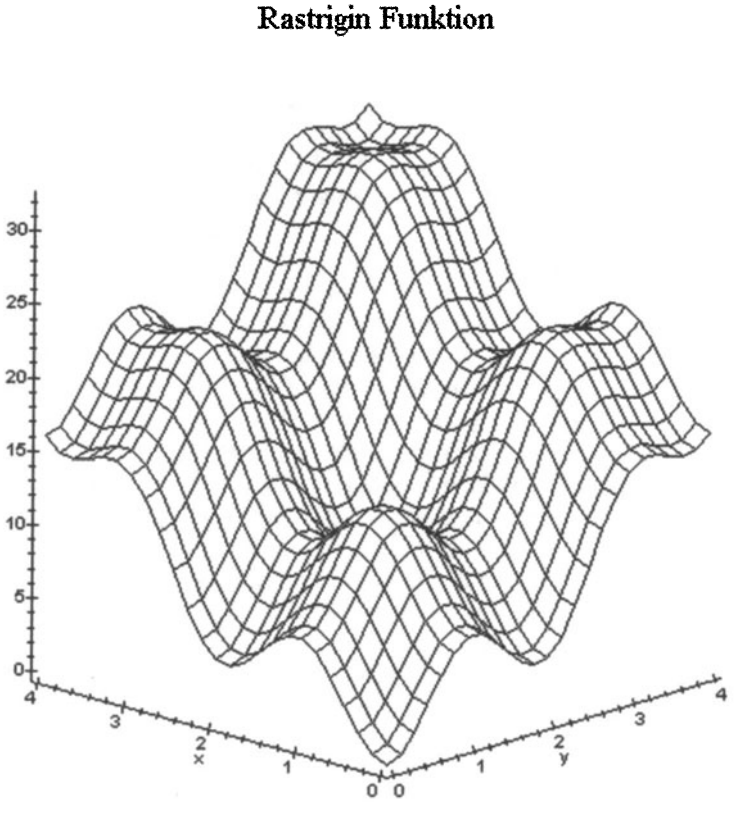


Abbildung 2.1 Beispiel für einen Lösungsraum

Abbildung 2.1 zeigt einen exemplarischen Verlauf einer derartigen Zielfunktion. Die gesuchten Minima und Maxima sind im dargestellten Bereich deutlich erkennbar. Das Minimierungs- oder Maximierungsproblem besteht dann im Auffinden des tiefsten "Tals" bzw. des höchsten Gipfels in dieser "Gebirgslandschaft". Dadurch, dass in diesem Fall nur zwei Parameter zu optimieren sind, lässt sich die Zielfunktion als eine dreidimensionale Gebirgslandschaft – eine Dimension für den Wert der Zielfunktion und je eine Dimension für die beiden Parameter – darstellen. Bei n zu optimierenden Parametern muss man sich die Zielfunktion als abstrakte $(n+1)$-dimensionale Gebirgslandschaft vorstellen, die sich nicht mehr grafisch veranschaulichen lässt.

Optimierungsprobleme können durchaus weit mehr als nur zwei zu optimierende Parameter beinhalten. Als Beispiel sei hier eine Variante der multidimensionalen Skalierung genannt. Bei der multidimensionalen Skalierung versucht man, einen höherdimensionalen Datensatz $D \subset \mathbb{R}^p$ – meistens in der Ebene – zu visualisieren. Das bei der multidimensionalen Skalierung zu lösende Optimierungsproblem stellt sich wie folgt dar. Gegeben ist ein p-dimensionaler Datensatz $D = \{\mathbf{x_1}, \ldots, \mathbf{x_n}\} \subset \mathbb{R}^p$. Gesucht ist eine zweidimensionale Repräsentation $\{\mathbf{y_1}, \ldots, \mathbf{y_n}\} \subset \mathbb{R}^2$ der Daten, die so weit wie möglich die Struktur des p-dimensionalen Datensatzes D erhält. Mit Strukturerhaltung meint man, dass die euklidischen Distanzen zwischen den Daten in D möglichst mit den euklidischen Distanzen der Daten in der zweidimensionalen Repräsentation übereinstimmen sollten. Man definiert daher als zu minimierende Zielfunktion

$$f(\mathbf{y_1}, \ldots, \mathbf{y_n}) = \frac{1}{\sum_{i=1}^{n} \sum_{j=i+1}^{n} \| \mathbf{x_i} - \mathbf{x_j} \|^2} \sum_{i=1}^{n} \sum_{j=i+1}^{n} \left(\| \mathbf{y_i} - \mathbf{y_j} \| - \| \mathbf{x_i} - \mathbf{x_j} \| \right)^2,$$

die auch als *Stress* bezeichnet wird. Da jeder Punkt $\mathbf{y_i} \in \mathbb{R}^2$ ein Parameter der Zielfunktion ist, müssen in diesem Optimierungsproblem $2n$ Parameter angepasst werden. Datensätze mit ca. $n = 1000$ Daten, die zu 2000 zu optimierenden Parametern führen würden, gelten in der Datenanalyse noch nicht als groß.

Auch wenn viele Optimierungsprobleme mehr als nur zwei zu optimierende Parameter erfordern, lässt sich mit Abbildung 2.1 recht gut veranschaulichen, was die eigentliche Problematik ist. Man sucht in einem „Gebirge", das durch die Zielfunktion über dem Suchraum definiert wird, nach den höchsten Gipfeln, wenn die Zielfunktion zu maximieren ist, und nach den tiefsten Tälern bei einer zu minimierenden Zielfunktion. Dieses „Gebirge" muss man sich im Allgemeinen allerdings eher abstrakt vorstellen. Zum einen haben wir bereits gesehen, dass dieses „Gebirge" durch eine größere Anzahl von zu optimierenden Parametern nicht als drei-, sondern als höherdimensionale Struktur anzusehen ist. Zum anderen besteht der Suchraum unter Umständen nicht nur aus einer Anzahl von reellen, zu optimierenden Parametern. Es können diskrete Parameter auftauchen, wie im Beispiel des TSP. Beim TSP besteht der Suchraum auch nicht einfach nur aus dem Produktraum der diskreten Parameter, sondern aus einer komplexeren Substruktur dieses Produktraums, da jede Stadt auf einer Rundreise genau einmal besucht werden sollte. Man spricht von *kombinatorischer Optimierung*, wenn der der Suchraum endlich ist.

Suchräume können eine sehr gemischte Struktur aufweisen. Sie können gleichzeitig kontinuierliche und diskrete Parameter sowie zusätzliche Nebenbedingungen enthalten, die nur bestimmte Kombinationen der Parameterwerte zulassen.

2.5 Was ist Optimierung?

Wir haben im vorhergehenden Abschnitt festgestellt, dass man ein Optimierungsproblem als die Suche nach Gipfeln oder Tälern in einem abstrakten Gebirge auffassen kann. Leider lässt sich dieses abstrakte Gebirge nur selten als Ganzes wie in Abbildung 2.1 veranschaulichen, so dass eine globale Sicht auf dieses Gebirge nur in den seltensten Fällen realisierbar ist.

Welche Möglichkeiten stehen zur Verfügung, um die Gipfel oder Täler in einem derartigen abstrakten Gebirge zu finden? Die Zielfunktion sollte sicherlich auf irgendeine Weise berechnet werden können. Dies bedeutet, dass wir die Höhe des Gebirges an jedem beliebigen Punkt in unserem Suchraum ausrechnen können. Da die Suchräume aber bei kontinuierlichen Parametern unendlich groß sind, erübrigt sich eine erschöpfende Suche. Selbst bei rein diskreten Parametern ist der Suchraum in den meisten Fällen so groß, dass eine erschöpfende Suche auch dann nicht in vernünftiger Zeit durchgeführt werden kann, wenn die schnellsten Rechner zur Verfügung stehen. Beim TSP mit nur 25 Städten besteht der Suchraum aus $25! \approx 1.55 \cdot 10^{25}$ Elementen. Ein Rechner mit 100 Gigahertz Taktfrequenz, der (unrealistischerweise) in jedem Takt eine Lösung berechnen könnte, bräuchte fast fünf Millionen Jahre, um den gesamten Suchraum zu durchforsten.

Das Auswerten der Zielfunktion kann ebenfalls sehr zeitaufwendig und kostenintensiv sein. Das Auswerten der Zielfunktion an einer Stelle des Suchraums kann in manchen Fällen mit wenigen Berechnungen im Rechner erfolgen wie beispielsweise beim TSP, es kann aber auch einen aufwendigen Versuchsaufbau bedeuten, etwa das Fräsen eines Modells, das anschließend in einem Windkanal auf unterschiedliche Eigenschaften hin untersucht werden muss. Im Beispiel der Leerlaufdrehzahlregelung wird entweder eine sehr gute Simulation benötigt, die unter Umständen viel Rechenzeit erfordert, oder es müssen sogar Testfahrten durchgeführt werden, was noch wesentlich mehr Zeit- und Kostenaufwand verursacht.

Welche grundsätzlichen Strategien – außer einer blinden Zufallssuche – lassen sich bei der Lösung von Optimierungsproblemen anwenden? Eine wesentliche Voraussetzung, um ein besseres Verfahren als eine reine Zufallssuche zu entwickeln, besteht darin, dass eine Ähnlichkeitsbeziehung zwischen den Elementen des Suchraums $\mathcal{S}$ gegeben ist. Bezüglich dieser Ähnlichkeitsstruktur sollte gelten, dass sich die Funktionswerte $f(s)$ und $f(s')$ für ähnliche Elemente $s, s' \in \mathcal{S}$ in den meisten Fällen nicht sehr stark unterscheiden. Anders ausgedrückt: Ähnliche Lösungen sollten im Allgemeinen auch ähnlich gut bzw. schlecht durch die Zielfunktion bewertet werden. Unter dieser Voraussetzung können ausgehend von beliebigen oder bereits gefundenen Lösungen weitere Lösungen konstruiert werden, indem man in der Nachbarschaft der bereits gefundenen Lösungen heuristisch oder systematisch nach besseren Lösungen sucht. Welche Strategien zur

Verbesserung gefundener Lösungen angewendet werden und wie erfolgreich diese Strategien sind, hängt sehr stark vom Optimierungsproblem und insbesondere der Struktur der Zielfunktion ab. Ein Problem aller Verfahren, die gezielt bessere Lösungen in der Nachbarschaft einer Lösung suchen, stellen lokale Optima dar. Diese Verfahren bleiben üblicherweise im nächsten lokalen Optimum stecken, das sehr weit vom globalen Optimum entfernt sein kann. Abbildung 2.1 weist eine Reihe lokaler Optima von sehr unterschiedlicher Güte auf.

Auch die Struktur lokaler Optima spielt eine wesentliche Rolle. Sehr schmale lokale Optima sind schwerer zu finden als breite. Bei einer zu maximierenden Funktion kann eine hohe, schmale Spitze zwar ein Optimum mit einem besonders guten Wert darstellen; sie ist aber schwer zu finden und stellt auch kein robustes Optimum dar. Robust meint hier, dass eine kleine Änderung der Parameter immer noch einen zumindest annähernd genauso guten Wert der Zielfunktion ergeben sollte. Wenn beispielsweise Motoren mit gewissen optimalen Eigenschaften produziert werden sollen, dann kann die mangelnde Robustheit eines Optimums dazu führen, dass die Motoren zwar theoretisch die optimalen Eigenschaften besitzen, aber durch Fertigungstoleranzen diese Eigenschaften bei weitem nicht erreichen. Man würde in diesem Fall ein etwas schlechteres, aber dafür robusteres Optimum vorziehen.

Ein wesentliches Charakteristikum von Optimierungsproblemen besteht noch darin, dass man im Allgemeinen auch mit suboptimalen Lösungen zufrieden ist. Es ist zwar wünschenswert, das globale Optimum zu finden. Ein nur etwas schlechteres lokales Optimum sollte aber trotzdem als gute Lösung akzeptiert werden. Man kann prinzipiell jedes (mathematische) Problem als Optimierungsproblem formulieren. Es ist allerdings wenig sinnvoll, deswegen alle Probleme im Sinne der Optimierung zu betrachten. Beispielsweise ist die Lösung der Gleichung $5x - 7 = 0$ gleichwertig dazu, die Zielfunktion $f(x) = (5x - 7)^2$ zu minimieren. Diese Betrachtungsweise ist jedoch wenig hilfreich. Man sieht dies noch deutlicher an der Gleichung $x^2 + 1 = 0$. Diese Gleichung besitzt keine reelle Lösung. Definiert man die zu minimierende Zielfunktion $f(x) = (x^2 + 1)^2$, besitzt diese zwar ein Minimum bei $x = 0$, das aber keine Lösung der ursprünglichen Gleichung ist. $x = 0$ ist keine suboptimale Lösung, sondern überhaupt keine Lösung.

Mit dieser sehr engen Betrachtungsweise werden auch viele Optimierungsprobleme als extrem schwer klassifiziert, obwohl dies in manchen Fällen nur theoretisch stimmt. Entsprechend werden auch Aussagen über die Eigenschaften NP-vollständig oder NP-hart im Zusammenhang mit Optimierungsproblemen gemacht. (Für eine kurze Erklärung dieser Begriffe verweisen wir auf den Anhang B.) Häufig soll mit dem Beweis, dass ein Problem NP-vollständig oder NP-hart ist, gezeigt werden, dass es kein effizientes Verfahren zur Lösung dieses Problems gibt bzw. dass dies zumindest zu vermuten ist.

Das TSP gehört zu den NP-vollständigen Problemen. Genauer gesagt, ist folgendes, aus dem TSP abgeleitetes Entscheidungsproblem NP-vollständig: Gibt es zu einem gegebenen TSP und vorgegebener Reiselänge r eine Rundreise, die kürzer als r ist? Gäbe es einen Algorithmus, der in polynomialer Zeit die kürzeste Rundreise bestimmt, könnte man auch die Frage, ob es eine Rundreise gibt, die kürzer als r ist, in polynomialer Zeit beantworten. Man müsste ja nur vergleichen, ob die in polynomialer Zeit berechnete kürzeste Rundreise kürzer als r ist. Auch wenn die Vermutung stimmt, dass NP-vollständige Probleme einen exponentiellen Rechenaufwand erfordern, bedeu-

tet dies nicht, dass das TSP in keiner Weise in akzeptablen Rechenzeit handhabbar ist. Dies stimmt nur, wenn man darauf besteht, dass man unbedingt die kürzeste Rundreise finden muss. Wenn man sich mit einer guten suboptimalen Lösung – einer kurzen Rundreise, die vielleicht etwas länger als die kürzeste Rundreise ist – zufrieden gibt, stehen einem durchaus brauchbare heuristische Verfahren zur Verfügung, die zu zufrieden stellenden Lösungen führen. In diesem Sinne kann man auch NP-vollständige Probleme (suboptimal) lösen, ohne dass ein exponentieller Rechenaufwand erforderlich ist. Das bedeutet jedoch nicht, dass das eigentliche NP-vollständige Problem in polynomialer Zeit lösbar ist.

3 Optimierungsverfahren

Bevor wir uns dem eigentlichen Thema dieses Buches – den evolutionären Algorithmen – widmen, soll in diesem Kapitel eine Auswahl bekannter, sowohl klassischer als auch moderner Optimierungsverfahren kurz besprochen werden. Eine detaillierte Diskussion der Verfahren würde den Rahmen dieses Buches bei weitem sprengen, so dass wir auf die entsprechende Literatur verweisen. Ein guten Überblick über klassische Optimierungsverfahren findet sich in [Pap96].

Die meisten Optimierungsverfahren beginnen mit einer zufällig konstruierten Anfangslösung. Einige Verfahren berechnen in den Folgeschritten die Lösungen nach einem deterministischen Schema, andere involvieren auch in den Folgeschritten Zufallssmechanismen. Die – außer im ersten Schritt – zufallsunabhängigen Vertreter der Optimierungsverfahren werden daher als exakte und die zufallsabhängigen Vertreter als probabilistische Algorithmen bezeichnet.

Es gibt neben einer Zufallssuche zwei grundlegende Strategien bei der Optimierung, die auch in Kombination verfolgt werden können. Viele Verfahren verwenden eine lokale Suche, bei der versucht wird, eine bessere Lösung in der Nähe einer bereits gefundenen Lösung zu bestimmen. Beim TSP könnte man beispielsweise versuchen, aus einer gegebenen Rundreise eine kürzere zu konstruieren, indem man die Positionen zweier Städte vertauscht. Neben der lokalen Suche werden auch problemspezifische Techniken angewendet, die Schritt für Schritt eine Lösung konstruieren. Beim TSP würde man die Rundreise z.B. in einer beliebigen Stadt beginnen und dann jeweils die am nächsten liegende Stadt wählt, die bisher noch nicht besucht wird.

3.1 Analytische Lösung des Optimierungsproblems

Die wohl einfachste und effizienteste Strategie besteht in der direkten Berechnung einer analytischen Lösung, d.h., das Optimum kann direkt berechnet werden. Leider sind die Voraussetzungen hierfür nur in seltenen Fällen gegeben. Sind beispielsweise folgende Voraussetzungen gegeben:

- Der Suchraum ist der $\mathbb{R}^n$.
- Die zu minimierende Zielfunktion ist stetig differenzierbar.
- Der Gradient der Zielfunktion wird an genau einer Stelle null, die auch analytisch berechnet werden kann.
- Lässt man einen der Parameter der Zielfunktion gegen $\pm\infty$ gehen, geht auch die Zielfunktion gegen $+\infty$.

Unter diesen Voraussetzungen besitzt die Zielfunktion besitzt die Zielfunktion genau ein lokales Minimum, das auch gleichzeitig das globale Minimum ist. Das globale Minimum befindet sich an der Stelle, wo der Gradient null wird.

Die Voraussetzungen sind etwa im Falle der Zielfunktion (2.1) auf Seite 8 erfüllt, die zur Berechnung einer Regressionsgeraden verwendet wird. Durch Berechnung der partiellen Ableitungen der Zielfunktion nach den beiden Parameter a und b, die die Regressionsgerade definieren, und Nullsetzen der Ableitungen erhält man zwei Gleichungen, die explizit nach a und b aufgelöst werden können. Selbst wenn die Zielfunktion differenzierbar ist und ein eindeutiges lokales Optimum besitzt, ist das Gleichungssystem, das man durch die partiellen Ableitungen erhält, im Allgemeinen nicht analytisch lösbar.

Lässt sich das Gleichungssystem nicht analytisch lösen, hilft es in manchen Fällen, dass man Parameter oder Parametergruppen separat optimiert. Wenn es für einen Parameter oder eine Parametergruppe unter der Annahme, dass die anderen Parameter fest gewählt sind, eine analytische Lösung existiert, kann man abwechselnd jeweils einzelne Parameter bzw. Parametergruppen optimieren, bis dieser Prozess konvergiert. Dies birgt sowohl die Gefahr, in einem lokalen Optimum als auch in einem Sattelpunkt der Zielfunktion zu enden. Auch wenn die alternierende Optimierung diese Nachteile mit sich bringt, führt sie in vielen Problemen, in denen sie anwendbar ist, zu recht guten Ergebnissen wie beispielsweise bei der Fuzzy-Clusteranalyse [HKKR99](vgl. auch 9.5).

3.2 Gradientenverfahren

Der Grundgedanke des *Gradientenverfahrens* ist, sich ausgehend von einem zufällig gewählten Startpunkt in die Richtung fortzubewegen, in der die zu maximierende Zielfunktion am stärksten ansteigt bzw. in die entgegengesetzte Richtung bei einer zu minimierenden Zielfunktion. Die Richtung dieses stärksten Anstiegs lässt sich durch das Bilden des Gradienten, d.h. der partiellen Ableitungen der Zielfunktion berechnen. Das *Gradientenverfahren* ist nur bei differenzierbaren Funktionen anwendbar, d.h. die Zielfunktion

$$f : \mathcal{S} \to \mathbb{R}, \qquad \mathcal{S} \subseteq \mathbb{R}^n$$

muss auf dem Suchraum $\mathcal{S}$ differenzierbar sein.

Der Ablauf des Verfahren ist wie folgt (siehe auch [Hei94]):

1. Wähle einen Startpunkt $\left(x_1^{(0)}, \ldots, x_n^{(0)}\right)^T$.
2. Bestimme
   $\nabla f \left(x_1^{(0)}, \ldots, x_n^{(0)}\right) = \left(\frac{\partial f}{\partial x_1}(x_1^{(0)}, \ldots, x_n^{(0)}), \ldots, \frac{\partial f}{\partial x_n}(x_1^{(0)}, \ldots, x_n^{(0)})\right)^T$.
3. Gehe vom Startpunkt aus solange in Richtung des Gradienten aus (2.) (bzw. bei Minimierungsproblemen in entgegengesetzte Richtung), wie sich der Wert von f vergrößert (verkleinert). Wähle diesen Punkt als neuen Startpunkt und fahre mit

(2.) fort.
Also: $\left(x_1^{(1)}, \ldots, x_n^{(1)}\right)^T = \left(x_1^{(0)}, \ldots, x_n^{(0)}\right)^T + \rho * \nabla f \left(x_1^{(0)}, \ldots, x_n^{(0)}\right)^T$

4. So lange mit Schritt 2.) weitermachen, wie es im vorhergehenden Schritt 3.) noch zu einer Verbesserung gekommen ist.

ρ muss so gewählt werden, dass die Zielfunktion f in Suchrichtung ein Maximum bzw. Minimum erreicht. Das kann z.B. mit einem – eventuell sehr aufwendigen – Liniensuchverfahren durchgeführt werden. Häufig wird aber auch eine feste Schrittweite ρ gewählt. Eine feste Schrittweite birgt die Gefahr des Oszillierens oder über das Ziel (Optimum) hinauszuschießen, wenn ρ zu groß gewählt wird. Bei zu kleinem ρ werden sehr viele Schritte beim Gradientenverfahren benötigt und im schlimmsten Fall bei sehr flachen lokalen Optima konvergiert der Algorithmus sogar, bevor das lokale Optimum erreicht wird.

Da das Gradientenverfahren direkt auf das nächste lokale Optimum zusteuert, ist es empfehlenswert, den Algorithmus mehrmals mit verschiedenen Startwerten auszuführen.

Die Berechnung des Gradienten kann in einigen Fällen sehr schwierig und aufwändig sein. Ein Beispiel dafür ist das Backpropagation-Lernalgorithmus, der bei bestimmten neuronalen Netzen [BKKN03] verwendet wird. Dieser Lernalgorithmus basiert auf einem Gradientenverfahren mit fester Schrittweite. Aufgrund der komplexen Struktur der neuronalen Netze muss die Zielfunktion zunächst geschickt in Einzelfunktionen zerlegt werden, bevor der Gradient berechnet werden konnte.

Beim Gradientenverfahren kann es passieren, dass die Gradienten jeweils so ausgerichtet sind, dass man sich spiralförmig auf ein lokales Optimum zubewegt, was zu einer großen Zahl von Iterationsschritten führt. Deshalb wird häufig die Technik der konjugierten Gradienten [Pap96] angewendet. Die Idee ist dabei, dass sich nach einer geeigneten Transformation möglichst orthogonale Gradienten ergeben. Im Idealfall gelangt man bei n zu optimierenden Parametern nach n Iterationsschritten zum (lokalen) Optimum, da die Schrittweiten durch die Orthogonalität der Gradienten unabhängig voneinander optimal gewählt werden können.

3.3 Newton-Verfahren

Das *Newton-Verfahren* (s. z.B. [Pap96]) ist ein Instrument zur Lösung nichtlinearer Gleichungssysteme

$$g_1(x_1, \ldots, x_n) = 0 \quad (i = 1, \ldots, n). \tag{3.1}$$

Man benötigt zunächst einen Startwert $x^{(0)} \in \mathbb{R}^n$, um dann iterativ verbesserte Lösungen $x^{(1)}, x^{(2)} \ldots \in \mathbb{R}^n$ zu berechnen. Um aus der Näherungslösung $x^{(i)}$ die Lösung $x^{(i+1)}$ zu bestimmen, linearisiert man die Funktion $g : \mathbb{R}^n \to \mathbb{R}^n$ an der Stelle $x^{(i)}$, d.h., man entwickelt sie in ihre Taylor-Reihe:

$$g(x^{(i)} + \Delta x) \approx g(x^{(i)}) + g_x(x^{(i)})\Delta x$$

Dabei ist

$$g_x(x) = \begin{pmatrix} \frac{\partial g_1}{\partial x_1} & \cdots & \frac{\partial g_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial g_n}{\partial x_1} & \cdots & \frac{\partial g_n}{\partial x_n} \end{pmatrix}$$

die Jacobi-Matrix der Vektorfunktion g.

Zur Lösung des nichtlinearen Gleichungssystem (3.1) löst man

$$\mathbf{0} = g(x^{(i)} + \Delta x) \approx g(x^{(i)}) + g_x(x^{(i)})\Delta x$$

und erhält

$$\Delta x^{(i)} = -g(x^{(i)})_x^{-1} g(x^{(i)}),$$

so dass man

$$x^{(i+1)} = x^{(i)} + \Delta x^{(i)}$$

als nächste Näherungslösung verwendet.

Bei Optimierungsproblemen mit einer Zielfunktion $f : \mathcal{S} \to \mathbb{R}$ mit $\mathcal{S} \subseteq \mathbb{R}^n$, die entsprechende Differenzierbarkeitsvoraussetzungen erfüllt, muss für lokale Optima gelten, dass alle partiellen Ableitungen dort null sind. Man erhält somit als notwendige Bedingung für ein lokales Optimum das – im Allgemeinen nichtlineare – Gleichungssystem

$$f_x(x_1, \ldots, x_n) = \begin{pmatrix} \frac{\partial f(x_1,\ldots,x_n)}{\partial x_1} \\ \vdots \\ \frac{\partial f(x_1,\ldots,x_n)}{\partial x_n} \end{pmatrix} = \begin{pmatrix} 0 \\ \vdots \\ 0 \end{pmatrix}.$$

Die Vektorfunktion $f_x : \mathbb{R}^n \to \mathbb{R}^n$, die sich aus den partiellen Ableitungen der Zielfunktion f ergibt, spielt somit die Rolle der Funktion g beim Newton-Verfahren. Wendet man das Newton-Verfahren auf die Funktion f_x an, ergibt sich als Iterationsvorschrift

$$x^{(i+1)} = x^{(i)} - f_{xx}(x^{(i)})^{-1} f_x(x^{(i)}). \tag{3.2}$$

Dabei ist

$$f_{xx} = H[f(x)] = \begin{pmatrix} \frac{\partial^2 f}{\partial^2 x_1} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \cdots & \frac{\partial^2 f}{\partial^2 x_n} \end{pmatrix}$$

die Hesse-Matrix der Funktion f. Teilweise wird noch ein in jedem Iterationsschritt anzupassender Faktor $\alpha^{(i)}$ eingeführt, der die Schrittweite steuert, d.h., man verwendet anstelle der Gleichung (3.2) die Gleichung

$$x^{(i+1)} = x^{(i)} - \alpha^{(i)} f_{xx}(x^{(i)})^{-1} f_x(x^{(i)}).$$

$\alpha^{(i)}$ kann beispielsweise durch Abtasten der Zielfunktion $f(x)$ an den Stellen $x = x^{(i)} - \alpha f_{xx}(x^{(i)})^{-1} f_x(x^{(i)})$ mit verschiedenen α-Werten bestimmt werden, je nachdem, an welcher Stelle sich der beste Wert der Zielfunktion ergibt.

Da die Anwendung des Newton-Verfahrens auf Optimierungsprobleme die erste Ableitung der Zielfunktion linearisiert, bedeutet dies, dass die Zielfunktion an der Stelle $x^{(i)}$ jeweils durch eine quadratische Funktion bzw. quadratische Form

$$f(x_1,\ldots,x_n) \approx \frac{1}{2}(x_1,\ldots,x_n)Q\begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} + (b_1,\ldots,b_n)\begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} + c$$

angenähert wird. Dabei ist $Q = f_{xx}$ die Hesse-Matrix an der entsprechenden Stelle und

$$f_x = Q\begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} + \begin{pmatrix} b_1 \\ \vdots \\ b_n \end{pmatrix}$$

der Gradient von f. Die in jedem Iterationsschritt neu zu berechnende quadratische Form besitzt jeweils ein eindeutiges globales Optimum, das beim Newton-Verfahren direkt bestimmt wird. Bei quadratischen Funktionen terminiert das Newton-Verfahren daher bereits nach dem ersten Schritt.

Das Newton-Verfahren ist wie das Gradientenverfahren nur auf Optimierungsprobleme mit rein reellen Parametern anwendbar und steuert ebenfalls mehr oder weniger direkt auf das nächste lokale Optimum zu. Neben der Berechnung des Gradienten muss zusätzlich die Hesse-Matrix der Zielfunktion bestimmt und in jedem Schritt invertiert werden.

Um den hohen Aufwand zur Invertierung der Hesse-Matrix zu verringern, gibt es verschiedene Approximationen der inversen Matrix, die sich ohne direkte Matrixinversion berechnen lassen. Diese Verfahren bezeichnet man als *Quasi-Newton-Verfahren.*

3.4 Optimierung mit Nebenbedingungen

Die bisher in diesem Kapitel besprochenen Optimierungsverfahren eignen sich für reelle Zielfunktionen $f : \mathcal{S} \to \mathbb{R}$ mit $\mathcal{S} \subseteq \mathbb{R}$. Häufig werden an die Lösung gewisse Nebenbedingungen in Form von Gleichungen oder Ungleichungen gestellt. Möchte man beispielsweise eine diskrete Wahrscheinlichkeitsverteilung, gegeben durch n Wahrscheinlichkeiten p_i für Elementarereignisse, schätzen oder optimieren, so gilt: $p_i \in [0,1]$, d.h. $\mathcal{S} = [0,1]^n$. Allerdings muss eine derartige Wahrscheinlichkeitsverteilung zusätzlich die Nebenbedingung

$$p_1 + \ldots + p_n = 1 \tag{3.3}$$

erfüllen. Ein konkretes Beispiel hierzu findet sich im Abschnitt 9.5 über Fuzzy-Clusteranalyse.

Wir betrachten Optimierungsprobleme mit einer Zielfunktion $f(x_1,\ldots,x_n)$ mit geeigneten Differenzierbarkeitsvoraussetzungen und Nebenbedingungen in der Form von Gleichungen

$$g_i(x_1,\ldots,x_n) = 0 \quad (i = 1,\ldots,m). \tag{3.4}$$

Jede Gleichungsnebenbedingung lässt sich in der Form (3.4) darstellen. Beispielsweise würde man die Bedingung (3.3) zu

$$g(p_1, \ldots, p_n) \;=\; p_1 + \ldots + p_n - 1 \;=\; 0$$

gleichwertig umformulieren.

Derartige Optimierungsprobleme werden häufig mit der Methode der *Lagrange-schen Multiplikatoren* gelöst [Pap96]. Dazu bildet man die Lagrange-Funktion

$$L(x_1, \ldots, x_n, \lambda_1, \ldots, \lambda_m) \;=\; f(x_1, \ldots, x_n) + \sum_{i=1}^{m} \lambda_i g_i(x_1, \ldots, x_n).$$

Notwendige Bedingungen für ein lokales Optimum der Zielfunktion unter den Gleichungsnebenbedingungen sind dann, dass die partiellen Ableitungen der Lagrange-Funktion nach den zu optimierenden Parametern x_j $(j = 1, \ldots, n)$ und den Lagrange'schen Multiplikatoren λ_k $(k = 1, \ldots, m)$ verschwinden:

$$\frac{\partial L(x_1, \ldots, x_n, \lambda_1, \ldots, \lambda_m)}{\partial x_j} \;=\; \frac{\partial f(x_1, \ldots, x_n)}{\partial x_j} + \sum_{i=1}^{m} \lambda_i \frac{\partial g_i(x_1, \ldots, x_n)}{\partial x_j} \;=\; 0$$

$$\frac{\partial L(x_1, \ldots, x_n, \lambda_1, \ldots, \lambda_m)}{\partial \lambda_k} \;=\; g_k(x_1, \ldots, x_n) \;=\; 0$$

Auf diese Weise erhält man ein Gleichungssystem mit $(n+m)$ Variablen, das im günstigsten Fall analytisch oder im anderen Fall z.B. mit einer Technik wie dem Newton-Verfahren gelöst werden kann.

Bei einigen Optimierungsproblemen treten zusätzlich zu Gleichungsnebenbedingungen auch Nebenbedingungen in Form von Ungleichungen

$$h_j(x_1, \ldots, x_n) \;\leq\; 0 \quad (j = 1, \ldots, k) \tag{3.5}$$

auf. Jede Ungleichung lässt sich in dieser Form schreiben, indem man alle Terme der Ungleichung auf die linke Seite bringt und ggf. mit (-1) multipliziert, um aus einem $\geq$-Zeichen in ein $\leq$-Zeichen zu erhalten.

Eine gängige Technik zur Berücksichtigung derartiger Ungleichungen bei einem Optimierungsproblem besteht darin, dass man die Ungleichungen durch Einführung von Schlupfvariablen z_j in Gleichungsnebenbedingungen überführt:

$$\widehat{h}_j(x_1, \ldots, x_n, z_j) \;=\; h_k(x_1, \ldots, x_n) + z_j^2 \;=\; 0 \tag{3.6}$$

Diese Nebenbedingungen lassen sich dann wieder mit Hilfe von Lagrange'schen Multiplikatoren behandeln. Die allgemeine Lagrange-Funktion für ein Optimierungsproblem mit m Gleichungsnebenbedingungen der Form (3.4) und k Ungleichungsnebenbedingungen der Form (3.5) lautet daher:

$$\begin{aligned} L(x_1,\ldots,x_n,\lambda_1,\ldots,\lambda_m,\mu_1,\ldots,\mu_p) &= f(x_1,\ldots,x_n) \\ &\quad + \sum_{i=1}^{m} \lambda_i g_i(x_1,\ldots,x_n) \\ &\quad + \sum_{j=1}^{k} \mu_j \widehat{h}_j(x_1,\ldots,x_n,z_j) \end{aligned}$$

Dabei sind die Funktionen $\widehat{h}_j$ die aus den Ungleichungen (3.5) abgeleiteten Funktionen (3.6). Die dazu gehörigen Faktoren μ_j bezeichnet man auch als *Kuhn-Tucker-Multiplikatoren.*

3.5 Tabu-Suche

Die *Tabu-Suche* versucht, das beim Gradientenverfahren aufgetretene Problem der lokalen Optima dadurch zu umgehen, dass nicht nur nach der stärksten Verbesserung gesucht wird, sondern ggf. auch eine geringe Verschlechterung in Kauf genommen wird. Dieses Verfahren wird hauptsächlich in der kombinatorischen Optimierung angewendet (TSP, Steiner Trees).

Wie zuvor bei den Gradientenverfahren im Abschnitt 3.2 wird von einem zufälligen Punkt aus gestartet. Die Grundlage des Tabu-Verfahrens ist eine Nachbarschaftsbeziehung zwischen den Lösungen (beispielsweise in Matrixform), die z.B. über einen Abstand von Lösungen untereinander definierbar ist. Bei der kombinatorischen Optimierung lassen sich beispielsweise zwei Lösungen a und b als direkt benachbart bezeichnen, wenn a durch eine einzige Änderung (z.B. beim TSP das Vertauschen zweier aufeinander folgender Städte in der Rundreise) in b überführt werden kann. Im Falle einer Lösungsdarstellung in Binärform wären zwei Lösungen benachbart, wenn sie sich an genau einer Position unterscheiden, also durch einen Wechsel von 0 zu 1 oder umgekehrt in einer Position ineinander überführen lassen.

Jeder Schritt zu einer benachbarten Lösung wird als *Move* bezeichnet. Als komplementärer Schritt $\overline{m}$ zu Schritt m wird der Schritt bezeichnet, der die für m ausgeführte Änderung rückgängig macht. Bezeichnet man im folgenden Beispiel bei einer binären Kodierung den Move, der das i-te Bit auf den Wert $j \in \{0,1\}$ setzt, mit m_i^j, so ergibt sich mit:

$$\begin{aligned} c &= (1111) \\ c' &= m_3^0(c) = (1101) \text{ und } m_3^1(c') = (1111) = c \\ \text{und} \quad m_3^0 &= \overline{m_3^1}, \text{ d.h. } m_3^0 \text{ und } m_3^1 \text{ sind komplementäre Schritte} \end{aligned}$$

Die Idee der Tabu-Suche besteht darin, keine – oder zumindest möglichst wenige – Schritte durchzuführen, die die Wirkung von vorher durchgeführten Schritten wieder aufheben, um auf diese Weise möglichst effizient den Lösungsraum zu durchsuchen. Um das zu erreichen, werden während des Ablaufs zwei Listen mitgeführt: Die *Tabu*-Liste T und die *Running*-Liste R.

In die Running-Liste werden alle im Verlauf der Optimierung durchgeführten Bewegungen eingetragen. Die Tabu-Liste hingegen hat eine vorher festgelegte Länge k. Sie nimmt dem entsprechend nur auf die k aktuellsten Schritte Bezug. In ihr werden die Komplemente der durchgeführten Schritte (bzw. ihre Attribute) gespeichert, die dann als tabu gelten, unabhängig davon, wie lange die Ausführung des komplementären Schrittes zurückliegt. So kann leicht verglichen werden, ob ein beabsichtigter Schritt einen früheren – noch gesperrten – Schritt rückgängig machen würde. Dadurch, dass die in der Tabu-Liste aufgeführten Schritte nicht ausgeführt werden dürfen, soll das ständige Wiederholen einer Schrittfolge (Cycling) verhindert, das üblicherweise eintritt, wenn die gefundene Lösung einem lokalen Optimum entspricht. Die Vorgehensweise bei diesem Verfahren ist:

1. Startpunkt $s_0 \in S$ wählen.
2. Prüfen, welche Lösung s_1 in der Nachbarschaft von s_0 die größte Verbesserung, bzw. die geringste Verschlechterung gegenüber s_0 darstellt und mit einem nicht verbotenem Schritt m_1 erreicht werden kann.
3. Wechsel von der Lösung s_0 zur Lösung s_1 durch Schritt m_1.
4. Eintragen des Schrittes in die Running-Liste: $R_1 = R_0 \cup \{m_1\}$.
5. Eintragen des komplementären Schrittes in die Tabu-Liste $T_1 = T_0 \cup \{\overline{m}_1\}$ und Kürzen der Liste um das erste Element.
6. Überprüfen eines vorher festgelegten Stopp-Kriteriums. Bei Nichterfüllung weiter mit 2.).

In der oben dargestellten einfachen Form wird diese Methode als *Static Tabu Search* (STS) bezeichnet, da die Tabu-Liste immer eine feste Länge aufweist. Weitere bekannte Methoden, die mit variabler Tabu-Liste arbeiten, sind die *Reverse Elimination Method* (REM) und die *Cancellation Sequence Method* (CSM) (siehe [ADV96]).

Das Hauptproblem des STS ist es, dass auch dann Züge verboten werden, wenn sie nicht direkt wieder zu einer vorher besuchten Lösung führen, da inzwischen weitere Änderungen zum Tragen gekommen sind. Es würde reichen zu verhindern, dass eine ganze Sequenz von Änderungen zurückgenommen wird. Dieses Prinzip wurde innerhalb von REM und CSM verwirklicht, wenn auch auf unterschiedliche Weise. Insgesamt müssen bei der Anwendung die folgenden Probleme beachtet werden:

- Bei STS muss ein Kompromiss zwischen der Länge der Tabu-Liste und der Gefahr des Cyclings gefunden werden.

- Tabu-Suche ist nur bei Problemen anwendbar, bei denen es eine direkte Nachbarschaftsbeziehung gibt.
- Der Aufwand durch das Mitführen der Listen und die Überprüfung auf Tabus ist höher als bei anderen Verfahren.

Einen Überblick zum Thema Tabu Search findet man von Glover in [Glo89, Glo90].

3.6 Greedy-Heuristiken

Unter dem Begriff *Greedy Heuristiken* werden all jene Algorithmen zusammengefasst, deren Zweck es ist, durch Hinzunahme von Teillösungen eine Gesamtlösung zu schaffen, wobei in jedem Schritt die beste Teillösung hinzugenommen wird. Es wird also immer eine Zwischenlösung gebildet. Ein typischer Vertreter eines Problems, bei dem ein derartiges Verfahren anwendbar wäre (ohne das Auffinden der optimalen Lösung zu garantieren), ist das Rucksackproblem (siehe unten).

Zur Lösung des TSP gibt es eine sehr einfache Greedy-Heuristik zur Konstruktion einer Rundreise. Man startet in einer beliebigen Stadt und besucht jeweils die am nächsten liegende noch nicht besuchte Stadt. Diese Heuristik liefert zwar relativ schnell eine nicht völlig abwegige Rundreise. Das Ergebnis wird allerdings in den meisten Fällen wesentlich schlechter als die optimale Lösung sein. Abbildung 3.1 zeigt ein derartiges Beispiel. Man startet in der Stadt, die durch das Quadrat symbolisiert wird. Die Greedy-Heuristik besucht die Stadt oben links erst ganz am Ende der Reise. Es ist aber offensichtlich günstiger, diese Stadt als dritte Station zu besuchen, wenn das Quadrat die erste Station repräsentiert.

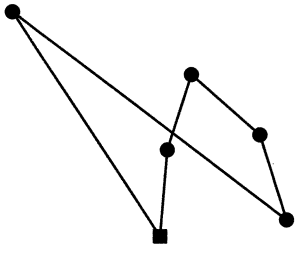


Abbildung 3.1 Eine Greedy-Lösung des TSP

Für Greedy-Heuristiken ist die Zielfunktion $f : \mathcal{S} \rightarrow \mathbb{R}$ auf einem diskreten Suchraum $\mathcal{S}$ definiert. Es muss möglich sein, auch partielle Lösungen (Zwischenlösungen) zu bewerten, was beispielsweise durch die Addition der Bewertungen der in der Zwischenlösung enthaltenen Teillösungen geschehen kann. Greedy-Heuristiken lassen sich sehr einfach definieren, wenn die Lösung des Optimierungsproblems als ein Folge von Elementen – wie z.B. eine Rundreise beim TSP – interpretiert werden kann.

Das Verfahren läuft wie folgt ab:

1. Starte mit einer leeren Zwischenlösung ℓ.
2. Erweitere die Lösung so, dass sich die beste Bewertung ergibt und die neue Zwischenlösung nicht ungültig macht
3. Prüfe, ob die eine maximale Zwischenlösung (insgesamt gültige Lösung) gefunden wurde, ansonsten weiter mit Punkt 2.

Beispiel 3.1 (Rucksackproblem) Beim Rucksackproblem besteht die Aufgabe darin, einen Rucksack mit festgelegtem Volumen mit einer Anzahl unterschiedlich großer Gegenstände so zu füllen, dass der Inhalt z.B. möglichst wertvoll oder der Rucksack optimal gefüllt ist. Es wird also mit einem leeren Rucksack gestartet, in den dann beispielsweise der Reihe nach die Gegenstände mit dem besten Verhältnis von Größe und Wert hineingepackt werden, die noch hineinpassen.

Problemstellung : Gegeben sind k Gegenstände mit einem Volumen von jeweils v_k und einem Wert von w_k, sowie ein Rucksack mit einem Volumen von V.

Gesucht : Eine Kombination von n Gegenständen, die zusammen in den Rucksack passen und einen möglichst hohen Wert haben.

Zielfunktion : $f = \sum_{i=1}^{n} w_k$.

Zum Auffinden einer Lösung mittels der Greedy-Heuristik wird für alle Gegenstände das Verhältnis $b_k = w_k/v_k$ von Wert zu Gewicht berechnet. Danach wird in jedem Schritt der Gegenstand mit dem höchsten Wert b_k in den Rucksack gepackt, der noch hineinpasst, bis kein derartiger Gegenstand mehr zur Verfügung steht. Bei gleichem Verhältniswert mehrerer Gegenstände erfolgt die Auswahl zufällig.

Wir betrachten ein konkretes Zahlenbeispiel:

Anzahl der Gegenstände: $k = 10$

Volumen der Gegenstände v_k: $\{1, 1, 4, 3, 6, 2, 8, 8, 7, 9\}$

Zugehörige Werte w_k: $\{2, 5, 3, 6, 9, 1, 4, 6, 8, 3\}$

Volumen des Rucksacks: $V = 25$

Verhältniswerte $\{2, 5, 0.75, 2, 1.5, 0.5, 0.5, 0.75, 1.14, 0.33\}$.

Nummern der Gegenstände in einer möglichen Lösung:
$L := \{2, 1, 4, 5, 9, 3, 6\}$ mit $f(L) = 24$.

3.7 Hillclimbing

Hillclimbing ist eine der einfachsten Methoden zur Optimierung von Lösungen, sowohl was den zur Implementation nötigen Aufwand, als auch den Anwendungsbereich angeht. Die Grundidee ist es, mit einer zufällig gewählten Lösung zu starten, um dann einen ebenfalls zufälligen Schritt in Richtung eines Nachbarn auszuführen. Ist die Bewertung der neuen Lösung besser als die vorhergehende, so wird die neue Lösung behalten. Ansonsten wird in der Umgebung der alten Lösung weitergesucht.

Soll die Zielfunktion $f : \mathcal{S} \to \mathbb{R}$ maximiert werden, sieht der Algorithmus folgendermaßen aus: (Für zu minimierende Zielfunktionen verläuft der Algorithmus analog; nur im dritten Schritt muss $\geq$ durch $\leq$ ersetzt werden.)

1. Wähle einen Startpunkt $s_0 \in \mathcal{S}$
2. Wähle $s_1 \in \mathcal{S}$ in der „Nähe“ von s_0
3. Falls $f(s_1) \geq f(s_0)$, wähle s_1 als neuen Startpunkt und fahre mit 2.) fort; ansonsten fahre mit s_0 als Startpunkt fort und wähle ein neues s_1 in 2.)

Aufgrund seiner Einfachheit hat dieser Algorithmus mit einigen Problemen zu kämpfen:

- Er ist unfähig, ein lokales Optimum zu verlassen, nachdem er es erreicht hat.
- Da der Algorithmus nur in seiner direkten Umgebung nach einer Verbesserung sucht und jede Verbesserung sofort akzeptiert, läuft er mit hoher Wahrscheinlichkeit in das nächstgelegene lokale Optimum.
- Der Algorithmus ist relativ langsam, da er nicht gezielt nach Verbesserungen sucht.
- Die Festlegung der optimalen Schrittweite (Nachbarschaft) ist insbesondere im stetigen Fall schwierig.

Dieser Algorithmus ist die Grundlage für die Mehrheit der in den folgenden Abschnitten erläuterten Verfahren, die so weiterentwickelt wurden, dass sie in der Lage sind, mit den oben erwähnten Problemen umzugehen.

Um die Gefahr zu umgehen, in einem lokalem Optimum zu enden und es nicht mehr verlassen zu können, werden häufig eine größere Anzahl von Durchläufen mit zufällig gewählten Startpunkten durchgeführt. Eine weitere Variante des Hillclimbing probiert systematisch alle mit einem Schritt erreichbaren Lösungen durch und behält die beste. Diese Variante wird *Steepest-Ascent*-Hillclimbing genannt (siehe [Mit96]). Diese Algorithmenform bringt jedoch mit sich, dass die Suche immer in Richtung der größten Verbesserung durchgeführt wird, was lediglich zum Auffinden der steilsten in der Nähe liegenden Spitze und nicht direkt zum Optimum führt.

3.8 Simulated Annealing

Beim *Simulated Annealing* Algorithmus (*Simuliertes Abkühlen/Ausglühen*) handelt es sich um die Weiterentwicklung eines bereits 1953 veröffentlichten Verfahrens von Metropolis et. al. [MRR+53] zum Verhalten einzelner Teilchen in einer Schmelze. Daraus ging der so genannte Metropolis-Algorithmus hervor. Der physikalische Hintergrund sieht wie folgt aus:

Teilchen in einer Flüssigkeit nehmen bei Vorhandensein eines bestimmten Energieniveaus nicht immer die energetisch günstigste Position in der Umgebung ein, sondern können sich, beeinflusst durch eine Umgebungstemperatur T_U, vom Energieniveau her auch verschlechtern. Die Wahrscheinlichkeit dafür beträgt:

$$p = e^{\frac{-\Delta E}{k_B \cdot T_U}}$$

ΔE steht für die Differenz zwischen dem angestrebten und dem alten Energiezustand, k_B für die Boltzmann-Konstante und T_U für die Temperatur . Indem man Boltzmann-Konstante und Temperatur in einer neuen Konstante T zusammenfasst, lässt sich diese Formel

$$p = e^{\frac{-\Delta E}{T}}$$

gut auf Optimierungsprobleme anwenden. Geht man dann dazu über, den Wert für T mit der Zeit zu verringern, erhält man den eigentlichen Simulated Annealing Algorithmus für ein Minimierungsproblem.

Langsames Abkühlen ist eine Möglichkeit, z.B. im Bereich der Metallschmelze zu verhindern, dass die Atome im Metallgitter bei schnellem Abkühlen in energetisch ungünstigen Positionen stehenbleiben und so die Qualität des erzeugten Materials verschlechtern. Analog soll beim Algorithmus Simulated Annealing durch langsames Reduzieren von T die Möglichkeit zur Akzeptanz einer Verschlechterung immer weiter eingeschränkt werden. Der Abkühlvorgang kann dabei entweder sehr langsam, oder in diskreten Sprüngen (Level) erfolgen. Der Gedanke bei der zweiten Möglichkeit ist der Versuch, dem Algorithmus unter stabilen Bedingungen die Gelegenheit zu geben, den umgebenden Lösungsraum zu durchsuchen.

Wie schon bei den vorhergehenden Algorithmen soll für die Zielfunktion

$$f : \mathcal{S} \rightarrow \mathbb{R}$$

ein Minimum gefunden werden. Dabei geht man wie folgt vor:

1. Wähle einen Startpunkt $s_0 \in \mathcal{S}$.
2. Wähle $s_1 \in \mathcal{S}$ in der Nachbarschaft von s_0.
3. Falls $f(s_1) \leq f(s_0)$, wähle s_1 als neuen Startpunkt und fahre mit (2.) fort. Ist $f(s_1) > f(s_0)$, wähle s_1 mit Wahrscheinlichkeit $p = e^{\frac{\Delta f}{T}}$ mit $\Delta f := f(s_1) - f(s_0)$ als neuen Startpunkt, ansonsten behalte s_0 und fahre mit (2.) fort.

Die „Temperatur“ T sollte im Laufe des Verfahrens langsam gegen 0 gehen ($T \to 0_+$). Ist T sehr groß, so gilt für die Annahmewahrscheinlichkeit p einer Verschlechterung: $p \approx 1$. Mit $T \to 0_+$ gilt auch für die Akzeptanzwahrscheinlichkeit $p \to 0$. Die Folge der Temperaturverringerungen wird als *Kühlschema* bezeichnet. Dabei kann u.a. unter den folgenden Formen unterschieden werden (siehe [Vos93]):

Konstant: $T(n) = k$ konstant, wobei n die Nummer des Iterationsschrittes ist.

Arithmetisch: $T(n) = T(n-1) - k$ mit einer konstanten Temperaturreduktion k. Dabei muss die Anfangstemperatur $T(0)$ so groß gewählt werden, dass die Temperatur auch nach der maximalen Anzahl von Iterationsschritten nicht negativ wird.

Geometrisch: $T(n) = \alpha(n) \cdot T(n-1)$ (α typischerweise zwischen 0.8 und 0.99)

Durch Verwendung entsprechender Schemata lässt sich also steuern, mit welcher Wahrscheinlichkeit und wie lange Verschlechterungen akzeptiert werden. So kann ggf. anfangs sehr weiträumig gesucht werden, während sich mit fortschreitender Absenkung von T die Suche auf einen zuvor gefundenen erfolgversprechenden Bereich des Suchraums beschränkt. Für $T \to 0$ geht Simulated Annealing in Hillclimbing über. Der Vorteil dieses Verfahrens gegenüber dem Hillclimbing liegt darin, dass einmal gefundene – insbesondere schlechte – lokale Optima mit einer Wahrscheinlichkeit größer Null wieder verlassen werden können.

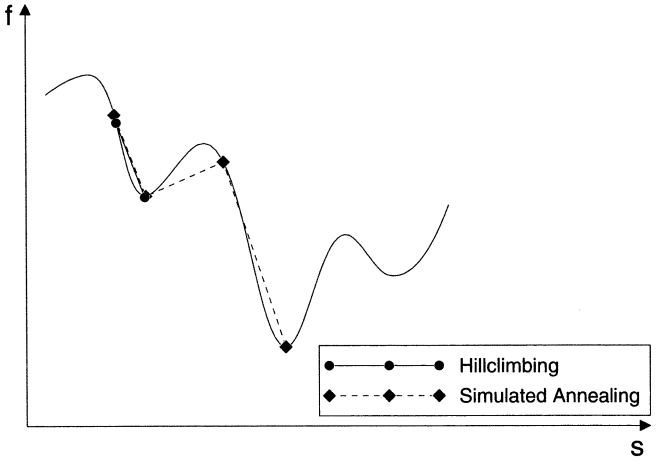


Abbildung 3.2 Simulated Annealing im Vergleich zum Hillclimbing

Generell ist es für einen Simulated Annealing Algorithmus einfacher, ein lokales Minimum zu verlassen, dessen Umgebung relativ flach verläuft. In diesem Fall sind die in Kauf zu nehmenden Verschlechterungen nur gering, während im anderen Fall eine Akzeptanz durch die große zu erwartende Verschlechterung sehr unwahrscheinlich wäre.

3.9 Threshold Accepting

Treshold Accepting wurde von Duck et. al. (siehe [DS90]) entwickelt. Auch dieser Algorithmus ist in der Lage, ein einmal erreichtes lokales Optimum wieder aufzugeben und Verschlechterungen in Kauf zu nehmen. Gegenüber dem Simulated Annealing stellt er eine Vereinfachung dar, die in vielen Bereichen mit großem Erfolg eingesetzt wurde.

Im Gegensatz zum Simulated Annealing verwendet dieser Algorithmus kein Kühlschema, sondern behält einen Schwellenwert T_k solange bei, bis während einer vorher festgelegten Anzahl von Schritten keine Verbesserung mehr aufgetreten ist. Der Schwellenwert geht im Laufe der Zeit gegen 0, beispielsweise durch Multiplikation mit einem vorher festgelegten Faktor $0 < \alpha < 1$. Jede neue Lösung s_{i+1}, die den Wert von s_i um nicht mehr als T unterschreitet, wird akzeptiert, jede andere zurückgewiesen. Die Berechnung einer Annahmewahrscheinlichkeit fällt ebenso weg wie das Auswürfeln einer Zufallszahl, wodurch der Rechenaufwand reduziert wird. Der zugehörige Algorithmus hat die folgende Form:

1. Wähle einen Startpunkt $s_0 \in S$ und eine Starttemperatur T_0.
2. Wähle $s_1 \in S$ in der Nachbarschaft von s_0.
3. Ist $f(s_1)$ um mehr als T_0 schlechter als $f(s_0)$, behalte s_0, ansonsten wähle s_1 und fahre mit (2.) fort. Falls über einen vorher festgelegten Zeitraum keine Verbesserung eintritt, senke T_t langsam auf 0.

Durch den Verzicht auf ein vorher festgelegtes Kühlschema ist es unwahrscheinlicher, dass es zu einer Situation kommt, in der T_k reduziert wird, bevor die Umgebung der aktuellen Lösung ausreichend erforscht wurde. Auf der anderen Seite wird durch die erfolgsabhängige Reduktion von T dafür gesorgt, dass nicht unnötig lange versucht wird, bessere Lösungen in der Umgebung zu finden. Stattdessen wird in diesem Fall zügig zu einer genaueren Untersuchung der Nachbarschaft der aktuellen Lösung übergegangen.

In [Nis97] wird eine leicht modifizierte Variante vorgestellt, die über ein konkretes Schema zur Reduktion von T und zur Berechnung der Anzahl der Generationen bis zur nächsten Änderung verfügt. Dabei wird auch die Anzahl der Schritte, die ohne Verbesserung durchlaufen werden, bevor T verringert wird, bei Erreichen eines neuen Simulationslevels angepasst.

3.10 Sintflut-Algorithmus

Wie auch der im vorhergehenden Abschnitt beschriebene Algorithmus zum Treshold Accepting stammt auch der *Sintflut Algorithmus* von Duck et. al. [DSW93]. Der Ansatz zur Problemlösung ist jedoch fast umgekehrt, denn hier wird eine neue Lösung nicht genau dann angenommen, wenn sie maximal um T schlechter ist als die vorige Lösung, sondern es wird jede Lösung angenommen, die über einem bestimmten Wert W liegt. Man geht dabei von einer zu maximierenden Zielfunktion aus. W ist unabhängig von der Bewertung der letzten Lösung.

W wird als Wasserstand bezeichnet, da man sich den Algorithmus am ehesten als einen Wanderer in einem immer stärker überfluteten Gelände vorstellen kann. Das einzige, was der Wanderer nicht darf, ist, Wasser zu durchschreiten. Mit steigendem Wasserstand wird er immer weiter hügelaufwärts gedrängt, bis er schließlich ganz vom Wasser umschlossen ist. Auch dieser Algorithmus entscheidet nicht mehr per Zufall über Annahme oder Ablehnung einer neuen Lösung. Das Schema des Algorithmus sieht folgendermaßen aus:

1. Wähle einen Startpunkt $s_0 \in \mathcal{S}$.
2. Wähle $s_1 \in \mathcal{S}$ in der Nachbarschaft von s_0.
3. Falls $f(s_1) \geq W$, wähle s_1, sonst behalte s_0 und fahre mit (2.) fort. Erhöhe W langsam, bis die aktuelle Lösung schlechter als W ist, d.h., bis der Wanderer nasse Füße bekommt.

Dieser vom Grundprinzip her sehr einfache Algorithmus arbeitet insbesondere bei durch Zielfunktionen beschriebenen „Gebirgen" gut, deren Oberfläche nicht aus einer Vielzahl von stark ausgeprägten Strukturen besteht. Bei derartigen Landschaften besteht die Gefahr, dass ein lokales Optimum plötzlich nicht mehr verlassen werden kann, obwohl der Wasserstand nur geringfügig gestiegen ist.

3.11 Ameisenkolonieoptimierung

Die relativ neue Idee der *Ameisenkolonieoptimierung* (*Ant Colony Optimization, ACO*) stammt von Dorigo, Maniezzo und Colorni [DMC96]. Einen guten Überblick und weiterführende Literatur über diesen Ansatz findet man in [CHS02].

Diese Optimierungsstrategie orientiert sich an dem Prinzip, wie Ameisen kürzeste Wege z.B. zu Nahrungsquellen finden. Eine Ameise hinterlässt Pheromone – Duftstoffe, die von anderen Ameisen wahrgenommen werden – auf einem Pfad, den sie entlang läuft. Wenn eine Ameise keine Pheromone in ihrer Umgebung wahrnimmt, durchläuft sie mehr oder weniger einen Zufallspfad. Findet sie jedoch in ihrer direkten Umgebung eine Pheromonspur, so folgt sie dieser mit hoher Wahrscheinlichkeit. Sind mehrere Pheromonspuren in ihrer direkten Umgebung vorhanden, folgt sie der stärksten mit der höchsten Wahrscheinlichkeit. Auf diese Weise – auch wenn dieses Modell nur ein stark vereinfachtes Bild der Realität wiedergibt – finden und folgen die Ameisen dem kürzesten Weg zu einem Ziel, ohne direkt miteinander kommunizieren zu müssen. Existieren mehrere Wege unterschiedlicher Länge zum Ziel, wird der kürzeste Pfad mit der Zeit von immer mehr Ameisen gewählt werden. Selbst wenn am Anfang nur ein Teil der Ameisen dem kürzesten Pfad folgt und viele zunächst auf einem längeren bleiben, wird der kürzeste Pfad mit der Zeit immer mehr Ameisen anlocken. Das liegt darin begründet, dass die Pheromone sich im Laufe der Zeit kontinuierlich verflüchtigen, so dass die Konzentration der Pheromone auf einem kürzeren Pfad bei gleicher Frequentierung durch Ameisen höher ist als auf einem längeren.

Der Zusammenhang dieser Beobachtungen bei Ameisenkolonien zu Optimierungsproblemen besteht darin, dass man die Aktionen der Ameisen als Konstruktion eines kürzesten Pfades – also als Lösung eines Optimierungsproblems – interpretiert. Ähnlich wie bei Greedy-Heuristiken erfordert die Anwendung der Ameisenkolonieoptimierung die Möglichkeit, eine Lösung schrittweise über Zwischenlösungen zu konstruieren. Eine Ameise konstruiert einen Pfad, indem sie an jeder Verzweigung entscheidet, welcher Pheromonspur sie folgt.

Formal erfordert die Anwendung der Ameisenkolonieoptimierung die folgende Problemstruktur:

- Der Suchraum bzw. die möglichen Lösungen des Optimierungsproblems bestehen aus Sequenzen der Form $(n_1, \ldots, n_l)$, die nicht notwendigerweise die gleiche Länge haben müssen.
- Nicht alle möglichen Sequenzen werden als zulässige Lösungen betrachtet. Es gibt eine Menge Ω von Constraints oder Einschränkungen, durch die bestimmte Sequenzen verboten werden.
- Es existiert eine Nachbarschaftsbeziehung auf der Menge aller Sequenzen. Dabei werden auch unvollständige Sequenzen zugelassen, d.h., Sequenzen die zwar nicht den Constraints aus Ω widersprechen, aber erst zu einer Lösungssequenz ergänzt werden müssen. Die Nachbarschaftsbeziehung beschreibt üblicherweise, wie man von einer kürzeren Sequenz zu einer im Allgemeinen um ein Element längeren Sequenz gelangt, die nicht den Constraints aus Ω widerspricht.
- Es ist vorteilhaft, aber nicht notwendig, dass die Zielfunktion auch auf Teilsequenzen anwendbar ist, die noch keine Gesamtlösung repräsentieren.

Das elementare Prinzip der Ameisenkolonieoptimierung läuft dann nach dem folgenden Schema ab:

- Es werden immer wieder neue „Ameisen" auf den Weg geschickt. Eine Ameise konstruiert eine Lösung, indem sie ausgehend von der leeren Sequenz ihre Sequenz über die Nachbarschaftsbeziehung schrittweise erweitert, bis sie eine vollständige Lösung erarbeitet hat.
- Eine Ameise hinterlässt auf ihrem Lösungsweg in der Sequenz „Pheromone", d.h. an den Übergängen von einem Zustand zu einem benachbarten. Die Menge der Pheromone hängt von der Qualität der Gesamtsequenz oder auch – wenn die Zielfunktion dies zulässt – von der Bewertung der Teillösung im entsprechenden Schritt ab. Je besser die Lösung ist, desto mehr Pheromone hinterlässt die Ameise. Dieser Aspekt entspricht nicht dem natürlichen Ameisenmodell, da eine Ameise ja nicht nachträglich auf ihrem Pfad Pheromone absondert, je nachdem als wie nützlich sich der Pfad erwiesen hat.
- Wenn eine Ameise eine neue Lösung schrittweise konstruiert, lässt sie sich von den Pheromonen leiten, die sie vorfindet. Das bedeutet, dass die Ameise mit höherer Wahrscheinlichkeit einen mit vielen Pheromonen behafteten Schritt wählt, wenn ihr mehrere Auswahlmöglichkeiten zur Verfügung stehen.

- Die Pheromone verflüchtigen sich mit der Zeit, d.h., die Pheromonwerte, die sich an Übergängen von einem Zustand zum nächsten angesammelt haben, werden im Laufe der Zeit verringert, so dass sich der Pheromonwert eines Transition nur dann halten oder erhöhen kann, wenn dieser Übergang immer wieder von Ameisen verwendet wird.

Beim TSP könnte man als Transitionen oder Zustandsübergänge beispielsweise die Verbindung von einer Stadt zu einer anderen verwenden. Die Pheromone, die einer solchen Verbindung zugeordnet werden, hängen dann von der Länge der Verbindung und der Länge der insgesamt konstruierten Rundreise ab. Je kleiner diese beiden Werte sind, desto mehr Pheromone hinterlässt eine Ameise auf dieser Verbindung.

3.12 Grundsätzliche Elemente von Optimierungsstrategien

Wir haben in diesem Kapitel einen kurzen, aber bei weitem nicht erschöpfenden Überblick über eine Reihe von Optimierungsverfahren gegeben. Bevor wir uns dem eigentlichen Thema dieses Buches – den evolutionären Algorithmen – widmen wollen, sollen noch einmal die grundsätzlichen strategischen Elemente betrachtet werden, die von den verschiedenen Optimierungsverfahren verwendet werden.

Sieht man von der in den seltensten Fällen möglichen direkten Berechnung einer analytischen Lösung des Optimierungsproblems ab, besteht eine Strategie in der gezielten lokalen Suche oder Konstruktion einer Verbesserung bzw. guten Lösung. Auf diesem Prinzip beruhen z.B. das Gradientenverfahren und die Greedy-Heuristiken. Auch Hillclimbing handelt im Sinne dieser Strategie, auch wenn die Suche nach einer verbesserten Lösung rein zufällig in der Nähe geschieht. Der Vorteil dieser Verfahren besteht darin, dass sie meistens relativ schnell eine zumindest halbwegs gute Lösung produzieren. Allerdings bewegen sich alle diese Verfahren geradewegs auf ein lokales Optimum zu, aus dem sie nicht entkommen können.

Um dieses Problem der lokalen Optima zu vermeiden oder zumindest abzuschwächen, werden Zufallsmechanismen wie beim Simualted Annealing, beim Sintflut-Algorithmus oder Threshold Accepting eingeführt.

Die Ameisenkolonieoptimierung verwendet neben den beiden eben erwähnten Prinzipien noch ein weiteres Strategieelement. Während bei allen anderen vorgestellten Verfahren Informationen über erfolgreiche oder weniger erfolgreiche Schritte nicht weiter berücksichtigt werden, findet bei der Ameisenkolonieoptimierung ein Informationsaustausch über gute und schlechte (Teil-)Lösungen über die Pheromone statt. Durch diesen Informationsaustausch werden mehr Ressourcen (Ameisen) auf vielversprechende Teile des Suchraums konzentriert.

Zusammenfassend lassen sich folgende Strategieelemente bei der Optimierung identifizieren, die auch in einer Kombination gleichzeitig verwendet werden können:

- **gezielte Suche, Konstruktion oder Verbesserung**, um möglichst schnell eine gute Lösung zu erhalten.

- **Zufallssuche oder -elemente**, die für eine breiteres Abtasten des Suchraums sorgen.
- **Informationsaustausch über vielversprechende Regionen des Suchraums**, um die Ressourcen bzw. die Suche in den entsprechenden Regionen zu verstärken.

Wir werden sehen, dass sich diese Elemente in den evolutionären Algorithmen wiederfinden. Insbesondere das dritte Strategieelement spielt eine zentrale Rolle bei den evolutionären Algorithmen. Dieses Element wird durch Populationen von Lösungen verwirklicht, bei denen sich vor allem die besseren Lösungen vermehren dürfen. Das Vermehren besteht üblicherweise in einem Zufallsmechanismus, der eine (lokale) Zufallssuche realisiert, das das zweite Strategieelement repräsentiert.

Das erste Element, die gezielte Suche oder Verbesserung, wird bei evolutionären Algorithmen häufig vernachlässigt. Dies liegt daran, dass evolutionäre Algorithmen – im Gegensatz zur Ameisenkolonieoptimierung – im Allgemeinen generische Optimierungsverfahren sind, die keine besonderen Voraussetzungen an den Suchraum oder die Zielfunktion stellen. Gezielte Suche oder Verbesserung kann nur problemspezifisch unter Berücksichtigung der speziellen Eigenschaften des Optimierungsproblems realisiert werden. Auch wenn dieser Aspekt bei evolutionären Algorithmen häufig vernachlässigt wird, sollte man ihn, wenn möglich, ebenfalls integrieren.

4 Genetische Algorithmen und Optimierung

In diesem Kapitel sollen die grundlegende Terminologie evolutionärer Algorithmen eingeführt und deren Gebrauch am Beispiel kanonischer genetischer Algorithmen veranschaulicht werden.

4.1 Biologische Evolution

Der Ursprung der evolutionären Algorithmen liegt in der Theorie der biologischen Evolution begründet. Die dahinter stehende Idee knüpft an die Fähigkeiten der Natur an, sich durch Adaption an eine verändernde Umgebung anzupassen, durch Innovation Probleme auf völlig neue Art und Weise zu lösen und Lösungen für sehr komplexe Probleme in kurzer Zeit und durch Prüfung einer relativ geringen Anzahl von Möglichkeiten zu erzeugen. Die eben genannten Fähigkeiten entsprechen genau dem, was zur Lösung einer Vielzahl von heute in Wirtschaft und Technik auftretenden Problemen benötigt wird.

Um den Grundgedanken evolutionärer Algorithmen nachvollziehen zu können, ist es notwendig, sich mit den ursprünglichen, biologischen Begriffen und Konzepte vertraut zu machen. Die evolutionären Algorithmen verwenden diese Begriffe und Konzepte allerdings nur in einem sehr abstrakten Sinne. Sie orientieren sich an der biologischen Evolution und lassen sich von ihr inspirieren, haben aber nicht eine möglichst genaue Abbildung oder Modellierung der biologischen Evolution zum Ziel. Genau wie der Konstrukteur eines Flugzeugs sich von der Natur – den Vögeln – inspirieren lassen kann, ohne das Modell „Vogel" vollständig zu übernehmen. Die aerodynamisch günstige Flügelform der Vögel wird auch bei Flugzeugen verwendet. Die natürliche Antriebsform der Vögel mittels Schwingen der Flügel findet sich aber bei Flugzeugen nicht wieder.

Da die evolutionären Algorithmen ebenfalls nur ein sehr schematisches, abstraktes Modell der biologischen Evolution darstellen, genügen äußerst rudimentäre Kenntnisse über die biologische Evolution zum Verständnis evolutionärer Algorithmen.

In der folgenden Liste sind daher die innerhalb dieses Buches wichtigsten Begriffe aus der Genetik aufgeführt. Es wird jeweils die Bedeutung für die biologische Evolution sowie innerhalb der evolutionären Algorithmen erläutert. Ein etwas vertieftere Darstellung der biologischen Evolution befindet sich im Anhang A.

Name	Biologische Evolution	Evolutionäre Algorithmen
Chromosom:	legt den Bauplan bzw. die Eigenschaften eines Lebewesens (bzw. eines Teils davon) fest	String, Zeichen- oder Zahlenkette nicht notwendig fester Länge
Gen:	einzelnes Teilstück eines Chromosoms, das eine „Teileigenschaft" festlegt, z.B. „Ein Gen für die Augenfarbe"	Character, Feature, Zeichen, Variable
Allel:	Ausprägung eines Gens (z.B. Augenfarbe: blau, grün, ...)	Wert eines Zeichens
Locus:	Ort eines Gens im Chromosom	Position eines Zeichens
Phänotyp:	Erscheinungsbild des Lebewesens	Chromosom-kodierte Problemlösung
Genotyp:	Das Erscheinungsbild des Chromosoms	Die formale Kodierung einer Lösung. *Unterschiedliche Genotypen können den gleichen Phänotyp zur Folge haben (Beginn der Rundreise in einer anderen Stadt beim TSP).*
Population:	Menge von Lebewesen	Menge bzw. Familie (Bag) von Chromosomen
Fitness:	Überlebens- / Vermehrungsfähigkeit eines Lebewesens	Güte einer durch ein Chromosom kodierten Lösung
Generation:	Population zu einem Zeitpunkt	Analog
Reproduktion:	Erzeugen von Nachkommen aus einem oder mehreren Lebewesen.	Erzeugen von Chromosomen aus einem oder mehreren Chromosomen

Charakteristisch für die biologische Evolution ist insbesondere die Fähigkeit der Lebewesen, sich an veränderte Umstände anzupassen und somit die Bewertung der Überlebenschance eines Individuums mit der Zeit zu variieren. Diese genetische Drift wird dazu benötigt, so genannte Fitnesstäler zu überspringen. Zusätzlich findet die

Suche nach einem biologischen Optimum massiv parallel statt, was sein Pendant in den evolutionären Algorithmen findet (siehe Abschnitt 7.5).

Bei der Chromosomenstruktur wird zusätzlich noch in Haploid und Diploid unterschieden. Haploide Strukturen enthalten pro Zelle nur ein Chromosom, während eine diploide Struktur zwei Chromosomen aufweist. Wenn im diploiden Fall die Chromosomen verschieden sind, können im Falle der Reproduktion bei zwei möglichen Allelen nicht beide Ausprägungen zum Einsatz kommen. Sind beide Allele identisch, so werden sie als homozygot bezeichnet, ansonsten als heterozygot. Das unterdrückte Allel wird dann als rezessiv bezeichnet, das ausgedrückte Allel als dominant. Die meisten Anwendungen der evolutionären Algorithmen arbeiten mit haploiden Strukturen, während der Mensch über eine diploide Struktur verfügt.

Die notwendige Neuorganisation der Chromosomen im Fall der Verknüpfung zweier Chromosomen oder Chromosomenpaare wird *Crossover* oder *Rekombination* genannt. Hierbei hängt die Anzahl der Crossover-Punkte von der Länge des Chromosoms ab. Beim so genannten *One-Point-Crossover* werden beispielsweise die beiden relevanten Eltern-Chromosomen jeweils an einer Stelle aufgetrennt und überkreuz wieder zusammengesetzt (Austausch von zwei Teilstücken). Durch diese Rekombination kommt es bei den erzeugten Chromosomen üblicherweise zu großen Unterschieden zu beiden Elternteilen.

Zusätzlich zu den mit Rekombination erzeugten Veränderungen gegenüber den Originalchromosomen kann es noch zu *Übermittlungsfehlern* bei einzelnen Allelen kommen, bei denen die Erbinformation beliebig verändert werden kann. Solche *Mutationen* lassen sich in die folgenden drei Gruppen (siehe [Bäc96]) teilen:

Gen-Mutation: Mit vernachlässigbaren oder deutlichen Auswirkungen auf den Organismus.

Chromosomen-Mutation: Verdopplung von Teilketten (diploider Fall), Umbau und Verlust von Teilketten.

Genom-Mutation: Änderung der Anzahl der Genome oder Chromosomen.

Unter Genom versteht man bei einem Organismus die Gesamtheit der Chromosomen. Eine ausführliche Darstellung des biologischen Hintergrunds findet man bei Schöneburg, Heinzmann und Feddersen [SHF94].

Die Evolution selbst ist ein ständiger Kreislauf von Neuschaffung, Überlebensprüfung, Vermehrung der Besten und Veränderung des Vorhandenen, um sich immer wieder neu anzupassen und zu verbessern. Derselbe Kreislauf wird bei den evolutionären Algorithmen durch die Folge

1. Schaffen einer Startpopulation,
2. Berechnen der Fitness,
3. Selektion und
4. Anwendung genetischer Operatoren (Rekombination / Mutation),

nachempfunden, wobei Schritt 1 nur in der ersten Generation durchgeführt wird.

4.2 Kanonische Genetische Algorithmen: Struktur und Operatoren

In diesem Kapitel sollen Struktur und Operatoren einfacher *kanonischer genetischer Algorithmen (kGA)* am Beispiel von Optimierungsproblemen näher betrachtet werden. Die genetischen Algorithmen bilden eine Untergruppe der evolutionären Algorithmen (EA), auf deren weitere Ausprägungen bzw. Varianten in den Kapiteln 6 und 7 eingegangen werden wird.

Als Suchraum $\mathcal{S}$ wird zukünftig die Menge aller möglichen Lösungen eines Optimierungsproblems betrachtet. Auf diesem Suchraum wird eine Bewertungs- oder auch Fitnessfunktion $fit : \mathcal{S} \rightarrow \mathbb{R}$ definiert, die jedem Element des Suchraums (Individuum) einen Wert als Maß seiner Qualität zuweist. Gesucht wird nach dem Individuum mit der maximalen bzw. minimalen Bewertung (Optimum), wobei die Erbinformation eines Individuums in Form eines Chromosoms c gespeichert wird. Die Entwicklung neuer, bzw. die Veränderung alter Chromosomen wird dabei innerhalb einer Population P von insgesamt $popsize$ Chromosomen durchgeführt, die sich mit vorher festgelegten Wahrscheinlichkeiten p_c und p_m für Kreuzen (Crossover) und Mutation von Generation t zu Generation $t + 1$ verändern.

Die folgende Liste beinhaltet nochmals die wichtigsten der zukünftig verwendeten Bezeichnungen:

$\mathcal{S}$	Suchraum
c	Chromosom
$popsize$	Populationsgröße: Anzahl der Chromosomen pro Generation
l	Länge des Chromosoms
p_m	Wahrscheinlichkeit für Mutation eines Gens
p_c	Wahrscheinlichkeit für Crossover zweier Chromosomen
fit	Zielfunktion, Evaluierungsfunktion, Bewertung, Fitness
$p(c)$	Überlebenswahrscheinlichkeit, Auswahlwahrscheinlichkeit
$P(t)$	Population zum Zeitpunkt t
$P^*(t)$	Hilfspopulation für die Population $P(t)$, auf die die genetischen Operatoren angewendet werden
Abbruchkriterium	z.B. Durchlauf von $t_{\max}$ Generationen; bis eine genügend gute Lösung gefunden ist; bis keine Verbesserungen pro Generation auftreten; bis die mittlere Verbesserung der Chromosomen in den letzten t_{last} Generationen einen Wert unterschreitet;

Neben den oben angegebenen Parametern sind auch Struktur von Chromosomen und Suchraum, sowie die zur Weiterentwicklung der Population notwendigen genetischen Operatoren zu definieren. Demnach müssen die folgenden Randbedingungen festgelegt werden:

- eine Repräsentiation des Suchraums,
- eine Startpopulation,
- eine Fitnessfunktion,
- Selektion und
- Genetische Operatoren.

Repräsentation des Suchraums: Die Elemente des Suchraums müssen als Chromosomen (Zeichenketten mit fester Länge) über einem meistens endlichen Alphabet kodiert werden. Die Zeichenkette wird in einzelne Segmente aufgeteilt, wobei jedes Segment einem Gen entspricht. Bei genetischen Algorithmen wird ein binäres Alphabet verwendet. Das verwendete Alphabet entspräche in diesem Fall also $Z_2 = \{0, 1\}$. Ist das Minimum oder Maximum einer reellen Funktion einer Variablen gesucht, könnte beispielsweise eine Binärdarstellung des Arguments/Eingabewertes der Funktion als Chromosom verwendet werden. Um Teilmengen der reellen Zahlen als Lösungsraum verwenden zu können, ist es notwendig, eine Kodierungs- /Dekodierungsfunktion zu definieren, mit deren Hilfe reelle Zahlen auf eine zugehörige Binärdarstellung abgebildet werden können. Sollen beispielsweise bei einer Optimierung einer Funktion eine reellen Variablen Lösungen auf k Kommastellen genau berechnet werden, so kann man das Argument der Funktion mit 10^k multiplizieren, so dass man nur ganzzahlige Werte zu betrachten braucht. Diese ganzzahligen Werte werden für die Repräsentation als Chromosom in ihre Binärdarstellung umgerechnet (Bsp.: $10.37 \overset{\cdot 10^2}{\longmapsto} 1037$). Die Anzahl der Gene (Bits), die benötigt werden, hängt davon ab, in welchem Intervall die Funktion optimiert werden soll.

Startpopulation: Die Startpopulation ist eine Menge von $popsize \in \mathbb{N}$ Chromosomen, die meist zufällig erzeugt werden. Je nach Problemstellung kann es vorkommen, dass Randbedingungen beachtet werden müssen, um zulässige Lösungen zu erhalten. Eine einfache Heuristik, mit der sich die Anfangsqualität der Population positiv beeinflussen lässt, fügt bereits bekannte „gute" Chromosomen (mit großer Fitness) in die Startpopulation ein. Dabei geht man allerdings das Risiko ein, dass sich diese Chromosomen sehr schnell in der Population durchsetzten, selbst wenn sie lediglich in ein lokales Optimum führen. Dies führt dann zur so genannten *vorzeitigen Konvergenz* (siehe auch Abschnitt 5.3).

Fitnessfunktion: Die Fitnessfunktion entspricht dem, was bei Optimierungsproblemen als Zielfunktion bezeichnet wird. Sie bildet die Grundlage genetischer Algorithmen. Bei ihr handelt es sich um eine Abbildung, die jedem Element des Suchraums eine üblicherweise positive reelle Zahl als Bewertung zuweist:

$$fit : \mathcal{S} \to [0, \infty)$$

Die mit ihr für jedes Chromosom berechnete Bewertung ist entscheidend für dessen Überlebenswahrscheinlichkeit, denn generell gilt innerhalb einer Population das Prinzip, dass im Wesentlichen die Stärksten überleben (*Survival of the Fittest*). Das bedeutet, dass Individuen mit guten oder gut angepassten Eigenschaften bzgl. der Fitnessfunktion die besten Überlebens- und Vermehrungschancen haben.

Falls z. B. das Maximum einer Funktion f gesucht wird ($f \stackrel{!}{=} \max$), dann repräsentiert das Chromosom c_1 genau dann eine bessere Lösung als ein weiteres Chromosom c_2, wenn gilt: $fit(c_1) > fit(c_2)$. Wird nicht nach dem Maximum, sondern nach dem Minimum einer Funktion gesucht ($f \stackrel{!}{=} \min$), so lässt sich dieser Fall sehr leicht auf die Maximierung zurückführen, sofern es eine Konstante $const \in \mathbb{R}$ gibt, für die gilt: $\forall\ c \in \mathcal{S} : f(c) \leq const$. Dadurch lässt sich die Fitnessfunktion umschreiben in die Form $fit(c) = const - f(c)$. fit nimmt nur nichtnegative Werte an und die ursprüngliche Zielfunktion f besitzt genau dann ein Minimum an der Stelle c, wenn fit dort ein Maximum hat.

Ein weiteres Problem bei der Definition einer Fitnessfunktion tritt auf, wenn Chromosomen, die von der Struktur her korrekt sind, kein Element des Lösungsraums kodieren. Ein Beispiel dafür wäre die Suche nach dem Maximum einer Funktion, beschränkt auf ein Intervall. Dadurch wären alle Chromosomen, die reelle Zahlen außerhalb dieses Intervalls kodieren, unzulässige Lösungen. Es besteht dann einerseits die Möglichkeit, derartige Chromosomen zu „reparieren" (so zu verändern, dass sie im Intervall liegen), andererseits können diese Chromosomen entsprechend ihres Fehlers bestraft werden („Penalty-Faktor" > 0). Die Fitnessfunktion müsste in diesem Fall folgendermaßen angepasst werden:

$$\forall\ c \notin S\ :\ fit(c) = f(c) - \text{ Penalty-Faktor}$$

Um aus der vorhandenen Population in Abhängigkeit von der jeweiligen Bewertung die Chromosomen auszuwählen, die die Grundlage für die nächste Population bilden, ist es notwendig, jedem Chromosom c_0 eine Überlebenswahrscheinlichkeit (Auswahlwahrscheinlichkeit) $p(c_0)$ zuzuweisen. Das geschieht bei genetischen Algorithmen durch das Berechnen des Quotienten aus Fitness des Chromosoms und der Fitness der gesamten Population.

$$p(c_0) = \frac{fit(c_0)}{\sum_{c \in P(t)} fit(c)}$$

Durch diese Definition der Überlebenswahrscheinlichkeit folgt:

$$\sum_{c \in P(t)} p(c) \ = \ 1 \quad \text{und } 0 \leq p(c_0) \leq 1.$$

Mit $F_t = \sum_{c \in P(t)} fit(c)$ wird die Gesamtfitness der Population zum Zeitpunkt t bezeichnet. Ist diese Gesamtfitness gleich Null, so erhalten alle Chromosomen c der Population die gleiche Auswahlwahrscheinlichkeit von $p(c) := \frac{1}{popsize}$.

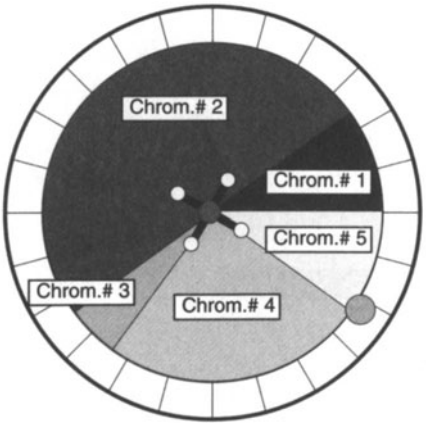
Selektion („Survival of the fittest"): Aus der im vorigen Abschnitt berechneten Überlebenswahrscheinlichkeit $p(c)$ eines Chromosoms c ergibt sich direkt seine Selektionswahrscheinlichkeit. Um die Chromosomen für die Folgepopulation $P(t)$ aus der Vorgängerpopulation $P(t-1)$ auszuwählen, wird daher folgendermaßen vorgegangen:

- Realisieren eines (Pseudo-)Zufallszahlengenerator für eine Gleichverteilung auf dem Intervall $[0,1]$.
- Berechnen einer Zufallszahl $r \in [0,1]$.
- Wähle $c_i \in P(t-1), i \in \{1,\ldots,popsize\}$ genau dann, wenn gilt:

$$\sum_{j=0}^{j<i} p(c_j) \;\leq\; r \;<\; \sum_{j=0}^{j\leq i} p(c_j)$$

Wähle $popsize$-mal zufällig ein Chromosom aus $P(t-1)$ *(mit Zurücklegen!)* mit der Wahrscheinlichkeit $p(c)$[1] für $c \in P(t-1)$ aus und speichere sie in einer Hilfsmenge $P^*(t)$. Dies Art der Selektion wird *Rouletterad-Selektion* genannt.

Bild 4.1 zeigt ein einfaches Beispiel für eine Rouletterad-Selektion. Die Größe der zu den fünf Chromosomen gehörenden Abschnitte des Rades sind proportional zu ihrer Selektionswahrscheinlichkeit (siehe oben), die sich aus der nebenstehenden Tabelle ergibt. Die Kugel des Rouletterades übernimmt die Rolle der Zufallszahl.



Chromosom	eval(c)	fit(c)
1	30	0.1
2	150	0.5
3	15	0.05
4	75	0.25
5	30	0.1
	300	1.00

Abbildung 4.1 Überlebens-/Vermehrungswahrscheinlichkeit

[1] In $P^*(t)$ und $P(t)$ können Chromosomen mehrfach vorkommen.

Genetische Operatoren: Mit Hilfe der genetischen Operatoren wird die Population der ausgewählten Chromosomen $P^*(t)$ verändert und in die Nachfolgepopulation $P(t)$ überführt. Sie dienen zur Erzeugung neuer und insbesondere fitterer Chromosomen durch Veränderung der Erbinformation bzw. durch eine Erhöhung des Verbreitungsgrades nützlicher Gene. Die bei einem kanonischen genetischen Algorithmus verwendeten Operatoren sind die

Mutation, die zufällige Änderung eines oder mehrerer Gene im Chromosom, und der

Crossover-Operator, der aus zwei Chromosomen durch Austausch von Teilketten zwei neue Chromosomen entstehen lässt.

Welche Chromosomen mittels Crossover gekreuzt werden, wird auf der Basis einer Gleichverteilung zufällig bestimmt. Es werden je zwei noch nicht gewählte Chromosomen aus der neuen Menge $P^*(t)$ ausgewählt und mit einer Wahrscheinlichkeit p_c miteinander gekreuzt. Das wird solange durchgeführt, bis alle Chromosomen aus $P^*(t)$ eine Chance zum Crossover hatten. Für den Fall, dass $popsize$ ungerade ist, erhält das letzte Chromosom keine Chance zum Crossover.

Für den Fall, dass Crossover durchgeführt werden soll, wird die Kreuzung zweier Chromosomen folgendermaßen durchgeführt:

1. Wähle zufällig einen Crossoverpunkt $p \in \{1, \ldots, l-1\}$ auf der Basis einer Gleichverteilung.
2. Tausche alle Gene ab dem $(p+1)$-ten Gen zwischen den ausgewählten Chromosomen aus (siehe Bild 4.2):

   (a) $a_1 a_2 \ldots a_{p-1} a_p a_{p+1} \ldots a_l \longmapsto a_1 a_2 \ldots a_{p-1} a_p A_{p+1} \ldots A_l$
   (b) $A_1 A_2 \ldots A_{p-1} A_p A_{p+1} \ldots A_l \longmapsto A_1 A_2 \ldots A_{p-1} A_p a_{p+1} \ldots a_l$

3. Übergabe der neu entstandenen Chromosomen an den nächsten Operator.

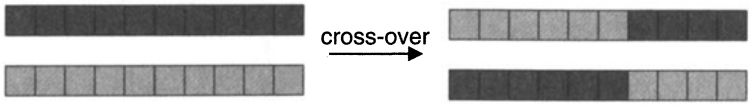


Abbildung 4.2 Cross-Over

Dieser recht einfache Crossover-Operator wird als *One-Point-Crossover* bezeichnet. Verschiedene Tests mit diesem Operator haben gezeigt, dass er relativ schlechte Ergebnisse liefert, da die Wahrscheinlichkeit für ein Allel, ausgetauscht zu werden, von seiner Position im Chromosom abhängt (siehe hierzu Abschnitt 6.5).

Nach Beendigung des Crossover-Verfahrens für die Chromosomen von $P^*(t)$ erfolgt die Anwendung des Mutationsoperators. Zu diesem Zweck wird für jedes Gen jedes Chromosoms (also insgesamt $m \times popsize$ mal) „gewürfelt", ob es mutiert wird. Die Wahrscheinlichkeit für eine Mutation liegt jeweils bei p_m (meist $p_m \approx \frac{1}{m}$).

Falls ein Gen mutiert werden soll, wird das entsprechende Allel zufällig durch ein anderes Allel nach dem folgenden Schema ersetzt:

Falls es k mögliche Allele (Belegungen) für das Gen gibt, erwürfelte eines der $(k-1)$ nicht vertretenen Allele nach einer Gleichverteilung aus und ersetze das ursprüngliche Allel durch das neu ausgewürfelte Allel. Da bei genetischen Algorithmen immer ein binäres Alphabet verwendet wird, es also ohnehin nur zwei Allelen $(0, 1)$ gibt, wird bei der Mutation das eine Allel durch das jeweils andere ersetzt.

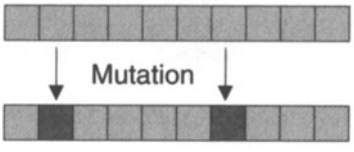


Abbildung 4.3 Mutation

Nach Abschluss der Anwendung der genetischen Operatoren werden die Elemente der Menge $P^*(t)$ in die neue Population $P(t)$ kopiert. Damit startet der Evolutionsprozess zur Bildung der nächsten Generation $P(t+1)$ von neuem.

Genereller Ablauf eines kGA

```
Procedure kGA
        begin
            t := 0
            Initialisiere P(0)
            Bewerte P(0)
            Setze P*(t) := ∅
            while ( nicht Abbruchkriterium ) do
            begin
                t := t + 1
                FOR (i = 0; i < popsize; i++)
                    Auswahl des i-ten Chromosoms für P*(t)
                    aus P(t − 1)
                Anwendung genetischer Operatoren auf P*(t)
                P(t) := P*(t)
                Bewerten von P(t)
            end
        end
```

Warum Mutation und nicht nur Crossover?

Genetische Algorithmen leiden unter dem Problem der *vorzeitigen Konvergenz* (*premature convergence*), sofern nicht mit zusätzlichen Maßnahmen dafür gesorgt wird, dass sich der Algorithmus nicht in einem lokalen Optimum verfängt. Ansonsten können sich frühzeitig und zufällig Chromosomen durchsetzen, die lokalen Optima entsprechen. Diese dominieren die Population und im Extremfall kann dies dazu führen, dass alle Chromosomen identisch sind. Durch Crossover kann keine Verbesserung erreicht werden, denn haben zum Beispiel alle Chromosomen an der i-ten Stelle eine 1, so kann durch Crossover dort keine 0 erzeugt werden. Spears [Spe95] vertritt die Ansicht, dass Mutation immer dann vorteilhaft gegenüber dem Crossover-Operator ist, wenn die Population klein ist. Das Umgekehrte ist der Fall, wenn die Population sehr groß ist, da sie von vornherein eine größere Vielfalt an genetischem Material beinhaltet, so dass viel versprechende Lösungen durch Mutation eher zerstört werden. Wichtig ist das Vorhandensein eines Mutationsoperators zudem immer dann, wenn sich die Fitnesslandschaft ständig ändert, da durch Mutation relativ schnell neue Allele in die Population eingefügt werden können. Crossover hingegen greift nur auf die vorhandenen Allele zu (Spears [Spe93]).

Vorsicht: Es muss garantiert werden, dass die durch Mutation und Crossover neu entstandenen Chromosomen wiederum gültige Zeichenketten darstellen, also Elemente des Suchraums repräsentieren. Oft wird dies durch eine geeignete Repräsentation von Lösungen in den Chromosomen sicher gestellt (siehe auch Abschnitt 6.2). Ansonsten muss es zur Anwendung geeigneter *Reparaturmechanismen* kommen (siehe Abschnitt 6.5 und 6.6).

Beispiel 4.1 (Travelling Salesman Problem (TSP))

Problemstellung: Gegeben sind n Städte, die ein Vertreter auf einer Rundreise besuchen muss.

Gesucht: Die Rundreise mit der kürzesten Länge, also $f \stackrel{!}{=} \min$.

Kodierung: Eine Rundreise ist eine Sequenz von n Städten, in der jede Stadt genau einmal vorkommt.

Der Suchraum $\mathcal{S}$ ist die Menge aller gültigen Rundreisen, $f(s) =$ ist die Länge der Rundreise $s = (a_1, a_2, \ldots, a_n)$, also die Summe der Distanzen zwischen den durch a_1 und a_2, a_2 und a_3, ..., a_{n-1} und a_n, a_n und a_1 repräsentierten Städten.

Im Falle des TSP ergibt sich eine mögliche Repräsentation in Chromosomenform kanonisch: Jedes Gen beinhaltet die Binärkodierung der an der Stelle in der Rundreise besuchten Stadt. Die Bewertung einer Route ist die Gesamtlänge der Rundreise. Da bei dieser Form der Kodierung beim Crossover die Gefahr besteht, ungültige Chromosomen zu erzeugen (doppelt oder nicht besuchte Städte), muss ggf. nach Anwendung der genetischen Operatoren ein Reperaturmechanismus gestartet werden (siehe Abschnitt 6.5).

Beispiel 4.2 (Schnellster Weg)

Problemstellung: Gegeben ist eine Geländeformation, in der man sich je nach dem lokalen Gegebenheiten (Untergrund, Bewuchs usw.) unterschiedlich schnell fortbewegen kann.

Gesucht: Die schnellste Verbindung von einem Punkt A zu einem Punkt B.

Kodierung: Eine Sequenz von drei nummerierten Wegpunkten als Route, die entsteht, wenn man auf einer Geraden jeweils von einem Punkt zum nächsten läuft. Dies stellt eine starke Vereinfachung oder Einschränkung des Suchraums dar. Grundsätzlich könnte man beliebige Kurven durch das Gelände als potentielle Lösungen zulassen.

Die gegebene Landschaft wird mit einem Gitter überzogen (siehe Abbildung 4.4), die einer groben Einteilung in Geschwindigkeitsstufen entsprechen. Jedem Quadrat ist eine Zahl zugeordnet, die angibt, wie lange man benötigt, um dieses Quadrat zu durchqueren (Weiß 1; Hellgrau 2; Dunkelgrau 3). Alle Gitterpunkte werden nummeriert und als Wegpunkte zugelassen.

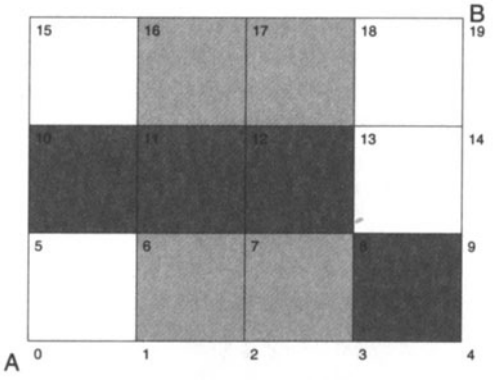


Abbildung 4.4 Beispiel: Schnellster Weg

Vereinfachte Routen: Eine Route besteht aus Geradenstücken; dabei beginnt und endet jedes Geradenstück auf einem Gitterpunkt. Gehört außer einem Eckpunkt ein innerer Punkt eines Quadrates zu einem Geradenstück, so muss die volle Zeit für das Durchqueren des Quadrates berechnet werden. Die Ränder der Quadrate werden bei Mehrdeutigkeit jeweils zum linken bzw. unteren Quadrat gerechnet. Im folgenden werden die Nummern der Gitterpunkte direkt verwendet, obwohl damit jeweils die zugehörige Binärdarstellung gemeint ist.

Chromosom: Zeichenkette aus drei Zeichen (Genen)

Allele: $1, \ldots, 18$ (Gitterpunkte).

Suchraum: $\mathcal{S} := \{(a_1, a_2, a_3) \mid a_1, a_2, a_3 \in \{1, \ldots, 18\}\}$

Ein Beispiel für die Interpretation des Chromosom $(5, 11, 11)$ zeigt die Abbildung 4.5), in der die Route als Linie eingezeichnet wurde.

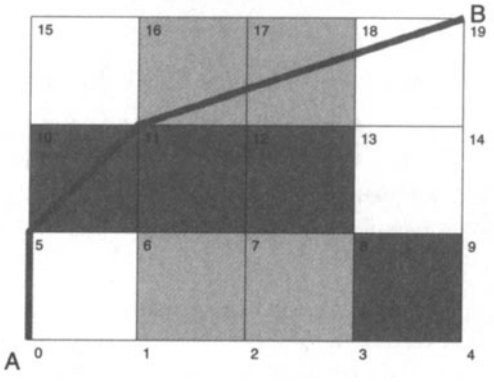


Abbildung 4.5 Eine mögliche Lösung (5,11,11)

Bemerkung: Das Besondere an dieser Form der Kodierung ist, dass verschiedene Chromosomen (Genotypen) dieselbe Lösung (Phänotyp) repräsentieren können. Beispiele dafür sind die Chromosomen $(6, 12, 18) \equiv (6, 6, 18) \equiv (18, 18, 18) \equiv (6, 18, 18)$, deren Phänotyp in Abbildung 4.6 dargestellt ist.

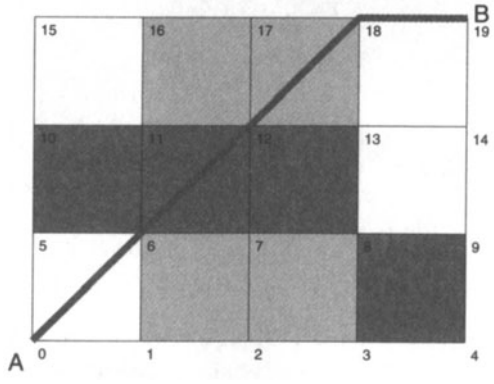


Abbildung 4.6 Der Phänotyp von (6,12,18), (6,6,18), (18,18,18), (6,18,18)

Eine Beispielrechnung mit einer (unrealistisch) kleinen Population soll im folgenden für eine Generation exemplarisch durchgeführt werden. Hierbei ist $popsize = 8$, $p_c = 0.25$, $p_m = 0.05$, und als Abbruchkriterium wird die Durchführung von 30 Generationen verwendet. $P(0)$ wird zufällig erzeugt mit $P(0) = [(8, 15, 7),\ (10, 5, 10),\ (15, 16, 15),\ (16, 3, 14),\ (6, 6, 6),\ (13, 14, 14),\ (5, 15, 16),\ (5, 15, 16)]$

Die eigentliche Bewertungsfunktion soll maximiert werden. Die Fitnessfunktion

kann dann so definiert werden, dass die Routendauer jedes Chromosoms von der maximalen – in der Population vorhandenem – Routendauer oder dem ggf. schlechtest möglichem Wert abgezogen wird. Ein Maximierungsproblem ergibt sich kanonisch, wenn man statt der Durchquerungszeit der Felder eine Bewertungszahl vergibt, die umso höher ist, je leichter ein Feld durchquert werden kann. Als Fitnessfunktion wird hier die zuerst genannte Variante verwendet:

$$fit(c) = \max\{0, \max_{i \in \{1,\ldots,popsize\}} \{\text{Routendauer Chromosom } i\} - \text{Routendauer}(c)\}$$

Startpopulation $P(0)$:

$c \in P(0)$	Routendauer	fit(c)	p(c)
(8,15,7)	22	0	0.000
(10,5,10)	16	6	0.097
(15,16,15)	13	9	0.145
(16,3,14)	20	2	0.032
(6,6,6)	10	12	0.193
(13,14,14)	11	11	0.177
(5,15,16)	11	11	0.177
(5,15,16)	11	11	0.177
		$\sum 62$	

Die Anwendung der Rouletterad-Selektion könnte dann zu folgendem Ergebnis führen:

$$(8,15,7)^*,\ (15,16,15)^*,\ (16,3,14),\ (6,6,6)^*,$$
$$(6,6,6),\ (6,6,6),\ (13,14,14),\ (13,14,14)^*$$

Auf die mit * gekennzeichneten Chromosomen wird der Crossover-Operator angewendet. Bei vier ausgewählten Chromosomen ergibt sich eine Crossover-Anzahl von zwei, wobei der Operator beim ersten Paar zufällig an der Position 1 und beim zweiten Paar an der Position 2 angewendet wird. Damit ergibt sich:

$$\begin{matrix}(8, & 15, & 7)\\ (13, & 14, & 14)\end{matrix} \Rightarrow \begin{matrix}(8, & 14, & 14)\\ (13, & 15, & 7)\end{matrix} \quad \text{und} \quad \begin{matrix}(15, & 16, & 15)\\ (6, & 6, & 6)\end{matrix} \Rightarrow \begin{matrix}(15, & 16, & 6)\\ (6, & 6, & 15)\end{matrix}$$

Nach der Durchführung des Crossovers sieht die Population $P^*(1)$ folgendermaßen aus:

$$(8,14,14),\ (15,16,6),\ (16,3,14),\ (6,6,15),$$
$$(6,6,6),\ (6,6,6),\ (13,14,14),\ (13,15,7)$$

Durch Anwendung des Mutationsoperators ergibt sich dann letztendlich die Population $P(1)$:

$(8, 12, 14)$, $(15, 16, 6)$, $(16, 3, 14)$, $(8, 12, 15)$,
$(6, 6, 6)$, $(6, 6, 6)$, $(13, 14, 14)$, $(13, 4, 7)$

Das Verfahren wird analog für die nächsten 29 Generationen durchgeführt.

Dieses Beispiel zeigt eine Vorstufe des in Abschnitt 6.8 vorgestellten Tools ROGENA, das Flugrouten für den Anflugverkehr auf einen Flughafen erstellt. An diesem Tool werden dann verschiedene Erweiterungen, die in den vorhergehenden Abschnitten desselben Kapitels vorgestellt wurden, näher erläutert (siehe Kapitel 6).

5 Theoretischer Hintergrund

Da der theoretische Untersuchungen über genetische Algorithmen hauptsächlich bei einer Restriktion auf das binäre Alphabet (Allele nur $0, 1$) und für die Verwendung des einfachen, in Abschnitt 4.2 vorgestellten Algorithmus durchgeführt wurden, werden wir uns in diesem Kapitel auf diesen Bereich einschränken.

Ziel dieses Kapitels wird es sein zu charakterisieren, auf welche Weise evolutionäre Algorithmen den Suchraum durchforsten. Aufgrund der Einschränkung der Theorie auf das binäre Alphabet, der üblicherweise kleinen Populationsgrößen und neuerer Erkenntnisse in der Forschung ist insbesondere die im folgenden vorgestellte „Building-Block-Hypothese" nur beschränkt anwendbar.

Um die theoretische Seite der evolutionären Algorithmen betrachten zu können, müssen zuvor einige Begriffe definiert werden, die in den folgenden Abschnitten verwendet werden.

Definition 5.1 (Charakteristische Funktion einer Menge) *Für eine Menge B über einer Grundmenge A mit $B \subseteq A$ heißt die Funktion*

$$1_B : A \to \{0,1\}, \quad 1_B(a) \mapsto \begin{cases} 1 & \text{falls } a \in B \\ 0 & \text{sonst} \end{cases}$$

charakteristische Funktion *der Menge B.*

Diese Funktion nimmt genau dann den Funktionswert 1 an, wenn die Variable ein Element der Menge B ist.

Definition 5.2 (Hammingdistanz) *Die* Hammingdistanz h *zweier gleichlanger Zeichenketten $c_1 = c_{11} \ldots c_{1l}$ und $c_2 = c_{21} \ldots c_{2l}$ ist definiert als die Anzahl der Loci (Stellen), deren Zeichen (Allele) in beiden Ketten verschieden sind.*

$$h(c_1, c_2) = \sum_{i=1}^{l} \mid c_{1,i} - c_{2,i} \mid$$

Die Hammingdistanz wird im folgenden zusammen mit der charakteristischen Funktion auf der Menge $\{0,1\}$ als ein Maß für die Verschiedenheit ganzer Chromosomen untereinander und insbesondere von Teilbereichen mehrerer Chromosomen verwendet. Sie ist auch wichtig bei der Auswahl einer möglichst passenden Kodierung (siehe 6.2), da dabei darauf geachtet werden sollte, dass im Suchraum nahe beieinander liegende Lösungen auch ähnlich kodiert sind. In diesem Sinne gibt die Hammingdistanz die Nähe zweier Chromosomen an.

Definition 5.3 (Schema) *Ein* Schema H *ist eine Zeichenkette der Länge l, in der neben den Zeichen für die Allele auch das Zeichen $*$* („don't-care-Symbol") *vorkommen darf.*

Ein Chromosom c *passt zum Schema* H (in Zeichen: $c \lhd H$), wenn c mit H an allen Stellen übereinstimmt, außer an den Positionen, wo in H $*$ steht. Also:

$$\sum_{\substack{i=1 \\ H_i \neq *}}^{l} \mid c_i - H_i \mid = 0$$

Bei den folgenden theoretischen Betrachtungen soll die Verteilung der Schemata anstelle der einzelnen Chromosomen im Verlauf eines genetischen Algorithmus verfolgt werden, in der Hoffnung, dass sich ein Zusammenhang zwischen guten Lösungen und bestimmten Schemata entdecken und nutzen lässt. Über die Schemata besteht die Möglichkeit, den Suchraum in nicht disjunkte Mengen aufzuteilen.

Definition 5.4 (Ordnung) *Die Differenz zwischen der Länge l eines Schemas H und der Anzahl der don't-care-Symbole wird als* Ordnung *des Schemas H bezeichnet:*

$$ord(H) = \sum_{i=1}^{l} 1_{\{0,1\}}(h_i)$$

Im Fall binärer Chromosomen mit einer Länge von l gibt es insgesamt 2^l Chromosomen und mit dem zusätzlichen don't-care-Symbol 3^l Schemata . Jedes Chromosom passt zu $\sum_{i=0}^{l} \binom{l}{i} = 2^l$ Schemata, da jedes zu einem Chromosom passende Schema der Ordnung $l - i$ genau i don't-care-Symbole auf beliebigen Positionen des Chromosoms hat. Damit ergeben sich $\binom{l}{i}$ Möglichkeiten zur Auswahl der Positionen. Goldberg hat nachgewiesen, dass die Anzahl der in einer Population vertretenen Schemata bei $popsize^3$ liegt [Gol89]. Das bedeutet, dass wesentlich mehr Schemata durchsucht werden, als Chromosomen in einer Population vorhanden sind, was als implizite Parallelität bezeichnet wird.

Definition 5.5 (Definierende Länge) *Die Differenz zwischen der Positionsnummer des letzten nicht-don't-care-Symbols und der Positionsnummer des ersten nicht-don't-care-Symbols heißt* definierende Länge *eines Schemas H und wird mit $dl(H)$ bezeichnet.*

$$dl(H) = \max_{i \in \{1,\ldots,l\}} \{h_i \neq *\} - \min_{i \in \{1,\ldots,l\}} \{h_i \neq *\}$$

Die definierende Länge wird in dem Moment wichtig, in dem es zur Anwendung von Crossover-Operatoren kommt, da Schemata mit großer definierender Länge wesentlich leichter durch ein Crossover unterbrochen werden als Schemata mit kurzer definierender Länge.

In die völlig gegenteilige Richtung zu Schemata zielt der Begriff der *reduzierten Surrogate*. Darunter versteht man bei Betrachtung von mindestens zwei Chromosomen die auf die unterschiedlichen Allele reduzierten Strings. Positionen mit gleichen Einträgen werden durch einen Strich ersetzt.

Beispiel 5.1 $H =$ **0*11*10*
$c_1 =$ 1100111100 passt zu H
$c_2 =$ 1111111101 passt nicht zu H
$h(c_1, c_2) = 3$
$ord(H) = 10 - 5 = 5$
$dl(H) = 9 - 3 = 6$
Reduziertes Surrogat von c_1 mit c_2: $--00-----0$
Zu H passen $2^{l-ord(H)} = 2^5 = 32$ Chromosomen aus $\mathcal{S}$. Die Wahrscheinlichkeit, dass ein Crossover-Punkt in das Schema fällt, liegt bei $2/3$: Alle Stellen hinter der ersten 0 und vor der letzten 0.

Veranschaulichung von Schemata in Hyperwürfeln und Hyperebenen:
Sei $l = 3$:
Jedes Schema H beschreibt eine Hyperebene der Dimension $l - dl(H)$ im Würfel (vgl. Abbildung 5.1). Sind zwei Allele festgelegt, so handelt es sich um eine Kante: $*00 \equiv$

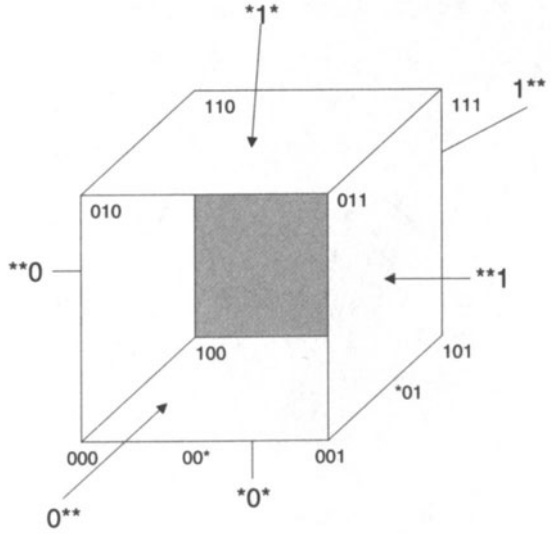


Abbildung 5.1 Veranschaulichung von Schemata in Hyperwürfeln und -ebenen

Kante von 000 nach 100. Bei nur einem festgelegten Allel ergibt sich eine Ebene: $0** \equiv$ Ebene vordere Fläche. Legt man kein Allel fest, so deckt das Schema den gesamten Suchraum (Würfel) ab: $*** \equiv$ Würfel.

Veranschaulichung durch Funktionen:
Sei $f : [0, 1) \rightarrow \mathbb{R}^+$ eine positive reelle Funktion. Dann kann der Definitionsbereich wie folgt in disjunkte Intervalle zerlegt, d.h. diskretisiert werden:

$$[0,1) = \bigcup_{i=0}^{15} [i/16\ ,\ (i+1)/16),$$

wobei jedem von i erzeugtem Intervall die Zahl i eindeutig als Vertreter zugewiesen wird. Jedes Schema kann dem entsprechend durch eine zugehörige Menge von Integerzahlen dargestellt werden.

Als Optimierungsaufgabe soll das Intervall $k, k \in \{0, \ldots, 15\}$, bestimmt werden, für das der Flächeninhalt unter dem Funktionsgrafen einer gegebenen positiven Funktion f maximal ist. Also:

$$fit(c) \stackrel{!}{=} \int_{i/16}^{(i+1)/16} f(x)dx \quad \text{maximal für } i \in \{0, \cdots, 15\}$$

Die folgenden Bilder (Bild 5.2 bis Bild 5.4) zeigen für einige Schemata die von ihnen erzeugte Aufteilung des Suchraums. Zu dem Schema $(0 * **)$ passen alle Zahlen/Intervalle i, deren (vierstellige) Binärdarstellung mit einer 0 beginnt.

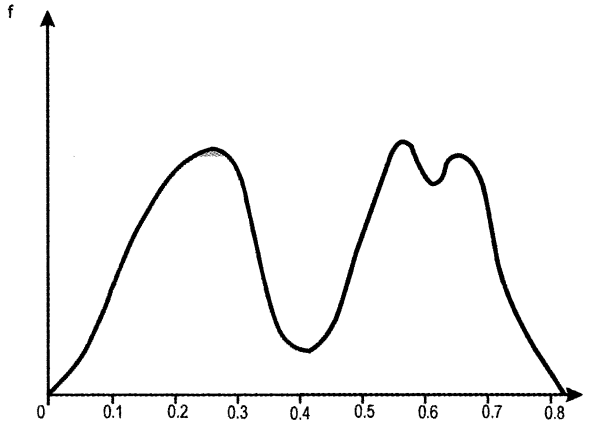


Abbildung 5.2 Funktion mit Schema $(0 * **) \approx \{0, 1, 2, 3, 4, 5, 6, 7\}$

5.1 Schema-Theorem und Building-Block-Hypothese

Definition 5.6 *$N(H,t)$ bezeichnet die mittlere Anzahl der Chromosomen in der Generation t, die zum Schema H passen.*

Gesucht ist ein Zusammenhang zwischen $N(H, t+1)$ und $N(H,t)$, um Aussagen über die Konvergenz der Population bzgl. die Güte des gefundenen Optimums machen

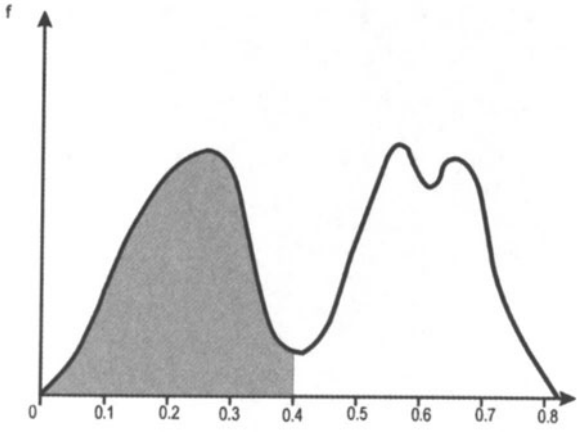


Abbildung 5.3 Funktion mit Schema $(*1**) \approx \{4, 5, 6, 7, 12, 13, 14, 15\}$

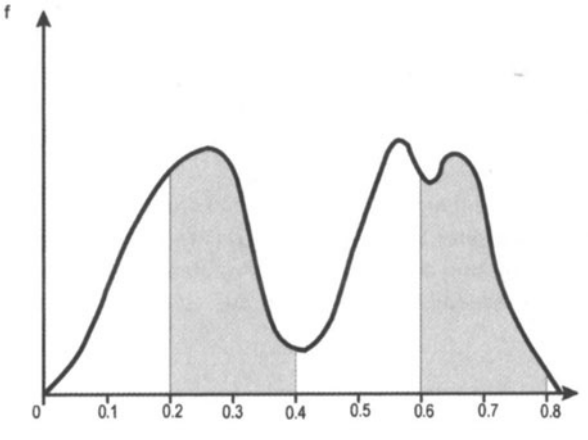


Abbildung 5.4 Funktion mit Schema $(01**) \approx \{4, 5, 6, 7\}$

zu können. Dazu müssen die Auswirkungen der Selektion und der genetischen Operatoren Crossover und Mutation auf die in einer Population vertretenen Schemata untersucht werden. Das Schema-Theorem in seiner ursprünglichen Form wurde von Holland [Hol94] entwickelt, der sich in den folgenden Jahren insbesondere um die Entwicklung des theoretischen Hintergrunds bemühte.

Um die durchschnittliche Anzahl zu erwartender Vertreter eines Schemas bestimmen zu können, werden im folgenden einige zur Vereinfachung der Formeln notwendige Begriffe eingeführt. Mit ihnen wird dann sukzessive aus $N(H,t)$ unter Berücksichtigung der Veränderungen durch den Übergang zur nächsten Generation der Wert für $N(H,t+1)$ berechnet.

Definition 5.7 *$N(H, t + \Delta t_{sel})$ bezeichnet die mittlere Anzahl der Chromosomen, die nach der Selektion in der Generation t zum Schema H passen.*

Analog werden $N(H, t + \Delta t_{sel} + \Delta t_{cross})$ und $N(H, t + \Delta t_{sel} + \Delta t_{cross} + \Delta t_{mut})$ definiert, wobei der letzte Term dem Ausdruck $N(H, t+1)$, also der Anzahl der Vertreter des Schemas in der Generation $t + 1$ entspricht.

Bei der im folgenden definierten Fitness für Schemata muss berücksichtigt werden, dass zwischen der Fitness eines Schemas bei Betrachtung aller möglichen Instanzen und der Fitness bei Einschränkung auf die aktuelle Population große Unterschiede bestehen können. Diese Unterschiede fallen umso größer aus, je kleiner der Wert des Bruches $\frac{l}{|\mathcal{S}|}$ ausfällt.

Definition 5.8

$$\overline{fit(H)} = \frac{\sum_{\substack{c \in \mathcal{S} \\ c \lhd H}} fit(c)}{N(H, t)}$$

bezeichnet die mittlere Fitness der Chromosomen *aus $P(t)$, die zu H passen, während man mit*

$$\overline{p(H)} = \frac{\sum_{\substack{c \in P(t) \\ c \lhd H}} p(c)}{N(H, t)}$$

die mittlere Überlebenswahrscheinlichkeit der Chromosomen *in $P(t)$ bezeichnet, die zu H passen.*

Da zum Aufbau der nächsten Generation die Selektion $popsize$-mal durchgeführt wird, ergibt sich mit Definition 5.8

$$N(H, t + \Delta t_{sel}) = \sum_{\substack{c \in P(t) \\ c \lhd H}} p(c) \cdot popsize = \underbrace{N(H, t)\overline{p(H)}}_{G} \cdot popsize. \tag{5.1}$$

Hierbei bezeichnet G in der Gleichung die Wahrscheinlichkeit, dass ein zu H passendes Chromosom bei der Selektion ausgewählt wird. Erweitert man Gleichung (5.1) um den Anteil, der durch Crossover nicht zerstört, bzw. erzeugt wird, so ergibt sich:

$$\begin{aligned} N(H, t + \Delta t_{sel} + \Delta t_{cross}) &= \underbrace{(1 - P_c) N(H, t + \Delta t_{sel})}_{A} \\ &+ \underbrace{p_c (N(H, t + \Delta t_{sel})(1 - P_{loss}) + \text{gaines})}_{B} \end{aligned} \tag{5.2}$$

mit:

A: Mittlere Anzahl der zum Schema H passenden ausgewählten Chromosomen, die nicht durch Crossover zerstört werden.

B: Mittlere Anzahl der zu H passenden Chromosomen nach der Anwendung des Crossover-Operators.

gaines: Anzahl der durch Crossover neu entstandenen zu H passenden Chromosomen.

P_{loss}: Wahrscheinlichkeit, dass zum Schema H passende Chromosomen durch Crossover zerstört werden.

Der Wert für P_{loss} kann wie folgt abgeschätzt werden, wobei der Einfluss der definierenden Länge klar hervortritt:

$$\begin{aligned} P_{loss} &\leq \text{[Wahrsch., dass Crossover zwischen ersten und letztem nicht-don't-care-Symbol stattfindet]} \times \text{[Wahrsch., dass das zweite Chromosom für Crossover nicht zu } H \text{ passt]} \\ &= \frac{dl(H)}{l-1}\left(1 - \underbrace{\frac{N(H, t+\Delta t_{sel})}{popsize}}_{C}\right) \end{aligned} \tag{5.3}$$

Hierbei ist C die mittlere Wahrscheinlichkeit, dass das ausgewählte Chromosom zu H passt. Es gilt im allgemeinen $P_{loss} <$ (5.3), da je nach Lage des Crossover-Punktes auch eines der neu erzeugten Chromosomen dem Schema entsprechen kann (siehe „Erhaltung" im folgenden Beispiel).

Beispiel 5.2 Beispiel für Verlust („loss"):

$$\begin{array}{lcll} H & = & **0*|1*1* & \\ c_1 \lhd H & = & 0000|1111 & \nearrow\!\!\!\!\searrow \quad 00000000 \not\lhd H \\ c_2 \not\lhd H & = & 1111|0000 & \nearrow\!\!\!\!\searrow \quad 11111111 \not\lhd H \end{array}$$

Beispiel für Erhaltung:

$$\begin{array}{lcll} H & = & **0*1|*1* & \\ c_1 \lhd H & = & 00001|111 & \nearrow\!\!\!\!\searrow \quad 00001000 \not\lhd H \\ c_2 \not\lhd H & = & 11001|000 & \nearrow\!\!\!\!\searrow \quad 11001111 \lhd H \end{array}$$

Beispiel für Gewinn („gaines"):

$$\begin{array}{lcll} H & = & **0*|1*1* & \\ c_1 \lhd H & = & 0000|0000 & \nearrow\!\!\!\!\searrow \quad 00001111 \lhd H \\ c_2 \not\lhd H & = & 1111|1111 & \nearrow\!\!\!\!\searrow \quad 11110000 \not\lhd H \end{array}$$

Mit Hilfe der Gleichungen (5.1) bis (5.3) lässt sich $N(H, t+\Delta t_{sel}+\Delta t_{cross})$ so abschätzen, dass in der entstehenden Ungleichung nur noch Werte aus der Population $P(t)$ vorkommen:

$N(H, t+\Delta t_{sel}+\Delta t_{cross})$

$$\overset{(5.2)(5.3)}{\geq} (1-p_c)N(H, t+\Delta t_{sel}) + p_c\left[N(H, t+\Delta t_{sel})\right.$$

$$\cdot \left(1 - \left(\frac{dl(H)}{l-1}\left(1 - \frac{N(H,t+\Delta t_{sel})}{popsize}\right)\right)\right)\Bigg]$$

$$\overset{\text{oh. Gaines}}{=} \quad N(H,t+\Delta t_{sel})\left[1 - p_c + p_c\left(1 - \frac{dl(H)}{l-1}\left(1 - \frac{N(H,t+\Delta t_{sel})}{popsize}\right)\right)\right]$$

$$= \quad N(H,t+\Delta t_{sel})\left[1 - p_c\frac{dl(H)}{l-1}\left(1 - \frac{N(H,t+\Delta t_{sel})}{popsize}\right)\right]$$

$$\overset{(5.1)}{=} \quad N(H,t)\underbrace{\overline{p(H)}popsize}_{*}\left[1 - p_c\frac{dl(H)}{l-1}\left(1 - N(H,t)\overline{fit(H)}\right)\right] \tag{5.4}$$

* läst sich auch direkt als Verhältnis der Fitness des Schemas zur Fitness der gesamten Population ausdrücken, indem die Definition der Schemafitness einbezogen wird:

$$\overline{p(H)}popsize = \frac{\sum_{\substack{c\in P(t)\\ c\lhd H}} p(c)}{N(H,t)}popsize = \frac{\sum_{\substack{c\in P(t)\\ c\lhd H}} fit(c)}{\sum_{c\in P(t)} fit(c)} \cdot \frac{popsize}{N(H,t)}$$

$$= \frac{\frac{\sum_{\substack{c\in P(t)\\ c\lhd H}} fit(c)}{N(H,t)}}{\frac{\sum_{c\in P(t)} fit(c)}{popsize}} = \frac{\overline{fit(H)}}{\overline{fit}} \tag{5.5}$$

Dabei ist $\overline{fit}$ die mittlere Bewertung aller Chromosomen aus $P(t)$. Mit Hilfe dieser Nebenrechnung lässt sich jetzt die Wahrscheinlichkeit für die Zugehörigkeit zu einem Schema definieren und im folgenden auf die Gleichung (5.4) anwenden.

Definition 5.9 $P(H,t) = \frac{N(H,t)}{popsize}$ *ist die mittlere Wahrscheinlichkeit, dass ein Chromosom der Population* $P(t)$ *zum Schema* H *passt.*

Dividieren der Gleichung (5.4) durch $popsize$ ergibt zusammen mit Gleichung (5.5) und Definition 5.9:

$$P(H,t+\Delta t_{sel}+\Delta t_{cross}) \geq P(H,t)\frac{\overline{fit(H)}}{\overline{fit}}\left[1 - p_c\frac{dl(H)}{l-1}\left(1 - P(H,t)\frac{\overline{fit(H)}}{\overline{fit}}\right)\right]$$

Berücksichtigt man zusätzlich die Mutation mit der Mutationsrate p_m, so ergibt sich die folgende Abschätzung:

$$P(H,t+\Delta t_{sel}+\Delta t_{cross}+\Delta t_{mut}) \geq P(H,t+\Delta t_{sel}+\Delta t_{cross})(1-p_m)^{ord(H)}$$

Dem entsprechend ergibt sich für die Wahrscheinlichkeit, dass ein Chromosom in der $(t+1)$-ten Generation zum Schema H passt, das so genannte Schema-Theorem :

Satz 5.1 (Schema-Theorem)

$$P(H,t+1) \geq \frac{\overline{fit(H)}}{\overline{fit}}\left[1 - P_c\frac{dl(H)}{l-1}\left(1 - P(H,t)\cdot\frac{\overline{fit(H)}}{\overline{fit}}\right)\right]\cdot (1-P_m)^{ord(H)}P(H,t)$$

Interpretation: Wie aus dem Schema-Theorem direkt ablesbar ist, wird der Wert für $P(H,t+1)$ umso größer, je stärker ein Schema den folgenden Aussagen entspricht:

- Hohe mittlerer Bewertung,
- Kurze definierende Länge.
- Geringe Ordnung.

Diese Schemata vermehren sich in einer Population besonders stark (etwa exponentiell). Das heißt, dass der Lösungsraum besonders gut in den Hyperebenen (Regionen) durchsucht wird, deren Schemata eine hohe Fitness, kurze definierende Länge und geringe Ordnung aufweisen. Solche Schemata heißen *Building Blocks*.

Building Block Hypothese Die Building Block Hypothese geht davon aus, dass sich ein genetischer Algorithmus bei seiner Suche nach dem Optimum auf die so genannten Building Blocks konzentriert, um möglichst schnell das Optimum zu erreichen. Die Vorstellung ist also, dass die beste Lösung sukzessive aus Teillösungen über Schemata mit niedriger Ordnung aufgebaut wird. Da die Anzahl der Schemata, die in einer Population vorkommen, im binären Fall am höchsten ist (siehe [Hol94]), wurde diese Hypothese in der Vergangenheit als Motivation zur Verwendung einer binären Kodierung verwendet. Dagegen sprechen jedoch die folgenden Kritikpunkte:

Probleme: Das Schema-Theorem gilt nur im Mittel, d.h. für $popsize \to \infty$. Es muss mit statistischen Abweichungen gerechnet werden:

- Zufälligkeit bei Selektion, Crossover, Mutation.
- $\overline{fit}$ ist die mittlere Bewertung der *aktuellen* Population, nicht die generelle mittlere Bewertung aller Chromosomen (analog $\overline{fit(H)}$).
- Es ist nicht garantiert, dass $\overline{fit(H)}$ für kurze Building Blocks der besten Lösung hoch ist (Deception, siehe auch Abschnitt 8.2.2).
- Nicht alle Schemata sind in der Population vertreten.
- Die vertretenen Schemata sind nicht unabhängig voneinander, und können die Population gegenläufig beeinflussen.

Aus dem zweiten Kritikpunkt geht hervor, dass man keine Rückschlüsse von der durchschnittlichen Fitness eines Schemas auf die Fitness der in der Population vertretenen Instanzen ziehen kann. In Umgehung dieses Problempunktes wurde die so genannte „Static Building Block Hypothesis" aufgestellt (SBBH), deren Aussage wie folgt lautet (Grefenstette[Gre93]):

> Sei eine Aufteilung des Suchraumes durch eine Hyperebenenpartition mit kurzer Ordnung gegeben. Dann kann erwartet werden, dass der genetische Algorithmus zu der Hyperebene konvergiert, die über die beste statische durchschnittliche Fitness verfügt (den erwarteten Gewinner).

Unter der statischen durchschnittlichen Fitness versteht man die Fitness in Bezug auf den gesamten Suchraum, also über die Population hinaus. Aber auch diese Hypothese verliert ihre Gültigkeit, wenn die Population bereits beginnt, zu konvergieren. Auch die innerhalb eines Schemas vorhandene Varianz ist weiterhin ein Problem. Letztendlich wurde das Problem der mittleren Bewertung aus dem zweiten Kritikpunkt nur durch ein ähnliches Problem ersetzt. Es lassen sich auch hier keine Aussagen über die tatsächliche Konvergenz in Richtung der besten Lösung treffen, da nicht bekannt ist, wie gut die in der Population vorhandenen Instanzen der Schemata eigentlich sind.

Dem entsprechend lässt sich aus dieser Hypothese nicht einmal der Grundsatz ableiten, dass Funktionen, bei denen die besten Lösungen über Schemata mit niedriger Ordnung erreicht werden können, für evolutionäre Algorithmen leicht zu bearbeiten sind (siehe auch Royal Road Funktionen im Abschnitt 8.2.1).

Bemerkung: Die hier vorgestellte Variante des Schema-Theorems ist diejenige, die sich an der einfachsten Form der genetischen Algorithmen orientiert. Es existieren noch verschiedene andere Varianten in Abhängigkeit von der Verwendung komplexerer Operatoren für Selektion, Crossover und Mutation. Im allgemeinen sind jedoch Aussagen über komplexere Formen nur eingeschränkt gültig, was insbesondere den Aufbau guter Lösungen aus Building Blocks angeht.

5.2 Konvergenzbetrachtungen

Gerade für Anwender sind Informationen über die Konvergenzeigenschaften eines Algorithmus von besonderer Bedeutung. Das betrifft einerseits die Konvergenz gegen das gesuchte Optimum, andererseits aber auch die Neigung zur vorzeitigen Konvergenz gegen ein Suboptimum („Premature Convergence"). Bezüglich der Konvergenz gegen das Optimum muss festgestellt werden, dass es bisher keine Aussage über den allgemeinen genetischen Algorithmus gibt, jedoch verschiedene Aussagen über Unterarten wie z.B. den kanonischen genetischen Algorithmus mit Elite-Prinzip (Sicheres Überleben der besten Lösung) [EAVH91], oder spezielle Evolutionsstrategien (Abschnitt 7.1). Allen Beweisen gemeinsam ist jedoch, dass sie bezüglich ihrer Voraussetzungen so stark eingeschränkt wurden (z.B. bzgl. ihre genetischen Operatoren), dass ihre Aussagekraft

für den Anwender gering ist. Insgesamt lässt sich die Konvergenz *relativ* leicht für solche Algorithmen beweisen, bei denen die Mitglieder der nächsten Population in der Reihenfolge ihrer Bewertung ausgewählt werden (Elite-Prinzip) und der Mutationsoperator dafür sorgt, dass der Bereich des Suchraumes, der das Optimum enthält, erreichbar ist [Rud94b].

Die Mehrheit der Beweise beruhen auf Überlegungen zur Markovkettentheorie [All78] [Fel57]. Im folgenden werden die grundlegenden Definitionen zur Theorie der Markovketten gegeben, so dass die zum kanonischen genetischen Algorithmus existierende Theorie vorgestellt werden kann.

Definition 5.10 (Bedingte Wahrscheinlichkeit) *Die Wahrscheinlichkeit für das Eintreten des Ereignisses A, nachdem Ereignis B bereits stattgefunden hat, wird als bedingte Wahrscheinlichkeit von A unter der Bedingung B bezeichnet. In Zeichen:*

$$P[A|B] = \frac{P[A \cap B]}{P[B]}$$

Definition 5.11 (Stochastischer Prozess) *Eine beliebige Familie von Zufallsvariablen $\{X(t), t \in T\}$ heißt* stochastischer Prozess. *Ist die Indexmenge $|T|$ abzählbar unendlich, so wird der Prozess als* diskret parametrisiert *bezeichnet, geschrieben $\{X_n\}$, ansonsten als* kontinuierlich parametrisierter *Prozess.*

Definition 5.12 (Markovprozess/Markovkette) *Ein stochastischer Prozess wird als* Markovprozess *bezeichnet, wenn gilt*

Sei $\{t_1, ..., t_{n+1}\} \subset T$ und seien $x_1, ..., x_{n+1}$ Zustände des Prozesses.

Dann gilt:

$$\begin{aligned} &P[X(t_{n+1}) = x_{n+1}|X(t_1) = x_1, X(t_2) = x_2, ..., X(t_n) = x_n] \\ &= P[X(t_{n+1}) = x_{n+1}|X(t_n) = x_n] \end{aligned}$$

Ist der Zustandsraum diskret, so bezeichnet man den Markovprozess als Markovkette.

Diese Definition sagt aus, dass die Wahrscheinlichkeit für den nächsten Zustand eines Markovprozesses nur von seinem aktuellen Zustand abhängt und nicht von der Folge der vorhergehenden Zustände. Die Wahrscheinlichkeit p_{ij}, vom Zustand x_i in den Zustand x_j zu kommen, wird als *Übergangswahrscheinlichkeit* bezeichnet:

$$p_{ij}(t_{k+1}) = P[X(t_{k+1}) = x_j|X(t_k) = x_i]$$

Dem entsprechend heißt die daraus erzeugbare Matrix *Übergangsmatrix*. Hängen die Übergangswahrscheinlichkeiten nicht von der Zeit ab, so heißt die Markovkette *homogen* und die Übergangsmatrix *stationär*. Mit p^0 wird der Zeilenvektor der Startverteilung bezeichnet.

Definition 5.13 *Eine quadratische $n \times n$-Matrix A heißt*

nichtnegativ, *wenn gilt:* $\forall i, j \in \{1, \ldots, n\} : a_{ij} \geq 0$.

positiv, *wenn gilt:* $\forall i, j \in \{1, \ldots, n\} : a_{ij} > 0$.

primitiv, *wenn gilt: A ist nichtnegativ und $\exists\, k \in \mathbb{N}$ mit A^k positiv.*

irreduzibel, *wenn gilt: A ist die Übergangsmatrix einer stationären Markovkette und jeder Zustand i kann von jedem Zustand j in endlicher Schrittzahl erreicht werden.*

zeilenstochastisch, *wenn gilt:* $\forall i \in \{1, \ldots, n\} : \sum_{j=1}^{n} a_{ij} = 1$.

stabil, *wenn gilt: A ist zeilenstochastisch mit identischen Zeilen.*

nicht degeneriert, *wenn gilt: A ist zeilenstochastisch und* $\forall j \in \{1, \ldots, n\}\ \exists\, i \in \{1, \ldots, n\} : a_{ij} > 0$.

Satz 5.2 *Seien Seien C, M und S zeilenstochastische Matrizen mit M positiv und S nicht degeneriert. Dann ist das Produkt CMS positiv.*

Beweis: Über Nachweis der Positivität von CM durch Ausmultiplizieren und anschließendem Multiplizieren mit der nicht degenerierten Matrix S [Rud94a].

Satz 5.3 *[Ios80] Sei P eine primitive stochastische Matrix. Dann gilt:*

$$\lim_{k \to \infty} P^k = P^\infty = \begin{pmatrix} p^\infty \\ \vdots \\ p^\infty \end{pmatrix} \text{ mit } p^\infty = p^0 \lim_{k \to \infty} P^k = p^0 P^\infty.$$

Dabei ist P^∞ eine positive, stabile und stochastische Matrix. p^∞ ist eindeutig bestimmt, unabhängig von der Startverteilung und die Einträge sind ungleich Null.

Schon aus der Definition der Markovkette wird der Zusammenhang zu den genetischen Algorithmen klar. Auch beim kanonischen genetischen Algorithmus ist die neue Generation unabhängig von der weiter zurückliegenden Vorgeschichte der Population. Dem entsprechend besteht die Aufgabe darin, den genetischen Algorithmus als eine Markovkette darzustellen. Dafür geht man von einer Population aus einzelnen Individuen über zu einem umfassenden Bitstring der Länge $(l \cdot popsize)$, in dessen $popsize$ Abschnitten die Chromosomen der Länge l gespeichert werden. Der gesamte Suchraum setzt sich aus allen derartigen Bitstrings zusammen und hat die folgende Form: $\{0,1\}^{l \cdot popsize}$. Jedes Element dieses Suchraums stellt einen möglichen Zustand des zu der Markovkette gehörenden Zustandsraumes dar. Die zugehörige Übergangsmatrix ist eine $(l \cdot popsize) \times (l \cdot popsize)$-Matrix.

Die Übergangsmatrix P setzt sich zusammen aus der Matrix C, welche die durch Crossover erzeugten Veränderungen umsetzt, sowie aus einer Matrix M (Mutationen) und einer Matrix S (Selektion), also $P = CMS$. Eine Projektion $\pi_k(i)$ wird so definiert, dass man bei ihrer Anwendung den k-ten Abschnitt des i-ten Elements des Zustandsraumes, also das k-te Chromosom erhält. Das bedeutet, dass sich jeder Zustand einer Population als derartiger Bitstring darstellen lässt, wobei die Bitstrings eineindeutig mit den von ihnen kodierten Integerzahlen verknüpft sind. Im folgenden wird daher statt des Bitstrings die zugehörige ganze Zahl verwendet.

Satz 5.4 *Die Übergangsmatrix P des kGA mit Mutationsrate $p_m \in (0,1)$ und Crossover-Wahrscheinlichkeit $p_c \in [0,1]$ ist primitiv.*

Beweis [Rud94a]: C, M und S sind zeilenstochastisch, da jede als Zustand von $\mathcal{S}$ abgebildete Population durch die Operatoren auf einen anderen Zustand von $\mathcal{S}$ abgebildet wird. Außerdem gilt für die Wahrscheinlichkeit, durch Mutation vom Zustand i in den Zustand j zu wechseln:

$$\forall\, i,j \in S \quad m_{ij} = (1-p_m)^{(l-h(i,j))} \cdot p_m^{h(i,j)} > 0 \quad \Rightarrow \quad M \text{ positiv.}$$

Die Wahrscheinlichkeit, ein in der Population t vorhandenes Chromosom k in die Population $n+1$ durch Selektion zu übernehmen, ist gleich:

$$fit(\pi_k(i))/(\sum_{k=1}^{l} fit(\pi_k(i))) \;>\; 0$$

Damit folgt für die Wahrscheinlichkeit, die gesamte Population durch Selektion nicht zu verändern:

$$s_{ii} = \frac{\prod_{k=1}^{l} fit(\pi_k(i))}{(\sum_{k=1}^{n} fit(\pi_k(i)))^n} \;>\; 0$$

Daraus folgt: S ist nicht degeneriert. Zusammen mit Satz 5.4 und Definition 5.13 ergibt sich: P ist primitiv. □

Um eine Aussage über die Konvergenzeigenschaften des kGA machen zu können, wird eine entsprechende Definition der Konvergenz benötigt.

Definition 5.14 *Sei $B_t := max_{k=1,\ldots,l}\{fit(\pi_k(i))\}$ das beste Chromosom der durch den Zustand i repräsentierten Population in der t-ten Generation. B_t ist dann eine Folge von Zufallszahlen. Ein genetischer Algorithmus konvergiert genau dann gegen das globale Optimum fit^*, wenn gilt:*

$$\lim_{t\to\infty} P(B_t = fit^*) = 1$$

Satz 5.5 *Der kGA mit $p_m \in (0,1)$ und $p_c \in [0,1]$ konvergiert nicht gegen das globale Optimum.*

Beweis: Sei $i \in S$ ein Zustand mit $B_t < fit^*$, p_i^t die Wahrscheinlichkeit, zum Zeitpunkt t in Zustand i zu sein. Da es wenigstens diesen einen Zustand gibt, in dem nicht das Optimum enthalten ist, gilt:

$$P\{B_t \neq fit^*\} \geq p_i^t \quad \Longleftrightarrow \quad P\{B_t = fit^*\} = 1 - P\{B_t \neq fit^*\} \geq 1 - p_i^t.$$

Außerdem folgt aus Satz 5.3:

$$\lim_{t\to\infty} p_i^t = p_i^\infty > 0 \qquad \Longrightarrow \qquad \lim_{t\to\infty} P\{B_t = fit^*\} \leq 1 - p_i^\infty < 1$$

Also konvergiert B_t nicht gegen das globale Optimum.□

Diese Aussage über den kGA ist ebenso unbefriedigend wie theoretisch, da üblicherweise bei einem genetischen Algorithmus die beste gefundene Lösung gesichert wird und damit die Wahrscheinlichkeit, den Selektionsprozess zu überleben, gleich 1 ist. Nur wenn man sich absolut strikt an die Rouletterad-Selektion hält und sich weigert, die zwischenzeitlich beste gefundene Lösung während des Ablauf des kGA zu speichern, gilt die negative Aussage dieses Satzes.

Wichtig ist die Aussage über die Nicht-Konvergenz im Zusammenhang mit dem Schema-Theorem, das sich ja ebenfalls mit dem kGA befasst. Es liefert dem entsprechend keine Aussage über die Konvergenz gegen das globale Optimum, da die Wahrscheinlichkeit, die beste Lösung wieder zu verlieren, größer als Null ist.

Weicht man insofern vom kGA ab, als dass die beste Lösung sicher überlebt, lässt sich zeigen, dass der neue Algorithmus kGA* gegen das globale Optimum konvergiert, da die beste Lösung nicht verloren gehen kann [Rud94a, Rud94b]. Die Vorgehensweise von Rudolph soll hier nur grob skizziert werden, da sie eine große Ähnlichkeit zum Vorgehen beim kGA aufweist.

Um die beste Lösung in einen genetischen Algorithmus mit einzubeziehen, muss sie nicht nur in die Population (an Position 0) mit aufgenommen werden, sondern auch die Übergangsmatrix P bedarf der Modifikation. Da für jeden Zustand aus S nun auch das bis dahin beste Element gespeichert werden muss, erweitert sich jede Zustandsbeschreibung um die l Gene des besten Chromosoms. Dem entsprechend vergrößert sich die Anzahl der Elemente des Zustandsraums von $2^{l \cdot popsize}$ auf $2^{(popsize+1)\cdot l} = 2^{popsize \cdot l} \cdot 2^l$. Für die Übergangsmatrix bedeutet dies, dass sie sich aus 2^l Matrizen CMS auf der Hauptdiagonalen zusammensetzt. Darüber hinaus wird eine Matrix benötigt, die nach der Anwendung der Operatoren prüft, ob das bisherige beste Element immer noch das beste ist. Wenn das nicht der Fall ist, muss ein Übergang vom bis dahin gültigen Zustand in den Zustand durchgeführt werden, in dem das neue beste Element an Position 0 steht.

Mit diesen Vorarbeiten lässt sich beweisen, dass der Algorithmus konvergiert, unabhängig davon, ob das beste Chromosom vor oder nach Anwendung der Selektion übernommen wird. Im ersten Fall wird verhindert, dass das beste Element durch die Selektion verloren geht. Ähnlich wird auch von Eiben et. al. [EAVH91] beim Beweis der Konvergenz des kGA vorgegangen. Es werden jedoch noch zwei Voraussetzungen benötigt, von denen eine verlangt, dass sich das beste Element der Population nie

verschlechtert, und die andere, dass jeder Zustand durch Mutation und Crossover erreichbar ist. Letztendlich wird also eine Monotonie verlangt, die durch den Einsatz einer Elite-Strategie erreicht werden kann.

Für den Fall der (1+1)-Evolutionsstrategie (siehe Abschnitt 7.1) gelang es Born [Bor78] und Rudolph [Rud92b] unter bestimmten Randbedingungen die Konvergenz mit Wahrscheinlichkeit 1 nachzuweisen. Auf diese Beweise wird in den zugehörigen Kapiteln eingegangen.

5.3 Vorzeitige Konvergenz

Eines der großen Probleme der evolutionären Algorithmen ist die vorzeitige Konvergenz der Population gegen ein Suboptimum. Diese Problematik ist bereits am Schema-Theorem zu erkennen. Wenn das Optimum nicht in der Population enthalten ist und die Bewertung für die in der Population enthaltenen Vertreter einer suboptimalen Lösung über dem Durchschnitt liegt, übernimmt das Suboptimum sehr schnell die gesamte Population. Um das zu verhindern, wurden insbesondere von Anwendern eine Reihe von Strategien, Verfahren und genetischen Operatoren entwickelt, die eine Diversifikation sicherstellen.

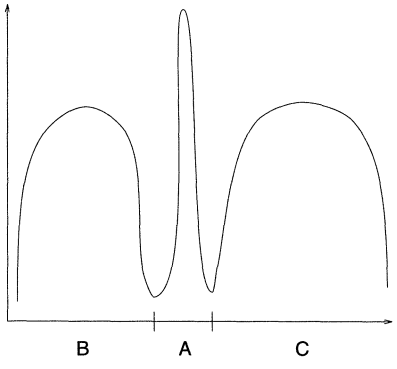


Abbildung 5.5 Problem der vorzeitigen Konvergenz

Als Maß für die Gleichartigkeit einer Population kann die durchschnittliche Hammingdistanz der Population dienen, wie sie von Louis und Rawlins verwendet wurde [LR93]. Hierbei wird für jedes Paar die Hammingdistanz berechnet und durch die Anzahl der möglichen Paarungen $popsize(popsize-1)/2$ geteilt. Bei Verwendung eines kGA ergibt sich als Mittelwert einer Binomialverteilung für die Startpopulation die durchschnittliche Hammingdistanz von $l/2$ mit Streuung $\sqrt{l}/2$. Mit abnehmender durchschnittlicher Hammingdistanz steigt die Gefahr der vorzeitigen Konvergenz.

Gerade das im vorhergehenden Abschnitt beschriebene Verfahren, die beste Lösung jeder Population zu sichern, leistet der vorzeitigen Konvergenz Vorschub. Um das zu verhindern, muss bei der Entwicklung eines evolutionären Algorithmus ein Gleichgewicht

zwischen gründlichem Durchsuchen des Suchraumes und Weiterentwicklung guter, in der Population vorhandener Lösungen gefunden werden (Exploitation versus Exploration). Das lässt sich teilweise über die Anpassung der Populationsgröße und der genetischen Operatoren erreichen (siehe Kapitel 6). Oft ist es jedoch darüber hinaus von großer Bedeutung, dass die Varianz innerhalb der Population durch zusätzliche Operatoren künstlich hoch gehalten wird. Beispiele für derartige Operatoren sind:

Fitnessteilung: Die Fitness eines Chromosoms wird in Abhängigkeit von seiner Ähnlichkeit zum Rest der Population berechnet (siehe Abschnitt 6.3.4).

Verbot von Inzest: Crossover wird nur durchgeführt zwischen Chromosomen, deren halbe Hammingdistanz über einem Schwellwert liegt (siehe Eshelman [Esh91], [ES91]), wobei dieser Schwellwert mit der Zeit sinkt.

Eingeschränktes Crossover: Crossover wird für jedes Gen einzeln ausgwürfelt (siehe „Uniform Crossover") und nur auf dem reduziertem Surrogat durchgeführt.

Populationsabhängige Selektion: Neue Chromosomen werden nur übernommen, wenn sie noch nicht in der Population vorkommen, bzw. alte Chromosomen werden nach einer bestimmten Generationenanzahl aus der Population entfernt.

Die Selektion in Verbindung mit dem Alter der Chromosomen allein garantiert noch keine Vermeidung von vorzeitiger Konvergenz. Sie ist keine Sicherung dagegen, dass ein Chromosom, das aufgrund seines Alters entfernt wurde, nicht sofort wieder durch ein neu entstandenes Pendant ersetzt wird. Eine weitere Möglichkeit ist, dafür zu sorgen, dass zumindest die Chromosomen, die unverändert in die neue Population übernommen werden, voneinander verschieden sind (siehe Abschnitt 7.4.2).

Hauptverursacher vorzeitiger Konvergenz ist der Selektionsoperator . Zu diesem Thema wurden verschiedene Studien zur so genannten „Übernahmezeit" durchgeführt ([GD91], [Bäc96], [Bäc94]). Unter der Übernahmezeit versteht man die erwartete Anzahl der Generationen, die das beste Chromosom der ersten Generation benötigt, um die Population komplett zu übernehmen, wenn nur Selektion durchgeführt wird. Der bisher vorgestellte Operator „Proportionale Auswahl" des kGA übte demnach den geringsten Druck auf die Population aus. Er hat eine Übernahmezeit der Ordnung $O(popsize \cdot \log(popsize))$ Generationen. Das wirkt sich am Beginn insofern positiv aus, als dass auch schlechtere Chromosomen eine Chance zur Weiterentwicklung bekommen. Negativ macht sich der mangelnde Selektionsdruck dann bemerkbar, wenn der Unterschied in der Bewertung der Chromosomen geringer wird.

6 Problemangepasste Operatoren und Verfahren

6.1 Anwendungsbereich

Angesichts der Vielzahl der Optimierungsalgorithmen stellt sich die Frage nach ihrer Anwendbarkeit bei konkreten, praktischen Problemen. Wann macht es Sinn, welchen Algorithmus zu nutzen. Eine eindeutige Antwort lässt sich auf diese Frage nur selten geben. Einige der in Kapitel 3 genannten Verfahren sind von vornherein auf stetige und/oder differenzierbare Zielfunktionen beschränkt, andere nur bei unimodalen Fitnesslandschaften mit Erfolg einzusetzen (z.B. Hillclimbing). Bezüglich der von evolutionären Algorithmen gut bearbeitbaren Probleme lassen sich einige allgemeine Aussagen machen (siehe [Mit96, Nis97]). Sie lassen sich gegenüber anderen Modellen erfolgreich einsetzen, wenn

- der Suchraum groß ist,
- die Fitnessfunktion einen sehr unregelmäßigen Verlauf hat,
- schnelles Entwickeln einer guten Lösung wichtiger ist als das Auffinden des globalen Optimums,
- der Implementationsaufwand gering gehalten werden soll (Vorstudie, Abschätzung von Möglichkeiten),
- das Problem parallelisierbar ist,
- die Fitnesslandschaft mutimodal ist, oder
- kaum Vorwissen über das betrachtete Problem vorhanden ist.

Insbesondere der letzte Grund ist der Gegenstand dieses Kapitels, denn das Anwenden des Basisalgorithmus ist zwar prinzipiell leicht möglich, die erzielten Ergebnisse sind aber häufig nicht von überzeugender Qualität. Dem entsprechend groß ist die Notwendigkeit, Operatoren und Parameter optimal an das betrachtete Problem anzupassen. Insbesondere für die Einstellung der Ablaufparameter wie Populationsgröße, Crossover- und Mutationsraten können jedoch keine bindenden Aussagen getroffen werden. Umfangreiche Tests zur Parametereinstellung wurden u.a. von De Jong [DJ75] und Schaffer et. al. [SCED89] durchgeführt.

6.2 Kodierung und Startpopulation

Die Kodierung des betrachteten Problems (Repräsentation von Lösungen) ist einer der wichtigsten Punkte bei der Anpassung eines evolutionären Algorithmus. Hierbei sollten bestimmte Grundregeln beachtet werden, von denen eine von Goldberg [Gol89] als „Principle of Minimal Alphabet" eingeführt wurde:

> Der Nutzer sollte das kleinstmögliche Alphabet nutzen,
> das dem natürlichen Ausdruck des Problems entgegenkommt.

Damit ist gemeint, dass die Verwendung von zu großen Alphabeten dazu führen kann, dass Rechenzeit durch das Durchsuchen nicht lösungsrelevanter Bereiche im Lösungsraum verloren geht. Bei zu kleinen Alphabeten können gute Lösungen ggf. nicht erreicht werden. In den Anfängen der evolutionären Algorithmen wurden hauptsächlich binäre Zahlen als Alphabet verwendet. Begründet wurde das mit dem impliziten Parallelismus (viele Schemata vertreten durch wenige Chromosomen), wie er durch das Schema-Theorem nachgewiesen wurde (siehe Abschnitt 5.1). Hinzu kommt die einfache Form der evolutionären Operatoren. Für diese Form der Kodierung liegen umfangreiche theoretische Untersuchungen vor, was bei ganzzahliger oder reellwertiger Kodierung nicht in dem Maße der Fall ist. Trotzdem werden die beiden letztgenannten Formen gerade im Bereich der Anwendungen häufig verwendet (z.B. [Hin94],[Ger97]). Ein ausführlichen Vergleich zwischen einer Binär- und einer Fließkommakodierung wurde von Michalewicz [Mic96] durchgeführt und zeigte, dass die Fließkommakodierung nicht nur schneller und genauer war als die entsprechende Binärdarstellung, sondern auch stabiler in den erreichten Lösungen (siehe auch [Gol90]).

Entscheidend bei der Auswahl des Alphabets ist es, dass die Entwicklung von Lösungen möglichst natürlich durch das Alphabet unterstützt wird. Das ist wichtig, um einerseits den Aufwand bei der Umrechnung vom Gen zum Allel (Funktion: Genotyp $\rightarrow$ Phänotyp) möglichst gering zu halten, andererseits, um dafür zu sorgen, dass ähnliche Lösungen auch im Lösungsraum nah beieinander liegen, da ansonsten eine kontinuierliche Verbesserung vorhandener Lösungen nicht möglich ist. Das lässt sich insbesondere durch die Verwendung von Fließkommakodierungen erreichen. Ebenfalls muss berücksichtigt werden, dass die Zerstörung guter Genkombinationen durch Crossover möglichst selten sein sollte (so genanntes „Linkage Problem"). Gerade dieser Punkt lässt sich nur sehr schwierig verwirklichen, da dafür ein Vorwissen über mögliche gute Lösungen vorhanden sein muss, was gerade bei komplexen Problemen nicht immer der Fall ist.

Beispiel 6.1 (schnellster Weg) Im Beispiel 4.2 zum Finden des „schnellsten Weges" wurden die Gitterpunkte auf kanonische Weise von 0 bis 19 durchnummeriert, was einer Integerkodierung entspricht. Für die Verwendung einer Binärkodierung ließe sich die Knotennummer direkt in eine Binärdarstellung umrechnen mit

$$Knotennummer \quad z \in \mathbb{Z} \leftrightarrow (b_1, ..., b_n) \quad \Leftrightarrow \quad z = \sum_{i=1}^{n} (b_i * 2^{n-i})$$

Problematisch bei der Kodierung der durchnummerierten Gitterpunkte ist die Tatsache, dass sich nicht alle in direkter Nachbarschaft zu einem gewählten Punkt liegenden Gitterpunkt auch tatsächlich in einem Schritt – also mit der Mutation genau einer Position – erreichen lassen. So liegen die Punkte 6 und 8 von Punkt 7 aus in einem rechnerischen Abstand von einem Schritt, das gilt jedoch nicht für die Punkte 1 bis 3 und 11 bis 13, obwohl sie in Wirklichkeit dieselbe Distanz aufweisen sollten. Dadurch wird ein geregeltes Absuchen der Nachbarschaft von Punkt 7 z.B. durch Mutation erschwert. Ebenso stellt sich dieser Fall bei binärer Kodierung dar, denn hierbei entspricht 7 der Binärzahl (00111) und 8 dem Tupel (01000). Da ein Mutationsschritt aus dem Wechsel von 0 auf 1 und umgekehrt besteht, haben diese Zahlen eine Hammingdistanz (Anzahl unterschiedlicher Stellen) von 3.

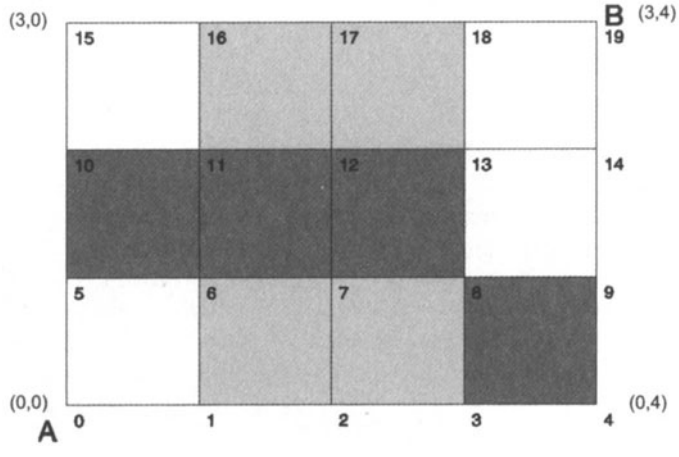


Abbildung 6.1 Beispiel: Schnellster Weg

Eine Möglichkeit, das oben genannte Problem der Nachbarschaft zu berücksichtigen, wäre die Kodierung der Gitterpunkte als Paare aus der Nummer der horizontalen Gitterlinien (0 bis 3, von unten nach oben) und der vertikalen Gitterlinien (0 bis 4, von links nach rechts). Mit dieser Kodierung ergibt sich dann:

$$\text{Knotennummer} \quad z \in \mathbb{Z} \leftrightarrow (p_1, p_2) \quad \Leftrightarrow \quad p_1 = \left\lfloor \frac{z}{5} \right\rfloor \quad \textit{Zeilennummer} \quad \wedge$$
$$p_2 = z - \left\lfloor \frac{z}{5} \right\rfloor * 5 \quad \textit{Spaltennummer}$$

Hierbei steht $\lfloor r \rfloor$ für die größte ganze Zahl, die kleiner oder gleich r ist. Mit dieser Kodierung lassen sich zumindest die über, unter, links und rechts des aktuellen Punktes liegenden Gitterpunkte in einem Schritt erreichen.

Integer Zahl	Std.-Binärcode Code	Distanz	Gray-Code Code	Distanz
0	(000)	-	(000)	-
1	(001)	1	(001)	1
2	(010)	2	(011)	1
3	(011)	1	(010)	1
4	(100)	3	(110)	1
5	(101)	1	(111)	1
6	(110)	2	(101)	1
7	(111)	1	(100)	1

Tabelle 6.1 Gegenüberstellung von Standard-Binärdarstellung und Gray-Code mit Hammingdistanz.

Gray-Code:
Eine weitere Möglichkeit der Kodierung ist die Verwendung des so genannten Gray-Codes (siehe auch [Bäc96]). Der Gray-Code geht mittels einer bijektiven Abbildung direkt aus der Standard-Binärdarstellung einer ganzen Zahl hervor und umgekehrt. Es handelt sich dabei ebenfalls um einen Binärcode. In der folgenden Formel steht das Symbol $\bigoplus$ für die binäre Addition.

$$\forall i \in \{1, \ldots, l\}:$$

$$\gamma: \quad g_i = \begin{cases} b_i & , \quad \text{falls} \quad i = 1 \\ b_{i-1} \bigoplus b_i & , \quad \text{falls} \quad i > 1 \end{cases}$$

$$\gamma^{-1}: b_i = \bigoplus_{j=1}^{i} g_j$$

γ beschreibt die Umrechnung von Graycode in Binärcode, γ^- entsprechend die umgekehrte Berechnung.

In Tabelle 6.1 sind die Kodierungen der Zahlen von 0 bis 7 in Standard-Binärkodierung und in Gray-Kodierung zusammen mit ihrer Hammingdistanz zum Vorgänger aufgeführt. Im Fall des Gray-Codes wird die gewünschte Hammingdistanz von 1 in keinem Fall überschritten. Gray-Code verhält sich zwar im allgemeinen deutlich gutartiger als der Standard-Binärkode, die Qualität hängt jedoch von der Länge der Kodierung ab.

Problematisch bei der Verwendung von Gray-Code ist auch sein Einfluss auf das zu suchende Optimum, da sich die Fitnesslandschaft durch die notwendige Abbildung von Binär- auf Gray-Code in ihrer Struktur deutlich verändern kann, was sowohl Anzahl als auch Lage lokaler Optima betrifft [MW94]. Schwierig wird es besonders dann, wenn die Ergebnisse eines EA mit Gray-Code mit einem anderen Verfahren verglichen werden sollen, in dem eine Binärkodierung verwendet wurde. Die Algorithmen arbeiten schlimmstenfalls auf völlig unterschiedlichen Fitnesslandschaften. Nutzbringend erweist

sich diese Kodierungsart immer dann, wenn über die Operatoren (insbesondere Mutation) versucht wird, die Nachbarschaft zwischen Lösungen zu berücksichtigen.

Nicht nur das Alphabet ist für den Erfolg evolutionärer Algorithmen von großer Bedeutung, sondern auch die Form der Kodierung selbst. Die naheliegendste Unterscheidung ist hierbei die in haploide und diploide Chromosomen (siehe Abschnitt 4.1), da sie direkt von der Natur übernommen werden kann. Der Nutzen der Verwendung einer diploiden Struktur wird ebenfalls durch einen Blick auf die natürlichen Vorbilder deutlich. So kann eine zweite mögliche Lösung, die ggf. unter anderen Bedingungen erfolgreich war, zusätzlich gespeichert werden. Dem entsprechend kann sie selbst dann überleben, wenn ein Individuum mit ihren Ausprägungen zum aktuellen Zeitpunkt nur eine geringe Überlebenschance hätte.

Unter gleichbleibenden Umständen (keine plötzlichen Wechsel) enthält eine haploide Struktur alle notwendigen Informationen und ist in der Lage, sich zügig an die optimale Lösung anzupassen. Eine diploide Struktur hat erst in dem Moment einen Vorteil, wenn sich die Umwelt sehr plötzlich stark verändert, so dass eine vorher unterdrückte Information zum tragen kommen und zu einer guten Anpassung an die neue Situation führen kann. Das setzt eine Änderung in der Dominanz des einen Informationssatzes gegenüber dem anderen voraus. Beispiele für derartige Vorkommnisse in der Natur finden sich bei Goldberg [Gol89]. Dem entsprechend braucht man bei Verwendung einer diploiden Struktur nicht nur zwei Chromosomen pro Paar, sondern auch eine Dominanzregel oder -funktion, die für die Auswahl der sich durchsetzenden Ausprägung sorgt. Für einen evolutionären Algorithmus bedeutet die Anwendung der Dominanzregel für den Ablauf einen Arbeitsschritt mehr, der durchgeführt werden muss.

Unter bestimmten Umständen kann es von Nutzen sein, wenn die Gene eines Chromosoms unabhängig von ihren Loci sind. Das kann z.B. durch die Verwendung von Genpaaren konstruiert werden, wobei jedes Gen zusammen mit seinem Locus im Chromosom abgespeichert werden kann.

$$\begin{matrix} Loci \\ Allel \end{matrix} \quad \begin{pmatrix} i \\ a_i \end{pmatrix}$$

Durch diese Konstruktion sind die folgenden Chromosomen identisch:

$$\binom{1}{7}\binom{2}{3}\binom{3}{3}\binom{4}{5} \quad = \quad \binom{2}{3}\binom{3}{3}\binom{1}{7}\binom{4}{5}$$

Verkürzt kann man für beide Varianten schreiben: (7335) wobei dann jedes Gen die Positionsnummer in der Zeichenkette als Platznummer hat. Wichtig wird diese Art der Kodierung erst dann, wenn als ein genetischer Operator auch die Vertauschung von Positionen im Chromosom verwendet wird, wie es zum Beispiel bei der Inversion (siehe Abschnitt 6.5.3) der Fall ist. Da Schemata mit kurzer definierender Länge seltener durch Crossover getrennt werden als solche mit großer definierender Länge, kann sich eine positionsunabhängige Programmierung als sinnvoll erweisen, da so bereits bewährte Gene im Chromosom nebeneinander liegen können, die in der üblichen Darstellung auf weit voneinander entfernt liegenden Plätzen liegen würden. Auf die für diese Art der Kodierung verwendbaren Crossover-Operatoren wird im Abschnitt 6.5 eingegangen.

Eine weitere wichtige Frage, die vor der Auswahl der Kodierung geklärt werden muss, ist die Frage nach der Länge des Chromosoms. Mehrheitlich werden Chromosomen mit fester Gesamtlänge verwendet. Aber das ist nur dort sinnvoll möglich, wo die Struktur einer Lösung im voraus bekannt ist.

Beispiel 6.2 Gesucht ist die kürzeste, konfliktfreie Route für ein Flugzeug zwischen den Flughäfen A und B, die durch Strecken zwischen festgelegten Wegpunkten definiert wird.

Konfliktfrei bedeutet dabei, dass das Flugzeug einen vorgeschriebenen Mindestabstand zu anderen, sich im gleichen Luftraum bewegenden Flugzeugen nicht unterschreitet. Die notwendige Anzahl von Punkten, an denen die Flugrichtung geändert wird (Wegpunkte), hängt von der Verkehrssituation im Luftraum ab. Ist die Anzahl weiterer Flugzeuge gering, so kann man mit Start- und Zielflughafen als Wegpunkte auskommen, ist die Anzahl hoch, kann auch die Zahl der notwendigen Wegpunkte beliebig steigen [Ger97]. Das bedeutet für eine Kodierung, dass sie entweder eine hohe Anzahl von nichtbeachteten Genen mitführt, die – falls nötig – aktiviert werden, oder dass das Chromosom eine variable Länge aufweist und zu den üblichen genetischen Operatoren noch „Löschen" und „Hinzufügen" von Genen oder Chromosomenstücken dazukommen (siehe auch Abschnitt 6.8).

Neben der Verwendung von einfachen, unterschiedlich langen Chromosomen mit entsprechend geänderten (Crossover) oder hinzugefügten Operatoren (Löschen, Hinzufügen) existieren mehrere komplexere Ansätze zur Lösung dieses Problems. Goldberg et. al. [GKD89], [GDK90] entwickelten den Messy-GA (siehe Abschnitt 7.4.1), der auf der oben erläuterten Locus-unabhängigen Struktur aufbaut und bei dem nicht notwendigerweise pro Locus genau eine Belegung vorhanden sein muss. Koza [Koz92, Koz94] entwickelte maßgeblich die genetische Programmierung (siehe Abschnitt 7.3), in der die Chromosomen die Struktur von abstrakten Bäumen [Sed95] haben, in die ganze Unterbäume eingebunden bzw. aus ihnen gelöscht werden können. Des weiteren wurden Kodierungen entwickelt, bei denen das Chromosom nicht als String, sondern als Matrix vorliegt [FM91], so dass entsprechende neue Operatoren konstruiert werden mussten (vgl. Abschnitt 6.5).

6.3 Fitnessfunktion

Die Fitnessfunktion ist entscheidend für die Berechnung der Überlebenswahrscheinlichkeit jedes Chromosoms. Die bereits in Abschnitt 4.2 vorgestellte Variante der relativen Fitness ist die am häufigsten benutzte. Dem entsprechend sollte die Fitness nur positive Werte annehmen, da sonst die daraus berechnete Selektionswahrscheinlichkeit negativ werden würde. Das ist nicht immer gegeben, wenn man z.B. bei der Maximierung einer Funktion die Funktionswerte direkt als Fitness übernimmt. Dem entsprechend ist die am häufigsten verwendete Klasse von Fitnessfunktionen die der Skalierungsfunktionen, die ggf. vorhandene negative Bewertungen nach $\mathbb{R}^+$ abbilden.

Da die Fitness in direktem Zusammenhang mit der Überlebenschance steht, ist sie durch geschickte Skalierung auch nutzbar, um Probleme wie vorzeitige oder zu langsame Konvergenz zu vermeiden. Zu diesem Zweck wird von der linearen Fitnessfunktion abgewichen zugunsten einer Funktion mit populations- oder generationsabhängigen Exponenten, über die die Fitness künstlich vergrößert oder verkleinert werden kann. In diesen Fällen ist die Fitness eines Chromosoms nicht mehr statisch, sondern eine veränderbare Größe. Beispiele dafür sind Ansätze, bei denen am Anfang der Suchraum relativ gleichmäßig durchsucht werden soll (Exponent kleiner oder gleich 1), während sich die Suche in späteren Phasen auf die unmittelbare Umgebung der besten Lösungen konzentriert, wofür der Exponent größer als 1 sein sollte.

Neben den beiden bereits erwähnten Formen sollen in diesem Abschnitt auch noch die von mehreren Kriterien abhängige Fitness, die zwischen gleichartigen Chromosomen geteilte Fitness, und Fitnessfunktionen mit Berücksichtigung von Randbedingungen durch Strafpunkte behandelt werden.

6.3.1 Lineare Skalierte Fitness

Greffenstette und Baker stellten bereits 1989 die grundlegenden Skalierungsformen vor (hier für Maximierungsprobleme) [GB89]:

Definition 6.1 *Sei* $a \in \mathbb{R}^+$, $b \in \mathbb{R}$*:*

- *Lineare statische Skalierung:*
  $fit(c) = a \cdot fit^*(c) + b$
- *Lineare dynamische Skalierung:*
  $fit(c, P(t)) = a \cdot fit^*(c) - \min\{fit^*(c_i) \mid c_i \in P(t)\}$

Insbesondere durch das Abziehen der kleinsten in der Population vorgekommenen Bewertung werden die Unterschiede in der Fitness deutlich verstärkt, was das Entstehen von Superindividuals (Chromosomen mit weit über dem Durchschnitt liegender Fitness) begünstigen kann. Sie lässt sich jedoch gut verwenden, um negative Bewertungen zu verhindern. Dieses Verfahren kann sich jedoch ungünstig auf den Rechenaufwand auswirken, da die Bewertungen in jeder Generation für alle Chromosomen neu berechnet werden müssen.

Das Addieren des Faktors b in der ersten Variante kann ebenfalls zu Problemen führen, allerdings genau in der entgegengesetzten Weise. Hier gilt:

$$\text{Sei } b \gg \max_{c \in P(t)} \{fit^*(c)\} \Rightarrow \forall\, c \in P(t) :\ p(c) = \frac{fit^*(c) + b}{\sum_{c_i \in P(t)} (fit^*(c_i) + b)} \approx \frac{1}{\text{popsize}}$$

Das heißt, dass bei zu groß gewählter Konstante b die eigentliche Fitness der Chromosomen nur noch eine untergeordnete Rolle spielt, die Überlebenschance für gute Chromosomen also nicht wesentlich größer ist als für schlechte Chromosomen. Soll also ein Faktor b bei der Skalierung verwendet werden, so muss die Konstante immer kleiner als die ursprüngliche Fitness des gesuchten Optimums gewählt werden. Das setzt jedoch Wissen über das gesuchte Optimum voraus.

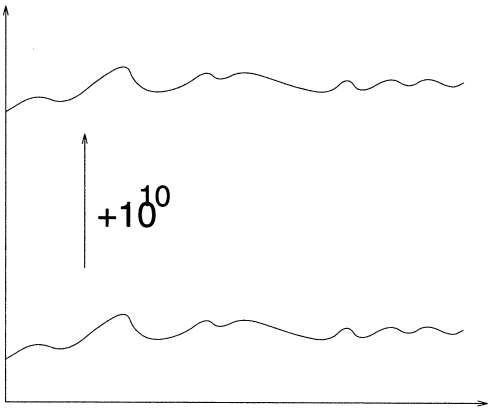


Abbildung 6.2 $\max fit^*$ und $\max(fit^* + b)$

6.3.2 σ-Skalierung

Die σ-Skalierung ist eine Erweiterung der linearen Skalierung um populationsabhängige Faktoren wie mittlere Fitness und Streuung. Ursprünglich entwickelt wurde sie von Forrest [For85], liegt aber inzwischen in verschiedenen Varianten auch von Goldberg [Gol89] und Michalewisz [Mic96] vor, wobei die beiden letztgenannten sich nur um ein Vorzeichen unterscheiden.

Definition 6.2 *Sei $a \in \mathbb{R}$, $\sigma(P(t))$ die Standardabweichung der Bewertung der Population. Die σ-Skalierung ist wie folgt definiert [Gol89]:*

$$fit(c, P(t)) = \begin{cases} fit^*(c) - (\overline{fit^*} - a \cdot \sigma(P(t))) & \text{falls } fit^*(c) \geq (\overline{fit^*} - a \cdot \sigma(P(t))) \\ 0 & sonst \end{cases}$$

Für die Variante von Michalewicz wird das erste Minus durch ein Plus ersetzt, die obere Bedingung reduziert sich auf $\overline{fit^*} \geq a \cdot \sigma(P(t))$. Schon durch die Definition wird klar, dass am Anfang alle Chromosomen gemäß ihrer Bewertung berücksichtigt werden, da praktisch keinem Chromosom die Fitness von Null zugewiesen wird. Je homogener die Population jedoch wird, desto mehr schlechte Chromosomen nehmen am Auswahlprozess nicht mehr teil. Daher ist der Vorteil dieser Form der Skalierung darin zu sehen, dass im Gegensatz zu den bisher betrachteten Skalierungsformen der Selektionsdruck mit der Homogenisierung der Population nicht zu stark sinkt. Ungünstig hingegen ist, dass dadurch ein gleitender Übergang in andere vielversprechende Bereiche des Suchraumes nach gewisser Zeit kaum noch möglich ist.

6.3.3 Exponentielle Skalierung

In eine ähnliche Richtung wie die σ-Skalierung zielt auch die exponentielle Skalierung. Sie wurde 1985 von Gillies als „Power Scaling" entwickelt [Gil85] und hatte entspre-

chend die einfache Form $fit = (fit^*)^k$ mit $k \in \mathbb{R}$. Die allgemeine Formulierung dieser Skalierungsart lautet jedoch:

Definition 6.3
Seien $a \in \mathbb{R}^+$, b, $k \in \mathbb{R}$ und $k(t,s) : \mathbb{Z} \times \mathbb{R} \to \mathbb{R}$ eine reellwertige Funktion. Man definiert folgende Skalierungen:

- *Exponentielle Skalierung [Gol89, GB89]:*

$$fit(c) = (a \cdot fit^*(c) + b)^k$$

- *Ungleichmäßige exponentielle Skalierung [Mic96]:*

$$fit(c) = (a \cdot fit^*(c) + b)^{k(t,s)}$$

Üblicherweise wird jedoch die einfache Form mit $a = 1$ und $b = 0$ verwendet. Bei festem k wurden gute Ergebnisse mit Werten knapp über 1 erreicht (1.005 bei Gillies [Gil85]). Auch hier ist der Hintergedanke, über die Dynamik von $k(t,s)$ den Effekt zu erreichen, dass bei sehr kleinem $k(t,s)$ (nahe 0) für kleine t in den ersten Generationen die Fitness nur eine geringe Rolle spielt, während sie sich bei monoton wachsendem $k(t)$ in späteren Generationen jedoch sehr stark ausprägt.

Die ungleichmäßige exponentielle Skalierung wurde von Michalewicz entwickelt, wobei er einen größeren Aufwand in die Entwicklung einer Formel für $k(t,s)$ steckte. Sie ist abhängig vom Populationsalter t und einem zusätzlichen, von der Anfangspopulation abhängigem Wert s_0, Spannweite genannt. Sie hat die folgende Form:

$$k(t,s_0) = \left(\frac{s^*}{s_0}\right)^{P_1} \left[\tan\left(\frac{t}{T+1} \times \frac{\pi}{2}\right)\right]^{P_2\left(\frac{s_0}{s^*}\right)^\alpha}$$

mit $s^* = 0.1$, $\alpha = 0.1$, $P_1 = 0.05$, $P_2 = 0.1$, $T =$ maximaler Anzahl von Generationen. Michalewicz hat ein spezielles Berechnungsverfahren für den Wert der Spannweite entwickelt, so dass s_0 dabei dem Quotienten aus der Streuung der Bewertungen der Anfangspopulation und der mittleren Bewertung aller Elemente der Anfangspopulation entspricht. Bild 6.3 zeigt die Oberfläche der Funktion $k(t,s_0)$ bei Verwendung der oben angegebenen Parameter.

Für ein fest gewähltes s_0 steigt die Kurve mit wachsendem t erst relativ stark, dann über einen längeren Zeitraum nur gering und dann extrem stark an. An dieser Form ist leicht zu erkennen, dass z.B. bei einer Wahl von s von ca. 0.1 die Fitness mit t die beabsichtigten Phasen „Unwichtig", „Einflussnehmend" und „Entscheidend" durchläuft.

Empirisch wurde festgestellt, dass $s_0 \approx 0.1$ ein gutes Verhältnis zwischen guter Konvergenz des genetischen Algorithmus und guter Durchforstung des Suchraums liefert. Weicht s_0 von diesem Wert ab, so kann versucht werden, den Wert durch Anpassung der Fitnessfunktion zu erreichen.

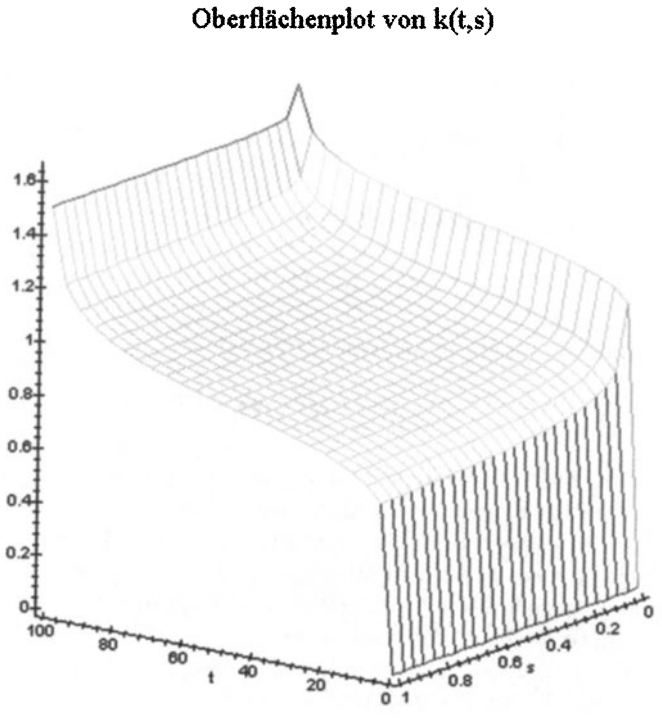


Abbildung 6.3 Darstellung der Fläche $k(t, s_0)$

6.3.4 Geteilte Fitness (Fitness Sharing)

Der Idee der geteilten Fitness hat ihren Ursprung im Problem der Superindividuals und der vorzeitigen Konvergenz. Um zu verhindern, dass eine Population sehr schnell von einem guten Individuum dominiert wird, wird für jedes Element der Population auf normale Weise die zugehörige Fitness berechnet, um dann in Abhängigkeit von der Ähnlichkeit zu den übrigen Chromosomen modifiziert zu werden. Die Ähnlichkeit wird als Abstand $d(c_1, c_2)$ berechnet (im binären Fall z.B. die Hammingdistanz) und durch eine „Sharing"-Funktion $Sh(d(c_1, c_2))$ in das Intervall $[0, 1]$ abgebildet. Je größer die Ähnlichkeit, desto höher sollte der Sharing-Wert sein. Die Fitness eines Chromosoms ergibt sich dann aus:

$$fit(c) = \frac{fit^*(c)}{\sum_{i=1}^{popsize} Sh(d(c, c_i))}$$

Den unter dem Bruchstrich stehenden Anteil bezeichnet man als „Nischen-Zähler" (Niche Count). Goldberg und Richardson schlugen als Sharingfunktion $Sh(.)$ [GR87] eine Funktion der folgenden Form vor:

$$Sh(d(c_1,c_2)) = \begin{cases} 1 - \left(\frac{d(c_1,c_2)}{\sigma_{Sh}}\right)^{\alpha} & \text{falls } d(c_1,c_2) < \sigma_{Sh} \\ 0 & \text{sonst} \end{cases}$$

mit zu wählenden Konstanten σ_{Sh} und α. Goldberg und Deb entwickelten auch Formeln zur Bestimmung eines passenden Wertes für σ_{Sh}, allerdings in Abhängigkeit von der Anzahl der zu erwartenden Optima. Da diese Zahl nur selten bekannt ist, bleibt die Wahl eines passenden Wertes für diese Konstante schwierig, da die Nischen auch über sehr unterschiedliche Form (Breite, Höhe) verfügen können. In der Literatur wird der Wert für α üblicherweise auf 1 gesetzt, so dass sich für die Sharing-Funktion eine dreieckige Form ergibt (bezeichnet als *Triangular Sharing Function*).

Durch die obige Konstruktion werden Suchbereiche genau dann gründlich durchforstet, wenn die dort vorhandenen Chromosomen von durchschnittlicher Fitness und das Gebiet in der Population nicht besonders häufig vertreten ist. Erfolg versprechende und bis dahin nicht untersuchte Bereiche des Suchraumes werden bei dieser Fitnessform also bevorzugt. Problematisch kann es werden, wenn eine Häufung von Chromosomen im Bereich um das Optimum stattgefunden hat, das Optimum aufgrund seines geringen Unterschieds zur Umgebung aber noch nicht erreicht wurde (mangelnder Selektionsdruck). Dieses Problem lässt sich durch die zusätzliche Verwendung einer populationsabhängigen exponentiellen Fitness in Grenzen halten.

Aufgrund der Notwendigkeit, jedes Element mit jedem anderen Chromosom zu vergleichen, kann dieser Algorithmus in Abhängigkeit vom verwendeten Ähnlichkeitsmaß sehr zeitaufwändig sein.

6.3.5 Mehrkriterienoptimierung

Bisher wurde immer der Fall betrachtet, bei dem das Optimum bezüglich eines einzigen Wertes gesucht wurde. Von dieser Form muss jedoch gerade bei Anwendungen häufig abgewichen werden zugunsten einer Mehrkriterienoptimierung. Hierbei setzt sich die Fitnessfunktion aus den zu k einzelnen Kriterien gehörenden Unterfunktionen zusammen, die so aufeinander abgestimmt sein müssen, dass sich daraus ergebende optimale Lösungen tatsächlich alle Aspekte berücksichtigen. Eine mittlere Bewertung bzgl. eines Kriteriums darf also keinen höheren Fitnesswert haben als die gute Bewertung eines anderen. Es gilt also:

$$fit(c) = \sum_{i=1}^{k} \alpha_i fit_i(c) \text{ mit}$$

$$\forall i \in \{1,\ldots,k\} \; : \; fit_i(c) \; : S \to [0,\infty)$$

Es gibt eine Vielzahl unterschiedlicher Varianten der Mehrkriterienoptimierung:

- Die Chromosomen werden jeweils vollständig von allen Fitnessfunktionen fit_i ausgewertet, jedoch nach unterschiedlichen Aspekten. Beispiel: Ein Traveling Salesman Problem, bei dem nicht nur die insgesamt zurückgelegte Strecke minimiert werden soll, sondern auch die Fahrtzeit, was aufgrund verschiedener Straßentypen zu einer völlig anderen Bewertung führen kann.

- Die Chromosomen können sich aus Teilketten zusammensetzen, die aus dem gleichen Suchraum stammen, jedoch von unterschiedlichen Fitnessfunktionen ausgewertet werden. Beispiel: Prozentuale Verteilung einer Anzahl herzustellender Gegenstände auf k Maschinen, wobei das Chromosom die Anteile jeder Maschine an der Arbeit enthält. Die Fitnessfunktion kann beispielsweise unterschiedliche Rüst- und Reparaturkosten berücksichtigen.

- Die Chromosomen setzten sich aus Teilketten zusammen, die jeweils aus einem speziellen Suchraum stammen und deren zugehörige Fitnessfunktion unabhängig voneinander sind.

Beispiel 6.3 (Optimierung eines PC-Kaufs) Die Fitnessfunktion setzt sich aus drei Gütekriterien mit unterschiedlichem Definitions- und Wertebereichen zusammen. Um daraus eine Gesamt-Fitnessfunktion zu bekommen, müssen die Kriterien gewichtet und ggf. genormt werden.

	Gütekriterium		
PC	Euro (Preis)	GHz (Geschwindigkeit)	GB (Platte)
c_1	1000	2.0	60
c_2	1500	2.8	120
c_3	1100	2.4	80

Da der Preis ein „negatives" Kriterium darstellt, während die beiden anderen „positive" Kriterien sind, muss der Wert des ersten Kriteriums vom Maximum der Bewertungen abgezogen werden. Für das zweite und dritte Kriterium sollten geeignete Gewichte so gewählt werden, dass sie annähernd gleichviel zur Gesamtfitness beitragen. Ein Beispiel für eine Gesamt-Fitnessfunktion ist:

$$fit(c_k) = (\max_{i=1,\ldots,3}\{Euro(c_i)\} - Euro(c_k)) \quad + \quad 100 \cdot GHz(c_k) \quad + \quad 5 \cdot GB(c_k)$$

Mit dieser Fitnessfunktion ergibt sich dann für die PC-Angebote 1, 2 und 3 eine Fitness 1000, 880 bzw. 1040. Ob dies die tatsächlichen Präferenzen des Käufers widerspiegelt ist allerdings fraglich. Problematisch ist hier die Auswahl der letztendlich alles entscheidenden Konstanten vor den Fitnesswerten für die einzelnen Kriterien. Vorteilhaft an der Verwendung eine gewichteten Summer über die Einzel-Fitnesswerte ist die Möglichkeit, Kriterien nach ihrer Wichtigkeit zu bewerten, wenn eine sinnvolle Gewichtung bekannt ist.

Bei diesem Beispiel wird auch deutlich, dass es keine optimale Lösung geben muss, die den anderen in jedem Bereich überlegen ist.

Definition 6.4 *Gegeben seien r Gütekriterien in Form von totalen Ordnungen $\leq_i$, $i = 1, \ldots, r$, auf dem Suchraum $\mathcal{S}$. Dann heißt bedeutet die Schreibweise $c_1 \leq_i c_2$, dass c_1 das Gütekriterium i mindestens genauso gut wie c_2 erfüllt. $c_1 \in \mathcal{S}$* dominiert *$c_2 \in \mathcal{S}$ (in Zeichen $c_1 <_p c_2$) (*echt*), wenn gilt:*

$$\Leftrightarrow \forall i \in \{1, \ldots, r\} : c_1 \leq_i c_2 \wedge \exists i_0 \in \{1, \ldots, r\} : c_1 <_{i_0} c_2$$

$c_1 \in \mathcal{S}$ heißt pareto-optimal, *wenn c_1 von keinem $c \in \mathcal{S}$ (echt) dominiert wird.*

Im vorhergehendem Beispiel wären die Lösungen alle pareto-optimal, da keine die anderen in wirklich allen Punkten bzw. allen Kriterien dominiert. Wenn bei einer gewichteten Summe der Einzelfitnesswerte als Gesamtfitness alle Gewichte positiv sind, dann ist eine bezüglich der Gesamtfitness optimale Lösung auch pareto-optimal.

Bei der Mehrkriterienoptimierung gibt es noch verschiedene andere Ansätze zur Auswahl der nächsten Generation als die Verwendung einer Gesamtfitness, z.B.:

Geteilte Selektion:
Hierbei wird die Selektion auf die verschiedenen Kriterien aufgeteilt, d.h. es werden bei r Kriterien jeweils $\frac{popsize}{r}$ Chromosomen nach fit_i, $(i = 1, \ldots, r)$ ausgewählt. Nachteil dieses Verfahrens ist, dass es Lösungen benachteiligt, die einen „Trade-Off" zwischen verschiedenen Kriterien beinhalten. Es werden also extreme Lösungen bevorzugt, die ein einzelnes Kriterium optimieren. Lösungen, die alle Kriterien relativ gut, aber keines optimal erfüllen, erhalten geringere Überlebenschancen.

Beispiel 6.4 Wir betrachten als Suchraum $\mathcal{S} = R$ und zwei Gütekriterien: $G_1(t) = t^2$, $G_2(t) = (t-2)^2$. Gesucht ist $t \in R$, so dass $G_1(t)$ und $G_2(t)$ möglichst klein sind.

Je weiter links und je weiter unten sich ein Punkt der Funktion in der Abbildung 6.4 befindet, desto besser ist er.

Das obige Verfahren würde vor allem die Extremlösungen $t = 0$ und $t = 2$ bevorzugen; die ebenfalls Pareto-optimalen Lösungen $0 < t < 2$ würden benachteiligt.

Bemerkung: Eine Fitnessfunktion der Form $\sum \alpha_i fit_i$ liefert immer eine pareto-optimale Lösung.

Wenn man bei der Mehrkriterienoptimierung nicht eine gewichtete Gesamtfitnessfunktion verwenden möchte, sondern allgemein pareto-optimale Lösung sucht, ist der Selektionsmechanismus rechenaufwendiger, da zur Feststellung von Pareto-Optimalität bzw. Dominanz von Lösungen viele paarweise Vergleiche von Chromosomen durchgeführt werden müssen. Durch geeignete Algorithmen lässt sich aber auch hier die Komplexität in Grenzen halten [Jen03]. Vergleichende Übersichten zur Performanz verschiedener Ansätze zur Mehrkriterienoptimierung geben [ZT99, ZTL+03]. Einen zusammenfassende Übersicht zur Mehrkriterienoptimierung mittels evolutionärer Algorithmen gibt das Buch von Deb [Deb01].

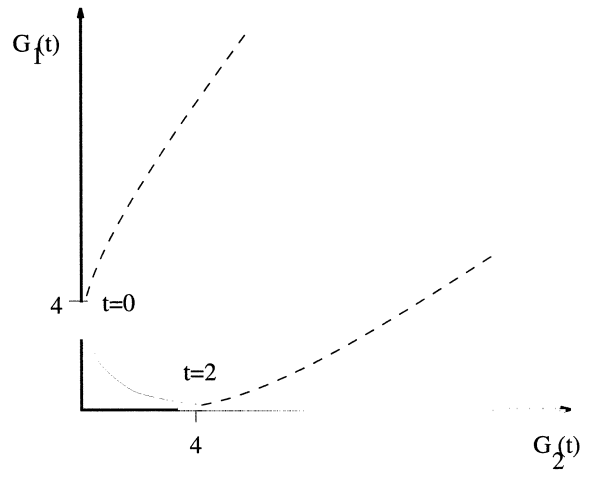


Abbildung 6.4 Pareto-optimale Lösungen ($0 \leq t \leq 2$)

Rangbasierte Fitness ist die Fitnessberechnung auf der Grundlage von *Ranking:* (siehe Abschnitt 6.4.4). Dafür wird die Population $P(t)$ mit Hilfe der totalen Ordnungen sortiert. Den Chromosomen, die nicht von anderen dominiert werden, wird der Rang 1 zugewiesen. Danach werden sie aus der ursprünglichen Population entfernt und in die Menge P_1 übernommen, wobei die Population $P(t)$ um diese Menge verringert wird ($P'(t) := P(t) - P_1$). Auf diese verkleinerte Population wird dasselbe Verfahren angewendet wie vorher auf ganz $P(t)$, jedoch wird den dominierenden Chromosomen der Rang zwei zugewiesen (Menge P_2). Mit diesem System wird fortgefahren, bis allen Chromosomen ein Rang zugeordnet wurde. Mit Hilfe der im folgendem Kapitel vorgestellten Selektion durch Ranking lässt sich dann die nächste Population erstellen.

6.3.6 Fitnessfunktion mit Zusatzbedingungen

Sehr häufig handelt es sich beim untersuchten Problem um eine Form der Optimierung, bei der eine Reihe von Nebenbedingungen zu berücksichtigen sind. Hierbei sollte unterschieden werden zwischen **Randbedingungen**, bei denen nur überprüft werden muss, ob sie erfüllt sind oder nicht (z.B. TSP), und zwischen **Nebenbedingungen** in Form von Gleichungen und Ungleichungen. In beiden Fällen bedeuten zusätzliche Bedingungen, dass der Lösungsraum von „Sperrgebieten" durchzogen ist, die nicht betreten werden dürfen: Also: $\mathcal{S} = \mathcal{S}_{\text{Korrekt}} \cup \mathcal{S}_{\text{Verboten}}$.

Aus Bild 6.5 wird deutlich, dass es unter diesen Umständen sehr problematisch ist, die Population vom Startpunkt aus langsam in die erfolgversprechenden Regionen driften zu lassen, da sie ggf. an den Barrieren nicht vorbeikommt. Das gilt insbesondere bei der Verwendung von Verfahren, bei denen die Anzahl der Mutationen pro Chromosom auf Eins beschränkt wurde (siehe Abschnitt 7.4.2), da die verschiedenen

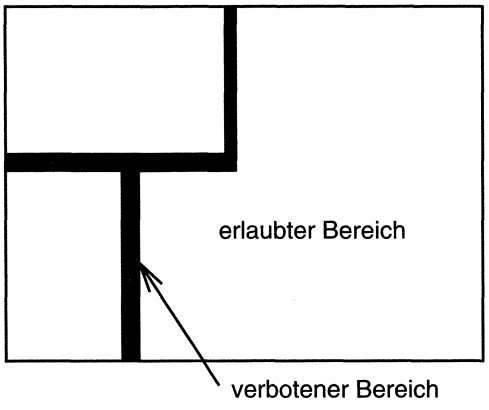


Abbildung 6.5 Beispiel für einen nicht-zusammenhängenden Suchraum.

Lösungsregionen meistens durch die Werte mehrerer Gene getrennt sind. Eine zusätzliche Schwierigkeit bedeuten derartige Sperrgebiete für die Kodierung, da nicht zulässige Lösungen vermieden werden müssen.

Die Benutzung so genannter Reparaturmechanismen (siehe Abschnitt 6.5) nach Anwendung der genetischen Operatoren ist eine Möglichkeit, unzulässige Lösungen zu vermeiden. Eine andere Möglichkeit, das Problem zu umgehen, ist die Zulassung der verbotenen Bereiche unter Zuweisung eines angemessenen *Strafwertes*, der so genannten *Penalty*. Hierfür gibt es zwei Möglichkeiten:

1. Fester Strafwert für jede Verletzung einer Nebenbedingung, unabhängig von dem Ausmaß des Fehlers.

2. Variabler Strafwert in Abhängigkeit vom Ausmaß des Fehlers.

Der Nachteil der ersten Methode liegt darin, dass der Übergang zwischen korrekten und falschen Lösungen nicht fließend ist. Andererseits kann man so nicht in eine Situation kommen, in der fehlerhafte Lösungen bessere Bewertungen aufweisen als korrekte Lösungen, was leicht möglich ist, wenn das Optimum nahe an einem durch eine Randbedingung gesperrten Bereich liegt.

In jedem Fall muss eine Straffunktion $pen : \mathcal{S} \rightarrow \mathbb{R}_0^+$ definiert werden. Der Wert $pen(c)$ wird als *Penalty* des Chromosoms c bezeichnet. Außerdem sollte pen so gewählt werden, dass gilt:

$pen(c) = 0 \Longleftrightarrow c$ ist eine korrekte Lösung (erfüllt die Randbedingung).

Beispiel 6.5 (Rucksackproblem) Ein Beispiel für ein Problem mit Nebenbedingung ist das Rucksackproblem, bei dem Gegenstände in einen Behälter mit festem Volumen gepackt werden sollen. Optimiert werden soll der Füllungsgrad des Behälters durch eine Auswahl der möglichen Gegenstände. Zu prüfen ist die Nebenbedingung, ob bei der Auswahl der Gegenstände das zur Verfügung stehende Volumen überschritten wurde. Als Penalty kann ein fester Wert (fester Strafwert) oder das überzählige Gewicht (variabler Strafwert) verwendet werden.

Beispiel 6.6 (Routenfindungsproblem) Das Beispiel 4.2, bei dem der schnellste Weg gefunden werden sollte, modifizieren wir, indem wir annehmen, dass einige Teilgebiete nicht nur langsam, sondern überhaupt nicht durchquert werden dürfen. Die Penalty-Funktion kann die folgende Form haben:

Fester Strafwert: $pen(c)$ = Anzahl der trotz des Verbotes durchquerten verbotenen Felder beim Chromosom c multipliziert mit einem Gewichtungsfaktor w.

Variabler Strafwert: $pen(c)$ = Länge der in verbotenen Bereichen zurückgelegten Strecke multipliziert mit einem Gewichtungsfaktor w.

In beiden Fällen handelt es sich um Randbedingungen und nicht um Nebenbedingungen in Form von Funktionen.

Beispiel 6.7 (Travelling Salesman Problem (TSP)) Bei einer nicht-binären Kodierung in der Form, dass das i-te Gen die Nummer der an der i-ten Stelle der Rundreise zu besuchenden Stadt enthält, kann es zu nicht oder doppelt besuchten Städten kommen. Es entsprechen also nur Chromosomen mit n verschiedenen Allelen einer Rundreise. Mögliche Penalty-Funktionen sind:

Fester Strafwert: $pen(c) = n - \text{card}(\{\text{Stadt} \mid \text{Stadt wird bei Reise } c \text{ besucht }\}) \cdot w$. Dabei bezeichnet $\text{card}(M)$ die Kardinalität der Menge M, d.h. bei einer endlichen Menge die Anzahl der Elemente.

Variabler Strafwert: $pen(c) = \text{card}(\{\text{Stadt} \mid \text{Stadt wird bei Reise } c \text{ nicht besucht}\}) \cdot \max_{\{c_i \in P(t)\}} \{fit(c_i), c_i \text{ vollständig}\}/l$

Mit der Definition des variablen Strafwertes soll vermieden werden, dass unvollständige Rundtouren besser abschneiden als vollständige. Voraussetzung ist natürlich, dass wenigstens eine korrekte Route existiert. Ansonsten muss eine Konstante gesetzt werden.

Definition 6.5 (Allgemeiner Lösungsansatz bei Randbedingungen)
Im Falle der Optimierung mit Randbedingungen hat die Fitnessfunktion $fit(c)$ die folgende Form:

$$fit(c) = fit^*(c) + w(t) \cdot \max\{0, pen(c)\}$$

wobei w(t) eine populationsabhängige Gewichtungsfunktion ist.

Das Problem bei dieser Art Gewichtungsfunktion ist das Auffinden des korrekten Verhältnisses $w(t)$ zwischen $pen(c)$ und der ursprünglichen Fitness. Die Bestrafungsfunktion muss also einerseits das Durchwandern eigentlich nicht zulässiger Gebiete ermöglichen, andererseits aber verhindern, dass in diesen Bereichen liegende Lösungen favorisiert werden. Ist $w(t)$ zu groß, spielt $fit(c)$ fast keine Rolle mehr, sondern es wird irgendeine (schlechte) korrekte Lösung geliefert. Ist $w(t)$ zu klein, dann werden Verstöße gegen Nebenbedingungen nur gering geahndet; es wird eine gute, inkorrekte Lösung gefunden. Häufig tritt auch die Schwierigkeit auf, dass bei genügend großem $w(t)$ und einer Initialpopulation ohne, oder nur mit wenigen unzulässigen Lösungen, sich die zuerst gefundenen schlechten zulässigen Lösungen schnell gegenüber den unzulässigen Lösungen durchsetzen. Dies führt zu vorzeitiger Konvergenz.

Günstig ist es daher, $w(t)$ als eine zeit- bzw. populationsabhängige monoton steigende Gewichtungsfunktion zu implementieren. $w(t)$ kann dabei z.B. die Form $w(t) = w \cdot t/T$ annehmen, wobei w eine Konstante ist und es sich bei T um die maximale Anzahl von Populationen handelt.

Der Ansatz im Falle von Nebenbedingungsfunktionen sieht ein wenig anders aus, da sich die Werte für die Funktionen problemlos direkt für die Penalty-Funktion verwenden lassen und sich die Frage der Verhältnismäßigkeit zur eigentlich zu optimierenden Funktion nicht mehr stellt. Hierzu hat Michalewicz [Mic96] eine Definition geliefert:

Definition 6.6 (Allgemeiner Lösungsansatz bei Nebenbedingungen)
Es seien Nebenbedingungen in Form von Ungleichungen $g_i \leq 0$ $(i = 1, \ldots, q)$ und Gleichungen $h_j = 0$ $(j = q+1, \ldots, m)$ gegeben. Außerdem sei

$$\forall j \in \{1, \ldots, m\}: \quad f_j := \begin{cases} \max\{0, g_j(c)\} & \text{falls} \quad j \leq q \\ |h_j(c)| & \quad\quad\; j > q \end{cases}$$

Dann lässt sich die Fitness wie folgt definieren:

$$fit(c) = \begin{cases} fit^*(c) & \text{falls } c \text{ eine korrekte Lösung ist} \\ fit^*(c) + pen(c) & \text{sonst.} \end{cases}$$

Dabei ist pen eine Straffunktion zur Berücksichtigung der Nebenbedingungen, die von den Werten f_j abhängt.

Darüber hinaus wurde von Michalewicz das Tool GENOCOP (GEnetic algorithms for Numerical Optimization of Constrained Problems [MN95]) entwickelt, dass sowohl bei linearen, wie auch bei nicht-linearen Gleichungen und Ungleichungen verwendet werden kann (siehe Abschnitt 7.4.3). Joines und Houck [JH94] präsentieren eine Variante, die weitestgehend der Definition entspricht. Der Penalty-Faktor ist dabei von der Form:

$$pen(c) = (C \cdot t)^\alpha \sum_{j=1}^{m} f_j^\beta(c)$$

Etwas anders definiert sind die zu den Ungleichungen gehörenden f_j, die analog zu den Gleichungen über den Betrag definiert wurden.

Einen speziellen Lösungsansätze mit Nebenbedingungen findet man bei Schoenauer und Xanthakis [SX93], in deren Variante allerdings die Nebenbedingungen sukzessive überprüft werden. Sobald die Population die vorhergehende Bedingung mit einem festgelegten Prozentsatz erfüllt, wird zur nächsten Bedingung übergegangen. Chromosomen, die bereits abgearbeitete Bedingung nicht erfüllen, werden entfernt.

6.4 Selektion

Zusammen mit der Fitness gehört das verwendete Selektionsverfahren zu den entscheidenden Faktoren für den generellen Verlauf der Konvergenz innerhalb der Population. Dem entsprechend viele unterschiedliche Verfahren wurden entwickelt, von denen sich jedoch nur relativ wenig durchsetzen konnten. Sie sollen zusammen mit von Bäck und Hoffmeister ([BH91], [Bäc96]) entwickelten Klassifizierungen vorgestellt werden.

Die bisher verwendete Methode der Rouletterad-Selektion ist in der Literatur auch unter dem Namen „Stochastic Sampling with Replacement" bekannt. Einer der größten Nachteile dieser Methode liegt in der ggf. sehr großen Diskrepanz zwischen der erwarteten Anzahl von Kopien, die in die nächste Generation übernommen werden, und der tatsächlich gezogenen Anzahl. Unter ungünstigen Umständen kann es vorkommen, dass das beste Chromosom gar nicht überlebt (siehe 5.2). Hinzu kommt, dass der Selektionsdruck geringer ist als bei anderen Verfahren. Andererseits ist wohl kein anderes Verfahren so intensiv theoretisch untersucht worden wie dieses (siehe Schema-Theorem).

Für die folgenden Klassifizierungen soll der Einfachheit halber davon ausgegangen werden, dass die Chromosomen gemäß ihrer Fitness absteigend sortiert vorliegen, das beste Chromosom also die Nummer 1 trägt. Eine mögliche Einteilung verwendet die Selektionswahrscheinlichkeit der Chromosomen als Einteilungskriterium in statische und dynamische Selektion. Als statisch gilt eine Selektion dann, wenn die Selektionswahrscheinlichkeit nicht vom Fitnesswert sondern nur vom Rang abhängt.

Definition 6.7 (statische Selektion) *Eine Selektionsfunktion bezeichnet man als* statisch, *wenn gilt:*

$$\forall i \in \{1, \ldots, popsize\} \forall t \geq 0 \ : \ p_t(c_i) = K_i \text{ mit } K_i \text{ konstant}$$

Bei einem dynamischen Selektionsverfahren variiert wenigstens einmal über die Zeit die Selektionswahrscheinlichkeit für ein Chromosom. Also:

Definition 6.8 (dynamische Selektion) *Eine Selektionsfunktion wird als* dynamisch *bezeichnet, wenn gilt:*

$$\exists i \in \{1, \ldots, popsize\} \ : \ \neg \ (\forall t \geq 0 \ : \ p_t(c_i) = K_i) \text{ mit } K_i \text{ konstant}$$

In diesem Sinne handelt es sich bei der Rouletterad Selektion um ein dynamisches Selektionsverfahren. Es werden im folgenden auch statische Verfahren vorgestellt. Typisch für dynamische Verfahren ist es, dass die Selektionswahrscheinlichkeit in jeder Generation aufs neue ausgerechnet wird, während bei statischen Verfahren nur eine Berechnung am Anfang durchgeführt werden muss.

Eine andere Form der Aufteilung orientiert sich direkt an der Überlebenswahrscheinlichkeit. Hierbei wird unterschieden zwischen *diskriminierender* und *nicht diskriminierender* Selektion. Bei einer diskriminierenden Selektion wird für einige Chromosomen die Überlebenswahrscheinlichkeit auf Null gesetzt. Das macht insbesondere dann Sinn, wenn schon aus der Höhe der Bewertung geschlossen werden kann, dass diese Chromosomen keine Chance haben, sich oder andere zu verbessern.

Definition 6.9 (diskriminierende Selektion) *Ein Selektionsverfahren wird als* diskriminierend *bezeichnet, wenn gilt:*

$$\exists \, i \in \{1, \ldots, popsize\} \, \forall \, t \geq 0 \ : \ fit(c_i(t)) > 0 \ \wedge p_t(c_i(t)) = 0$$

Durch die Anwendung einer diskriminierenden Selektion kann der Selektionsdruck stark erhöht werden, da weniger Chromosomen am Selektionsvorgang teilnehmen und die Auswahlwahrscheinlichkeiten die Qualitätsunterschiede besser widerspiegeln. Eine nicht diskriminierende Selektion hingegen durchsucht den Lösungsraum gründlicher, bevor ein vielversprechender Bereich gewählt wird.

Definition 6.10 (nicht diskriminierende Selektion) *Ein Selektionsverfahren wird als* nicht diskriminierend *bezeichnet, wenn gilt:*

$$\forall\, i \in \{1, \ldots, popsize\}\, \forall\, t \geq 0 \;:\; fit(c_i(t)) > 0 \;\Rightarrow\; p_t(c_i(t)) > 0$$

Üblicherweise werden im Falle einer diskriminierenden Selektion die Auswahlwahrscheinlichkeiten für die schlechtesten Chromosomen auf Null gesetzt. Es ist jedoch genauso gut möglich, dieses Vorgehen auf die besten Chromosomen anzuwenden, um vorzeitige Konvergenz zu verhindern. Je nachdem, welche dieser Vorgehensweisen gewählt wird, bezeichnet man die Selektion als links- (beste Chromosomen werden ausgeschlossen) bzw. rechtsseitige Selektion (schlechteste Chromosomen werden ausgeschlossen). An dieser Stelle muss bemerkt werden, dass in den Originalversionen der obigen Definitionen [BH91] keine Aussagen über die Fitness der betrachteten Chromosomen gemacht wurde, da generell von positiven Fitnesswerten ausgegangen wurden. In diesem Sinne wurden die Definitionen so angepasst, dass nur bei zwangsweiser Zuweisung einer Selektionswahrscheinlichkeit von Null von diskriminierender Wahrscheinlichkeit gesprochen wird.

Eine zusätzliche Unterscheidungsmöglichkeit ist die in Elite- und reine Selektion. Bei Eliteselektion handelt es sich um alle Selektionsalgorithmen, bei denen die k, $(k > 0)$ besten Chromosomen unverändert in die neue Population übernommen werden. Dem entsprechend wird bei reiner Selektion der gesamte Auswahlvorgang vom Zufall gesteuert.

Wichtige Merkmale zur Beurteilung der Qualität eines Selektionsverfahrens sind die Abweichung zwischen der Anzahl der tatsächlich in die nächste Generation übernommenen Kopien eines Chromosoms und der erwarteten Anzahl von Kopien („Bias"), sowie der Spannweite in der Anzahl der übernommenen Kopien („Spread").

6.4.1 Remainder Stochastic Sampling

Von dieser Selektionsart gibt es zwei Varianten, nämlich „Remainder Stochastic Sampling with/without Replacement". Die ersten Schritte der beiden Auswahlalgorithmen sind identisch. Es wird für jedes Chromosom auf die übliche Weise die Selektionswahrscheinlichkeit $p(c) = fit(c)/\sum_{i=1}^{popsize} fit(c_i)$ berechnet. Für jedes Chromosom $c \in P(t)$ werden dann genau $\lfloor p(c) \cdot popsize \rfloor$ Chromosomen direkt in $P^*(t)$ einfügt. Hierbei steht $\lfloor r \rfloor$ für die größte ganze Zahl, die kleiner oder gleich r ist. Danach werden die Auswahlwahrscheinlichkeiten wie folgt modifiziert:

Definiere $\widetilde{fit}(c) := p(c) \cdot popsize - \lfloor p(c) \cdot popsize \rfloor$. Die fehlenden Chromosomen werden im Fall „With Replacement" nach Berechnung neuer Selektionswahrscheinlichkeiten $\widetilde{p(t)}$ aus $\widetilde{fit}(t)$ mit Rouletterad-Selektion ausgewählt. Die Werte $\widetilde{fit}(c)$ werden bei „Without Replacement" direkt als Auswahlwahrscheinlichkeiten verwendet. Es wird der Reihe nach (nach Fitness sortiert) für jedes Chromosom ausgewürfelt, ob es in die nächste Generation übernommen wird.

Bei diesen Selektionsformen handelt es sich um dynamische, nicht diskriminierende Eliteverfahren, da die Selektionswahrscheinlichkeiten sich mit jeder Generation ändern, keine Chromosomen künstlich eine Auswahlwahrscheinlichkeit gleich Null erhalten und die besten Chromosomen immer überleben. Bias und Spread dieser Verfahren sind deutlich geringer als bei der normalen Rouletterad-Selektion, da eine Mindestanzahl von Kopien in Abhängigkeit von der Fitness immer ausgewählt wird.

Beispiel 6.8 (Remainder Stochastic Sampling with Replacement)
Wir betrachten eine Population mit $popsize = 6$ Chromosomen und den Selektionswahrscheinlichkeiten $p(c)$ für die Rouletterad-Selektion. Es werden zunächst so viele Kopien von jedem Chromosom übernommen, wie in der vierten Spalten angegeben sind. Die verbleibenden zwei noch zu selektierenden Chromosomen werden nach der Verteilung in der letzten Spalte zufällig ausgewählt.

$P(t)$	$p(c)$	$p(c) \cdot popsize$	$\lfloor p(c) \cdot popsize \rfloor$	$\widetilde{fit}(c)$	$\widetilde{p}(c)$
c_1	$\frac{9}{24}$	$2 + \frac{1}{4}$	2	$\frac{1}{4}$	$\frac{1}{8}$
c_2	$\frac{2}{24}$	$\frac{2}{4}$	0	$\frac{2}{4}$	$\frac{2}{8}$
c_3	$\frac{3}{24}$	$\frac{3}{4}$	0	$\frac{3}{4}$	$\frac{3}{8}$
c_4	$\frac{4}{24}$	1	1	0	0
c_5	0	0	0	0	0
c_6	$\frac{6}{24}$	$1 + \frac{2}{4}$	1	$\frac{2}{4}$	$\frac{2}{8}$

Automatisch werden in diesem Beispiel die Chromosomen c_1 zweimal, c_4 und c_6 einmal übernommen. Die zwei fehlenden Chromosomen werden abhängig von der Restwahrscheinlichkeit bestimmt. Für $\widetilde{fit}$ ergibt sich ein Wert von 2. Dividieren der Bewertungen $\widetilde{fit}(c_i)$ durch $\widetilde{fit}$ führt dann zu den Auswahlwahrscheinlichkeiten $\widetilde{p}(c_i)$ in der letzten Spalte.

6.4.2 Stochastic Universal Sampling

Das *Stochastic Universal Sampling-Verfahren* ist ein inzwischen recht beliebtes Verfahren, das sich insbesondere durch einen Bias-Wert von 0 und einen minimalen Spread auszeichnet [Bak87]. Es wird wie beim „Stochstic Sampling with Replacement" mit einem Rouletterad gearbeitet, auf dem jedem Chromosom ein Abschnitt in einer Größe proportional zur Fitness zugewiesen wird. Statt es jedoch $popsize$-mal zu drehen und jeweils ein Chromosom zu bestimmen, werden $popsize$ Zeiger in gleichem Abstand rund um das Rouletterad angebracht. Nach dem einmaligen Drehen des Rades wird für jeden Zeiger dasjenige Chromosom in die nächste Generation übernommen, auf dessen Feld der Zeiger zeigt.

Über die Anzahl der zu erwartenden Kopien lässt sich sehr einfach die folgende Aussage machen: Es werden von jedem Chromosom wenigstens $\lfloor p(t) \cdot popsize \rfloor$ und höchstens $\lceil p(t) \cdot popsize \rceil$ (kleinster darüber liegender ganzzahliger Wert) Kopien in die nächste Generation übernommen.

Auch bei dieser Selektionsform handelt es sich um eine dynamische, nicht diskriminierende Eliteselektion. Durch die gleichmäßige Anordnung der Zeiger um das Rouletterad herum ist sichergestellt, dass jedes Chromosom mit einer wenigstens durchschnittlichen Bewertung wenigstens eine Kopie für die neue Generation bereitstellt. Dadurch überlebt das beste Chromosom den Auswahlvorgang auf jeden Fall.

Man benötigt hier außerdem für die Selektion der gesamten Population nur eine einzige (Pseudo-)Zufallszahl, egal wie groß $popsize$ ist.

6.4.3 Boltzmann Selektion

Der Grundgedanke der Boltzmann Selektion ähnelt sehr stark dem Prinzip des Simulated Annealing. Auch hier ist eine Temperaturvariable in die Formel zur Bestimmung der Selektionswahrscheinlichkeit einbezogen worden, die mit zunehmender Generationszahl von einem anfänglich sehr hohem Wert immer weiter absinkt und dadurch den Selektionsdruck erhöht. Dieses Verfahren ist vergleichbar mit einigen der bereits im vorhergehenden Abschnitt vorgestellten Skalierungsverfahren für Fitnessfunktionen. Ein Beispiel für die Implementierung einer derartigen Selektionswahrscheinlichkeit ist :

$$p(c_i) = \frac{e^{fit(i)/T}}{\sum_{j=1}^{popsize} e^{fit(c_j)/T}}$$

T steht dabei für die „Temperatur". Die Temperatur sinkt hierbei gemäß eines vorher festgelegten Abkühlungsplans stufenweise ab. Mit sinkender Temperatur werden die Unterschiede zwischen Chromosomen mit hoher und niedriger Fitness schnell größer. Gemäß der vorher gegebenen Einteilung handelt es sich um ein dynamisches, nicht diskriminierendes reines Selektionsverfahren, da das beste Chromosom zwar die größte Überlebenswahrscheinlichkeit, aber keine Überlebensgarantie hat.

6.4.4 Lineares Ranking

Auch bei diesem Verfahren wird versucht, die vorzeitige Konvergenz gegen ein Suboptimum zu verhindern. Statt jedoch – wie zuvor – direkt Bezug auf die Fitness zu nehmen, geht sie bei diesem Verfahren nur indirekt ein. Es wird wie folgt vorgegangen:

Die Chromosomen werden auf der Basis der Fitness geordnet. Das Chromosom mit der größten (besten) Fitness erhält den Rang 1, das nächste den Rang 2 und so fort. Die Selektionswahrscheinlichkeit eines Chromosoms wird über eine lineare von seinem Rang abhängige Funktion berechnet. Dafür wird vorher festgelegt, wie hoch die Erwartungswerte E_{max} und E_{min} für die Anzahl der Kopien für das beste, bzw. das schlechteste Chromosom sein sollen. Über die folgende Formel lässt sich damit die Wahrscheinlichkeit für jedes Chromosom berechnen:

$$p(c_i) = \frac{1}{popsize} \cdot \left(E_{max} - (E_{max} - E_{min}) \cdot \frac{i-1}{popsize - 1} \right)$$

Der Wert für E_{max} muss im Bereich von $1 \leq E_{max} \leq 2$ liegen. Der Erwartungswert für das schlechteste Chromosom muss dann mit $2 - E_{max}$ gewählt werden, da die Nebenbedingungen

$$\begin{aligned} \forall\, i \in \{1, \ldots, popsize\} : p(c_i) &\geq 0 \text{ und} \\ \sum_{i=1}^{popsize} p(c_i) &= 1 \end{aligned}$$

berücksichtigt werden müssen. Auch bei diesem Verfahren handelt es sich wieder um eine reine Selektionsform, die jedoch diesmal statisch ist. Diskriminierend wäre sie nur für den Fall, dass E_{max} auf 2 und damit E_{min} auf 0 gesetzt wird. Spread und Bias sind relativ hoch, da nicht sichergestellt wird, dass sich die Anzahl der tatsächlich in die nächste Population übernommenen Individuen zwischen nahe zusammenliegenden minimal, bzw. maximal erlaubten Anzahlen bewegt.

Es sind kaum theoretische Resultate für das Ranking bekannt, zumal es normalerweise günstiger ist, die durch den Fitnesswert repräsentierte Information über die Qualität des Chromosoms zu nutzen. Nichtsdestotrotz hat dieses Verfahren den Vorteil, dass es wesentlich schneller ist als dynamische Verfahren, da die Selektionswahrscheinlichkeiten nur einmal berechnet werden müssen. Die Übernahmezeit dieses Verfahrens hat im allgemeinen Fall ($E_{max} < 2$) die Ordnung $O(popsize \cdot \ln(popsize))$ [GD91].

6.4.5 Wettkampf-Selektion

Bei der so genannten *q-Wettkampfselektion* (*tournament selection*) werden zufällig, also mit von der Fitness unabhängiger Wahrscheinlichkeit, q Chromosomen aus der Population ausgewählt. Das beste Chromosom wird in die neue Population übernommen. Das Verfahren wird $popsize$-mal durchgeführt, wobei bereits gewählte Chromosomen wieder in die Ursprungspopulation eingesetzt werden. Wie zuvor bei der rangbasierten Selektion entspricht der Index eines Chromosoms dem Platz in der fitnessbasierten Reihenfolge.

Bäck [Bäc96] gibt sowohl eine Formel für die Selektionswahrscheinlichkeit bei beliebigem ganzzahligem q als auch den zugehörigen Beweis an:

$$p_i = \frac{1}{popsize^q} \cdot ((popsize - i + 1)^q - (popsize - i)^q)$$

Auch für dieses Selektionsverfahren haben Goldberg und Deb [GD91] die Übernahmezeit berechnet und erhielten dafür einen Wert von ca.

$$\approx \frac{1}{\ln q} \cdot (\ln popsize + \ln(\ln popsize)).$$

Über den Wert von q und die Populationsgröße $popsize$ lässt sich der Selektionsdruck relativ gut steuern. Bei steigendem q verringert sich die Übernahmezeit, der Selektionsdruck wächst demnach. Der umgekehrte Effekt tritt bei einer Erhöhung der Populationsgröße auf.

Von dieser Selektionsform gibt es in der Literatur unterschiedliche Varianten. Bei Mitchell [Mit96] findet sich für den Binärfall eine Form, bei der das beste Chromosom nur mit einer gewissen Wahrscheinlichkeit ausgewählt wird. Diese binäre Variante wird als *Zweikampfselektion* bezeichnet. Für diesen Spezialfall ist die Übernahmezeit identisch zu der Übernahmezeit für lineares Ranking mit $E_{max} = 2$.

Da auch das schlechteste Chromosom noch eine Selektionswahrscheinlichkeit größer als Null aufweist, ist die Wettkampfselektion nicht diskriminierend. Dadurch, dass die Selektionswahrscheinlichkeit für das Chromosom i nur von der Platznummer und nicht von der Fitness abhängt, handelt es sich um eine statische (und reine) Selektionsform. Bias und Spread sind auch bei diesem Verfahren relativ hoch, da keine Vorsorge getroffen wurde, um die Anzahl der tatsächlich ausgewählten Kopien eines Chromosoms nahe am Erwartungswert der Kopienzahl zu halten.

6.4.6 Steady State Reproduktion

Dieses Verfahren wurde von Syswerda [Sys91] entwickelt und auch von Whitley [Whi89] beim GENITOR-Algorithmus verwendet. Bei der *Steady State Reproduktion* handelt es sich nicht – wie bei den bisher vorgestellten Algorithmen – um ein Verfahren zur Auswahl von Chromosomen, dem eine spezielle Art, die Auswahlwahrscheinlichkeit zu berechnen, zugrunde liegt. Statt dessen lassen sich verschiedene der bisher vorgestellten Berechnungsverfahren wie z.B. fitnessproportionale Auswahl oder Wettkampfselektion in dieses Reproduktionsverfahren integrieren.

Die Besonderheit der Steady State Reproduktion liegt darin, dass nur die für die Anwendung eines genetischen Operators notwendige Anzahl von Chromosomen zufällig gezogen wird. Auf sie wird der entsprechende Operator angewendet und die veränderten Chromosomen werden unverzüglich wieder in die Originalpopulation eingesetzt. Diese Vorgehensweise wird als „Steady State" bezeichnet, im Gegensatz zu der Bezeichnung „Generational" für alle Algorithmen, bei denen immer die gesamte Population gleichzeitig ausgetauscht wird. Dem entsprechend stehen bei einem Steady State Verfahren neu erzeugte Chromosomen sofort für den weiteren Selektionsprozess zur Verfügung.

Im Gegenzug für das sofortige Einsetzen muss jedoch dafür gesorgt werden, dass in der Population für das neue Chromosom ein Platz vorhanden ist. Hierfür gibt es verschiedene Verfahren wie z.B.

- das Ersetzen der schlechtesten Chromosomen durch die neu entstandenen Chromosomen,
- Ersetzen eines zufällig gewählten Chromosoms und
- die von Syswerda favorisierte Methode des „Exponential Ranking".

Bei von Syswerda durchgeführten Untersuchungen waren die Ergebnisse für die zufällige Auswahl am schlechtesten, während die Ergebnisse für die beiden übrigen in einen ähnlichen Bereich fielen.

6.4.7 Deterministisches Crowding

Bei diesem Vorgehen handelt es sich um eine so genannte Nischentechnik (siehe Abschnitt 6.7.1) und in diesem Sinne um das Pendant innerhalb der Selektionsverfahren zur geteilten Fitness für Fitnessfunktionen. Auch beim Deterministischen Crowding liegt das Hauptaugenmerk auf dem Erhalten möglichst vieler Lösungen (Nischen) aus unterschiedlichen Bereichen des Suchraumes. Dem entsprechend häufig werden derartige Selektionsverfahren im Bereich der Mehrkriterienoptimierung (Abschnitt 6.3.5) verwendet.

Der Unterschied zum kGA liegt darin, dass insgesamt $popsize$ Teilnehmer für die Anwendung des Crossover-Operators gewählt werden (mit Zurücklegen) und dass diese in Zweiergruppen zusammengefasst und dann mit Wahrscheinlichkeit 1 gekreuzt werden. Auf die neu entstandenen Chromosomen werden die über das Crossover hinaus verwendeten evolutionären Operatoren angewendet. Das Ergebnis wird mit den Elternchromosomen verglichen und jedes Chromosom wird dem Elternteil zugeordnet, dem es am ähnlichsten ist. Die Zuweisung zu einem Elternchromosom erfolgt phänotypisch. Es überlebt letztendlich aus diesen Paaren jeweils das Chromosom mit der besseren Fitness. Ansonsten läuft der evolutionäre Algorithmus genauso ab wie gewohnt.

Durch den Einsatz von deterministischem Crowding soll das zu schnelle Übergreifen einiger weniger Lösungstypen auf die Population verhindert werden. Dadurch, dass im Gegensatz zur Fitnessteilung die Bewertung unverändert bleibt und zumindest immer ein Vertreter jeder Lösungsklasse übernommen wird, wird eine zügige Konvergenz nicht behindert. Um diese Art der Selektion durchführen zu können, ist es notwendig, zuvor ein Ähnlichkeitsmaß zu definieren. Im Falle der Optimierung einer reellen Funktion könnten beispielsweise solche Chromosomen als ähnlich gelten, die auf der x-Achse relativ nah zusammen liegen und deren Fitnesswerte sich nur geringfügig unterscheiden.

Diese Selektionsform wurde zuerst von de Jong [DJ75] eingeführt (bezeichnet als „Crowding") und später von Mahfoud ([Mah95b, Mah95a]) genauer untersucht. In de Jongs ursprünglichem Ansatz wurde eine festgelegte Anzahl von Chromosomen für die Anwendung der evolutionären Operatoren mittels fitnessproportionaler Selektion ausgewählt. Nach Anwendung der Operatoren wurde für jedes neu in die Population eingebrachte Chromosom zufällig eine Subpopulation mit „CF" (Crowding-Faktor) Elementen gebildet, mit der das neue Chromosom verglichen wurde. Das ähnlichste Element wurde dann durch das neue Chromosom ersetzt, wobei unter Ähnlichkeit die Ähnlichkeit im genotypischen Sinne zu verstehen ist. Problematisch an diesem Ansatz ist jedoch, dass auf diese Weise nicht versucht wird, innerhalb der Population eine stabile Mischung von Chromosomenklassen aufzubauen, sondern statt dessen wird die in der vorhergehenden Population vorhandene Mischung erhalten.

Ähnlichkeit kann sowohl genotypisch, also auf der Basis der Kodierung, als auch phänotypisch auf der Basis der verschlüsselten Information berechnet werden. Gerade bei Verwendung von Niching-Operatoren wird meist nach pareto-optimalen Lösungen des eigentlichen Problems gesucht, so dass die phänotypische Ähnlichkeit von größerer Bedeutung ist.

6.5 Rekombination und Reparaturmechanismen

In diesem Abschnitt werden verschiedene Ansätze zur Rekombination (Crossover) von Chromosomen vorgestellt, die sich in Theorie und Praxis verschiedentlich bewährt haben. Aufgrund der starken Abhängigkeit der verwendeten Operatoren vom betrachteten Problem sollten die hier aufgeführten Operatoren als Hinweis bzw. als Grundlage zur Erstellung problemangepasster Operatoren verstanden werden.

Die verschiedenen Rekombinationsoperatoren werden im Folgenden in die Bereiche

- allgemeine Crossover-Operatoren,
- Crossover-Operatoren für Reihenfolgeprobleme und
- Inversion

eingeteilt. Der Unterschied zwischen den ersten beiden Gruppen ist darin zu sehen, dass die Chromosomen der zweiten Gruppe aus einer Abfolge von Nummern untersuchter Objekte bestehen. Dabei kann es sich beispielsweise um die Reihenfolge einer Städtetour beim TSP, genauso gut aber auch um die Abfolge der Lötstellen beim Aufbau einer Leiterplatte handeln. Dem entsprechend wird in diesem Fall nur selten eine Binärkodierung verwendet. Die im Abschnitt „Allgemeines Crossover" vorzustellenden Operatoren werden häufig zur Parameteroptimierung und Nullstellensuche bei Funktionen eingesetzt, wobei vorher festgelegt wird, welcher Bereich eines Chromosoms zu den einzelnen Parametern gehört.

Das beim kanonischen genetischen Algorithmus verwendete One-Point-Crossover gehört zur ersten Gruppe und zählt zu den am besten untersuchten Operatoren. Er geht zurück auf frühe Arbeiten von Holland [Hol94], wird aber nicht mehr häufig eingesetzt, da er sehr stark unter dem Problem des „Positional Bias" [ECS89] zu leiden hat. Darunter versteht man die Abhängigkeit der Austauschwahrscheinlichkeit beim Crossover für ein Gen von seiner Position im Chromosom. Beim One-Point-Crossover nimmt die Wahrscheinlichkeit mit steigender Positionsnummer zu, wobei sie wegen der gleichverteilten Auswahlwahrscheinlichkeit des Crossoverpunktes für das erste Allel 0 und für das letzte 1 beträgt. Dieser Effekt ist zumeist nicht erwünscht, da gute Schemata, die sich zumindest teilweise auf hintere Allele erstrecken, so wesentlich häufiger zerstört werden als im vorderen Bereich eines Chromosoms liegende Schemata. Generell ist die Zerstörungswahrscheinlichkeit für Schemata mit hoher definierender Länge wesentlich größer als bei kurzen Schemata.

Ein weiteres Problem liegt in der geringen Anzahl neuer Chromosomen, die bei Anwendung dieses Operators aus den Elternchromosomen entstehen können. Es ergeben sich nur Varianten, die sich aus zwei Teilstücken zusammensetzen lassen. Dem entsprechend wurde das One-Point-Crossover den Anforderungen angepasst und findet nur noch selten Verwendung.

6.5.1 Allgemeine Crossover-Operatoren

Two-point-crossover:
Einer der bekanntesten Operatoren zur Rekombination ist das „Two-Point-Crossover". Der einzige Unterschied zum One-Point-Crossover besteht darin, dass zwei statt eines Kreuzungspunktes zufällig gewählt werden. Ausgetauscht wird dann der zwischen diesen beiden Punkten liegende Bereich der Elternchromosomen. Das Verfahren läuft wie folgt ab:

Wähle zufällig zwei Kreuzungspunkte:

$$\begin{array}{c|c|c} a_1 \dots a_{i-1} & a_i \dots a_{j-1} & a_j \dots a_l \\ b_1 \dots b_{i-1} & b_i \dots b_{j-1} & b_j \dots b_l \end{array} \quad \rightarrow \quad \begin{array}{c|c|c} a_1 \dots a_{i-1} & b_i \dots b_{j-1} & a_j \dots a_l \\ b_1 \dots b_{i-1} & a_i \dots a_{j-1} & b_j \dots b_l \end{array}$$

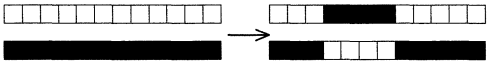


Abbildung 6.6 Two-Point-Crossover

Die Vorteile des Two-Point Crossover werden deutlich, wenn die Chromosomen nicht als Strings, sondern als Ringe aufgefasst werden:

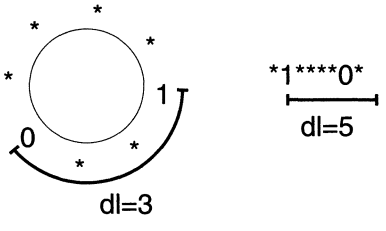


Abbildung 6.7 Definierende Länge beim 2-point-Crossover

Die Wahrscheinlichkeit, dass ein Schema zerstört wird, steigt nun nicht mehr mit der definierenden Länge, sondern hängt vom minimalem Abstand zwischen den Endpunkten des Schemas auf dem Ring ab, da ein Schema höchstens dann zerstört werden kann, wenn sich ein Crossoverpunkt innerhalb und einer außerhalb des Schemas befindet. In diesem Sinne muss für das Two-Point-Crossover die definierende Länge eines Schemas neu definiert werden. Dazu fasst das Chromosom als Ring auf und wählt das kürzere Stück als maßgeblich für die definierende Länge. Für das Chromosom $*1 * * * 0*$ ist die definierende Länge beim One-Point-Crossover 5; beim Two-Point-Crossover jedoch nur 3! Insgesamt findet sich ein günstigeres Verhalten von Schemata bei Anwendung des Two-Point-Crossover gegenüber dem One-Point-Crossover aus Sicht des Schema-Theorems. Beide Crossover-Arten weisen keinen Distributional Bias auf, worunter die Abweichung von einer Gleichverteilung für die Anzahl der ausgetauschten Gene verstanden wird [ECS89].

k-Point-Crossover
Analog zum Two-Point-Crossover lässt sich das k-Point-Crossover definieren, bei dem entsprechend k Kreuzungspunkte gewählt werden.

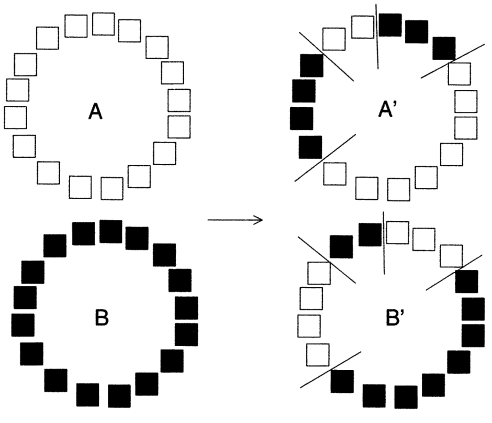


Abbildung 6.8 Chromosom als Ring aufgefasst beim 4-Point-Crossover

Falls k ungerade gewählt wurde, so wird ein zusätzlicher Kreuzungspunkt an der Position 0 hinzugenommen. Da auch in diesem Fall das Chromosom als Ring behandelt wird, wird also der letzte Abschnitt zusätzlich ausgetauscht. Der Positional Bias sinkt mit steigendem k, während der Distributional Bias ansteigt und gegen eine Binomialverteilung konvergiert [DJ75]. Eine leicht veränderte Form des k-Point-Crossovers ist das Segmented Crossover [ECS89], bei dem keine feste Zahl von Crossoverpunkten festgelegt wird. Stattdessen variiert die Anzahl um einen Erwartungswert herum.

Parametrisiertes Uniform Crossover:
Noch einen Schritt über das k-Point-Crossover hinaus geht das so genannte „Parametrisierte Uniform Crossover" [SDJ91]. Hierbei wird nicht mehr mit Kreuzungspunkten gearbeitet, sondern es wird für jedes Allel separat ausgewürfelt, ob es ausgetauscht wird. Die Austauschwahrscheinlichkeit p_c liegt dabei üblicherweise in einem Bereich zwischen 0.5 und 0.8 [Mit96, Nis97]. Der Fall $p_c = 0.5$ wird auch als „Uniform Crossover" bezeichnet und wurde von Syswerda entwickelt [Sys89]. Bild 6.9 zeigt ein Beispiel für Parametrisiertes Uniform Crossover.

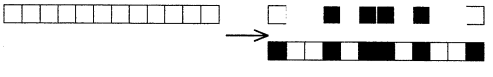


Abbildung 6.9 Uniform Crossover

Shuffle Crossover [ECS89]
Der Sinn des „Shuffle Crossovers" ist die weitgehende Beseitigung des Positional Bias. Das wird erreicht, indem vor der Anwendung eines beliebigen Crossover-Operators die Allele in den beteiligten Chromosomen zufällig vertauscht werden (gleich für beide Elternchromosomen). Nach Anwendung des eigentlichen Operators werden die Allele an den Originalplatz zurückgetauscht. Unabhängig davon, inwieweit der verwendete Crossover-Operator selbst einen hohen Positional Bias hat, lässt sich der Bias-Wert durch dieses Verfahren auf Null reduzieren.

Beim Shuffle Crossover spielt die definierende Länge eines Chromosoms keine Rolle mehr, allein die Ordnung und die Fitness sind von Bedeutung. Die Abschätzung für die Wahrscheinlichkeit, dass ein Schema H durch Crossover zerstört wird, beträgt

$$1 - \left(\frac{1}{2}\right)^{ord(H)} .$$

Empirischen Resultaten zufolge liefert von allen bisher vorgestellten Crossover-Arten das 1-Point-Crossover die schlechtesten Ergebnisse. Der Erfolg der anderen Crossover-Operatoren ist abhängig vom betrachteten Problem, bzw. von der Problemumsetzung.

Punctuated Crossover
Bei „Punctuated Crossover" handelt es sich um eine der wenigen adaptiven Crossoverformen. Angepasst werden die Punkte, an denen Crossover durchgeführt werden soll, um möglichst deutliche Verbesserungen zu erhalten („Hot Spots"). Das Verfahren wurde von Schaffer und Morishima entwickelt [SM87], hat sich aber nicht sehr weit durchsetzen können.

Um die Crossoverpunkte zusammen mit dem Chromosom entwickeln zu können, wurde jedes Chromosom durch einen Bitstring der Lange l ergänzt (Crossover-Template), bei dem eine 1 für einen zulässigen Crossoverpunkt und eine 0 für ein Verbot von Crossover an dieser Position steht. Als Crossoveroperator wird k -Point-Crossover verwendet, wobei alle in den Elternchromosomen als zulässig markierten Crossoverpunkte verwendet werden.

Dieser hinzugefügte String wird ebenso behandelt wie das übrige Chromosom, wobei das Crossover-Template immer zusammen mit den zugehörigen Bits des eigentlichen Chromosoms ausgetauscht wird. Mutation hingegen wird separat auf das Crossover-Template angewendet. Von diesem Verfahren erhofft man sich, dass sich Crossover-Punkte mit besonders positivem Einfluss bzw. mit möglichst geringem zerstörerischem Einfluss auf gute Schemata herauskristallisieren. Die tatsächliche Fitness der Chromosomen wird ausschließlich über den eigentlichen, die Lösung darstellenden Abschnitt der Chromosomen berechnet.

6.5.2 Crossover-Operatoren bei Reihenfolgeproblemen

Reihenfolgeprobleme treten sehr häufig bei den Anwendungen im Wirtschaftsbereich auf und ihre Implementationen mit evolutionären Algorithmen haben oft unter dem Pro-

blem der Lösungsgültigkeit zu leiden. Während es bei der allgemeinen Rekombination z.B. zur Lösung von Gleichungen oft genügt, den Lösungsraum so einzuschränken, dass nur Chromosomen auftreten können, die eine gültige Lösung darstellen, müssen bei Reihenfolgeproblemen zusätzlich oft Reparaturmechanismen implementiert werden. So ist es beispielsweise beim TSP notwendig zu verhindern, dass Städte mehr als einmal oder gar nicht angefahren werden. Das heißt insbesondere, dass jede durch ein Chromosom dargestellte Lösung auf ihre Korrektheit überprüft und ggf. repariert werden muss.

Reihenfolgeprobleme werden üblicherweise als Permutationen dargestellt, wobei die betrachteten Objekte nummeriert werden und über ihre Position im Chromosom ein Zusammenhang zu ihrer Reihenfolge für das Problem hergestellt wird. Es gibt auch Kodierungen, die nicht auf Permutationen aufbauen und ohne Reparaturmechanismen auskommen. Für beide Kodierungsarten werden auf den folgenden Seiten Beispiele vorgestellt.

Beispiel 6.9 (TSP mit beliebigem allgemeinen Crossover)
Gesucht ist die optimale Reihenfolge des Besuchs von sechs Städten. Die Chromosomen bestehen aus sechs Genen, deren Einträge sich auf eine Referenzliste beziehen und darüber die eigentliche Rundtour kodieren. Dabei wird für die Dekodierung eines Chromosoms $(a_1, \ldots, a_6)$ so vorgegangen, dass zuerst eine Referenzliste der Form $\{1, \ldots 6\}$ aufgestellt wird. Die erste anzufahrende Stadt ist dann diejenige, die sich auf Platz a_1 der Referenzliste befindet. Das zweite Gen a_2 bezieht sich auf die Referenzliste, aus der der a_1-te Eintrag gestrichen wurde. Das Verfahren wird solange ausgeführt, wie die Referenzliste noch Einträge enthält.

In diesem Sinne steht das Chromosom $c = [1, 3, 3, 2, 2, 1]$ für die Rundreise 1-4-5-3-6-1, da das erste Gen besagt, dass die erste Stadt aus der Referenzliste der Start der Rundreise ist. Streichen dieser aus der Referenzliste ergibt die neue Referenzliste $2, 3, 4, 5, 6$. Abarbeiten der Gene ergibt die folgenden neuen Referenzlisten: $2, 3, 5, 6$, dann $2, 3, 6$, dann $2, 6$, und schließlich 2 und die leere Liste am Ende der Rundreise.

Der Vorteil dieses Verfahrens ist, dass keine Reparaturmechanismen notwendig sind, da auch bei bei Anwendung der im vorhergehendem Abschnitt erläuterten Crossoververfahren keine ungültigen Chromosomen entstehen.

Der Nachteil liegt darin, dass der Crossover im allgemeinen zu keinen guten Ergebnissen führt, da die Bedeutung der letzten Gene von den vorderen Genen abhängig ist. Eine Lösung, die im hinteren Teilstück günstige Ergebnisse liefert, kann durch Kreuzen im vorderen Abschnitt zu einer insgesamt schlechten Lösung werden, da die Bedeutung des hinteren Teilstücks sich völlig ändern kann.

Partially Matched Crossover
Beim „Partially Matched Crossover" (PMX [GL85]) gibt die Position der Objektnummer im Chromosom direkt den Rang in der Reihenfolgeliste wieder. Um Fehler beim Kreuzen zweier Chromosomen (doppelte oder fehlende Platzeinträge) zu verhindern, wird wie folgt vorgegangen:

1. Auswürfeln der Position zweier Kreuzungspunkte.
2. Direktes Austauschen der Objekte auf den gleichen Positionen in diesem Abschnitt.
3. Eintragen der ursprünglichen Allele, sofern die Nummern in keinem der beiden ausgetauschten Chromosomenabschnitte vorkommen.
4. Ersetzen der doppelt vorkommenden Allele in den Chromosomen durch die im ausgetauschten Abschnitt durchgeführte Neuzuweisung.

Beispiel 6.10 (TSP mit Partially Matched Crossover)
Die Positionen der einzelnen Gene entsprechen der Reihenfolge in der Rundreise und die Allele beinhalten die den Städten zugewiesenen Stadtnamen, also gilt, dass alle c_i verschieden sein müssen. Wir benutzen nun 2-point-crossover mit Reparatur:

$$
\begin{array}{lllll}
Chromosom1 & = & 1\ 2 & 3\ 4\ 5 & 6 \\
Chromosom2 & = & 6\ 2 & 1\ 3\ 5 & 4
\end{array}
$$

$$
\begin{array}{lllll}
\rightsquigarrow & * \ 2 & 1\ 3\ 5 & 6 & \rightsquigarrow \quad (1 \to 3 \to 4)\ 2 \quad 1\ 3\ 5 \quad 6 \\
 & 6\ 2 & 3\ 4\ 5 & * & \qquad\quad 6 \qquad\ 2 \quad 3\ 4\ 5 \quad (4 \to 3 \to 1)
\end{array}
$$

$$
\begin{array}{lllll}
\rightsquigarrow & 4\ 2 & 1\ 3\ 5 & 6 & \\
 & 6\ 2 & 3\ 4\ 5 & 1 &
\end{array}
$$

Da die 3 ebenso wie die 1 innerhalb des ausgetauschten Abschnittes liegt, genügt es nicht, die außerhalb liegende 1 durch die 3 zu ersetzen, sondern es muss nachgeprüft werden, wohin die 3 letztendlich abgebildet wird.

Cycle Crossover
Das „Cycle Crossover" (CX [OSH87]) ist ein Verwandter des Uniform Crossover, da auch bei diesem Operator kein fester Abschnitt festgelegt wird, in dem die Informationen zwischen den Elternchromosomen ausgetauscht werden. Stattdessen wird das folgende Verfahren angewendet:

1. Das erste Gen an Position 1 des ersten Chromosoms c_1 mit dem Allel $a_{1,1}$ wird wieder an der Originalposition des ersten Chromosoms eingetragen. Da die erste Position des zweiten Chromosoms daher nicht mehr vom ersten Chromosom übernommen werden kann, wird auch hier das Originalallel eingetragen.
2. Da das Allel des Gens $a_{2,1}$ an Position 1 des zweiten Chromosoms c_2 nicht mehr in das erste Chromosoms übernommen werden kann, muss das identische Allel des ersten Chromosoms ebenfalls behalten werden. Da dieselbe Problematik dann für das ebenfalls neu hinzugekommene Allel des zweiten Chromosoms gilt, muss dieses Verfahren solange fortgesetzt werden, bis auch das Gen mit dem Allel von Position 1 von c_1 für c_2 übernommen wurde.
3. Nach Abschluss eines derartigen Durchlaufs werden alle fehlenden Plätze in c_1 mit den verbliebenen Genen von c_2 gefüllt und umgekehrt.

Beispiel 6.11 (TSP mit Cycle Crossover)
Auch hier entspricht die Reihenfolge der Allele im Chromosom wieder der Reihenfolge der Städte in der Rundreise.

$$
\begin{array}{lllllllllllllllllll}
Chromosom1 & = & 1 & 2 & 3 & 4 & 5 & 6 & & & & & & & & & & & \\
Chromosom2 & = & 3 & 6 & 4 & 1 & 2 & 5 & & & & & & & & & & & \\
\\
\rightsquigarrow & 1 & * & * & * & * & * & \rightsquigarrow & 1 & * & 3 & * & * & * & \rightsquigarrow & 1 & * & 3 & 4 & * & * \\
 & 3 & * & * & * & * & * & & 3 & * & 4 & * & * & * & & 3 & * & 4 & 1 & * & * \\
\\
\rightsquigarrow & 1 & 6 & 3 & 4 & 2 & 5 & & & & & & & & & & & & & & \\
 & 3 & 2 & 1 & 3 & 5 & 6 & & & & & & & & & & & & & &
\end{array}
$$

Mit der Auswahl des Allels 1 vom ersten Chromosom kann das Allel 3 nicht mehr vom zweiten Chromosom ausgewählt werden. Daher wird es vom ersten Chromosom auf die Originalposition übernommen. Dadurch wird die Übernahme des Allels 4 des zweiten Chromosoms unmöglich gemacht, so dass auch dieses Allel vom ersten Chromosom stammen muss. Damit schließt sich der Kreis („Cycle") und alle verbliebenen Allele des zweiten Chromosoms werden eingetragen.

Order Crossover
Das „Order-Crossover" (OX [Dav85]) wurde mit dem Hintergedanken entwickelt, dass bei Anwendung des Crossover-Operators die relative Reihenfolge der nicht ausgetauschten Gene erhalten werden soll. Das ist insbesondere dann wünschenswert, wenn die Reihenfolge im direktem Bezug zu den Kosten zwischen zwei aufeinanderfolgenden Allelen steht, statt vom Platz in der Rangfolge abzuhängen. Auch bei diesem Crossover-Verfahren lassen sich die Chromosomen gut als Ring behandeln. Der Ablauf des Verfahrens ist folgendermaßen:

1. Bestimme zufällig zwei Crossover-Punkte, um den dazwischen liegenden Abschnitt der beiden Chromosomen auszutauschen.
2. Ersetze im Chromosom c_1 alle Allele, die von c_2 nach c_1 verschoben werden sollen, durch Platzhalter. Analog bei c_2.
3. Beginne ab dem zweiten Kreuzungspunkt damit, die Platzhalter auf dem Ring nach links zu schieben, bis sich alle im Bereich zwischen den Kreuzungspunkten befinden.
4. Übernimm bei beiden Chromosomen die Originalallele vom anderer Chromosom auf die Platzhalter.

Beispiel 6.12 (TSP mit Order Crossover)
Die Bedeutung der Abfolge von Allelen ist analog zu den letzten beiden Beispielen.

$$
\begin{array}{lll}
Chromosom1 & = & 1\ 2 \mid 3\ 4 \mid 5\ 6 \\
Chromosom2 & = & 6\ 3 \mid 5\ 2 \mid 1\ 4
\end{array}
$$

$$
\rightsquigarrow \begin{array}{cc|cc|cc} 1 & * & 3 & 4 & * & 6 \\ 6 & * & 5 & 2 & 1 & * \end{array} \rightsquigarrow \begin{array}{cc|cc|cc} 3 & 4 & * & * & 6 & 1 \\ 5 & 2 & * & * & 1 & 6 \end{array}
$$

$$
\rightsquigarrow \begin{array}{cc|cc|cc} 3 & 4 & 5 & 2 & 6 & 1 \\ 5 & 2 & 3 & 4 & 1 & 6 \end{array}
$$

Enhanced Edge Recombination
Wie beim Order Crossover wird auch beim „Enhanced Edge Recombination" (EER [SMM+91]) Wert darauf gelegt, die Reihenfolgebeziehungen innerhalb der Elternchromosomen auf das neu entstehende Chromosom zu übertragen. Dabei wird allerdings davon ausgegangen, dass eine Umkehrung der Reihenfolge zumindest bezüglich der beteiligten Elemente keinen Unterschied ergibt. Der verwendete Ansatz ist jedoch völlig anders als beim Order Crossover. Beim EER wird aus zwei Elternchromosomen lediglich ein neues Chromosom konstruiert und es wird für die Elternchromosomen eine gemeinsame Liste aufgestellt, die die Verbindungen jedes Allele zu seinen Nachbarallelen enthält (Kanten), wobei die Chromosomen wie zuvor als Ringe aufgefasst werden. Dabei werden die Allele beider Chromosomen der Reihe nach abgearbeitet und für jedes Allel wird eine vo genanntes Kantenteilliste aufgestellt.

In diesem Sinne enthält beispielsweise das Chromosom (1234) die Kanten (12), (14), (23) und (34) und würde damit z.B. für das Allel 1 die Werte 2 und 4 zur Kantenteilliste von 1 beitragen. Danach werden die Kanten für das zweite Chromosom in dieselbe Liste eingetragen. Hat ein Allel in beiden Chromosomen einen gleichen Nachbarn, so wird dieses Nachbarallel nur einmal eingetragen, aber zusätzlich mit einem Minuszeichen versehen. Das Erzeugen des neues Chromosoms geht nach dem folgenden System vor sich:

1. Erstellen einer Kantenliste für die Elternchromosomen.
2. Auswahl des Startelementes. Entweder das erste Element eines der beiden Elternchromosomen oder das Allel mit der kürzesten Kantenteilliste.
3. Trage das gewählte Allel an die erste freie Position des neuen Chromosoms ein und entferne dieses Allel aus den Kantenteillisten aller verbliebenen Allele.
4. Wähle das nächste Allel nach dem folgenden Präferenzsystem und lösche dann die Kantenteilliste des zuletzt bearbeiteten Allels:
   (a) Negatives Element zuerst (da in beiden Elternchromosomen).
   (b) Element mit kürzester Kantenteilliste (da seine Vorgänger bzw. Nachfolger bereits teilweise abgearbeitet wurden).

(c) Zufällige Auswahl.

5. Führe die Schritte 3 und 4 durch, bis das neue Chromosom komplett ist.

Beispiel 6.13 (TSP mit Enhanced Edge Recombination)
Als Kodierung für das TSP wird wieder dieselbe Reihenfolgekodierung verwendet wie in den vorhergehenden Beispielen.

$$\begin{array}{lll} Chromosom1 & = & 1\ 2\ 3\ 4\ 5\ 6 \\ Chromosom2 & = & 6\ 2\ 1\ 3\ 5\ 4 \end{array} \quad \text{Startelement } 1$$

$$\rightsquigarrow \begin{array}{lll} \underline{1} & \to & -2,6,3 \\ 2 & \to & -1,3,6 \\ 3 & \to & 2,4,1,5 \\ 4 & \to & 3,-5,6 \\ 5 & \to & -4,6,3 \\ 6 & \to & 5,1,2,4 \end{array} \overset{1}{\rightsquigarrow} \begin{array}{lll} \underline{2} & \to & 3,6 \\ 3 & \to & 2,4,5 \\ 4 & \to & 3,-5,6 \\ 5 & \to & -4,6,3 \\ 6 & \to & 5,2,4 \end{array} \overset{2}{\rightsquigarrow} \begin{array}{lll} 3 & \to & 4,5 \\ 4 & \to & 3,-5,6 \\ 5 & \to & -4,6,3 \\ \underline{6} & \to & 5,4 \end{array}$$

$$\overset{6}{\rightsquigarrow} \begin{array}{lll} 3 & \to & 4,5 \\ \underline{4} & \to & 3,-5 \\ 5 & \to & -4,3 \end{array} \overset{4}{\rightsquigarrow} \begin{array}{lll} 3 & \to & 5 \\ \underline{5} & \to & 3 \end{array} \overset{5}{\rightsquigarrow} 1\,2\,6\,4\,5\,3$$

Das Crossover startet mit dem ersten Gen des ersten Chromosoms. In dessen Kantenteilliste ist das Allel 2 mit einem Minuszeichen versehen, daher wird es als nächstes ausgewählt. Die Liste für das Allel 1 wird gestrichen, ebenso wie die 1 aus den Listen der verbliebenen Allele. Die Listen der Allele 3 und 6 sind gleichwertig, so dass das nächste Element zufällig gewählt werden muss. Es wurde die 6 ausgewürfelt. Dasselbe Verfahren wird der Reihe nach für die 4, die 5 und die 3 durchgeführt.

Präzedenz-Matrix-Crossover
Bei dieser Form der Problemkodierung wird von der Darstellung eines Chromosoms als Zeichenkette vollständig abgewichen. Statt dessen werden Chromosomen als binäre $l \times l$-Matrizen $M = (m_{ij})$ dargestellt, in denen immer dann eine 1 eingetragen wird, wenn gilt:

$$m_{ij} = 1 \iff \text{Stadt } i \text{ wird vor Stadt } j \text{ besucht.}$$

Dem entsprechend stellt eine derartige Matrix genau dann eine vollständige Rundreise dar, wenn die folgenden Bedingungen erfüllt sind:

1. Die Anzahl der Einsen muss gleich $\frac{l(l-1)}{2}$ sein.
2. $m_{ii} = 0$
3. $m_{ij} = 1 \wedge m_{jk} = 1 \Rightarrow m_{ik} = 1$

Das folgende zugehörige Crossover für zwei $l \times l$-Matrizen M und N wird als „Präzedenz-Matrix-Crossover" (MX) bezeichnet und wurde von Fox und McMahon [FM91] entwickelt:

1. Bilde aus der Schnittmenge der Einträge der Matrizen M und N die neue Matriz K. Alle verschieden Werte werden weggelassen.
2. Wähle zufällig eine der beiden Elternmatrizen und daraus eine noch nicht in K enthaltene Eins. Trage sie an der entsprechenden Stelle in der Matrix K ein.
3. Füge alle Einträge hinzu, die durch die oben genannten Bedingungen erzwungen werden.
4. Führe die Schritte 2 und 3 aus, bis die Matrix K eine korrekte Rundreise darstellt (Bedingungen 1 bis 3 erfüllt).

Die in Schritt 2 ausgewählte Eins ist immer nur in einem der beiden Elternchromosomen vertreten, da alle in beiden vertretenen Einsen bereits im ersten Schritt in die Matrix K eingetragen wurden. Mit der Auswahl einer derartig eindeutig zuweisbaren Eins geht also immer Erbinformation eines bestimmten Elternteils in das neue Chromosom über. Im dritten Schritt müssen nun all diejenigen „Städte" k hinzugefügt werden, die bereits vor der Stadt i in der Sequenz vorkommen, die ihrerseits wiederum durch die Hinzunahme einer Eins an die Position (i, j) vor der Stadt j liegt.

Beispiel 6.14 (TSP mit Präzedenz-Matrix-Crossover)
Das Präzedenz-Matrix-Crossover wird am Beispiel der Rundreisen $(3, 6, 4, 1, 2, 5)$ und $(1, 2, 3, 4, 5, 6)$ dargestellt. Hierfür wird zuerst eine unvollständige Schnittmatrix gebildet, die genau die Einträge enthält, die in beiden, den Rundreisen zugeordneten Matrizen vorkommen.

$$
\begin{array}{cccccc}
0&1&0&0&1&0\\
0&0&0&0&1&0\\
1&1&0&1&1&1\\
1&1&0&0&1&0\\
0&0&0&0&0&0\\
1&1&0&1&1&0
\end{array}
\oplus
\begin{array}{cccccc}
0&1&1&1&1&1\\
0&0&1&1&1&1\\
0&0&0&1&1&1\\
0&0&0&0&1&1\\
0&0&0&0&0&1\\
0&0&0&0&0&0
\end{array}
\overset{(1)}{=}
\begin{array}{cccccc}
0&1& & &1& \\
0&0& & &1& \\
 & & 0&1&1&1\\
 & & 0&0&1& \\
0&0&0&0&0& \\
 & & 0& & &0
\end{array}
$$

$$
\overset{(2/3)}{\rightsquigarrow}
\begin{array}{cccccc}
0&1&\underline{0}& &1& \\
0&0&\underline{0}& &1& \\
\mathbf{1}&\underline{1}&0&1&1&1\\
 & &0&0&1& \\
0&0&0&0&0& \\
 & &0& & &0
\end{array}
\oplus
\overset{(2/3)}{\rightsquigarrow}
\begin{array}{cccccc}
 &0&1&0& &1\\
 &0&0&0&\mathbf{1}&1\\
1&1&0&1&1&1\\
 &\underline{0}&0&0&1&1\\
 &0&0&0&0&0\\
 &1&1&0&1&1
\end{array}
\overset{(2/3)}{\rightsquigarrow}
\begin{array}{cccccc}
 &0&1&0& &1\\
 &0&0&0&1&1\\
 &0&0&0&1&1\\
 &\underline{0}&0&0&1&1\\
 &0&0&0&0&0\\
 &1&1&0&1&1
\end{array}
\rightsquigarrow 3\ 6\ 1\ 2\ 4\ 5
$$

Danach wird die Eins an der Position $(3,1)$ der ersten Matrix ausgewählt (fett gedruckt), also die Aussage, dass die erste Stadt auf die dritte folgt. Hinzugenommen wird dann die Eins an Position $(3,2)$, da gilt: $3 \hookleftarrow 1 \overset{\hookleftarrow 2}{\underset{\hookleftarrow 5}{}}$. Des weiteren werden die komplementären Aussagen hinzugefügt. Alle zusätzlich hinzugenommenen Allele sind durch einen Unterstrich gekennzeichnet. Im nächsten Schritt wird die Eins von Position $(2,4)$ aus Matrix 2 eingetragen und um das Komplement ergänzt. In den nächsten Schritten wurde dasselbe Verfahren für die Eins auf den Positionen $(6,1)$ und $(1,4)$ angewendet, was letztendlich zu einer gültigen Lösung führt.

6.5.3 Inversion

Beim Inversionsoperator handelt es sich um einen der Natur abgeschauten Mechanismus, der – üblicherweise – zwischen zwei zufällig gewählten Schnittpunkten (wie beim Two-Point-Crossover) die Reihenfolge der Gene vertauscht. Angesichts der Tatsache, dass normalerweise bei einem evolutionären Algorithmus versucht wird, mit steigender Generationenzahl gute Problemlösungen zu erarbeiten und im Folgendem zu verbessern, erscheint es merkwürdig, derartige zeitaufwändig entwickelte Lösungen durch Anwendung der Inversion zu zerstören. Es muss daher an dieser Stelle darauf hingewiesen werden, dass dieser Operator für platzunabhängige Kodierungen entwickelt wurde, wie sie beispielsweise bei einer Reihenfolgekodierung mit angehängter Platznummer vorliegt (siehe auch Abschnitt 7.4.1).

Entwickelt wurde dieser Operator von Holland [Hol94], um gemäß der Building-Block-Hypothese (siehe Abschnitt 5.1) möglichst solche Gene direkt hintereinander zu setzen, die zusammen zu einer guten Lösung führen. Wie bereits vorher erläutert, ist bei Verwendung von One- und Two-Point-Crossover die Zerstörungswahrscheinlichkeit für ein Schema umso größer, je höher seine definierende Länge ist (siehe Abschnit 6.5).

Dieses Problem tritt also immer dann auf, wenn beispielsweise bei One-Point-Crossover ein Building-Block der folgenden Form vorliegt: 00*...*00. Um daraus einen Building-Block mit hoher Fitness, geringer Ordnung und kleiner definierender Länge zu machen, müsste statt der Kodierung $c_1 c_2 \ldots c_{l-1} c_l$ die Kodierung $c_1 c_2 c_l c_{l-1} c_{l-2} \ldots c_3$ gewählt werden, so dass aus 00*...*00 nun 0000*...* wird. Durch die Anwendung des unten definierten Inversionsoperator mit den Schnittstellen 3 und l würden die Gene automatisch entsprechend umsortiert. Damit das nicht zu einer völlig anderen Bedeutung des Chromosoms führt, sollte jedes Gen als Information auch seine Platznummer ν mit sich führen: $(1,c_1)(2,c_2)\ldots(l,c_l)$.

Eine Inversion wird folgendermaßen durchgeführt: Sei c ein Chromosom mit l Genen. Bestimme zufällig zwei Stellen i,j mit $1 \leq i < j \leq l$. Ordne die Gene in dem Chromosom nun so um, dass die Reihenfolge der Gene zwischen i und j umgedreht wird.

$$
\begin{aligned}
&\text{Sei } i,j \in \{1,\ldots l\}\ ,\ i<j\ : \\
&(\nu_1,c_{\nu_1})\ldots(\nu_{i-1},c_{\nu_{i-1}})|(\nu_i,c_{\nu_i})\ldots(\nu_j,c_{\nu_j})|(\nu_{j+1},c_{\nu_{j+1}})\ldots(\nu_l,c_{\nu_l}) \\
&\rightsquigarrow (\nu_1,c_{\nu_1})\ldots(\nu_{i-1},c_{\nu_{i-1}})|(\nu_j,c_{\nu_j})\ldots(\nu_i,c_{\nu_i})|(\nu_{j+1},c_{\nu_{j+1}})\ldots(\nu_l,c_{\nu_l})
\end{aligned}
$$

Bild 6.10 zeigt ein Beispiel für eine Inversion der Gene von 3 bis 6. Der Inversionsoperator stellt also eine sinnvolle Ergänzung zu einem Crossoveroperator dar, da es die Aufgabe der Inversion ist, Schemata mit hoher Fitness so umzuordnen, dass sich die definierende Länge verringert, während der Crossover-Operator derartige Schemata zu guten Gesamtlösungen zusammensetzen soll.

Abbildung 6.10 Inversion

Fasst man für den Inversionsoperator das Chromosom nicht als Ring auf, so leidet auch dieser Operator unter einer Art Positional Bias, da die Gene im mittleren Bereich der Chromosomen häufiger vertauscht werden als Gene in den Randbereichen. Um diesem Problem entgegenzutreten, entwickelte Frantz [Fra72] den „End-Inversions"-Operator, bei dem zufällig ein Endpunkt und ein Punkt im Chromosom als Randpunkte für den Inversionsoperator gewählt werden. Angewendet wurde dann diese Inversionsform zusammen mit der Standard-Inversion mit jeweils unterschiedlicher Wahrscheinlichkeit.

6.6 Mutation

In einem evolutionären Algorithmus hat der Mutationsoperator die Aufgabe, die in den Genen enthaltenen Informationen mit einer geringen Wahrscheinlichkeit zu verändern, um so zu verhindern, dass wichtige Allele aussterben oder nie in die Population eingefügt werden. Eine gängige Mutationswahrscheinlichkeit, die auch eine theoretische Rechtfertigung besitzt [Müh92a], liegt bei $1/n$, wobei n die Anzahl der Gene eines Chromosoms ist. Je nach Spezifikation des Mutationsoperators lässt sich

- Mutation wie eine Hill-Climbing-Variante verwenden, die durch das Erzeugen kleiner Änderungen in der Umgebung der durch ein Chromosom dargestellten Lösung zielgerichtet auf das Optimum zuläuft (lokale Optimierung).
- durch weitläufigere Mutation die Vielfältigkeit in der Population garantieren und dafür zu sorgen, dass lokale Optima wieder verlassen werden können, um das globale Optimum zu erreichen.

Für die Hill-Climbing Variante sollte vorher ein Maß für die Nachbarschaftsbeziehung eingeführt werden, da es ansonsten kaum möglich ist, ein vorhandenes Allel innerhalb der Nachbarschaft zu verändern. Von Vorteil ist bei Verwendung eines derartigen Operators eine Kodierung, die von vornherein eine Nachbarschaftsbeziehung berücksichtigt, wie es beim Beispiel 6.1 zum „Schnellsten Weg" der Fall ist.

Wie die Crossover-Operatoren müssen auch die Mutations-Operatoren an das untersuchte Problem angepasst werden, was ebenfalls über die Verwendung von Heuristiken oder die Einbindung von Standard-Optimierungsverfahren (siehe Kapitel 3) geschehen kann. Mit den zusätzlichen Verfahren können Regionen im Lösungsraum gefunden werden, in denen eine Suche erfolgversprechender erscheint, so dass es sich lohnt, Mutationen häufiger in Richtung dieser Bereiche vorzunehmen. Völlig andere Mutationsformen werden benötigt, wenn es nicht mehr um Binär-, Gray- oder Integer-Kodierung, sondern um Mutation für reellwertige Chromosomen geht. Eine derartige Mutationsform ist die

Non Uniform Mutation: Bei dieser Art von Mutation ist das Ausmaß der Mutation abhängig von der Generation, in der die Mutation angewendet wird. Die Mutation ist dabei folgendermaßen definiert [Mic96]:

Definition 6.11 (Non Uniform Mutation) *Sei* $c = (a_1, \ldots, a_l)$ *ein Chromosom und* a_k *gewählt für die Mutation und* $[u_k, \ldots, o_k]$ *der Definitionsbereich von* a_k. $\triangle(x, y)$ *sei eine mit* x *monoton fallende Funktion mit Wertebereich* $[0, \ldots, y]$. *Dann geht* a_k *unter dem Mutationsoperator über nach* a'_k *mit:*

$$a'_k = \begin{cases} a_k + \triangle(t, o_k - a_k) & \text{falls } z = 0 \\ a_k - \triangle(t, a_k - u_k) & \text{falls } z = 1 \end{cases}$$

wobei es sich bei z *um eine binäre Zufallszahl handelt.*

Durch die monoton fallende Funktion $\triangle(.,.)$ ist die Mutation am Anfang zufällig über den Suchraum verteilt, während sie mit zunehmender Generationenzahl immer stärker sinkt. Verzichtet man auf die generationsabhängige Komponente und verwendet statt dessen eine Gleichverteilung auf dem Wertebereich der Allele, so wird die Mutation als **„Uniform Mutation"** bezeichnet.

Eine andere Möglichkeit, unterschiedliche Mutationsraten zu verwenden, ist, sie abhängig vom Locus eines Gens zu machen. So lässt sich eine geringere Mutationswahrscheinlichkeiten für die Gene erzeugen, deren Veränderung eine starke Änderung der Bewertung bewirkt, und eine höhere für solche Gene, die die Fitness nur geringfügig beeinflussen. Die Notwendigkeit für den Einsatz einer Locusabhängigen Mutation ist im folgenden Beispiel gegeben:

Beispiel 6.15 Aufgabe: Maximiere $\varphi : [0, 1] \to [0, 1]$ (stetig).
Kodierung: Chromosom mit 20 binären Genen.

$$(a_1, \ldots, a_{20}) \rightsquigarrow x = \sum_{i=1}^{20} 2^{-i} a_i$$

Je höher der Exponent ist, desto geringer ist die Beeinflussung von x durch eine Veränderung der zugehörigen Variablen a_i. Dementsprechend wird die Mutation indexabhängig definiert.
Mutation: $p_m(i)$ abhängig vom Locus i mit

$$p_m^{(20)} > p_m^{(19)} > \ldots > p_m^{(2)} > p_m^{(1)}$$

Es soll also als Nebenbedingung gelten, dass die Mutationswahrscheinlichkeit $p_m^{(i)}$ für das i-te Gen kleiner ist als die des vorherigen und größer als die folgenden. Damit versucht man zu erreichen, dass gute Lösungen durch Mutation nicht mehr so leicht zerstört werden können, während durch eine entsprechend große Mutationswahrscheinlichkeit der niederwertigen Bits aber immer noch Feintuning möglich ist.

Berücksicht werden sollte auch immer die Möglichkeit, dass durch Mutation unzulässige Lösungen entstehen, die durch den Einsatz von Reparaturmechanismen analog zu den Crossover-Operatoren korrigiert werden müssen. Beispielsweise kann beim Travelling-Salesman-Problem durch Mutation eine Stadt mehrfach auftreten, sofern die Mutation lediglich eine Stadt durch eine andere ersetzt. Dadurch wird die Lösung unzulässig. Ein Operator, der gerade bei Reihenfolgeproblemen gut angewendet werden kann, ist die *Strukturmutation* („Structural Mutation" [vL91]). Hierbei handelt es sich nicht um eine Mutation im üblichen Sinne, wo lediglich ein ausgewähltes Allel verändert wird, sondern es findet ein Austausch zwischen dem ausgewähltem Allel und einem zufällig gewähltem anderen Allel statt. Dadurch wird ein Reparaturmechanismus überflüssig.

6.7 Weitere Aspekte

Neben den bisher vorgestellten Varianten der Standard-Darstellungen und -Operatoren sollen in diesem Abschnitt Ausprägungen evolutionärer Algorithmen vorgestellt werden, die in eine völlig andere Richtung zielen: Nischentechniken und adaptive Parameter. Sofern die Nischentechniken sich auf bekanntere Operatoren u.ä. beziehen, sind sie bereits in den zugehörigen Abschnitten behandelt worden und es wird nur noch auf bisher nicht erwähnte Varianten eingegangen.

6.7.1 Nischentechniken

Unter dem Begriff der „Nischentechniken" werden alle Verfahren, Algorithmen und Operatoren zusammengefasst, deren Ziel es ist, die Diversität in der Population aufrecht zu erhalten. Evolutionäre Algorithmen mit dem Ziel der Nischenbildung sind immer dann von Interesse, wenn es sich beim betrachteten Problem um ein Mehrkriterienproblem (siehe Abschnitt 6.3.5) handelt.

Der Grundgedanke bei allen Nischentechniken ist die Analogie zur Natur, dass sich Individuen in völlig verschiedene Richtungen entwickeln können, wobei sie mehr oder weniger ergiebige Nischen besetzten. Je nach Anzahl der in diesen Nischen existierenden Lebewesen und in Abhängigkeit von der Qualität der besetzten Nischen können sich die Individuen ungestört vermehren oder es kommt zu Ausleseprozessen, bei denen die Schwächsten entfernt werden. Das alles geschieht beschränkt auf die eigene Nische, ohne das Konkurrenz von Individuen aus anderen Nischen berücksichtigt wird. Gerade, wenn

damit zu rechnen ist, dass es mehrere gleichwertige, wenn auch völlig unterschiedliche Lösungen gibt (pareto-optimal), ist es günstig, dafür zu sorgen, dass sich nicht nur eine Lösung in der Population durchsetzt.

Obwohl Nischentechniken in diesem Zusammenhang sehr sinnvoll sind, gibt es nur wenige entsprechende Algorithmen, von denen die wichtigsten bereits in den vorhergehenden Abschnitten behandelt wurden:

- Fitness Sharing (Abschnitt 6.3.4).
- Deterministisches Crowding (Abschnitt 6.4.7).

Eine sehr einfache Methode zur Verminderung der vorzeitigen Konvergenz ist das „*Preselection*"-Verfahren von Cavicchio (siehe [Gol89]). Hierbei wird jedes veränderte Chromosom mit seinem Elternchromosom verglichen und nur das bessere von beiden wird in die neue Population übernommen. Auf diese Weise wird im Falle geringfügiger Veränderungen verhindert, dass zwei sehr ähnliche Chromosomen gleichzeitig in der nächsten Population enthalten sind. Das bereits vorgestellte Crowding-Verfahren von de Jong geht ebenso wie Deterministisches Crowding auf Preselection zurück.

Wesentlich bekannter geworden ist das von Schaffer entwickelte Modell *VEGA* (Vector Evaluated Genetic Algorithm) (siehe hierzu [Sch84]). Schaffer geht dabei ähnlich zur geteilten Selektion in Abschnitt 6.3.5 vor, indem er zu jedem Kriterium eine Subpopulation auswählt, die dann in die Gesamtpopulation übernommen wird. Die genetischen Operatoren werden auf die Gesamtpopulation angewendet, um auf diese Weise gute pareto-optimale Kombinationslösungen zu finden.

Ein neuerer Ansatz, der auf der Basis von Sharing und Wettkampfselektion (Abschnitt 6.4.5) arbeitet, ist der sogenannte „*Niched Pareto Genetic Algorithm*" (NPGA [HNG94]) von Horn, Nafpliotis und Goldberg. Da Wettkampfselektion für Probleme entwickelt wurde, die nur über ein Lösungskriterium verfügen, musste diese Selektionsform für die Anwendung in Multikriterienproblemen entsprechend angepasst werden. Statt zwei ausgewählte Chromosomen untereinander zu vergleichen, wird zusätzlich eine Vergleichsgruppe nach dem Zufallsprinzip gewählt. Jedes Chromosom aus der Wettkampfgruppe wird nun gegenüber jedem Chromosom der Vergleichsgruppe auf Dominanz geprüft. Dominiert nur eines der beiden Chromosomen die Vegleichsgruppe, so hat es den Wettkampf für sich entschieden. In den anderer Fällen geht der Wettkampf unentschieden aus. Danach wird zusätzlich geprüft, welches der beiden Chromosomen den niedrigeren Nischenzähler hat, für dessen Berechnung nur die Chromosomen der Vergleichsgruppe und die beiden Kandidaten herangezogen werden. Um die Diversität zu erhalten, wird dieses Chromosom gewählt.

Ebenfalls auf der Grundlage eines Sharing-Verfahren arbeitet der von Miller und Shaw entwickelte *Dynamic Niche Sharing*-Algorithmus [MS96]. Beim Dynamic Niche Sharing wird versucht, den beim Standard-Sharing-Algorithmus relativ hohen Zeitaufwand zum Vergleichen der Chromosomen beim Berechnen der Ähnlichkeit zu verringern. Diesem Ansatz liegen zwei Annahmen zugrunde:

- Die Anzahl q der Spitzen der betrachteten Funktion ist bekannt.
- Alle Funktionsspitzen weisen einen Mindestabstand von σ_{Sh} auf.

Darüber hinaus wird davon ausgegangen, dass sich bei einem Algorithmus, der mit Sharing arbeitet, die Nischen bei steigender Anzahl der Generationen immer stärker mit Chromosomen bevölkern. Die Fitness ist analog zum Sharing definiert:

$$fit(c_i) = \frac{fit(c_i)^*}{m_{dSh,i}} \quad \text{mit} \quad m_{dSh,i} = \begin{cases} n_j & \text{falls } c_i \text{ ein Element von Nische } j \text{ ist} \\ m_i & \text{sonst} \end{cases}$$

Hierbei ist n_j die Anzahl der Individuen in Nische j, während m_i dem Niche Count des Standard-Sharing-Verfahrens entspricht. m_i wird immer dann verwendet, wenn sich das Chromosom i nicht eindeutig einer Nische zuordnen lässt. Ein Chromosom wird dann einer Nische zugeordnet, wenn seine Distanz zum besten Element (Nischenvertreter) seiner Nische kleiner oder gleich σ_{SH} ist.

Benötigt werden noch die q Nischenvertreter, bei denen es sich jeweils um die besten Chromosomen handelt. Sie lassen sich über einen Greedy-Algorithmus (siehe Abschnitt 3.6) nach vorhergehender Sortierung der Chromosomen nach Fitness abschätzen, indem immer das nächstbeste Element, das zu keiner der bisher gefundenen Nischen gehört, als Nischenvertreter gewählt wird, bis q Nischen gefunden wurden. Diese Nischenvertreter müssen in jeder Generation neu berechnet werden. Die in [MS96] vorgestellten Ergebnisse eines Vergleichs mit dem Standard-GA, Deterministic Crowding, und dem Standard-Sharing-Verfahren kombinieren das Auffinden und Halten sämtlicher Funktionsspitzen mit einer deutlichen Reduzierung der Rechenzeit bei Verwendung von Dynamic Niche Sharing gegenüber dem Standard-Sharing.

6.7.2 Adaption

Adaption umfasst grob alle Bereiche der evolutionären Algoritmen, in denen in Abhängigkeit von der Anzahl der Generationen oder dem Zustand der Population Parameter oder Abläufe verändert werden. Prinzipiell ist jeder evolutionäre Algorithmus selbst ein adaptiver Prozess, bei dem sich eine Population an ein Optimum anpasst. Es liegt daher nahe, nicht nur das Ziel mittels Adaption zu erreichen, sondern auch den Weg dorthin durch automatische Anpassung zu erleichtern (Parameteranpassung, Anpassung der Operatoren).

Eine manuelle Anpassung wird bei der Entwicklung evolutionärer Algorithmen zur Lösung angewandter Probleme eigentlich immer durchgeführt, um Populationsgrößen, Mutations- und Crossoverraten im Vorfeld der eigentlichen Simulationsläufe zu optimieren. Beispiele dafür finden sich bei Grefenstette [Gre86] und de Jong [DJ75]. Generell gibt es nur grobe Richtlinien zur Parametereinstellung, so dass Anwendern lediglich die Möglichkeit bleibt, ihre Algorithmen von Hand an ihr Problem anzupassen. Diese von außen gesteuerte Form der Adaption wird als *statisch* bezeichnet. Sie erfordert einen erheblichen Zeitaufwand, da Parameter untereinander abhängig sein können oder mit der Anzahl der Generationen bzw. mit der Konvergenz der Population variieren. Dem entsprechend groß ist der zugehörige Suchraum, was die statische Adaption sehr fehleranfällig macht. Verfahren, bei denen der Algorithmus selbst Veränderungen gegenüber den ursprünglichen Einstellungen vornimmt, werden als *dynamisch* bezeichnet.

In letzter Zeit ist die Anzahl der Veröffentlichungen im Bereich Adaption wesentlich gestiegen. Um dieser Tatsache gerecht zu werden, haben Hinterding, Michalewicz und Eiben [Hin94] [EHM99] eine Klassifikation der Adaptionsformen in Bezug auf verschiedene Gesichtspunkte erarbeitet, auf die in diesem Kapitel mehrheitlich Bezug genommen werden wird. In den oben genannten Artikeln finden sich auch eine Vielzahl von Verweisen auf adaptive Verfahren, von denen hier nur einige – dafür ausführlicher – vorgestellt werden sollen. Auf verschiedene weitere Verfahren, die in anderen Kapiteln dieses Buches bereits beschrieben wurden, wird an entsprechender Stelle verwiesen.

Prinzipiell lässt sich Adaption in drei Gruppen teilen (vergleiche auch [Mic96]):

- Repräsentation der Chromosomen,
- Operatoren und
- Kontrollparameter.

Unter den ersten Punkt fallen alle Adaptionsvarianten, bei denen die Form der Chromosomen sich verändert. Das kann sowohl eine Veränderung in der Anzahl der Gene als auch das Mitführen eines Crossover-Templates mit sich anpassenden Einträgen bedeuten. Der zweite Punkt beinhaltet Verfahren, bei denen sich die Art und/oder die Häufigkeit des verwendeten Crossovers oder der Mutationsform im Laufe der Zeit verändert und sich automatisch auch an wechselnde Bedingungen anpasst. Die letzte Gruppe betrifft die eigentlichen Kontrollparameter. Enthalten sind hier Adaptionsverfahren, die die Mutations- und Crossoverhäufigkeiten – beispielsweise in Abhängigkeit von der Generation oder der Gesamtfitness der Population – anpassen.

Die oben gegebene Grobeinteilung wurde von Hinterding et. al. überarbeitet und wesentlich erweitert, um auch eine bessere Definition für Begriffe wie „Adaption", „Self-Adaption" (Selbst-Adaption) liefern zu können, die bis zu diesem Zeitpunkt nicht vorlag. Bei dieser Klassifikation wird zweidimensional vorgegangen, indem einerseits zwischen dem dynamischen Adaptionsmechanismus und andererseits dem Adaptionslevel unterschieden wird. Die Adaptionsmechanismen, auf die zuerst eingegangen werden soll, umfassen die folgenden drei Ausprägungen (Typen):

1. Deterministisch.
2. Adaptiv.
3. Selbst-Adaptiv.

Eine deterministische dynamische Adaption liegt dann vor, wenn die vorgenommenen Veränderungen nach einem vorher festgelegtem Schema ablaufen, ohne dass der Verlauf der Optimierung durch den evolutionären Algorithmus darauf Einfluss nimmt. Relativ häufig findet man diese Adaptionsform bei Mutationsoperatoren, die in Abhängigkeit von der Anzahl der Generationen definiert wurden. Die in Abschnitt 6.6 vorgestellte Mutationsvariante Non Uniform Mutation gehört zu den komplexeren derartigen Operatoren. Ebenfalls in diese Gruppe fällt die Fitnessfunktion mit Nebenbedingungen von Joines und Houck [JH94], die bereits in Abschnitt 6.3.6 präsentiert wurde. In ihr hängt der Penalty-Wert für die Nebenbedingungen von der Anzahl der durchlaufenen Generationen ab.

Um eine adaptive dynamische Adaption handelt es sich immer dann, wenn Richtung oder Ausmaß der Anpassung vom Verlauf der Optimierung abhängen. Ein typischer Vertreter dieser Gruppe ist die 1/5-Erfolgsregel der von Rechenberg [Rec73] und Schwefel [Sch75] entwickelten Evolutionsstrategien, bei denen die Mutationsrate von der Anzahl erfolgreicher Verbesserungen abhängt (siehe Abschnitt 7.1). Auch Penalty-Funktionen, deren Änderung beispielsweise von der Fitness des besten Chromosoms oder der Gesamtfitness abhängt, gehören zu dieser Gruppe.

Die adaptiven dynamischen Adaptionsverfahren zeichnen sich dadurch aus, dass die verwendeten Chromosomen nicht nur die zur Lösung des Problems benötigten Informationen enthalten, sondern darüber hinaus auch einige Parameter des evolutionären Algorithmus. Diese kodierten Parameter werden gemeinsam mit dem Hauptteil des Chromosoms den evolutionären Operatoren unterworfen und entwickeln sich dem entsprechend ebenfalls weiter. Man hat es hier also mit einer Evolution der Evolution zu tun, bei der die Chromosomen neben den eigentlichen zu optimierenden Parametern einen Teil der sie betreffenden Algorithmenparameter enthalten. Ein Beispiel dafür ist das Punctuated Crossover (Abschnitt 6.5), bei dem jedes Chromosom eine Auflistung der zulässigen Crossoverpunkte mit sich führt.

Die zweite Achse der Klassifizierung wird von den folgenden sechs Adaptionskomponenten gebildet, bei denen es sich um eine Erweiterung der oben gegebenen Aufteilung von Michalewicz handelt:

1. Adaption der Repräsentation.
2. Adaption der Bewertungsfunktion.
3. Adaption der evolutionären Operatoren.
4. Adaption der Selektionsstrategie.
5. Adaption der Ersetzungstrategie.
6. Adaption der Populationsgröße.

Die Repräsentationsadaption umfasst dieselben Punkte wie in der oben gegebenen Definition, einschließlich der Anpassung der Anzahl der pro Gen verwendeten Bits (z.B. bei ARGOT [Sch87a]) oder einer flexiblen Anzahl von Genen pro Chromosom, wie sie beim Messy-GA (Abschnitt 7.4.1) oder bei ROGENA (Abschnitt 6.8) zu finden sind.

Zur Gruppe der Bewertungsfunktionsadaptionen gehören alle Algorithmen, die mit sich anpassenden Fitnessfunktionen arbeiten. Das ist immer dann der Fall, wenn in einer Fitnessfunktion verwendete Penalty-Faktoren ihren Wert beispielsweise in Abhängigkeit vom Erfüllungsgrad der Randbedingung innerhalb der Population ändern oder die Fitnessfunktion durch das Besetzen neuer Nischen bei Nischenverfahren angepasst wird. Auch Co-Evolution (siehe Abschnitt 6.7.3), bei der mehrere Populationen sich in Abhängigkeit voneinander entwickeln, können dieser Adaptionsform zugerechnet werden, falls die Fitness der Chromosomen der einen Population vom Erfolg der anderen Population abhängt.

Die Anpassung von Crossover und Mutation, sei es in der Art oder der Anwendungswahrscheinlichkeit, gehört mit zu den besonders früh entwickelten Adaptionsformen. Einerseits fallen hierunter Veränderungen der Mutationswahrscheinlichkeit hervorgerufen durch die Qualität der Population, andererseits aber deterministische Verfahren wie bei der Non Uniform Mutation. Typische Vertreter dieser Adaptionsform sind auch Evolutionsstrategien (siehe Abschnitt 7.1), bei denen jedes Chromosom zusätzliche Gene mit einer individuelle Schrittweite für die Mutation aufweisen kann. Denkbar sind auch Algorithmen, bei denen verschiedene Mutations- oder Crossoverarten mit unterschiedlicher Anwendungswahscheinlichkeit zur Auswahl stehen, wobei die Anwendungswahrscheinlichkeiten erfolgsbedingt verändert werden (z. B. Davis [Dav91]). Auch Chromosomen, die zumindest teilweise Parameter für die anzuwendenden Operatoren beinhalten und analog zum eigentlichen Chromosom verändern, gehören in diese Gruppe. Dasselbe gilt für das Punctuated Crossover von Schaffer und Morishima.

Adaption der Selektions- und der Ersetzungsstrategie decken sehr ähnliche Bereiche ab. Selektionsadaption jedoch liegt immer dann vor, wenn die Überlebenswahrscheinlichkeit beispielsweise vom Alter der Chromosomen abhängt (Anzahl der unverändert überlebten Generationen), während eine Ersetzungsstrategie immer dann vorliegt, wenn ein Chromosom direkt mit seinem Nachfolger verglichen wird. Ein Beispiel für eine adaptive Ersetzungstrategie ist die Boltzmannselektion (siehe Abschnitt 6.4.3), bei der eine „Temperaturkonstante" nach einem festen Zeitplan gesenkt wird. Beide Adaptionsformen sind nicht sehr verbreitet.

Wenig anders verhält es sich mit der Adaption der Populationsgröße, da auch in diesem Bereich wenige Untersuchungen durchgeführt wurden. Die Anpassung der Anzahl der Chromosomen im Falle von miteinander konkurrierenden parallelen Populationen, hervorgerufen durch Belohnung und Bestrafung in Abhängigkeit von der Gesamtfitness, ist eine derartige Variante. Es wurden auch Verfahren getestet, bei denen Chromosomen erst in Abhängigkeit von ihrem Lebensalter aus der Population entfernt wurden [AMM94].

Im Folgenden werden verschieden Beispiele für Algorithmen bzw. Operatoren mit Adaption gegeben, die bisher unerwähnt geblieben sind. Eine sehr umfangreiche Sammlung von Beispielen und Literaturangaben findet sich in [EHM99].

Ein Vertreter der deterministischen dynamischen Repräsentationsadaption ist das von Schraudolph und Belew [SB92] entwickelte Prinzip des „*Dynamic Parameter Encoding*" (DPE). Bei dieser Kodierung arbeitet der Algorithmus auf einem Chromosomen mit fester Länge, wobei jeder für die Lösung notwendige reellwertige Parameter durch ein Gen mit l Bits als Integerwert kodiert wird. Dieser Integerwert wird über eine Abbildung in ein reelles Intervall in eine reelle Zahl umgewandelt. Prinzipiell ist die Genauigkeit der Annäherung an die optimalen Parameter durch die Anzahl der Bits beschränkt. Schraudolph und Belew gehen jedoch davon aus, dass in der Anfangsphase eines evolutionären Algorithmus hauptsächlich die führenden Bits optimiert werden, was die Genauigkeit in weniger signifikanten Bereichen stark einschränkt. Dem entsprechend optimieren sie die Population bis zum Erreichen eines Mindestfitnesswertes, um dann die bisher signifikanten Bits für jeden Parameter zu entfernen. Der durch die weggefallenen Bits repräsentierte Wert wird gespeichert. Das Suchintervall, das von den Genen

abgedeckt wird, wird durch dieses Vorgehen immer weiter eingeengt, bis die gewünschte Genauigkeit erreicht ist (DPE Zoom).

Von Spears [Spe95] wurde eine selbstadaptive dynamische Operatorenadaption entwickelt, bei der die Information über die anzuwendende Crossoverform in den Chromosomen mitgeführt wird. Zur Auswahl standen dabei Two-Point und Uniform Crossover, um zwischen möglichst gegensätzlichen Crossoverformen wählen zu können. Als Maß für die Gegensätzlichkeit dient dabei die Zerstörungswahrscheinlichkeit von Schemata, die für Two-Point-Crossover gering, für Uniform Crossover jedoch sehr hoch ist. Der Hintergedanke dabei ist, dass es bis heute keine klaren Richtlinien zur Auswahl problemspezifisch möglichst effizienter Operatoren gibt, und es sich dem entsprechend empfiehlt, auch Art und Anwendungshäufigkeit der Operatoren zu optimieren. Spears verwendete zur Implementation des Verfahren eine so genannte *1-Bit-Adaption*, indem er an die eigentlichen Chromosomen ein zusätzliches Bit anfügte. Der Wert dieses Bit entschied dann bei einem für das Crossover ausgewählten Chromosomenpaar über die Crossoverart: Zweimal der Eintrag 1 stand für das Two-Point-Crossover, zweimal der Eintrag 0 für die Anwendung von Uniform Crossover, und bei gemischten Einträgen entschied der Zufall. Wie bei selbstadaptiven Verfahren üblich, wurden die evolutionären Operatoren auch auf das letzte Gen angewendet. Von Tuson und Ross [TR98] wurde eine ähnliche Chromosomenkonstruktion entwickelt, die neben der Crossover- auch die Mutationswahrscheinlichkeit enthält. Darüber hinaus wurde ein Gen angehängt, das eine Wahrscheinlichkeit dafür enthält, dass der Crossover-Operator benutzt wird (ansonsten Mutation).

Ebenfalls um eine Operatoren-Adaption – allerdings um eine adaptive – handelt es sich bei dem von Corne, Ross und Fang [CRF94] für Timetabling entwickelten *COBRA*-Algorithmus (Cost Operator-Based Rate Adaption). Wie bei nicht-adaptiven Algorithmen üblich, werden vor dem Start des Verfahrens für jeden verfügbaren Operator Anwendungswahrscheinlichkeiten festgelegt, die sich zu 1 addieren müssen. Die Menge dieser Wahrscheinlichkeiten bleibt über den gesamten Verlauf der Optimierung dieselbe, die einzelnen Werte werden jedoch im Laufe der Zeit unterschiedlichen Operatoren zugewiesen. Nach einer festen Anzahl von durchgeführten Bewertungen wird für jeden Operator der Faktor von erzeugten Verbesserungen (Gewinn) zur aufgewendeten Rechenzeit (Kosten) kalkuliert und die verfügbaren Anwendungswahrscheinlichkeiten werden gemäß des Ergebnisses neu auf die Operatoren verteilt.

Ein eher ungewöhnliche Variante eines evolutionären Algorithmus mit einem selbstadaptiven „Mutationsverfahren" ist der von Harik, Lobo und Goldberg entwickelte *Compact Genetic Algorithm* (cGA [HLG97]). Konzipiert wurde er zur zeitsparenden Simulation einfacher genetischer Algorithmen (binäre Kodierung) mit Uniform Crossover bei deutlich verringertem Speicheraufwand. Statt eine ganze Population zu erzeugen, wird der Zustand der Population durch eine Wahrscheinlichkeitsverteilung über der Menge der Lösungen wiedergegeben. Es wird also bei Chromosomen der Länge l zu Beginn des Simulationslaufs ein reeller Wahrscheinlichkeitsvektor der Länge l erzeugt, dessen Einträge alle den Wert 0.5 aufweisen. Dieser Wert gibt für jedes Bit die Wahrscheinlichkeit an, dass sich bei einem Chromosom an dieser Stelle eine 1 befindet. Zu Beginn jedes Durchlaufs werden zufällig gemäß des Wahrscheinlichkeitsvektors $popsize$ Chromosomen erzeugt und bewertet. Danach werden die Chromosomen paarweise verglichen. Der

Wahrscheinlichkeitsvektor wird jeweils bitweise angepasst, indem der k-te Eintrag um einen festen Faktor erhöht wird, wenn das bessere der beiden betrachteten Chromosomen dort eine 1 aufweist. Ansonsten wird der Eintrag um denselben Faktor erniedriegt. Dieses Verfahren wird solange durchgeführt, bis der Wahrscheinlichkeitsvektor konvergiert ist, also nur noch Einträge der Form 0 oder 1 aufweist. In diesem Zustand entspricht der Wahrscheinlichkeitsvektor direkt der optimalen Lösung.

6.7.3 Co-Evolution

Unter dem Begriff *Co-Evolution* versteht man eine Spezialform evolutionärer Algorithmen, in der statt auf einer Population auf zwei oder mehr Populationen gearbeitet wird. Die Anzahl der Populationen ist üblicherweise für den Verlauf der Optimierung festgelegt ist. Zwischen den Populationen findet kein Austausch von Informationen oder Lösungen statt. Die Theorie auf diesem Spezialgebiet ist nicht sehr weit fortgeschritten, es lassen sich jedoch einige Aussagen über Formen und auftretene Probleme machen. Unterschieden werden können die co-evolutionären Algorithmen (CEA) nach der Art des Miteinanders der Populationen in

- kooperative CEA (cooperative CEA))und
- nicht-kooperative CEA (non-cooperative CEA).

Unter einem kooperativen co-evolutionären Algorithmus (cCEA) versteht man eine Algorithmenform, deren Populationen jeweils auf Teilziele des eigentlich zu lösendem Problem angesetzt wurden. Ein Beispiel dafür aus dem Bereich der Funktionsoptimierung wurde von Potter und De Jong [PDJ94] vorgestellt. Entgegen des üblichen Verfahrens bei dem alle Funktionsparameter in einem Chromosom kodiert wurden, verwendet das getestete Verfahren ebenso viele Populationen, wie Parameter in der Funktion zu optimieren waren. Jede Population beschränkte sich also auf die Optimierung eines einzelnen Parameters, so dass ein zusätzlicher Integrationsschritt notwendig wurde, der aus den Ergebnissen der Populationen eine Gesamtlösung erstellte. Verwendet wurde für jede Population der gleiche normale genetische Algorithmus (siehe Abschnitt 4). Die Fitness für jede Teillösung berechnete sich über die Gesamtfitness der Funktion unter Hinzunahme der Menge der besten Teillösungen aus den anderen Populationen. Da die damit erhaltenen Ergebnisse nicht den Erwartungen entsprachen, wurde eine weitere Algorithmenvariante entwickelt, bei der zur Berechnung der Fitness nicht nur die besten Teillösungen, sondern auch zufällig gezogenen Teillösungen herangezogen wurden. Der Vorteil dieser Algorithmenart ist darin zu sehen, dass bei der Berechnung der Gesamtlösung durch Entwicklung der Teillösungen keine Probleme mit *parasitären* Genen auftreten. Darunter versteht man Gene, die die in einer frühen Phase zufällig in Chromosomen vorhanden sind, die durch einige andere Allele überdurchschnittlich gute Lösungen darstellen. Diese Chromosomen vermehren sich zu Beginn sehr stark, was letztendlich auch den weniger guten, aber immer mit repräsentierten parasitären Allelen zugute kommt. Im weiteren Verlauf der Optimierung ist es dann sehr schwierig, die Werte der parasitären Gene so zu verändern, dass sie die Lösung weiter verbessern.

Bei Verwendung eines cCEA können sich zumindest Lösungen für Parameter, die unterschiedlich stark in die Gesamtlösung einfließen, nicht mehr auf diese Weise verbinden.

Die Gruppe der nicht-kooperativen co-evolutionären Algorithmen lässt sich nochmals in zwei Gruppen aufspalten, nämlich in

- prädative CEA's (Predator-Prey CEA) und
- konkurrierende CEA's (competing CEA).

Ein prädativer CEA zeichnet sich dadurch aus, dass er zwei Populationen aufweist, von denen die eine Strategien zur Lösung eines Problems und die andere Testfälle enthält. Die Fitness der Chromosomen aus den beiden Populationen verhält sich invers zueinander (*invertierte Fitness*). Sie wird üblicherweise in der Form berechnet, dass eine Strategie umso fitter ist, je mehr Testfälle sie korrekt verarbeitet, während die Fitness eines Testfalles über die Anzahl der Strategien berechnet wird, die den Fall nicht lösen konnten. Diese Form eines CEA wird häufig bei der Entwicklung von Gewinnstrategien bei Spielen und bei der Entwicklung von Zellularautomaten verwendet. Die Fitness für beide Populationen wird üblicherweise nicht für die gesamte gegnerische Population berechnet, sondern es wird jeweils eine kleinere Testgruppe gebildet.

Eines der frühesten Beispiele dieser Algorithmenform stammt von Hillis [Hil92], der einen prädativen CEA verwendete, um Sortierverfahren für 16 Zahlen in Netzwerkform zu entwickeln (Sorting Networks). Die Aufgabe dieser Verfahren war die Entwicklung von Regeln, um korrekt und möglichst effizient (mit wenigen Vergleichen) einen Datensatz zu sortieren. Hillis testete vor der Verwendung eines CEA auch einen genetischen Algorithmus mit sich nicht verändernder Menge von Testbeispielen, war aber mit dem Resultat nicht zufrieden, da er hinter den Ergebnissen der von Hand entwickelten Verfahren anderer Wissenschaftler lag. Für seinen Ansatz mit einem CEA ließ er von einer Population neue Verfahren und von der anderen neue Testbeispiele entwickeln. Die Fitness eines Sortierverfahrens berechnete sich über die Anzahl korrekt verarbeiteter Tests, während die Fitness der Tests sich aus der Prozentzahl der Sortierverfahren berechnet, die das jeweilige Testbeispiel nicht lösen konnten.

Relativ genau untersucht wurden auch die Zellularautomaten zum Zwecke der Klassifizierung eines binären Strings in überwiegend Nullen oder Einsen [JP98]. Bei dieser Art von Problem lässt sich leicht verdeutlichen, dass es nicht genügt, eine Population von Testfällen blind zu entwickeln. Sonst kann es vorkommen, dass die aktuellen Testfälle zwar in einem ausgwogenem Verhältnis bzgl. der Schwierigkeit zu den Strategien stehen, dass aber bereits vorher gelöste Probleme „vergessen" wurden. Das kommt insbesondere dann vor, wenn die Testfälle nicht aufeinander aufbauen, sondern völlig unterschiedliche Anforderungen stellen. In diesem Fall können sich die beiden Population zyklisch zueinander verhalten, durchlaufen also immer wieder bereits geklärte Fälle.

Die Aufgabe der Population der Testfälle ist es also, eine stabile Menge von Tests zu entwickeln, die sowohl schwierig, als auch deutlich voneinander verschieden sind. Zu diesem Zweck wurden von Rosin und Belew zwei neue Verfahren entwickelt [RB95, RB97], die mit den in Abschnitt 6.7.1 erläuterten Nischentechniken verwandt sind. Es handelt sich dabei um die beiden folgenden Strategien:

Competitive Fitness Sharing: Die Fitness einer Strategie ist umso höher, je niedriger die Anzahl der konkurrierenden Strategien ist, die dieselben Testfälle lösen können:

$$fit(c) = \sum_{j \in X} \frac{1}{N_j}$$

mit X Indexmenge der gelösten Testfälle, N_j Anzahl Strategien, die Testfall j lösen.

Shared Sampling: Die Testfälle für die nächste Generation werden so ausgewählt, dass alle Nischen abgedeckt werden. Dabei wird wie folgt vorgegangen:

1. Initialisiere die Testzähler $beat[i]$ aller Strategien i der letzten Generation auf Null und setze die Menge T aller gewählten Tests auf die leere Menge.
2. Berechne für alle bisher nicht in T enthaltenen Tests j die Fitness durch $fit_{sample}[j] = \sum_{i \in Y} \frac{1}{1+beat[i]}$, wobei Y die Menge der Indices der von Test j geschlagenen Strategien ist.
3. Füge den Test k mit maximalem $fit_{sample}[k]$ zur Testgruppe T hinzu.
4. Erhöhe für alle Strategien i, die von Test k geschlagen wurden, den Wert $beat[i]$ um 1.
5. Führe Schritte 2 bis 4 durch, bis die Testgruppe voll ist.

Hinter beiden Berechnungsvarianten steckt der Versuch, entweder die Bandbreite der Strategien, oder die Bandbreite der Testfälle zu erhöhen, indem Chromosomen aus möglichst unterschiedlichen Bereichen gewählt werden. Die von Rosin und Belew durchgeführten Tests bezüglich verschiedener Spielestrategien (Tic Tac Toe, NIM, GO) zeigten bei Verwendung der neuen Varianten deutliche Verbesserungen. Inwieweit diese Verbesserungen durch das neue Competitive Fitness Sharing, von der Co-Evolution selbst oder von einer Kombination von beidem verursacht wurden, haben Werfel, Mitchell und Crutchfield [WMC00] untersucht, da die Anwendung einer nicht modifizierten Co-Evolution bei Zellularautomaten zu schlechteren Ergebnissen führte als die Verwendung eines normaler evolutionären Algorithmus. Sie verwendeten allerdings die von Juille und Pollack entwickelte Variante der Sharing-Funktion (Resource Sharing genannt) und fanden heraus, dass auch die Verwendung der Sharing-Funktion ohne Co-Evolution zu einer deutlichen Verbesserung in der Performance führte. Durch die Hinzunahme von Co-Evolution verbesserten sich die Ergebnisse nochmals.

Angewendet wurden die Prinzipien der Co-Evolution bisher mit einigem Erfolg in den Bereichen Zellularautomaten, Prozess- [Par96] und Roboterkontrolle [PDJG95], sowie Spielestrategien. Unter dem letzten Punkt findet man auch Implementierungen konkurrierender CEA's. Sie werden so implementiert, dass eine Population versucht, die beste Gewinnstrategie zu entwickeln, während die andere Population nach einer Möglichkeit sucht, den Sieg zu verhindern. Beispiele dafür wurden von Rosin und Belew entwickelt [RB95].

6.8 Beispiel ROGENA: Evolutionäre Algorithmen zur Optimierung von Flugrouten

In diesem Abschnitt soll ein Anwendungsbeispiel für evolutionäre Algorithmen aus dem Bereich des Luftverkehrs ausführlich vorgestellt werden. Es stellt eine wesentliche Erweiterung des bereits vorher verwendeten Routenfindungsproblems (siehe Beispiel 4.2: Schnellster Weg) dar. Der Anwendungsbereich ist der Anflug auf die Landebahn eines Flughafens.

Problembeschreibung:
Flugzeuge bewegen sich im Luftraum üblicherweise auf vorgeschriebenen Standardrouten, da es auf diese Weise für die den Luftraum kontrollierenden Fluglotsen leichter ist, die Sicherheit – das bedeutet im Wesentlichen Kollisionsvermeidung – zu gewährleisten. Zusätzlich dazu bewegen sich Flugzeuge abhängig von ihrer Flugrichtung auf unterschiedlichen Höhen. Dem entsprechend sind die Kreuzungspunkte zwischen verschiedenen Standardrouten die einzigen schwierigeren Stellen, vorausgesetzt, die Flugzeuge halten auf den Standardrouten den vorgeschriebenen Mindestabstand ein. Durch diese Konstruktion ist zwar der Sicherheitsstandard im Flugverkehr hoch, es wird jedoch nur ein relativ kleiner Anteil des Luftraumes genutzt. Zuwenig für den fortwährend ansteigenden Flugverkehr.

In Abbildung 6.11 ist ein Ausschnitt des oberen Luftraumes im Bereich des Flughafens Frankfurt zu sehen. Die dunklen Linien zeigen die Standard-Flugrouten, die grau gepunkteten Linien die von den Flugzeugen tatsächlich geflogenen Routen. Deutlich ist zu erkennen, dass weite Bereiche des Luftraumes ungenutzt bleiben.

Lösungsidee:
Um für zukünftige Verkehrssteigerungen gewappnet zu sein, muss auch der Luftraum zwischen den Routen genutzt werden. Das bedeutet für die Lotsen einen Anstieg in der Arbeitsbelastung, da sich frei bewegende Flüge wesentlich schwieriger zu kontrollieren und zu koordinieren sind als auf festen Strecken geführte Flugzeuge. Es besteht also die Notwendigkeit, ein Lotsenunterstützungstool zu entwickeln, das den Fluglotsen zumindest die Konstruktion sicherer (konfliktfreier) und effizienter Routen zwischen Eintritts- und Austrittspunkt in den von den Lotsen kontrollierten Bereich des Luftraumes abnimmt. Zu diesem Zweck wurde das Tool ROGENA (free ROuting with GENetic Algorithms) entwickelt [Ger94, Ger95]. Es verwendet evolutionäre Algorithmen, um eine Population von Flugrouten gemäß der vorgegebenen Randbedingungen wie

- Ein- und Austrittspunkt im kontrollierten Bereich,
- Eintrittszeitpunkt,
- Flugeigenschaften des benutzten Flugzeugtyps,
- Bewegung weiterer Flugzeuge, und
- Sperrgebiete (aufgrund von militärischer Nutzung oder schlechtem Wetter).

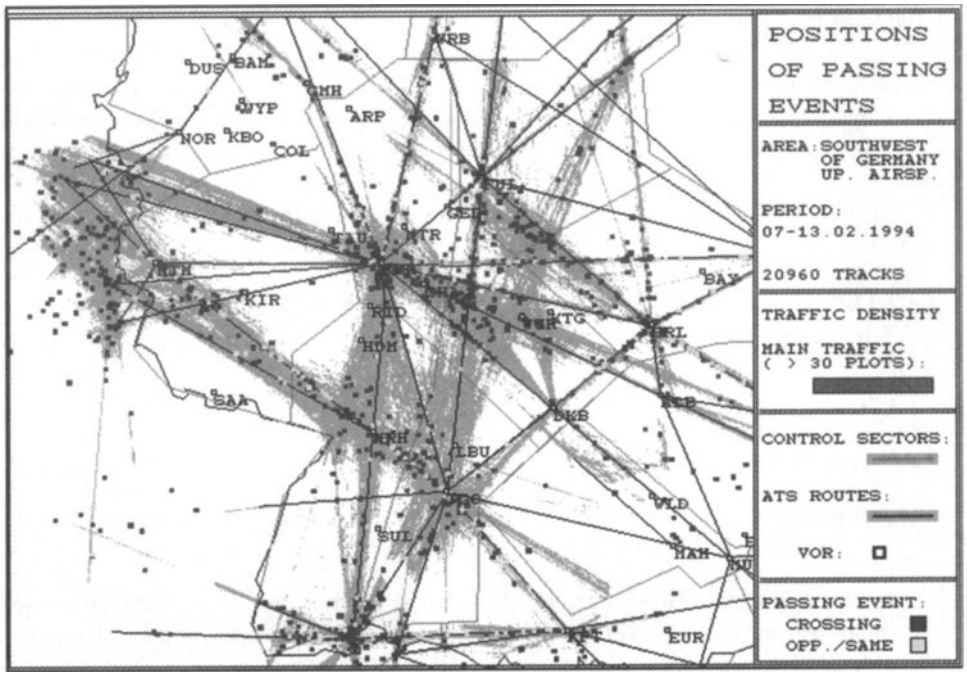


Abbildung 6.11 Luftraum über Deutschland [SH94]

Umsetzung

Die Flugrouten wurden als Sequenz von Vektorabschnitten im Raum definiert, wobei jedes Gen der Chromosomen einen zu überfliegenden Wegpunkt darstellt. Die Chromosomen wurden mit variabler Länge implementiert, jedes Gen besteht dabei aus dem Quadrupel der Koordinaten im Raum und der Überfluggeschwindigkeit dieses Wegpunktes. Die Struktur ist demnach haploid und die Kodierung reell (siehe auch Bild 6.12). Es wurde ein Ausschnitt von 200 mal 200 Nautischen Meilen (NM) mit einer Höhe von 0 bis 10000 Fuß (ft) aus dem Luftraum betrachtet. Die notwendigen Daten und Berechnungsformeln über die Flugeigenschaften bzw. das Flugverhalten von Flugzeugen wurden der BADA-Datenbank der EUROCONTROL entnommen [Byr95]. Um die Sicherheit zu gewährleisten, wurden in dem den Flughafen umgebenden Luftraum Mindestabstände zwischen Flugzeugen von 5 NM angenommen. Beim Endanflug auf den Flughafen variieren die einzuhaltenden Mindestabstände in Abhängigkeit vom Gewicht der beiden beteiligten Flugzeuge zwischen 2.5 und 6 NM. Jede Unterschreitung des Sicherheitsabstandes wird als Konflikt gewertet. Der evolutionäre Algorithmus zur Routengenerierung fängt nur dann an zu arbeiten, wenn es zu Konflikten zwischen den Routen eines neu hinzugekommenen und eines bereits fliegenden Flugzeuges kommt.

Evolutionärer Algorithmus
Der evolutionäre Algorithmus wurde auf der Basis eines modGA (siehe Abschnitt 7.4.2) entwickelt. Er arbeitet mit einer Population von 60 Chromosomen, die nach einem speziellem System um die Originalroute herum ausgewürfelt wurden. In jeder Generation wurden 20 Chromosomen unverändert übernommen, darunter die 5 besten Chromosomen.

Für die Fitnessfunktion mussten verschiedene Randbedingungen in Betracht gezogen werden, wie z.B.

- einzuhaltender Sicherheitsabstand zu den anderen Flugzeugen,
- kein Durchfliegen von gesperrtem Luftraum,
- Länge der Flugroute,
- Pünklichkeit,
- mmöglichst geringe Abweichung bei Sinkverfahren von der nach BADA berechneten optimalen Sinkrate und
- keine zu spitzen Winkel zwischen den Vektorabschnitten, da die tatsächlich geflogenen Routen sonst aufgrund des Kurvenflugverhaltens der Flugzeuge zu sehr von den Berechnungen abweichen.

Abweichungen von den obigen Randbedingungen gehen mit einem Gewichtungsfaktor in die Formel für die Fitness ein. Erschwert wurde die Schätzung korrekter Gewichtungsfaktoren durch die für den Nutzer bestehende Möglichkeit, seine persöhnliche Sicht der Wichtigkeit einzelner Faktoren über Schieberegler auf einen Wert zwischen 0 und 100 Prozent einzustellen.

Als Selektionsoperator wurde eine Variante der Methode „Stochastic Sampling with Replacement" verwendet, wobei allerdings Chromosomen, die wesentlich über einem Fitnessschwellwert lagen, nicht in die Berechnung der Selektionswahrscheinlichkeit einbezogen wurden. Die verwendete Selektionsform ist demnach dynamisch und diskriminierend. Zusätzlich erniedrigt sich auch der Schwellwert mit der Anzahl der Generationen (abhängig vom Fitnessdurchschnitt in der Population), um den Selektionsdruck mit der Anzahl der Generationen zu verstärken.

Mit Hilfe des Selektionsoperators wurden 20 Chromosomen ausgewählt, wobei jeweils zwei nacheinander gezogene Chromosomen mittels Two-Point-Crossover rekombiniert wurden (siehe Bild 6.12). Diese Form des Crossovers wurde gewählt, um Teilstücke von Routen auszutauschen und auf diese Weise die nützlichen Anteile verschiedener Routen zu kombinieren. Dadurch, dass auch Flughöhen und Geschwindigkeiten ausgetauscht werden, ist es notwendig, nach jedem Crossover einen Reparaturmechanismus auf die beiden neuen Chromosomen anzuwenden. Da eine Kombination von Sink- und Steigvorgängen in einer Route nicht ökonomisch ist, werden die neuen Höhen so an die bereits im Originalchromosom enthaltenen Höhen angepasst, dass die Flughöhen monoton fallen. Ebenso verhält es sich mit den Geschwindigkeiten, bei denen eine Abfolge von Beschleunigungs- und Bremsphasen vermieden werden soll.

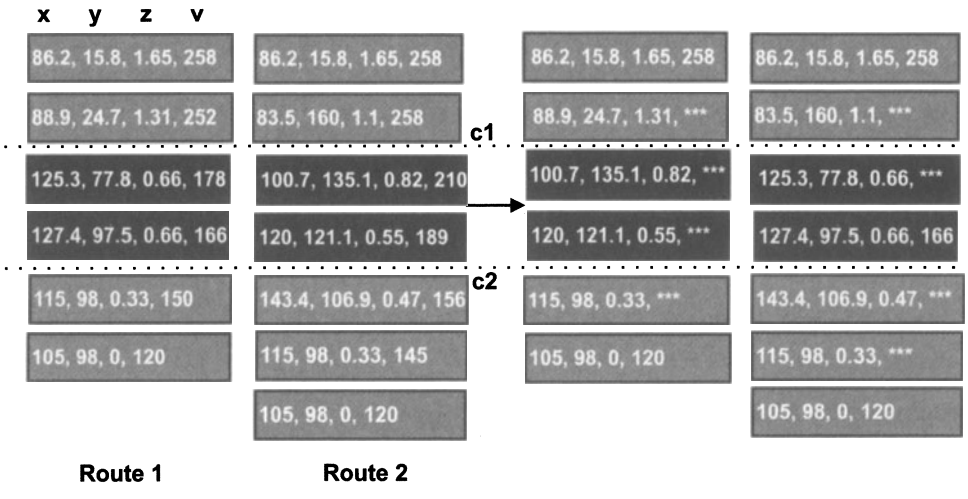


Abbildung 6.12 Cromosomenstruktur mit Two-Point-Crossover bei ROGENA

Bei weiteren 20 Routen wurde ein Mutationsoperator angewendet. Dafür standen verschiedene Operatoren zur Auswahl, nämlich eine völlig zufällige Koordinatenveränderung, eine mittlere Veränderung in einen Punkt aus der Umgebung, oder eine geringfügige Veränderung in einen in unmittelbarer Nähe liegenden Punkt. Es wurde auf diese Weise eine Art Hillclimbing integriert, um insbesondere den in der Endphase notwendigen geringeren Veränderungen entgegenzukommen. Es handelt sich also bei der verwendeten Mutation um eine Kombination von lokaler und globaler Optimierung. Über diese Gen-Mutationen hinaus wurde mit geringer Wahrscheinlichkeit auch die Anzahl der Gene verändert, wobei diese Wahrscheinlichkeit von der Anzahl der vorhandenen Gene abhängt, um das Ausarten in sehr lange Chromosomen zu verhindern. Das zu streichende Gen, bzw. der Punkt, an dem ein neues Gen eingefügt werden sollte, wurden zufällig bestimmt. Der neue Wegpunkt wurde in der Nähe der ihn umgebenden Punkte platziert und die Höhe entsprechend ausgewürfelt.

Vergleiche der Ergebnisse von Simulationsläufen unter Verwendung realer Daten mit den ursprünglichen geflogenen Routen zeigten eine deutliche Verkürzung der Länge der Routen, ohne dabei Konflikte mit anderen anfliegenden Luftfahrzeugen zu erzeugen.

Bild 6.13 zeigt den zur Optimierung gehörenden Bildschirm von ROGENA. Mögliche Routen sind als helle Linien dargestellt, wobei die dicke Linie die zur Zeit beste Route unter Berücksichtigung aller Nebenbedingungen darstellt. Die zu optimierende Originalroute ist die direkte Verbindung zwischen Start und Zielpunkt. Das linke Display zeigt eine Zeitleiter mit den Landezeiten aller Flugzeuge, wobei für das zu optimierende Flugzeug auf der linken Seite eine Linie die Differenz zum geplanten Landezeitpunkt zeigt. Das untere Display zeigt das Höhenprofil der aktuell besten Route. Auf der rechten Seite werden sowohl Informationen über die aktuell beste Route angezeigt, als auch über die in der Population erzeugten Durchschnittswerte angezeigt.

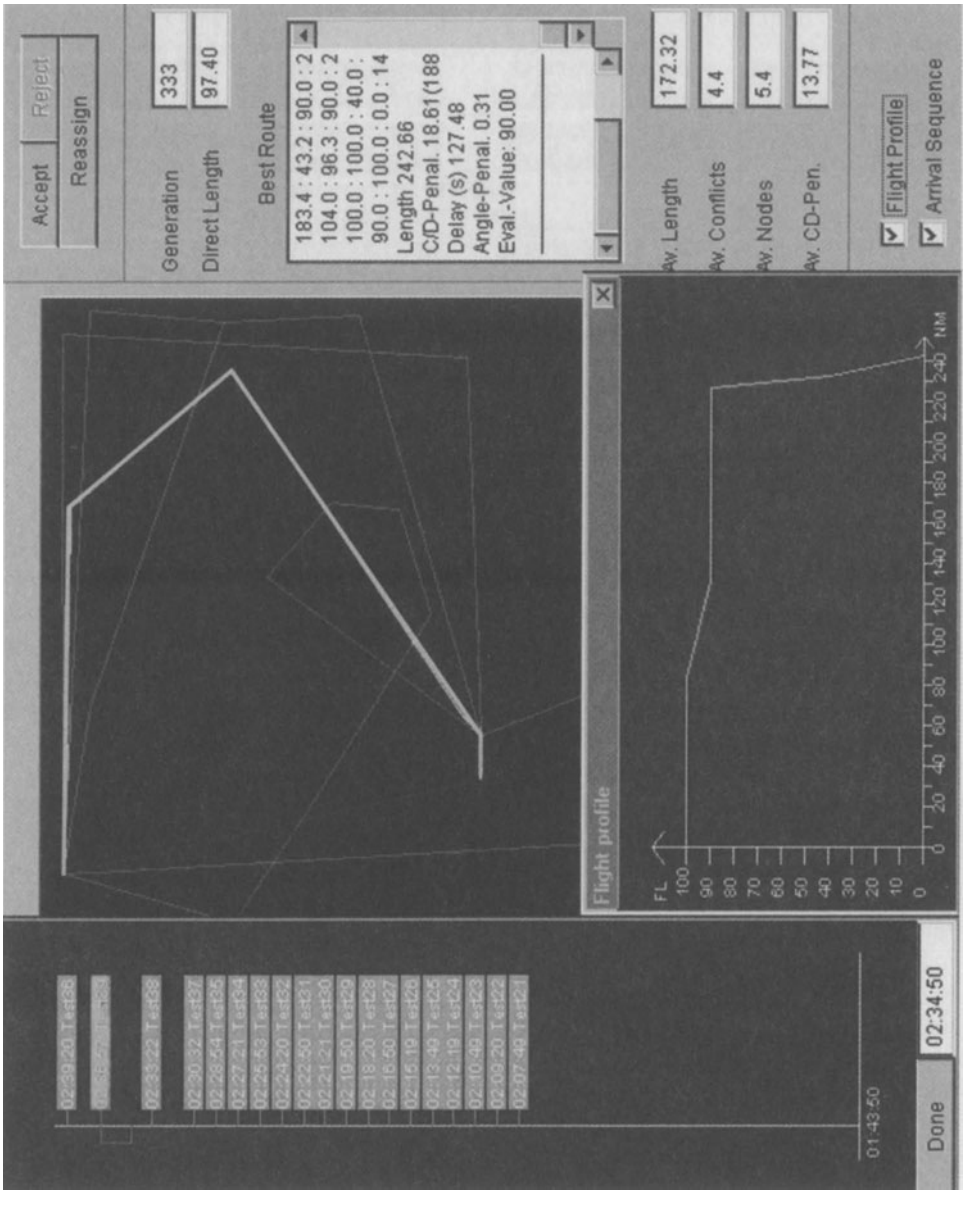


Abbildung 6.13 Optimierungsoberfläche von ROGENA

7 Klassifizierung evolutionärer Algorithmen

Wir haben bereits den kanonischen genetischen Algorithmus kennen gelernt sowie zahlreiche Varianten von genetischen Operatoren und Verfahren wie Selektion, Mutation und Crossover diskutiert. In diesem Kapitel wird eine Klassifizierung evolutionärer Algorithmen beschrieben, die zum Teil auf historischen Entwicklungen beruht. Man sollte die einzelnen Modelle evolutionärer Algorithmen allerdings nicht als konkurriende, sich gegenseitig ausschließende Verfahren betrachten, sondern bei einem realen Optimierungsproblem die geeigneten Komponenten und Techniken der einzelnen Verfahren sinnvoll miteinander kombinieren. Wo eine binäre Kodierung angebracht ist, sollte man sie verwenden. Bei reellen Parametern ist meistens auch eine Fließkommakodierug einer binären Kodierung vorzuziehen. Und wenn sowohl diskrete als auch kontinuierliche Parameter zu optimieren sind, dürfen in einem Chromosom Gene beiden Typs auftreten.

7.1 Evolutionsstrategien

Evolutionsstrategien (*ES*) wurden in den sechziger Jahren von Rechenberg und Schwefel für Optimierungsprobleme aus dem technisch-physikalischen Bereich entwickelt und später z.B. in [Rec73, Sch75] publiziert. Monografien, die sich vorwiegend auf Evolutionsstrategien konzentrieren, sind [Sch95, Bäc96].

7.1.1 Eine einfache Variante der Evolutionsstrategie

Bei den Evolutionsstrategien geht es um die Optimierung einer reellwertigen Zielfunktion $f : \mathbb{R}^n \to \mathbb{R}$ mit n reellen, zu optimierenden Parametern, wobei die Zielfunktion f auch auf einer Teilmenge des $\mathbb{R}^n$ definiert sein darf. Entsprechend bestehen die Chromosomen im einfachsten Fall aus n Genen, die jeweils reelle Werte annehmen können. Die Populationsgröße wird bei Evolutionsstrategien üblicherweise mit μ (d.h. $popsize = \mu$) bezeichnet. Die Startpopulation $P(0)$ wird entweder zufällig, d.h. möglichst gleichmäßig über den Suchraum verteilt, erzeugt oder, falls entsprechendes Vorwissen über das Problem vorhanden ist, werden gezielt einige Chromosomen an den entsprechenden Stellen des Suchraums platziert.

Die Nachfolgepopulation $P(t+1)$ der Population $P(t)$ wird folgendermaßen erzeugt. Die Chromosomen der Generation $P(t)$ generieren insgesamt λ Nachkommen durch Mutation, die wir im Folgenden noch im Detail beschreiben werden. Ein übliches Verhältnis zwischen μ und λ ist etwa eins zu sieben [Sch87b]. Häufige Größenordnungen für μ und λ sind $\mu = 15$ und $\lambda = 105$. Dies würde bedeuten, dass jedes Chromosom sieben Nachfolger erzeugt. Die Mutation eines Chromosoms erfolgt, indem zu dem reellen Wert (Allel) jedes Gens jeweils eine nach einer Normalverteilung mit Erwartungswert 0 und (kleiner) Streuung σ erzeugte (Pseudo-)Zufallszahl addiert wird.

Die Streuung σ, die die Mutationsschrittweite steuert, sollte so klein gewählt werden, dass man mit großer Wahrscheinlichkeit in der „Nähe" des Elternchromosoms bleibt. Die Mutation soll in den meisten Fällen schrittweise kleine Änderungen bewirken und nicht ein neues Chromosom erzeugen, das mehr oder weniger zufällig irgendwo im Suchraum liegt. Die geeignete Wahl des Parameters σ hängt sehr vom Problem ab und kann sich als äußerst schwierig erweisen. Wählt man σ (deutlich) zu groß, weisen die erzeugten Nachfolgerchromosomen eher die Charakteristik einer zufällig generierten Startpopulation auf als Ähnlichkeit zu ihren Eltern. Bei einem zu kleinen σ kann die Anzahl der benötigten Generationen, bis man eine gute Lösung erzeugt hat, und damit die Rechenzeit, beliebig groß werden. Außerdem sollte σ nicht konstant sein, sondern für jedes Chromosom abhängig von der lokalen Struktur der Zielfunktion f gewählt werden. Variiert f sehr stark in der Umgebung eines Chromosoms, ist ein sehr kleiner Wert von σ zu empfehlen. Ist f sehr flach in der Nähe eines Chromosoms, darf σ auch größere Werte annehmen. Aus diesem Grund wird in den meisten Fällen nicht ein fester Wert für σ festgelegt, sondern σ wird individuell für jedes Chromosom während des Evolutionsprozesses permanent neu adaptiert. Auf diese Technik werden wir später noch genauer eingehen. Zunächst betrachten wir die einfache Variante einer Evolutionsstrategie mit festem Wert σ.

Diese einfache Variante der Evolutionsstrategie verwendet nur Mutation und keine Rekombination. Nachdem die λ Nachkommen durch Mutation erzeugt wurden, wird direkt die Selektion angewendet. Bei den Evolutionsstrategien wird nach dem *Eliteprinzip* selektiert, d.h., es werden die μ besten Chromosomen in die Folgegeneration übernommen. Man unterscheidet dabei zwischen der *Plus-* und der *Komma-Strategie*:

Plus-Strategie: Hierbei werden die μ besten Chromosomen aus der Vereinigung der Elternpopulation mit der Menge der durch Mutation erzeugten Nachkommen ausgewählt. Die Nachfolgepopulation wird somit aus $\mu+\lambda$ Chromosomen ausgewählt. Man bezeichnet diese Form der Evolutionsstrategie auch als $(\mu + \lambda)$-ES.

Komma-Strategie: Die Nachfolgepopulation wird allein aus den μ besten Nachkommen gebildet. Die Elternchromosomen werden bei der Selektion nicht berücksichtigt. Diese Art der Evolutionsstrategie wird mit (μ, λ)-ES bezeichnet. Bei der Komma-Strategie muss sichergestellt werden, dass $\lambda > \mu$ gilt, damit genügend viele Chromosomen für die Folgepopulation bei der Selektion zur Verfügung stehen.

Der Vorteil der Plus-Strategie besteht darin, dass die beste gefundene Lösung auf jeden Fall in die Folgepopulation übernommen wird. Der Nachteil besteht darin, dass die Plus-Strategie eine höhere Gefahr der vorzeitigen Konvergenz in ein lokales Optimum birgt. Die Nachkommen einer Lösung, die zumindest in einer frühen Generation deutlich besser als die anderen noch relativ zufälligen Lösungen ist, können die gesamte Population bei der Plus-Strategie sehr schnell dominieren, so dass der Suchraum nur noch in der Nähe dieser einen Lösung durchforstet wird. Diese Gefahr ist bei der Komma-Strategie geringer. Man sollte bei der Anwendung der Komma-Strategie die beste bisher gefundene Lösung in jedem Fall speichern, auch wenn diese Lösung nicht mehr an der Erzeugung von Nachkommen und bei der Selektion beteiligt ist.

Teilweise wird auch ein Wechsel zwischen Plus- und Kommastrategie während des Evolutionsprozesses vorgenommen. Dabei wendet man im Regelfall die Plus-Strategie an. Wenn sich allerdings über mehrere Generationen keine Verbesserung des besten Chromosoms ergibt, wird für einige Generationen auf die Komma-Strategie umgeschaltet, um die Diversität in der Population wieder zu erhöhen.

Beispiel 7.1 ((1+1)-ES) Der Spezialfall der $(1+1)$-ES entspricht einer Realisierung des Hillclimbing. Die Population besteht nur aus einem einzigen Individuum bzw. Chromosom. Die Startpopulation enthält ein zufällig erzeugtes Chromosom $x^{(0)} \in \mathcal{S} \subseteq \mathbb{R}^n$. Um aus der Population $\{x^{(t)}\}$ die Population $\{x^{(t+1)}\}$ zu erhalten, erzeugt man einen Zufallsvektor $\varepsilon^{(t)} \in \mathbb{R}^n$ durch Realisation von n unabhängigen normalverteilten (Pseudo-)Zufallszahlen mit Erwartungswert 0 und Streuung $\sigma(t)$ und addiert diesen Zufallsvektor auf die Lösung $x^{(t)}$. Die bessere der beiden Lösungen bildet die Folgepopulation:

$$x^{(t+1)} = \begin{cases} x^{(t)} + \varepsilon^{(t)} & \text{falls } f(x^{(t)} + \varepsilon^{(t)}) \leq f(x^{(t)}) \\ x^{(t)} & \text{sonst} \end{cases}$$

Falls die Zielfunktion f maximiert werden soll, muss das $\leq$-Zeichen durch das $\geq$-Zeichen ersetzt werden. Die Selektion entspricht der Auswahlregel beim Hillclimbing. Das beim Hillclimbing nicht näher spezifizierte Erzeugen einer Lösung in der Nähe einer gegebenen Lösung wird hier konkret durch Addieren eines Zufallsvektors mit normalverteilten Komponenten realisiert.

Da bei den Evolutionsstrategien die Selektion auf dem Eliteprinzip basiert, muss die Zielfunktion nicht wie bei der Rouletterad-Selektion in eine Fitness-Funktion umgewandelt werden, die nur nichtnegative Werte annimmt und die zu maximieren ist. Ist die Zielfunktion nur auf einem Teil des $\mathbb{R}^n$ definiert, d.h. $\mathcal{S} \subset \mathbb{R}^n$, so kann man f einfach auf den $\mathbb{R}^n$ fortsetzen, indem man für Werte außerhalb des Definitionsbereichs der Zielfunktion den Funktionswert $+\infty$ im Falle einer zu minimierenden Zielfunktion wählt bzw. $-\infty$ im Falle einer zu maximierenden Zielfunktion. Man kann natürlich auch „Reparaturmechanismen" anwenden, wenn eine Mutation ein Chromosom außerhalb des Definitionsbereichs der Zielfunktion erzeugt. Beispielsweise könnte man den Wert der Mutation soweit verkleinern, dass das Chromosom wieder innerhalb des Definitionsbereichs liegt.

7.1.2 Adaption der Mutationsschrittweite

Wie bereits erwähnt, sind Evolutionsstrategien, bei denen die Mutation durch eine Normalverteilung mit fester Varianz durchgeführt wird, nur für einfache Probleme geeignet. Deswegen werden bei Evolutionsstrategien in der Regel Adaptionstechniken (siehe auch Abschnitt 6.7.2) angewendet, die die Varianzen der Normalverteilungen für die Mutation während des Evolutionsprozesses anpassen und somit die Stärke oder (mittlere) Schrittweite bei Mutationen beeinflussen. Es gibt dabei zwei grundsätzliche Strategien

deterministische Adaption: Da der Suchraum am Anfang eher in der Breite durchsucht werden soll, empfiehlt es sich, zunächst etwas größere Änderungen durch die Mutation zuzulassen und später die Schrittweite bei der Mutation immer weiter einzuschränken, um eine lokale Optimierung zu ermöglichen. Man kann hier ähnlich wie beim Simulated Annealing eine Art „Abkühlungsschema" definieren, mit dem die Mutationsschrittweite, d.h. der Wert σ, schrittweise verkleinert wird. Diese Strategie lässt sich in Form einer deterministischen Adaption realisieren. Es gibt ein festes Schema, nach dem die Streuung σ verringert wird, z.B. $\sigma(t+1) = \alpha \cdot \sigma$ mit $0 < \alpha < 1$, wobei der Wert von α sehr nahe bei 1 liegen sollte.

dynamische Adaption: Im Gegensatz zur deterministischen Adaption wird die Streuung σ bei der dynamischen Adaption nicht zwangsläufig immer verkleinert, sondern kann auch im Laufe der Evolution vergrößert werden, wenn dies zwischenzeitlich günstiger erscheint oder σ am Anfang zu klein gewählt wurde. Es gibt hier wiederum zwei wesentliche Ansätze für die dynamische Adaption, die wir im Folgenden diskutieren wollen.

7.1.3 Rechenbergs 1/5-Regel

Der erste Ansatz verfährt nach dem Prinzip, dass die Streuung, d.h. die Schrittweite der Mutationen, vergrößert werden sollte, wenn Mutationen relativ häufig zu Verbesserungen führen. In [Rec94] wird eine (1+1)-ES näher untersucht und als Faustregel die 1/5-Erfolgsregel empfohlen, die besagt, dass die Mutationsschrittweite optimal gewählt ist, wenn etwa ein Fünftel der Mutationen erfolgreich sind. Wenn die Rate erfolgreicher Mutationen geringer ist, sollte man die Mutationsschrittweite verringern. Ist sie größer, sollte die Mutationsschrittweite erhöht werden. Man sollte die Rate erfolgreicher Mutationen nach k Mutationen, d.h. bei der (1+1)-ES nach k Generationen, überprüfen und dabei die letzten $10k$ Mutationen berücksichtigen. Ist die Anzahl erfolgreicher Mutationen kleiner als $2k$, wird die Schrittweite σ mit 0.85 multipliziert; ist die Anzahl größer als $2k$, teilt man σ durch 0.85.

Rechenberg ging von der Annahme aus, dass man σ so wählen sollte, dass die (mittlere) Konvergenzrate der Evolutionsstrategie möglichst groß ist. Er bestimmte σ so, dass die Konvergenzrate optimal war für die Zielfunktionen $f_1(x_1, \ldots, x_n) = a + bx_1$ und $f_2(x_1, \ldots, x_n) = \sum_{i=1}^{n} x_i^2$. Eine Abschätzung der Wahrscheinlichkeit für eine erfolgreiche Mutation (bei optimalen σ) ergab die folgenden Werte: Für f_1 erhielt er $p_1 \approx 0.184$ und für f_2 den Wert $p_2 \approx 0.270$. Daraus leitete er die 1/5-Erfolgsregel ab.

Für größere Populationen und bei eher komplizierten Zielfunktionen ist die 1/5-Erfolgsregel teilweise noch zu optimistisch und führt zu einer permanenten Reduktion der Mutationsschrittweite, die vorzeitige Konvergenz bedingt. Dies kann z.B. auftreten, wenn das Optimum in der Nähe des Randes des Definitionsbereichs der Zielfunktion liegt [Sch95].

7.1.4 Die Mutationsschrittweite als zusätzlicher zu optimierender Parameter

Die Anwendung der 1/5-Erfolgsregel bedeutet entweder, dass man eine globale Mutationsschrittweite mit dieser Regel anpasst oder eine sehr große Anzahl von Nachkommen pro Chromosom erzeugen muss, um die Regel sinnvoll überprüfen zu können, wenn man die Mutationsschrittweiten individuell für die Chromosomen anpassen möchte. Selbst in diesem Fall kann die Mutationsschrittweite nur gleichmäßig für alle Parameter eines Chromosoms adaptiert werden.

Ein anderer Ansatz besteht darin, die Mutationsschrittweiten für jedes Chromosom als zusätzliche zu optimierenden Parameter aufzunehmen. Ein Chromosom besteht dann aus $2n$ reellwertigen Genen:

$$c = (x_1, \ldots, x_n, \sigma_1, \ldots, \sigma_n)$$

Die Mutation der Mutationsschrittweiten σ_i $(i = 1, \ldots, n)$ erfolgt dann nach dem folgenden Schema:

$$\sigma_i^{(\text{neu})} = \sigma_i^{(\text{alt})} \cdot \exp\left(\tau_1 \cdot N(0,1) + \tau_2 \cdot N_i(0,1)\right) \tag{7.1}$$

Dabei ist $N(0,1)$ eine einmalig am Anfang der Evolutionsstrategie erzeugte (Pseudo-)Zufallszahl, der eine Standardnormalverteilung zugrunde liegt. $N_i(0,1)$ ist ebenfalls nach einer Standardnormalverteilung ausgewürfelten (Pseudo-)Zufallszahl, die allerdings in jedem Schritt und individuell für jedes Chromosom und jedes σ_i neu erzeugt wird. Der erste Term in der Exponentialfunktion ist somit eine Realisierung einer normalverteilten Zufallsvariable mit Erwartungswert 0 und Streuung τ_1, der zweite Term ist entsprechend eine Realisierung einer normalverteilten Zufallsvariable mit Erwartungswert 0 und Streuung τ_2. Empfohlene Werte für die beiden Parameter τ_1 und τ_2 sind [BS93]

$$\begin{aligned} t_1 &\approx \frac{1}{\sqrt{2n}} \\ t_2 &\approx \frac{1}{\sqrt{2\sqrt{n}}} \end{aligned}$$

oder nach [Nis97]

$$\begin{aligned} t_1 &\approx 0.1 \\ t_2 &\approx 0.2 \end{aligned}$$

Gleichung (7.1) garantiert, dass die Streuungen σ_i immer positiv bleiben und dass der Erwartungswert des Wertes in der Exponentialfunktion bei 0 liegt und sich somit die mittlere Änderung der σ_i-Werte zu 0 ergibt. Es kann trotzdem vorkommen, dass ein Wert σ_i fast 0 wird. Üblicherweise legt man noch eine untere Schranke σ_{min} für die Mutationsschrittweiten fest. Falls diese Schranke durch die Berechnung (7.1) unterschritten wird, wird σ_i auf σ_{min} gesetzt. Lässt man beliebig kleine Wert für σ_i zu, können Chromosomen einen Selektionsvorteil erhalten, indem sie sich auf einen extrem kleinen Unterraum in der Nähe eines lokalen Optimums konzentrieren, was zu vorzeitiger Konvergenz führt [Sch95].

Für die Berechnung der Bewertung, also für die Zielfunktion, spielen die zusätzlichen Parameter σ_i keine Rolle. Allerdings werden Chromosomen mit gut angepassten Mutationsschrittweiten im Mittel bessere Nachkommen erzeugen, so dass auf diese Weise die Mutationsschrittweiten automatisch mit angepasst werden. Man erhofft sich, dass schlechte σ_i-Werte indirekt durch die Selektion ausgetilgt werden, da sie die eigentliche Lösung, den x-Vektor, bei zu kleinen σ_i-Werten zu langsam voranbringen bzw. für zu große σ_i-Werte den x-Vektor zu leicht verschlechtern.

Die Anpassung der Mutationsschrittweiten σ_i erfolgt zum einen individuell für jedes Chromosoms, d.h. die Mutationsschrittweiten werden lokal im Suchraum adaptiert. Zum anderen wird die Mutationsschrittweite für jeden der n Parameter im Chromosom einzeln angepasst, so dass auch der Einfluss der einzelnen Parameter auf die Zielfunktion berücksichtigt wird. Dadurch, dass die Mutationen für die einzelnen Parameter unabhängig voneinander ausgewürfelt werden, kann es bei stark abhängigen Parametern dazu führen, dass die Mutationsschrittweiten zu klein gewählt werden müssen. Aus diesem Grund wird zum Teil eine Korrelation der einzelnen Normalverteilungen benutzt.

Die „Hoffnung" ist, dass schlechte σ's indirekt durch die Selektion ausgetilgt, da sie den x-Vektor zu langsam voranbringen (für zu kleine σ_i-Werte) bzw. den x-Vektor zu leicht verschlechtern (für zu große σ_i-Werte).

7.1.5 Korrelierte Mutation

Formal haben wir bei der Mutation bisher einen Zufallsvektor einer n-dimensionalen Normalverteilung mit einer Diagonalmatrix als Kovarianzmatrix erzeugt. Die Diagonalelemente der Matrix entsprechen unseren Varianzen σ_i^2.

Um die Idee der korrelierten Varianzen besser zu verstehen, betrachten wir eine Zielfunktion F mit zwei zu optimierenden reellen Parametern. Nehmen wir an, ein Chromosom repräsentiert die Lösung (x_0, y_0) und der Gradient der Zielfunktion f an der Stelle (x_0, y_0) sei (1,0). Zumindest im Sinne des Gradientenverfahrens wäre es sinnvoll, primär in x-Richtung – positiv oder negativ, je nachdem, ob f maximiert oder minimiert werden soll – zu suchen. Eine günstige Wahl der Mutationsschrittweiten für das Chromosom könnte daher $\sigma_1 = 1$ und $\sigma_2 \approx 0$. Dies könnte sich auch im Laufe der Evolution ergeben. Betrachten wir nun den Fall, dass der Gradient nicht (1,0), sondern (1,1) beträgt. Die Wahl $\sigma_1 = 1$ und $\sigma_2 = 1$ führt nicht wie im vorhergehenden Fall dazu, dass primär in der vielversprechenden Richtung des Gradienten gesucht wird, sondern dass der Suchraum in der Nähe des betrachteten Chromosoms in alle Richtungen mit gleicher Wahrscheinlichkeit untersucht wird. Eine Anpassung an einen derartigen Gradienten kann nicht erreicht werden, wenn man bei der Mutation wie bisher die Änderungen der einzelnen Parameter unabhängig voneinander erwürfelt. Es ist erforderlich, Korrelationen zwischen den einzelnen Parameter zu berücksichtigen. Das bedeutet, dass wir für die Erzeugung des Zufallsvektors bei der Mutation nicht mehr von einer diagonalen Kovarianzmatrix ausgehen können.

Da eine Kovarianzmatrix keine beliebige Matrix sein kann, sondern symmetrisch und positiv definit sein muss, lässt sie sich durch die Werte σ_i $(i = 1, \ldots, n)$ und $\frac{n(n-1)}{2}$ Rotationswinkel φ_{pq} $(p, q \in \{1, \ldots, n\}, p < q)$ eindeutig charakterisieren. Bisher

bestimmte σ_i die Mutationsschrittweite für den zu optimierenden Parameter x_i. Damit konnte man beispielsweise erreichen, dass primär in x_1-Richtung gesucht wird, wenn man für σ_1 einen relativ großen Wert und für alle anderen σ_i sehr kleine Werte wählt. Wenn primär in der Richtung der Hauptdiagonalen gesucht werden soll, lässt sich dies nicht mehr allein mit den σ_i Werten realisieren. Für die Hauptdiagonale könnte man beispielsweise wiederum für σ_1 einen relativ großen Wert und für alle anderen σ_i sehr kleine Werte wählen. Zusätzlich muss dann noch durch die φ_{pq}-Werte eine Rotation erzeugt werden, die die bisherige Hauptsuchrichtung x_1 auf die Hauptdiagonale rotiert.

Bei korrelierten Mutation bedeutet das, dass man neben den n zu optimierenden Parametern σ_i zusätzlich $n+n(n-1)/2$ Parameter φ_{pq} während des Evolutionsprozesses adaptiert. Auch die Berechnung der Zufallsvektoren ist dann entsprechend aufwendiger. Konkret sieht die Berechnung bei korrelierter Mutation folgendermaßen aus. Wir bezeichnen mit $R_{pq}(\varphi)$ die Rotationsmatrix

$$R_{pq}(\varphi) = \begin{pmatrix} 1 & & & & & & & & & \\ & \ddots & & & & & & & & \\ & & 1 & & & & & & & \\ & & & \cos\varphi & & & & -\sin\varphi & & \\ & & & & 1 & & & & & \\ & & & & & \ddots & & & & \\ & & & & & & 1 & & & \\ & & & \sin\varphi & & & & \cos\varphi & & \\ & & & & & & & & 1 & \\ & & & & & & & & & \ddots \\ & & & & & & & & & & 1 \end{pmatrix}$$

Die beiden Einträge $\cos\varphi$ stehen in der p-ten und q-ten Zeile. Alle außerhalb der Diagonalen nicht angegebenen Einträge sind 0. Bei unkorrelierter Mutation wurde die Mutation der Lösung $x = (x_1, \ldots, x_n)^\top$ durch Addition eines Vektors $\Delta = (\Delta_1, \ldots, \Delta_n)^\top$ realisiert. Dabei wurden die Werte Δ_i nach unabhängigen Realisierungen von normalverteilten Zufallsvariablen mit Erwartungswert 0 und Streuung σ_i erzeugt. Bei der korrelierten Mutation berechnet man zunächst wie bisher den Vektor Δ, um daraus den eigentlichen Mutationsschritt zu berechnen:

$$\Delta_{\text{korreliert}} = \left(\prod_{p=1}^{n} \prod_{q=p+1}^{n} R_{pq}(\varphi_{pq}) \right) \Delta$$

Das durch korrelierte Mutation erzeugte Chromosom lautet dann

$$x_{\text{mutiert}} = x + \Delta_{\text{korreliert}}$$

Die Winkel φ_{pq} werden mittels

$$\varphi_{pq}^{(\text{neu})} = \varphi_{pq}^{(\text{alt})} + \tau \cdot N(0,1)$$

mutiert. Dabei bezeichnet $N(0,1)$ wiederum die Realisierung einer nach einer Standardnormalverteilung realisierten Zufallsvariablen, die für jede Generation, jedes Chromosom und jeden Winkel erneut und unabhängig ausgewürfelt wird. Der fest zu wählende Parameter τ wird üblicherweise auf ca. $\tau = 0.0873$, d.h. auf etwa $5°$ festgelegt. Die Art, wie wir implizit die Kovarianzmatrix der n-dimensionalen Normalverteilung für die Mutation berechnen, erfordert, dass die Rotationswinkel φ_{pq} im Bereich $[-\pi, \pi]$ liegen. Sollte ein Winkel durch Mutation aus diesem Bereich gelangen, wird er durch ggf. mehrfache Addition oder Subtraktion von 2π wieder in das Intervall $[-\pi, \pi]$ zurückgebracht.

Obwohl die Mutation im korrelierten Fall wesentlich flexibler ist, bringt sie auch Nachteile mit sich. Zum einen müssen wesentlich mehr Parameter adaptiert werden. Die Anzahl der Rotationswinkel φ_{pq} wächst quadratisch mit n, der Anzahl der im Optimierungsproblem ursprünglich zu optimierenden Parameter. Zum anderen findet die Anpassung der Rotationswinkel φ_{pq} eher beiläufig im Evolutionsprozess statt. In manchen Fällen ist es fraglich, ob diese Adaption der Winkel, d.h. die Anpassung der Hauptsuchrichtung, dem mutierten Chromosom überhaupt schnell genug folgen kann. Ein mutiertes Chromosom findet lokal eine andere Struktur der Zielfunktion vor. Ändert sich der Gradient der Zielfunktion relativ schnell, nutzt die korrelierte Mutation relativ wenig.

Für eine detaillierte Darstellung der Technik der korrelierten Mutation sei auf [Rud92a] verwiesen.

7.1.6 Rekombination

Im Rahmen der Evolutionsstrategien lassen sich ebenfalls Rekombinationsmechanismen anwenden. Rekombination hat die Aufgabe, dass sich gute Lösungen „austauschen", um so eventuell zu noch besseren Lösungen zu gelangen. Bei binären Genen der genetischen Algorithmen besteht der Austausch darin, dass man die Werte einzelner Gene zweier Chromosomen austauscht. Dies kann durch verschiedene Crossover-Arten wie 1-Point-, 2-Point- oder allgemeiner durch k-Point-Crossover oder sogar durch uniformes Crossover erfolgen, bei dem für die Nachkommen jedes Gen zufällig von einem der Elternchromosomen übernommen wird. Bei den Evolutionsstrategien wird häufig das uniforme Crossover verwendet, das man dann auch als *diskrete Rekombination* bezeichnet.

Da die Gene bei Evolutionsstrategien reellwertig sind, bietet sich hier noch eine andere Crossover-Variante an: die *intermediäre Rekombination*. Dabei wird für ein Gen der Mittelwert der Werte des entsprechenden Gens der beiden Elternchromosomen verwendet.

diskrete Rekombination: zufällige Auswahl der Gene, z.B.

$$\begin{matrix}(x_1^{(t)}, \ldots, x_n^{(t)}) \\ (y_1^{(t)}, \ldots, y_n^{(t)})\end{matrix} \rightsquigarrow (x_1^{(t)}, y_2^{(t)}, x_3^{(t)}, y_4^{(t)}, \ldots, x_{n-1}^{(t)}, y_n^{(t)})$$

intermediäre Rekombination: Mittelwertbildung bei jedem Gen:

$$\begin{matrix}(x_1^{(t)},\ldots,x_n^{(t)})\\(y_1^{(t)},\ldots,y_n^{(t)})\end{matrix} \rightsquigarrow \frac{1}{2}(x_1^{(t)}+y_1^{(t)},\ldots,x_n^{(t)}+y_n^{(t)})$$

Häufig wird eine Kombination der beiden Rekombinationstechniken verwendet. Auf die zu optimierenden Parameter, die die eigentliche Lösung kodieren, wird diskrete Rekombination angewendet, während die für die Adaption zuständigen Parameter – die Mutationsschrittweiten σ_i und ggf. die Rotationswinkel φ_{pq} – intermediärer Rekombination unterzogen werden.

7.1.7 Abbruchkriterien

Als Abbruchkriterium, wann die Generationenfolge beendet wird, wird eines der bei evolutionären Algorithmen übliches Kriterien verwendet:

- das Erreichen einer maximalen Anzahl von Generationen
- das Überschreiten einer vorgegebenen maximalen Rechenzeit
- keine Verbesserung der besten Lösung in den letzten t_0 Generationen
- Unterschreitung einer vorgegebenen mittleren Verbesserung in den letzten zurückliegenden Generationen
- das Finden einer Lösung mit einer vorgegebenen Mindestgüte
- Konvergenz der Population im dem Sinne, dass der Unterschied zwischen bester und schlechtester Lösung in einer Generation (absolut oder relativ) klein ist

7.1.8 Theoretische Resultate

Es gibt eine Reihe theoretischer Ergebnisse über Evolutionsstrategien, insbesondere über die (1+1)-ES, die aber nicht praxisrelevant ist. Die meisten dieser Resultate gehen von sehr speziellen Annahmen aus oder machen nur sehr allgemeine Aussagen, die für die reale Anwendung der Algorithmen keine große Bedeutung haben.

Um ein besseres Verständnis für derartige Aussagen zu bekommen, beweisen wir hier ein einfaches Resultat, das zeigt, dass unter gewissen Umständen jede Zufallssuche beliebig nahe an das globale Optimum gelangt, wenn sie den Suchraum nur genügend zufällig abtastet.

Wir betrachten als Suchraum $\mathcal{S} = [0,1]^n$ und eine zu maximierende Zielfunktion $f : \mathcal{S} \to \mathbb{R}$. Wir setzen voraus, dass f stetig ist. Aus diesen Annahmen folgt, dass das globale Maximum von f auf $\mathcal{S}$ angenommen wird, d.h., es existiert ein $x^* \in \mathcal{S}$ mit $f(x^*) = \sup\{f(x) \mid x \in \mathcal{S}\}$. Dass wir hier nur den Einheitswürfel $\mathcal{S} = [0,1]^n$ betrachten, stellt keine besondere Einschränkung dar. Wenn jeder zu optimierende Parameter Werte in einem Intervall annehmen darf, kann jedes dieser Intervalle auf das Einheitsintervall transformiert werden, so dass wir den Einheitswürfel als Suchraum erhalten.

Wir können nicht erwarten, dass wir die Lösung x^* exakt finden werden. Wir interessieren uns für Lösungen, die beliebig nahe an die beste Lösung x^* herankommen. Es sei $\varepsilon > 0$. Für $x \in \mathcal{S}$ sei

$$B_\varepsilon(x) = \{y \in \mathcal{S} \mid \| x - y \| < \varepsilon\}$$

die offene ε-(Hyper-)Kugel mit Radius ε um x. Aufgrund der Kompaktheit von $\mathcal{S}$ gibt es eine endliche Menge $\{x_1, \ldots, x_k\} \subset \mathcal{S}$, so dass die zugehörigen ε-Kugeln $\mathcal{S}$ überdecken, d.h.

$$\mathcal{S} \subseteq \bigcup_{i=1}^{k} B_\varepsilon(x_i).$$

Wir gehen von einem Zufallssuchverfahren aus, bei dem die beste Lösung immer überlebt. Außerdem sollte das Verfahren die folgende Bedingung erfüllen. Es existiert ein $\delta > 0$, so dass für jedes $i \in \{1, \ldots, k\}$ – egal in welcher Generation oder bei welcher/welchen Zwischenlösungen sich das Verfahren befindet – die Wahrscheinlichkeit, dass eine Lösung in $B_\varepsilon(x_i)$ erzeugt wird, größer als δ ist. Diese Bedingung ist beispielsweise für jede $(\mu + \lambda)$-ES erfüllt, bei der die Mutationsschrittweiten nicht angepasst werden. Daher gilt für jede Generation, dass die Wahrscheinlichkeit, dass keine Lösung in $B_\varepsilon(x_i)$ untersucht wird, kleiner als $1 - \delta$ ist. Damit ist die Wahrscheinlichkeit, dass innerhalb von t aufeinanderfolgenden Generationen keine Lösung in $B_\varepsilon(x_i)$ untersucht wird, kleiner als $(1-\delta)^t$. Somit wird für $t \to \infty$ eine Lösung in $B_\varepsilon(x_i)$ mit Wahrscheinlichkeit 1 untersucht.

Wir geben nun einen Toleranzschwelle ε_0 vor, d.h., wir wollen untersuchen, ob unser Verfahren eine Lösung $x_0 \in \mathcal{S}$ mit einer Güte $f(x_0) \geq f(x^*) - \varepsilon_0$ findet. Aufgrund der Stetigkeit der Zielfunktion f gibt es ein $\varepsilon > 0$, so dass gilt:

$$\forall x \in (B_\varepsilon(x^*) \wedge \mathcal{S}) : \; f(x) \geq f(x^*) - \varepsilon_0$$

Die vorhergehenden Überlegungen zeigen, dass die Kugel $B_\varepsilon(x^*)$ im Laufe der Suche mit Wahrscheinlichkeit 1 besucht wird und somit eine Lösung mit einer Mindestgüte von $f(x^*) - \varepsilon_0$ gefunden wird.

Das gleiche Resultat gilt für eine reine Zufallssuche, bei der in jeder Generation eine neue Lösung nach einer Gleichverteilung über $\mathcal{S}$ ausprobiert wird. Da dieses Ergebnis bereits für die reine Zufallssuche gilt, kann man es nicht als besonders wertvoll für Evolutionsstrategien werten.

Eine Reihe von allgemeineren theoretischen Ergebnissen findet man in [Bey93, Bey94, Bey95, Bey96, Bey00, Rud92b, Rud94a, Rud94b, Rec94, Bäc96]. Neben Betrachtungen zur Konvergenz zum globalen Optimum finden sich in diesen Arbeiten zum Teil auch ausführliche Betrachtungen zur Fortschrittsgeschwindigkeit bei Evolutionsstrategien. Damit wird die mittlere Verbesserungsrate pro Generation oder über einen Zeitraum von mehreren Generationen bezeichnet.

7.2 Evolutionäre Programmierung

Der Bereich der *evolutionären Programmierung* nimmt unter den evolutionären Algorithmen nur einen sehr kleinen Teil ein. Das ist sowohl auf die völlig andere Auffassung der Evolution, als auch auf die große Ähnlichkeit zu den fast zeitgleich entwickelten Evolutionsstrategien (siehe hierzu auch Bäck [Bäc96]) zurückzuführen. Hinzu kommt, dass die evolutionären Programme ursprünglich von L.J. Fogel [FOW66] zur Generierung „intelligenter" endlicher Automaten und nicht zu Optimierungszwecken entwickelt wurden. Dem entsprechend wird diese Algorithmenvariante in diesem Buch nur kurz vorgestellt, auch wenn sie mittlerweile insbesondere von D.B. Fogel zu einem Optimierungsalgorithmus weiterentwickelt wurde [Fog92]. Optimiert werden in der Regel mehrdimensionale reellwertige Funktionen, deren globales Optimum beim Nullpunkt liegt.

Das besondere an evolutionärer Programmierung ist, dass die Evolution von einem phänotypischen Standpunkt betrachtet wird, statt wie sonst üblich von einem genotypischen Standpunkt. Es wird also davon ausgegangen, dass sich eine Evolution durch eine Anpassung (Mutation) von Individuen einer Population an äußere Umstände vollzieht und nicht durch eine Änderung der Chromosomen. Dem entsprechend verfügen evolutionäre Programme auch nur über einen Mutationsoperator und nicht – wie andere Algorithmen – über Rekombinationsoperatoren (siehe Abschnitt 6.5), da nicht versucht wird, genetische Abläufe nachzubilden.

Der generelle Ablauf eines evolutionären Programmes (EP) ist analog zu einer Evolutionsstrategie. Als Chromosomen werden ebenfalls reellwertige Vektoren verwendet. Für die Startpopulation werden die Einträge der Gene zufällig bestimmt, jedoch so, dass die Werte mittels einer Gleichverteilung in ein vorgegebenes Intervall fallen. Es gilt also für ein Chromosom $c = (a_1, \ldots, a_n)$:

$$\forall\, i \in \{1, \ldots, n\} \;:\; a_i \in [u_i, o_i] \subset \mathbb{R}$$

Diese Intervalle gelten nur während der Initialisierungsphase, danach können die Gene beliebige reelle Werte annehmen (also: $c \in \mathbb{R}^n$). Als Bewertungsfunktion wird meistens der Wert der zu optimierenden Funktion verwendet, wobei die Positivität der Bewertungsfunktion durch eine entsprechende Skalierung sichergestellt werden kann (siehe auch Abschnitt 6.3.1 und 6.3.3).

Zur Erstellung der nächsten Population $P(t+1)$ wird von jedem Chromosom aus $P(t)$ eine Kopie erzeugt, so dass eine Population $P^*(t)$ mit doppelter Populationsgröße entsteht. Auf die doppelten Chromosomen wird dann der Mutationsoperator angewendet wird. Im Gegensatz zur Mutation bei Evolutionsstrategien ist der Operator nicht selbstadaptiv, sondern verändert sich in Abhängigkeit von der Fitness des Elternchromosoms und einem normalverteiltem Zufallswert:

$$\forall\, i \in \{1, \ldots, popsize\} \;:\; a^*_{ij} = a_{ij} + \sqrt{k_i \cdot fit(c_i) + z_j} \cdot N(0,1)$$

Die Parameter k_i und z_j (Grundvarianz) werden von außen bestimmt und müssen dem entsprechend an das betrachtete Problem angepasst werden. Häufig wird für die k_i der Wert 1 und für die z_j der Wert 0 verwendet, so dass das Ausmaß der Mutation nur noch von der Fitness des Elternchromosoms abhängt. Die Problematik der Verwendung

von Funktionen, deren globales Optimum nicht bei Null liegt, ist aus dieser Formel leicht zu erkennen. In diesem Fall würde die Mutationsschrittweite nie den Wert Null annehmen, so dass auch bereits optimale Gene immer wieder verändert werden würden.

Nach der Verdopplung der Population durch Anwendung der Mutation muss die Populationsgröße wieder auf $popsize$ Chromosomen reduziert werden. Das geschieht – analog zu einer $(\mu + \lambda)$-Evolutionsstrategie – durch die Anwendung des Selektionsoperators, nach vorhergehender Bewertung der neu hinzugekommenen Chromosomen. Die Selektion selbst unterscheidet sich deutlich von den bei den Standard-Algorithmen üblichen Selektionsformen, da sie nicht direkt über die Fitness durchgeführt wird. Statt dessen werden für jedes der $popsize$ Chromosomen der neuen Population zufällig q weitere Chromosomen ausgewählt, gegen die dann eine Wettkampfselektion durchgeführt wird. Für jedes Chromosom wird der eigene Fitnesswert mit denen der q anderen Chromosomen verglichen. Dabei wird die Zahl der Vergleiche berechnet, aus denen das zu testende Chromosom als Sieger hervorging. Es sei

$$P_i \;=\; \{c_j \in P^*(t) \mid j \in \{1,\ldots,popsize\} : fit(c_j) < fit(c_i)\}$$

und X eine Zufallsvariable, die die Werte aus auf der Menge $\{1,\ldots,2\cdot popsize\}$ nach einer Gleichverteilung annimmt. Dann wird die Bewertung w_i des Chromosoms c_i wie folgt berechnet:

$$w(c_i) \;=\; \sum_{k=1}^{q} \mathbf{1}_{P_i}(c_{x_k})$$

Dabei sind die x_k unabhängige Realisierungen der Zufallsvariable X.

Auf der Basis der Bewertungen wird eine Rangfolge erstellt, auf deren Grundlage die q besten Chromosomen ausgewählt werden. Durch diese Art der Auswahl überlebt das beste Chromosom immer. Insgesamt handelt es sich bei dieser Selektionsform um eine statische diskriminierende Elitist-Selektion (siehe Abschnitt 6.4).

Bei dem oben beschriebenen Algorithmus handelt es sich um die Grundform eines evolutionären Programmes. Der dabei verwendete Mutationsoperators kann jedoch immer dann zu Schwierigkeiten führen, wenn die Fitness einen sehr hohen Wert annimmt, da die Mutation nicht mehr zielgerichtet um den ursprünglichen Wert herum stattfindet.

Um den gerade beim Mutationsoperator auftretenden Schwierigkeiten aus dem Weg zu gehen, wurde von Fogel [Fog92] der so genannte Meta-EP entwickelt, der mit einer selbstadaptiven Mutationsschrittweite arbeitet. Wie schon bei den Evolutionsstrategien wird jedes Chromosom dafür um einen Vektor ν der Länge n ergänzt $(a \mapsto (a,\nu))$, der die zu jeder Position des Chromosoms a gehörenden Mutationsparameter enthält. Die Vektoren ν_i werden in der Initialisierungsphase zufällig mit Werten aus einem Intervall $[0,b]$, $b > 0$ belegt. Die Mutation des Standard-EP wird durch die folgende Mutation ersetzt:

$$\begin{aligned} \forall\, i \in \{1,\ldots,popsize\} \quad &: \quad \nu_{ij}^* = \nu_{ij} + \sqrt{\zeta \cdot \nu_i} \cdot N(0,1) \;\text{ mit} \\ \forall\, i \in \{1,\ldots,popsize\} \quad &: \quad a_{ij}^* = a_{ij} + \sqrt{\nu_i} \cdot N(0,1) \end{aligned}$$

ζ ist ein vorher festgelegter Parameter, der dafür sorgen soll, dass die Mutationsparameter positiv bleiben. Sollte der neue Wert ν^* kleiner als Null sein, so wird er auf einen kleinen, positiven Wert ε gesetzt.

Ein grundsätzliches Problem bei der evolutionären Programmierung besteht darin, dass angenommen wird, dass das globale Minimum der Zielfunktion bei 0 liegt. In den meisten Anwendungen kennt man aber den Wert des globalen Optimums nicht.

7.3 Genetische Programmierung

Bei den bisher betrachteten evolutionären Algorithmen sind wir fast immer davon ausgegangen, dass die Chromosomen eine feste Länge haben. In diesem Sinne waren die Optimierungsprobleme durch Parameteroptimierung gekennzeichnet. Eine gewisse Ausnahme stellt das Beispiel der Flugroutenoptimierung in Abschnitt 6.8, bei dem die Anzahl der Zwischenknoten und damit die Anzahl der Gene im Laufe der Evolution variiert werden konnte.

Chromosomen fester Länge sind jedoch ungeeignet für die Suche nach Problemlösungsstrategien etwa in Form von Programmen, Plänen oder im Bereich der Regelung wie im Beispiel 2.3. Deshalb soll jetzt ein neuer Ansatz vorgestellt werden, die *genetische Programmierung*. Hierbei weisen die Chromosomen eine völlig andere Struktur auf, da Chromosomen variabler Länge verwendet werden, die als Programme interpretiert werden können. Mit der genetischen Programmierung werden Programme erzeugt, die ihr Ein-/Ausgabe-Verhalten bzgl. vorgegebener Ziele optimieren. Genetische Programmierung wurde vor allem durch die Bücher von Koza [Koz92, Koz94] populär.

7.3.1 Listen und die Programmiersprache LISP

Bei der genetischen Programmierung muss zuerst einmal eine endliche Menge von Funktionen oder Funktionssymbolen $\mathcal{F}$ und eine endliche Menge von Terminalsymbolen (Konstanten, Variablen) $\mathcal{T}$ festgelegt werden.

Beispiel 7.2 Es soll eine Boole'schen Funktion $f : \{0,1\}^n \to \{0,1\}$ erlernt werden. Als Menge der Funktionen oder Funktionssymbole könnte man z.B.

$$\mathcal{F} = \{\texttt{and, or, not, if..then..else}\}$$

wählen. Die Menge der Terminalsymbole enthält die Eingabevariablen $x_1, \ldots, x_n$ und eventuell geeignete Konstanten, in diesem Fall die Boole'schen Konstanten 0 und 1:

$$\mathcal{T} = \{x_1, \ldots, x_n, 0, 1\}$$

Die Chromosomen sind aus $\mathcal{C} = \mathcal{F} \cup \mathcal{T}$ zusammengesetzte Funktionen.

Bei der Festlegung der Menge der Funktionen $\mathcal{F}$ für ein Problem, das mittels genetischer Programmierung gelöst werden soll, sind folgende Eigenschaften zu berücksichtigen:

vollständige Definiertheit: Jede Funktion aus $\mathcal{F}$ muss vollständig für alle Argumente, die auftreten können, definiert sein. Gegebenenfalls muss man den Definitionsbereich einzelner Funktion künstlich fortsetzen, z.B. für Funktionen wie die Division $\texttt{div}(x, y) = x/y$ oder den Logarithmus:

- $\texttt{div}(x, 0) := 1$ oder
- $\log(x) := 0$ für $x \leq 0$.

Zum Teil ist es sinnvoll, bei der Fortsetzung der Funktionen gleichzeitig den Wertebereich zu erweitern, z.B. $\pm\infty$ mit in den Wertebereich mit aufzunehmen und

- $\texttt{div}(x, 0) := +\infty$ für $x \geq 0$ bzw. $\texttt{div}(x, 0) := -\infty$ für $x < 0$ oder
- $\log(x) := -\infty$ für $x \leq 0$

zu definieren. In diesem Fall eines erweiterten Wertebereichs müssen aber auch die anderen Funktion für die neuen Argumente definiert werden und alle Kombinationen von Eingaben definiert werden, d.h., man muss auch Werte für

$$\texttt{div}(\pm\infty, \pm\infty) \quad \text{oder für} \quad \texttt{plus}(x, \pm\infty)$$

festlegen. Dabei ist `plus` die Funktion $\texttt{plus}(x, y) = x + y$.

In diesem Fall wäre es vermutlich sinnvoller, den Wertebereich nicht um $\pm\infty$ zu ergänzen, sondern um `NaN` (Not A Number) und zu definieren, dass das Ergebnis aller Funktionen wiederum `NaN` ist, falls mindestens eines der Argumente `NaN` ist.

erschöpfend: Diese Eigenschaft besagt, dass $\mathcal{F}$ reichhaltig genug sein muss, um das gegebene Problem zu lösen. Beispielsweise würde man mit der Festlegung

$$\mathcal{F} = \{\texttt{and, or}\}$$

für das Erlernen Boole'scher Funktionen keine erschöpfende Funktionenmenge haben, da sich mit dieser Funktionsmenge z.B. keine Negation ausdrücken lässt.

feste Arität: Jedem Funktionssymbol besitzt eine feste Arität, d.h., jeder Funktion ist eine feste Anzahl von Argumenten zugeordnet. Wir fassen somit die Funktionen $\texttt{plus}(a, b)$ und $\texttt{plus}(a, b, c)$ als zwei verschiedene Funktionen auf, eine mit zwei und eine mit drei Argumenten.

Die Funktionen werden in LISP-artiger Notation verwendet. LISP (*Lis*t-*P*rocessing) ist eine funktionale Programmiersprache. Dabei besteht LISP in unserer vereinfachenden Sicht aus verschiedenen Symbolen (Funktionssymbolen, Variablensymbolen, Konstantensymbolen und den öffnenden und schließenden Klammern) und *symbolischen Ausdrücken* oder *s-expressions* bzw. *Listen*.

Die folgenden Ausdrücke sind Listen:

1. Einzelne (auch geklammerte) Konstanten- oder Variablensymbole x oder (x) sind Listen.

2. $(fx_1 \ldots x_n)$ ist eine Liste, falls f ein Symbol für eine n-stellige Funktion ist und $x_1 \ldots x_n$ Konstanten oder Variablen sind. $(fx_1 \ldots x_n)$ liefert den Funktionswert $f(x_1, \ldots, x_n)$.

3. Ist f eine n-stellige Funktion und sind $\ell_1, \ldots, \ell_n$ Listen, so ist auch

$$(f\ (\ell_1)\ \ldots\ (\ell_n)\ )$$

eine Liste. Liefert die Liste ℓ_i den Wert l_i $(i = 1, \ldots, n)$, so liefert die obige Liste den Wert

$$f\,(l_1, \ldots, l_n)\,.$$

Nur die mit den Vorschriften 1. bis 3. konstruierbaren Listen sind Listen.

Beispiel 7.3 Die Liste `( + 3 ( * 5 6 ) )` liefert den Wert $3 + (5 \times 6) = 32$.

Die Liste `(+ 2 3 (IF ( > x y ) 5 6 ) )` liefert den Wert $2+3+5 = 10$, falls die Eingabe (Variable) x größer als die Eingabe y ist, und $3 + 3 + 6 = 11$ sonst.

Die Repräsentation in Form einer Liste kann für die Darstellung im Rechner bzw. zur Speicherung sinnvoll sein. Eine grafische Veranschaulichung der Listen kann mit der Darstellung in Form eines Baumes erreicht werden. Jedes Funktions-, Variablen- und Konstantensymbol in der Liste wird durch einen Knoten des Baums repräsentiert. Variablen- und Konstantensymbole bilden die Blätter des Baums. Jeder Knoten, der ein Funktionssymbol einer Funktion mit n Argumenten repräsentiert, hat genau n Nachfolgerknoten. Die Unterbäume, die durch die Nachfolgerknoten definiert werden, enthalten genau die Listen, die als Argumente der n-stelligen Funktion fungieren. Die Darstellung der beiden Listen aus Beispiel 7.3 als Baum ist in Abbildung 7.1 wiedergegeben.

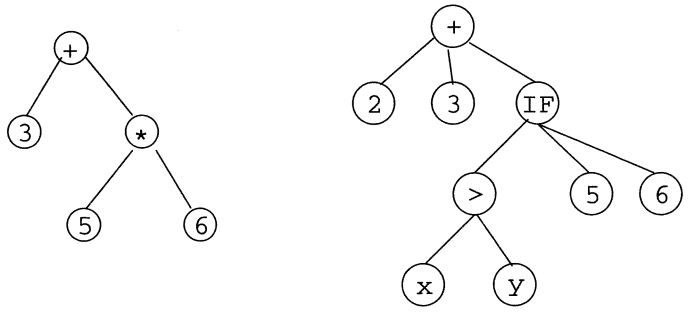


Abbildung 7.1 Beispielprogramme als Bäume

Will man die Funktionen in der Baumrepräsentation auswerten, setzt man – sofern vorhanden – die Eingabewerte für alle Variablen in den Wurzelknoten ein und wertet die Funktionsknoten von den Blättern des Baumes aufsteigend aus.

Im Folgenden werden wir die Begriffe Liste, Baum, Chromosom und das dazugehörige LISP-Programm synonym verwenden.

7.3.2 Ablauf der genetischen Programmierung

Das prinzipielle Ablaufschema der genetischen Programmierung sieht folgendermaßen aus. Die einzelnen Schritte werden wir im Folgenden näher erläutern.

1. Zunächst müssen für das gegebene Problem geeignete Terminalsymbole, d.h. (Eingabe-)Variablen und Konstanten, und geeignete Funktionssymbole festgelegt werden.
2. Es wird eine zufällige Startpopulation von Listen erstellt.
3. Bestimme die Fitness für jede Liste der Population, also wie gut die Liste – bzw. das Programm, das durch die Liste definiert wird – die gegebene Aufgabe erfüllt oder das gegebene Problem löst.
4. Führe Selektion nach Fitness durch.
5. Führe Crossover durch, so dass immer korrekte Listen entstehen.
6. Wiederhole Schritte 3 bis 5, bis ein Abbruchkriterium erfüllt ist.

Konkret werden die einzelnen Schritte folgendermaßen ausgeführt:

Die Startpopulation wird erzeugt nach dem Grundprinzip, dass zuerst zufällig ein Symbol s aus $\mathcal{C} = \mathcal{F} \cup \mathcal{T}$ gewählt wird. Falls $s \in \mathcal{T}$ gilt, ist eine Liste (ein vollständiges Chromosom) erzeugt. Falls $s \in \mathcal{F}$ und s eine n-stellige Funktion repräsentiert, werden zufällig n weitere Symbole aus $\mathcal{C}$ ausgewählt. Falls dabei wieder Funktionssymbole ausgewählt werden, muss für diese auf dieselbe Weise fortgefahren werden, bis schließlich nur noch Terminalsymbole gewählt wurden.

Auf diese Weise könnten in der Startpopulation Bäume beliebiger Tiefe entstehen, was wiederum zu unnötig komplizierten Lösungen führen kann. Aus diesem Grund legt man häufig eine maximale Anfangstiefe $indepth$ der Bäum fest. Typischerweise wird $indepth = 6$ gewählt. Das bedeutet, dass an den Stellen, wo ein Baum bereits eine Tiefe von 5 erreicht hat, Symbole nur noch aus der Menge der Terminalsymbole für diesen Teil des Baums ausgewählt werden.

Wir gehen wiederum von einer Populationsgröße von $popsize$ Chromosomen aus. Üblicherweise werden die Bäume der Startpopulation nicht nur in ihrer Tiefe begrenzt. Man versucht auch eine ausgewogene Verteilung der Bäume zu erreichen. Dazu werden jeweils $\frac{popsize}{indepth-1}$ Bäume der Tiefe $2, 3, \ldots,$ indepth erzeugt.

Außerdem verlangt man meist, dass 50% der Bäume voll sind, d.h. nur auf der untersten Ebene eines Baums tauchen Funktionssymbole auf. Jeder Blattknoten eines vollen Baums hat die gleiche Tiefe.

Man erreicht diese Eigenschaften, indem man an den entsprechenden Knoten jeweils nur aus der Menge $\mathcal{F}$ der Funktionssymbole, falls die gewünschte Tiefe noch nicht erreicht ist, bzw. aus der Menge $\mathcal{T}$ der Terminalsymbole auswählt, wenn die gewünschte Tiefe mit dem zu erzeugenden Knoten erreicht wird.

Wählt man in einem Knoten nur aus der Menge $\mathcal{T}$ oder nur aus der Menge $\mathcal{F}$, so geschieht dies üblicherweise nach einer Gleichverteilung. Darf in einem Knoten sowohl ein Funktionssymbol als auch ein Terminalsymbol gewählt werden, wählt man mit jeweils 50% Wahrscheinlichkeit ein Symbol aus einer der beiden Mengen aus.

Doppelte Bäume (Listen) sollten vermieden werden.

Die Fitness der Bäume wird auf der Grundlage von N Fallbeispielen ermittelt. Je nach Art des Problems können entweder zufällig erzeugte Eingaben, bei nicht zu großer Anzahl alle möglichen Eingaben oder repräsentative Eingaben benutzt werden.

Die *rohe Fitness* (*raw fitness*) der Liste ell berechnet man als

$$r(\ell) \;=\; \sum_{i=1}^{N} err(\ell, i),$$

wobei $err(\ell, i)$ der Fehler ist, den die Liste ℓ beim i-ten Fallbeispiel macht. Löst die Liste ℓ die gegebene Aufgabe optimal oder fehlerfrei für das i-te Fallbeispiel, würde man $err(\ell, i) = 0$ setzen. Wird die Aufgabe nicht gelöst, könnte man $err(\ell, i) = 1$ oder

$$err(\ell, i) \;=\; |output(\ell, i) - \text{ gewünschter Output im Fallbeispiel } i|$$

definieren.

Die rohe Fitness kann anstelle eines Fehlermaßes auch ein Gütemaß sein: Beispielsweise bei einer regelungstechnischen Aufgabe – z.B. der Leerflaufdrehzahlregelung aus Beispiel 2.3 – wie lange die Liste es schafft, den Stellwert für eine gegebene Anfangssituation in einem eventuell zeitlich variierenden Toleranzbereich zu halten. Im Gegensatz zu einem Fehlermaß ist die Liste umso besser, je größer ihre rohe Fitness ist.

Die *standardisierte Fitness* (*standardized fitness*) $s(l)$ einer Liste ℓ wird so definiert, dass sich kleinere Werte von $s(l)$ für bessere Listen ergeben und $s(\ell) = 0$ für eine optimale Liste ℓ gilt. Entspricht die rohe Fitness einem Fehlermaß, kann die rohe Fitness auch als standardisierte Fitness verwendet werden: $s(\ell) = r(\ell)$. Ist die rohe Fitness ein Gütemaß wie im regelungstechnischen Beispiel, könnte man $s(\ell) = r_{\max} - r(\ell)$ definieren, wobei $r_{\max}$ eine genügend große Zahl ist, so dass $s(\ell)$ niemals negativ werden kann.

Die *adjustierte Fitness* (*adjusted fitness*) einer Liste ℓ wird durch

$$a(\ell) \;=\; \frac{1}{1 + s(\ell)} \quad \in [0, 1].$$

Die adjustierte Fitness nimmt nur Werte im Einheitsintervall an.

Die *normalisierte Fitness* (*normalized fitness*) einer Liste ℓ ist durch

$$norm(\ell) = \frac{a(\ell)}{\sum_{i=1}^{popsize} a(\ell_i)} \in [0,1]$$

gegeben. Die Term im Nenner ist die Summe über die adjustierten Fitnesswerte der entsprechenden Population.

Die normalisierte Fitness folgende Eigenschaften:

- Sie liegt zwischen 0 und 1.
- Die Summe der normalisierten Fitnesswerte aller Chromosomen einer Generation ergibt 1.
- Je größer die normalisierte Fitness eines Chromosoms, desto besser ist es.

Damit kann die normalisierte Fitness direkt als Wahrscheinlichkeit für die Selektion verwendet werden.

Die Selektion wird durch zufälliges Wählen von $popsize$ Listen nach normalisierter Fitness mit Zurücklegen realisiert. Das entspricht dem Prinzip der Rouletterad-Selektion.

Die ersten $crosssize$ ausgewählten Listen nehmen am Crossover teil.

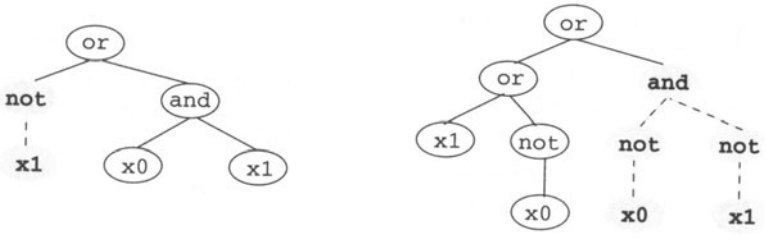


Abbildung 7.2 Die Bäume für die Listen `(or (not x1) (and x0 x1))` und `(or (or x1 (not x0)) (and (not x0) (not x1)))`

Crossover wird für Chromosomen in Form von Bäumen folgendermaßen realisiert. Wähle in zwei gegebenen Bäumen jeweils zufällig einen Knoten aus. Vertausche die durch die Knoten induzierten Unterbäume. Crossover produziert immer wieder korrekte Listen (Bäume).

Abbildung 7.2 zeigt zwei Chromosomen, die mittels Crossover gekreuzt werden sollen. Im ersten Chromosom wurde zufällig der Knoten `not` als Crossover-Punkt ausgewählt, im zweiten der Knoten `and`. Abbildung 7.3 zeigt die beiden Chromosomen, die durch Crossover, d.h. durch Vertauschung der zu den beiden ausgewählten Knoten gehörenden Teilbäume, entstehen.

Falls beim Crossover bei beiden Eltern die Wurzeln ausgewählt werden, ergeben sich durch Crossover wieder die beiden Eltern.

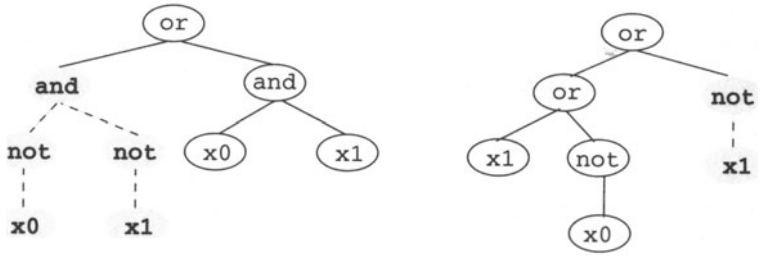


Abbildung 7.3 Bäume aus der vorhergehenden Abbildung nach Crossover, dem Austausch der markierten Unterbäume

Bei den genetischen Algorithmen und den Evolutionsstrategien ergibt die Rekombination eines Chromosoms mit sich selbst wieder das Chromosom, egal welchen Crossover-Operator man wählt. Bei der genetischen Programmierung kann die Kreuzung eines Chromosoms mit sich selbst zwei völlig neue Chromosomen erzeugen, wie Abbildung 7.4 zeigt.

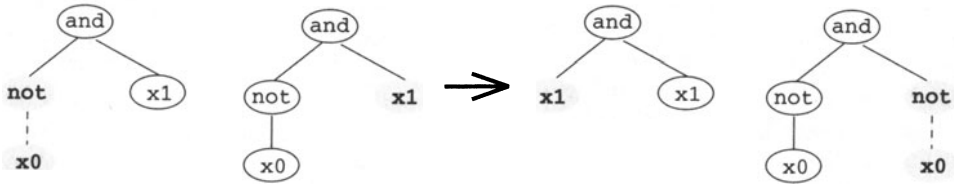


Abbildung 7.4 Crossover kann aus zwei gleichen unterschiedliche Chromosomen erzeugen

Wie schon bei der Startpopulation sollte auch die maximale Tiefe der Bäume, die beim Crossover entstehen, begrenzt werden. Man wählt für diese Begrenzung üblicherweise einen größeren Wert als bei der Startpopulation. Eine typischer Wert für eine maximale Tiefe ist $maxdepth = 17$, während die Tiefe der Bäume der Startpopulation häufig auf $indepth = 6$ begrenzt wird. Gängige Werte für die Populationsgröße und die Anzahl der am Crossover teilnehmenden Chromosomen sind $popsize = 500$ und $crosssize = 450$.

Im allgemeinen soll bei Crossover mit 90%-iger Wahrscheinlichkeit ein interner Knoten gewählt. Mit 10% Wahrscheinlichkeit wird nach einer Gleichverteilung ein Blatt, also ein Knoten, der ein Terminalsymbol enthält, gewählt. Der Zweck ist, dass nicht so häufig nur Terminalsymbole ausgetauscht werden sollen. Bei einer Gleichverteilung würde bei einem vollen Binärbaum mit ca. 50% Wahrscheinlichkeit ein Terminalsymbol gewählt werden.

Mutation ist bei der genetischen Programmierung ein optionaler Operator. Da Crossover „reichhaltiger" als bei den genetischen Algorithmen ist – das Kreuzen zweier identischer Listen kann zu zwei neuen Listen führen – ist Mutation nicht unbedingt erforderlich.

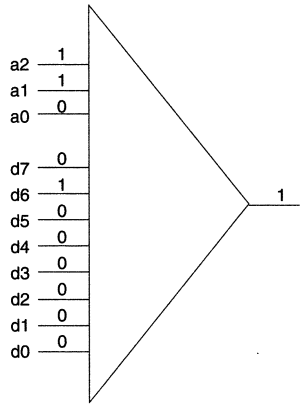


Abbildung 7.5 Boole'scher 11-Multiplexer

Beispiel 7.4 (Boole'scher 11-Multiplexer) Es soll in diesem Beispiel aus [Koz92] die Funktion eines Boole'schen 11-Multiplexer mittels genetischer Programmierung erlernt werden. Ein Boole'scher 11-Multiplexer arbeitet folgendermaßen. Er hat insgesamt elf Eingänge. Die Eingänge sind $a_2, a_1, a_0, d_7, \ldots, d_0$. Die Binärzahl $a_2a_1a_0$, die sich aus den ersten drei Eingängen ergibt, wird als Auswahl oder Index für die anderen acht Eingänge verwendet. Der Multiplexer gibt den Wert aus, der am Eingang $d_{a_2a_1a_0}$ anliegt, wobei $a_2a_1a_0$ als Binärzahl zu interpretieren sind. Die Belegung $a_2 = 1$, $a_1 = 1$ und $a_0 = 0$ ergibt als Binärzahl interpretiert den Wert 6. Somit gibt der Multiplexer bei dieser Belegung von a_2, a_1, a_0 den Wert aus, der am Eingang d_6 anliegt (siehe Abbildung 7.5).

Gesucht ist ein Baum, der den Boole'schen 11-Multiplexer realisiert. Da der Boole'sche Multiplexer eine Funktionen $\{0,1\}^{11} \to [0,1]$ ist, besteht unser Suchraum aus allen derartigen Funktionen. Von diesen Funktionen gibt es $2^{2^{11}} > 10^{616}$.

Als Terminalsymbole definieren wir die Eingabevariablen

$$\mathcal{T} = \{\texttt{A0, A1, A2, D0,...,D7}\}.$$

Die Funktionen, die wir verwenden wollen, sind

$$\mathcal{F} = \{\texttt{AND, OR, NOT, IF}\}.$$

Die rohe Fitness soll in diesem Beispiel die Anzahl der Fälle sein, bei denen von allen 2048 möglichen Eingaben die richtige Ausgabe geliefert wird. Entsprechend verwenden wir hier als standardisierte Fitness $s(\ell) = 2048 - r(\ell)$, wobei $r(\ell)$ die rohe Fitness des Chromosoms ℓ ist. Die adjustierte und die normalisierte Fitness werden wie bereits beschrieben aus der der standardisierten Fitness bestimmt.

Wählt die Parameter des Algorithmus nach den Werten, wie wir sie oben angegeben, allerdings unter Verwendung einer größeren Population mit $popsize = 4000$, erhält man für die erste Startpopulation folgendes Ergebnis. Die rohen Fitnesswerte liegen in der Startpopulation zwischen 768 und 1280, 23 Listen besitzen eine rohe Fitness von

1280, eine davon, `(IF A0 D1 D2)`, entspricht einem 3-Multiplexer. Die mittlere rohe Fitness liegt bei 1062.6, was in etwa dem zu erwartenden Wert von 1024 entspricht. Nach bereits neun Generationen wurde das Chromosom aus Abbildung 7.6 gefunden, das eine rohe Fitness von 2048 besitzt, d.h., das bei allen 2048 Eingaben die korrekte Ausgabe liefert.

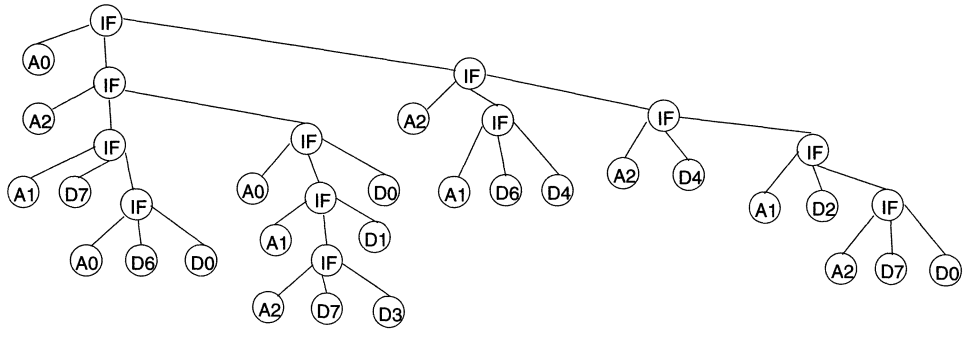


Abbildung 7.6 Multiplexer: 100% korrekte Lösung in Generation 9

7.3.3 Symbolische Regression

Bei der *symbolischen Regression* soll ein Regressionsproblem durch genetische Programmierung gelöst werden. Gegeben sind n (in der Regel mit Rauschen behaftete) Stützstellen $(x_i, y_i) \in \mathbb{R}^p \times \mathbb{R}$ $(i = 1, \ldots, n)$ einer unbekannten Funktion. Mittels genetischer Programmierung soll eine Funktion (Liste) gefunden werden, die die unbekannte Funktion möglichst gut annähert. Die Vorteile der symbolischen Regression bestehen darin, dass

- es für die genetische Programmierung egal ist, welches Fehlermaß (mittlerer/maximaler, quadratischer/absoluter Fehler, siehe auch Beispiel 2.2) man verwendet und
- dass man sich nicht auf eine spezielle Klasse von Funktionen einschränken muss.

Beispiel 7.5 Als anschauliches Beispiel dient uns die Funktion $f(x) = x^2 \cdot sin(x)$, die im Intervall $[-10, 10]$ approximiert werden soll. Wir verwenden $n = 41$ äquidistante Stützstellen:

$$\{(x_i, y_i) \mid x_i = -9.5 + i \cdot 0.5,\ y_i = f(x_i),\ i \in \{1, \ldots, 41\}\}$$

Abbildung 7.7 zeigt die Stützstellen der Funktion. Jedes Chromosom wird anhand dieser Stützstellen bewertet.

Als Menge der Terminalsymbole verwenden wir

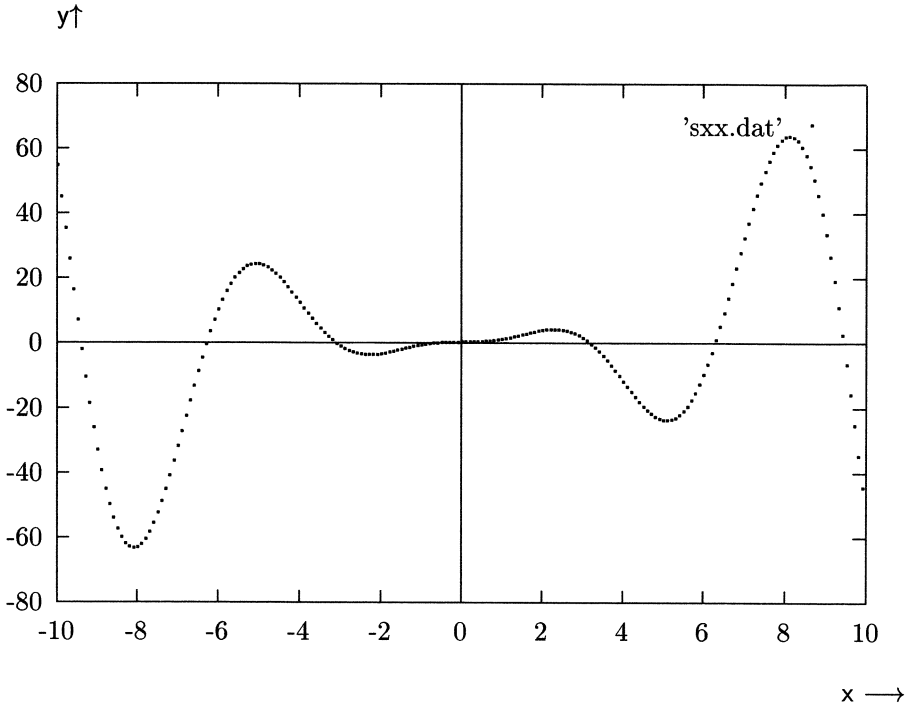


Abbildung 7.7 Die Stützstellen der Funktion

$$\mathcal{T} = \{x\}.$$

Man könnte zusätzlich noch reelle Konstanten wie 0 oder 1 in die Menge der Terminalsymbole aufnehmen. Dies führt allerdings zu einer deutlichen Vergrößerung des Suchraums.

Unsere Funktionssymbole sind

$$\mathcal{F} = \{+, *, -, /, \sin, \cos, \exp, \log, \sqrt{}, \text{neg}\}.$$

Die unäre Funktion neg kehrt das Vorzeichen ihres Argumentes um, d.h. $\text{neg}(\ell) = -\ell$. Da die Funktionen log und $\sqrt{}$ nicht für alle reellen Zahlen definiert sind, erweitern wir unseren Wertebereich um den Wert `NaN`. Alle Funktionen liefern als Wert `NaN`, falls mindestens einer ihrer Argumente `NaN` ist. Außerdem liefern log und $\sqrt{}$ den Wert `NaN` falls ihr Argument nichtpositiv bzw. negativ ist.

Wir wählen für dieses Beispiel eine Populationsgröße von 50 Chromosomen. Die Startpopulation wird wie bereits beschrieben zufällig erzeugt mit einer anfänglichen Maximaltiefe der Bäume von 6.

Als rohe und auch als standardisierte Fitness eines Chromosoms ℓ verwenden wir die Summe über die absoluten Fehler an den Stützstellen:

$$r(\ell) = s(\ell) = \sum_{i=1}^{41} |\ell(x_i) - y_i|$$

Falls ℓ für wenigstens eine der Stützstellen den Wert `NaN` liefert, wird die adjustierte und damit auch die normalisierte Fitness auf 0 gesetzt. Ein derartiges Chromosom hat somit keine Chance, in die nächste Generation übernommen zu werden.

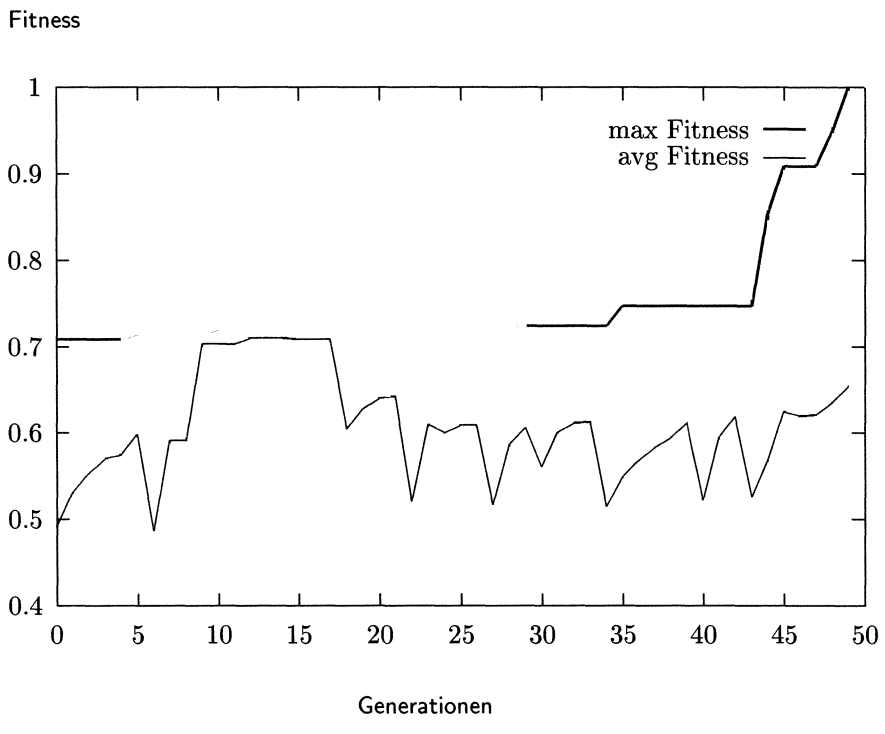


Abbildung 7.8 Fitnessentwicklung der Beispielanwendung

Abbildung 7.8 zeigt die Entwicklung der durchschnittlichen und der jeweils besten standardisierten Fitness pro Generation. Nach 48 Generationen wurde das Chromosom

$$\ell_{opt} = (\sin(x) * (x * (x/(\cos(x - x)))))$$

erzeugt, das in Abbildung 7.9 als Baum dargestellt ist. Das Chromosom lässt sich noch vereinfachen, da

$$x/(\cos(x - x)) \;=\; x/\cos(0) \;=\; 1$$

gilt. Daher entspricht das Chromosom genau der Funktion, von der wir die Stützstellen gewonnen haben.

Das Problem, dass ein recht komplexer Unterbaum eine sehr einfache Funktion realisiert, tritt häufiger bei der genetischen Programmierung auf. Daher sollen im nächsten Abschnitt weitere Techniken der genetischen Programmierung besprochen werden, die zur Behandlung dieses und anderer Problem eingesetzt werden können.

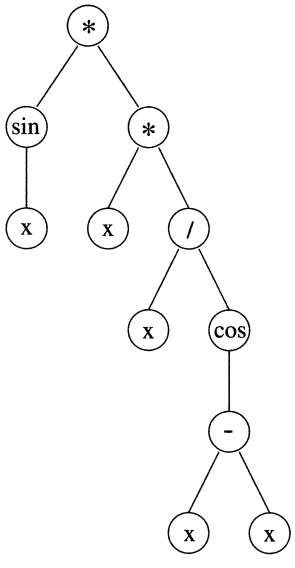


Abbildung 7.9 Das beste erzeugte Chromosom

7.3.4 Spezielle Techniken für die genetische Programmierung

Auch wenn Crossover der wichtigste Operator in der genetischen Programmierung ist, werden oft auch Mutation und weitere Operatoren angewendet.

Für die *Mutation* werden einzelne Chromosomen zufällig ausgewählt. Von dem zu mutierenden Chromosom (Baum) wird dann ebenfalls per Zufall ein Knoten als Mutationspunkt ausgewählt. Der Teilbaum unterhalb des Mutationspunktes wird dann durch einen neu erzeugten Baum ersetzt, das herausgeschnittene Fragment wird gelöscht. Der neu erzeugte Baum ist ein Chromosom, das nach den gleichen Regeln wie die Chromosomen der Startpopulation aufgebaut wird. Mutation wird bei der genetischen Programmierung üblicherweise nur auf einen sehr kleinen Teil der Population angewendet. Abbildung 7.10 veranschaulicht den Vorgang der Mutation.

Mutation hat bei der genetischen Programmierung meistens sehr starke Auswirkungen auf die Fitness des Chromosoms. Mutation wird vor allem dazu benötigt, um die Diversität in einer Population zur Vermeidung vorzeitiger Konvergenz zu erhöhen und ggf. aus der Population verschwundene Terminal- und Funktionssymbole wieder einzuführen.

Editing dient der Vereinfachung und Kompaktifizierung von Chromosomen. Editing behandelt das Problem, das am Ende des Beispiels 7.5 aufgetreten ist. Ein Chromosom oder ein Teil davon kann auf sehr komplizierte Weise eine recht einfache Funktion realisieren. Editing ist in jedem Fall als Nachbearbeitung der gefundenen besten Lösung nach Beendigung des Evolutionsprozesses empfehlenswert. Editing kann aber auch während

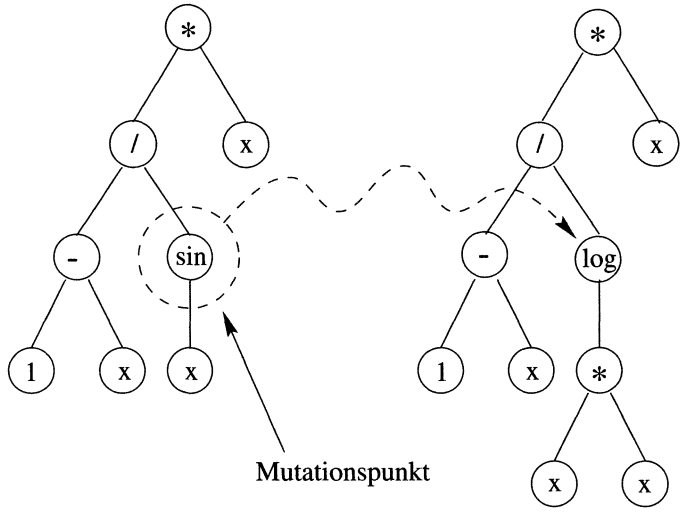


Abbildung 7.10 Durchführung einer Mutation

des Evolutionsprozesses – im Extremfall in jeder Generation und auf jedes Chromosom angewendet werden.

Die Anwendung von Editing hängt von der Menge der Funktionen ab, die bei der genetischen Programmierung zugelassen werden, und den Vereinfachungsregeln, die für diese Funktionen gelten bzw. beim Editing angewendet werden sollen. Der Boole'sche Ausdruck als Unterbaum einer Liste

$$a \vee ((b \vee (\neg a)) \wedge ((\neg b) \vee (\neg a)))$$

könnte zu a vereinfacht werden.

Editing kann auf das gesamte Chromosom angewendet werden und, wenn dies zu keiner Vereinfachung führt, rekursiv auf die Teilbäume, was allerdings einen hohen Rechenaufwand bedeutet. Neben der Vereinfachung von Lösungen kann Editing zwei völlig entgegengesetzte Effekte haben. Es kann vorkommen, dass ein gutes Chromosom durch einen relativ großen redundanten Teilbaum die Wahrscheinlichkeit verringert, durch Crossover oder Mutation in seiner Fitness verschlechtert zu werden. Ein Teilbaum, der durch die Liste `(x-x)*(x+((x*x)-1))` repräsentiert wird, liefert immer den Wert 0. Wenn Crossover oder Mutation im rechten Teil stattfinden, hat dies keine Auswirkung auf das Ergebnis, das der Baum liefert. Durch Editing würde diese „Schutzfunktion" zerstört werden.

Andererseits kann bei redundanten Chromosomen der gegenteilige Effekt eintreten. Der Teilbaum `(-1)*(-(-(-(-(-1)))))` liefert immer den Wert 1. Die Wahrscheinlichkeit, dass dieses Ergebnis durch Crossover oder Mutation zerstört wird, ist aber viel höher, als wenn man nur einen Blattknoten mit 1 hätte. Wäre dies auch die Ausgabe des

Teilbaums, die zu einer guten Fitness führt, wäre für das entsprechende Chromosom vorteilhaft, da die Wahrscheinlichkeit verringert wird, den Teilbaum durch Crossover oder Mutation zu zerstören.

Die *Kapselung* (*encapsulation*) wird – im Gegensatz zum Editing – gezielt dazu eingesetzt, um potentiell gute Teilbäume vor Zerstörung durch Crossover und Mutation zu schützen. Für einen Teilbaum wird eine neue Funktion definiert, die genau dieselbe erzeugt, wie der Teilbaum. Der gesamte Teilbaum wird dann durch einen einzigen Knoten, der die neue Funktion enthält, ersetzt.

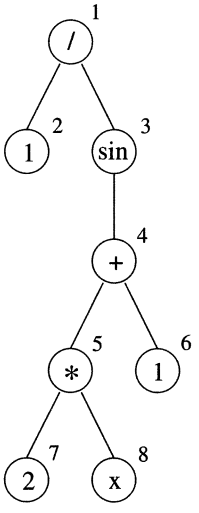


Abbildung 7.11 Der Teilbaum unterhalb Knotenpunkt 4 wird für die Kapselung selektiert.

Das Chromosom in Abbildung 7.11 besteht aus acht Knoten und berechnet die Funktion $\frac{1}{\sin((2*x)+1)}$. Für die Anwendung der Kapselung wählt man zufällig einen inneren Knoten des Baums aus, beispielsweise den Knoten 4. Der entsprechende Teilbaum berechnet die Funktion $(2*x)+1$. Als neue Funktion definieren wir somit $e_0 = (2*x)+1$. Die Anzahl der Argumente von e_0 entspricht der Anzahl von Blattknoten im zugehörigen Teilbaum, d.h. in diesem Fall drei Argumente. Die Ersetzung des Teilbaums führt zu dem vereinfachten Baum in Abbildung 7.12, wobei dem Knoten e_0 noch die Argumente $2, x$ und 1 zugeordnet sind.

Die so definierte Funktion e_0 wird in die Menge $\mathcal{F}$ aufgenommen, was nur bei Anwendung von Mutation im weiteren Verlauf des Evolutionsprozesses eine Rolle spielt. Durch Crossover und Mutation kann die Funktion e_0 dann später auch in anderen Chromosomen auftreten.

Automatisch definierte Funktionen (*ADF's*) haben eine ähnliche Funktion wie die Kapselung. Bei der Kapselung werden die gekapselten Funktionen zu dem Pool der Funktionssymbole hinzugefügt und stehen zumindest bei Mutation allen Chromosomen zur Verfügung. Bei automatisch definierten Funktionen hat jedes Chromosom die Möglichkeit, eine oder mehrere eigene Teilfunktionen zu entwickeln, die in dem zu dem

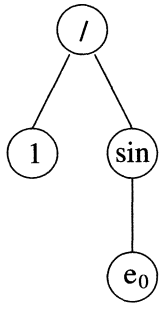


Abbildung 7.12 Chromosom mit neu geschaffener Funktion e_0

Chromosom gehörenden Baum ähnlich wie gekapselte Funktionen aufgerufen werden können. Ein Chromosom enthält bei Verwendung automatisch definierter Funktionen neben dem Baum, der wie bisher die eigentliche Funktion kodiert, die das Chromosom berechnet, noch eine üblicherweise feste Anzahl von weiteren, meist kleineren Bäumen, die Hilfsfunktionen repräsentieren. Jede dieser Hilfsfunktionen kann in der eigentlichen Funktion an einer oder mehreren Stellen jeweils durch einen einzelnen Knoten aufgerufen werden.

Automatisch definierte Funktionen fungieren in diesem Sinne als individuelle Unterprogramme der Chromosomen, die auch im Laufe der Evolution verändert werden können.

Es gibt eine Reihe von Ansätzen zur Erweiterung der Funktionenklasse im Rahmen der genetischen Programmierung. In [Lan98] werden komplexere Datenstrukturen eingeführt. [Mus97] ermöglicht die Benutzung von Quantoren $(\forall, \exists)$. [YC98b] führt LISP-typische Konstruktion zur Listenverarbeitung wie `MAP` und `FILTER` zu. Rekursion wird in [KBAK99] behandelt.

Bei der genetischen Programmierung kommen häufig unterschiedliche Datentypen vor. Beispielsweise könnte es bei symbolischer Regression sinnvoll sein, die Funktion `if a then b else c` in die Menge der Funktionen mit aufzunehmen, um eine stück- oder bereichsweise Definition der Regressionsfunktion zu ermöglichen. Das Argument a dieser Funktion oder die Funktion, die an die Stelle von a tritt, sollte einen Boole'schen Wert liefern. Da man die volle Definiertheit für die Funktionen bei der genetischen Programmierung verlangt, könnte man an dieser Stelle auch reelle Zahlen zulassen und beispielsweise definieren, dass der `then`-Teil genau dann ausgeführt werden soll, wenn a einen Wert kleiner oder gleich null liefert. Dies würde allerdings die Interpretierbarkeit oder Lesbarkeit des Chromosoms erschweren. Man sollte dann eher direkt die Funktion `if (a≤0) then b else c` definieren, die ausschließlich reelle Argumente verwendet. Wenn man allerdings komplexere Bedingungen wie `if ((x≥y) ∧ (z<x))...` zulassen möchte, erschwert die Erweiterung Boole'scher Funktionen auf reelle Werte auch in diesem Fall die Interpretierbarkeit. Das gleiche gilt für Probleme, bei denen gemischte Eingangsgrößen vorliegen wie häufig bei Klassifikationsproblemen. Bei der Diagnose von Krankheiten könnten z.B. das Geschlecht als binäre, die Anzahl der Herzinfarkte als

ganzzahlige und der Blutdruck als kontinuierliche Variable als Eingangsgrößen auftreten. Natürlich lässt sich ein Boole'scher Wert als reeller Wert aus der Menge $\{0, 1\}$ interpretieren. Es ist allerdings fraglich, ob dies für die genetische Programmierung sinnvoll ist. Als Alternative bietet sich an, *Typisierung* zu verwenden. Das bedeutet, dass jeder Variablen sowie jedem Argument und Rückgabewert einer Funktion ein Typ (z.B. Boolean, Integer, Real) zugeordnet wird. Bei der Erzeugung der Startpopulation und später bei der Anwendung von Operatoren wie Crossover und Mutation wird dann darauf geachtet, dass jeder Knoten des Baums einen passenden Typ hat. Wenn wir beispielsweise symbolische Regression für eine Funktion in einer Variablen x betreiben wollen und dabei als Funktionsmenge

$$\mathcal{F} = \{+, -, *, /, <, >, \texttt{if...then...else...}\}$$

verwenden, darf bei der Erzeugung eines Chromosoms der Startpopulation weder die Funktion $<$ noch die Funktion $>$ als erste Funktion gewählt werden, da bei der symbolischen Regression jedes Chromosom für eine Eingabe einen reellen Ausgabewert berechnen muss. Die Funktionen $<$ und $>$ haben aber jeweils zwei reelle Argumente und liefern einen Boole'schen Wert zurück. Wählt man als erste Funktion eines Chromosoms

$$\texttt{if...then...else...}$$

aus, muss das erste Argument entweder die Funktion $<$ oder die Funktion $>$ sein, da dies die einzigen Funktionen in $\mathcal{F}$ sind, die einen Boole'schen Wert zurückgeben. Auch das Terminalsymbol x ist nicht zugelassen, da es ebenfalls reell und nicht vom Typ Boolean ist. Das zweite und dritte Argument der If-Konstruktion kann wieder eine reelle Funktion oder direkt das Terminalsymbol x sein.

Ausführlichere und weitergehende Darstellungen zur Typisierung bei der genetischen Programmierung finden sich in [Mon95, Har97, MHR98, YC98a].

In [Vir03] wird ein Ansatz vorgestellt, der es erlaubt, geometrische Strukturen, wie sie bei Design- und Architekturanwendungen benötigt werden, mittels Techniken ähnlich der genetischen Programmierung zu entwerfen. Das Konzept der „Grammatical Evolution" wird in [OR01] vorgeschlagen, um nicht nur LISP-artige Programme, sondern Programme in beliebigen Programmiersprachen mittels evolutionärer Techniken entwickeln zu können.

7.3.5 Theoretische Untersuchungen

Schon bei genetischen Algorithmen muss ein sehr großer Aufwand betrieben werden, um theoretische Aussagen beispielsweise über Konvergenzeigenschaften machen zu können. Teilweise sind die Betrachtungen auch zu vereinfachend, so dass – wie beim Schema-Theorem und der Building-Block-Hypothese – die Interpretation der Resultate in Frage gestellt werden muss. Die wesentlich höhere Komplexität der genetischen Programmierung schränkt die Möglichkeiten für theoretische Untersuchungen noch weiter ein. Koza hat bereits in [Koz92, Koz94] festgestellt, dass die genetische Programmierung dazu tendiert, relativ große Programme mit schwer zu interpretierender Struktur zu generieren. Allerdings sind dies eher empirische Ergebnisse, die theoretisch schwer zu belegen

sind. Einige Analysen zur Komplexität der bei der genetischen Programmierung erzeugten Programme finden sich in [NB95]. Das Problem, wie Diversität in einer Population gemessen werden kann und die Korrelation mit der Fitness wird in empirisch untersucht [BGK04].

7.4 Weitere evolutionäre Algorithmen

7.4.1 Messy-GA

Der *MessyGA* (*mGA*) wurde von Goldberg, Deb und Korb entwickelt [GDK90, GKD89] und seine Struktur unterscheidet sich substantiell vom üblichen Ablauf evolutionärer Algorithmen. Konstruiert wurde er, um die Grundgedanken der Building-Block-Hypothese, also das Zusammensetzen guter Gesamtlösungen aus guten Teillösungen, auf einen Algorithmus zu übertragen (siehe Abschnitt 5.1). Hierbei ist zu beachten, dass die Building-Block-Hypothese nicht allgemein als gültig anerkannt wird. Daher wird der hier vorgestellte Ansatz eher aus Gründen der Vollständigkeit und aufgrund seiner ausgefallenen Struktur erwähnt.

Die Struktur der Chromosomen entspricht im Aufbau den für Permutationen verwendeten Chromosomen, bei denen jedes Gen zusätzlich zum Allel auch die Platznummer mit sich führt. Außerdem können die Chromosomen unterschiedlich lang sein und sowohl über-, als auch unterbestimmt vorliegen. Darunter ist zu verstehen, dass im überbestimmten Fall mehrere Gene zu denselben Platznummern vorkommen, die dann nach dem „First Come, First Served"-Prinzip ausgewertet werden. Schwieriger ist der unterbestimmte Fall. Zur Auswertung eines unterbestimmten Chromosoms wird ein so genanntes Template herangezogen, dass die beste Lösung der vohergehenden Generation darstellt. Mit Hilfe dieser Templates werden die fehlenden Gene für den Bewertungsvorgang eingefügt. Im Falle von Unterbestimmtheit lassen sich die Chromosomen als Äquivalenzklassen auffassen.

Entsprechend der Building-Block-Hypothese wird davon ausgegangen, dass sich die Lösung in Blocks der Länge k zerlegen lässt. Daher versucht der Algorithmus des mGA mit Hilfe wachsender Werte für k (Level), sukzessive die optimale Lösung zu erzeugen. Um keine wichtigen Building-Blocks außer acht zu lassen, wird die Startpopulation nicht zufällig gewählt, wie es sonst bei evolutionären Algorithmen üblich ist, sondern es werden alle möglichen Kombinationen von k Genen konstruiert und bewertet. Dem entsprechend wird mit wachsendem k diese so genannte *Initialisierungsphase* sehr zeitaufwendig, da insgesamt $2^k \binom{l}{k}$ Bewertungen durchgeführt werden müssen.

Auf die Initialisierungsphase folgt die *Ursprungsphase*, in der über eine Variante der binären Wettkampfselektion (Schwellwertakzeptanz) in mehreren Generationen eine Population geschaffen wird, die die vielversprechendsten Chromosomen der Länge k beinhalten soll. Unter Schwellwertselektion versteht man hierbei, dass nur solche Chromosomen einen Wettkampf gegeneinander ausführen, für die wenigstens eine feste Anzahl von Allelen (Schwellwert θ) für gleiche Gene vorliegen. Ansonsten wäre ein fairer Vergleich nicht möglich. Bei dieser Schwellwertselektion wird zu einem zufällig

gewähltem ersten Chromosom solange – ebenfalls zufällig – ein zweites Chromosom ausgewählt, bis beide Chromosomen wenigstens θ Gene gemeinsam haben. Durch dieses Verfahren wird eine Selektion innerhalb von Äquivalenzklassen der Ordnung θ durchgeführt. Mit einem Schwellwert von $\theta = l_1 l_2 / l$, wobei l_i für die Länge der beteiligten Chromosomens i steht, wurden gute Ergebnisse erzielt.

Nach Beendigung der Ursprungsphase folgt die so genannte *Vergleichsphase*. Auch in dieser Phase wird die oben erläuterte Schwellwertselektion zur Auswahl der Chromosomen für die nächste Generation genutzt. Zusätzlich wurde jedoch in den einzelnen Generationen ein „Cut and Splice"-Operator auf die ausgewählten Chromosomen angewendet. Dieser Operator trennt zwei zuvor ausgewählte Chromosomen jeweils an einer zufällig ausgewürfelten Position und setzt sie überkreuz wieder zusammen. Da in dieser Phase neue Chromosomen entstehen, ist es notwendig, nach Bildung jeder neuen Generation sämtliche Chromosomen neu zu bewerten. Wie zuvor in der Ursprungsphase wird auch in der Vergleichsphase eine feste Anzahl von Durchläufen (Generationen) durchgeführt. Nach Beendigung sämtlicher Durchläufe wird das beste Chromosom als Template für den nächsten Level $(k+1)$ gespeichert und der Algorithmus startet mit dem nächsten Level. Das Verfahren wird solange durchgeführt, bis ein Stoppkriterium erfüllt wurde.

Einer der großen Nachteile des mGA ist der große Rechenaufwand in der Initialisierungsphase jedes Levels. Ebenfalls als Nachteil wird von einigen Wissenschaftlern (z.B. Mitchell [Mit96]) die Verwendung von Täuschungsfunktionen als Testfunktionen (siehe Abschnitt 8.2.2) und der Mangel an theoretischem Hintergrund gesehen. Zur „Ausarbeitung eines theoretischen Hintergrundes" geben Goldberg, Deb, Kargupta und Harik [GKKH93] 1993 das folgende Statement ab:

> „Diejenigen unter uns, die komplexere Analysemethoden benutzt haben, wissen, dass diese Werkzeuge – Sachen wie Differentialgleichungen, Diffusionsmodelle, Markov-Ketten, Informationstheorie und ähnliches – mächtig sind, aber unhandlich, und in der Entwicklungsphase können unhandliche Werkzeuge den Todesstoss geben. Stattdessen haben wir unsere Intuition benutzt, wir haben unsere Vorstellungskraft benutzt, wir haben Dimensionsanalyse genutzt, wir haben sorgfältige Eingrenzungsexperimente benutzt und wir haben flexibel Sicherheitsdraht und Kaugummi eingesetzt, um eine Serie einfacher Modelle zu verbinden, die sich leicht anwenden lassen und genug Vorhersagefähigkeit besitzen, um den Entwicklungsprozess zu leiten."

Um den Aufwand der Initialisierungsphasen zu reduzieren, entwickelte dieselbe Forschergruppe den so genannten „*Fast Messy GA*" (fmGA [Kar95]), der auf eine vollständige Auflistung sämtlicher Blöcke der Länge k verzichtet. Stattdessen wird mit Hilfe der so genannten „probabilistic complete initialization" eine Startpopulation geschaffen. Bei diesem Verfahren wird die explizite Auflistung aller Blöcke der Länge k durch impliziten Parallelismus ersetzt, indem versucht wird, die beim mGA vorhandenen Äquivalenzklassen der Länge k in einer geringeren Anzahl längerer Strings unterzubringen. Je mehr Gene l' einer möglichen Lösung spezifiziert wurden, desto mehr unterschiedliche, durch k $(k < l')$ Gene charakterisierte Klassen sind in jeden Chromosomen enthalten. Um sicherzustellen, dass bei diesem Verfahren alle möglichen Building-Blocks der Länge k

in den Strings vorkommen, müssen im Mittel $\binom{l}{l'}/\binom{l-k}{l'-k}$ Zeichenketten der Länge l' verwendet werden. Da auch damit gerechnet werden muss, dass erwünschte Building-Blocks durch die Anwendung evolutionärer Operatoren zerstört werden, sollte die oben erwähnte Zahl zur Ermittlung der optimalen Populationsgröße noch mit einem Faktor multipliziert werden.

Da es auch beim fmGA letztendlich darum geht, die Building-Blocks der Länge k herauszufiltern, ist es notwendig, die Länge l' der Zeichenketten entsprechend zu reduzieren. Dafür wurde das so genannte „Building-Block-Filtering" eingeführt, das während der Ursprungsphase angewendet wird. Es handelt sich dabei um die vorher benutzte Schwellwertselektion, zu der noch ein Löschoperator hinzugenommen wurde. Es wird ein Löschplan verwendet, der angibt, wie oft Selektion zwischen den Löschphasen durchgeführt wird, und auf welche Genzahl λ_{i+1} der String von einer Anzahl λ_i in Phase i gekürzt werden soll. Der Löschfaktor sollte dabei so gewählt werden, dass im Mittel während der reinen Selektionsphasen mehr gute Lösungen dupliziert werden, als später durch das Löschen von Genen zerstört werden. Nach Ablauf der geänderten Ursprungsphase verläuft der fmGA analog zum mGA.

Es wurden verschiedene Test sowohl mit mGA, als auch mit fmGA zur Lösung von Täuschungsproblemen durchgeführt, in denen sich beide Algorithmenvarianten bewährt haben. Eine Anwendung im größerem Rahmen, bzw. auf eine größere Anzahl von Problemen fand jedoch nicht statt.

7.4.2 modGA

Der *modGA* wurde von Michalewicz [Mic96] entwickelt und stellt eine Variante des kGA dar. Hierbei unterscheidet sich der modGA insofern vom kGA, als dass nach der Selektionsphase genau festgelegt ist, welche Chromosomen mit welchem Operator verändert werden, bzw., ob sie unverändert übernommen werden. Das wird dadurch erreicht, dass eine Anzahl $(popsize - r)$ **verschiedener** Chromosomen aus der Population $P(t-1)$ für die Übernahme in die Population $P(t)$ ausgewählt wird, was gleichbedeutend mit dem Entfernen von r Chromosomen aus der vorhergehenden Population ist. Um diese *entfernten* Chromosomen zu ersetzen, werden anschließend r Chromosomen fitnessabhängig gewählt, für die sofort festgelegt wird, welcher Operator jeweils angewendet werden soll. Es wird also immer exakt ein Operator auf jedes Chromosom angewendet.

Durch dieses Vorgehen wird verhindert, dass Chromosomen mit herausragender Bewertung in der nächsten Population zu häufig vorkommen und ggf. sehr schnell die gesamte Population übernehmen. Durch die zwangsweise verschiedenen Chromosomen in der Gruppe der „Unveränderten" bekommen automatisch Chromosomen aus völlig unterschiedlichen Nischen eine Chance zu überleben, auch wenn die Bewertung mit den Besten nicht mithalten kann. Es handelt sich demnach bei dieser Algorithmenart um eine Nischentechnik. Im Gegensatz zum deterministischem Crowding (siehe Abschnitt 6.4.7) haben hier sowohl Eltern- als auch die daraus hervorgegangenen Chromosomen eine realistische Überlebenschance, zumal sie nicht direkt miteinander konkurrieren.

Für die Zuweisung der anzuwendenden Operatoren zu den Chromosomen kann unterschiedlich vorgegangen werden. Zum einen kann für jeden Operator vorher festgelegt

werden, wie viele Chromosomen mit ihm verändert werden sollen (direkte Zuweisung); es kann jedoch auch mit einer Wahrscheinlichkeit für die Zuweisung zur Crossovergruppe P_C gearbeitet werden, so dass alle Chromosomen mit einem Zufallswert größer als P_C automatisch der Mutationsgruppe zugewiesen werden. Ein möglicher Ablauf für das Erzeugen einer neuen Population für den modGA mit einem Parameter $0 < r < popsize$ ist:

1. Wähle aus $P(t)$ zufällig[1] r Elternchromosomen aus (*mit* Zurücklegen). Für jedes ausgewählte Chromosom bestimme eine Zufallszahl p mit $0 \leq p < 1$. Gilt $p < P_C$, so teile das Chromosom der Crossovergruppe zu, ansonsten der Mutationsgruppe.
2. Wähle zufällig $popsize - r$ *verschiedene* Chromosomen nach Fitness aus $P(t)$ aus und füge diese zu $P(t+1)$ hinzu.
3. Führe mit den r ausgewählten Chromosomen Crossover bzw. Mutation durch.
4. Füge die so erzeugten Nachkommen zu $P(t+1)$ hinzu.

Dieser Ablaufplan lässt sich problemlos auf die Anwendung von mehr als zwei Operatoren (beispielsweise bei zusätzlicher Verwendung von Inversion) erweitern, indem man das Intervall $[\,0, 1)$ in zu den jeweiligen Operatoren gehörende Abschnitte einteilt.

Michalewicz vertritt die Ansicht, dass durch die Art der Selektion nur sehr selten Chromosomen doppelt vorkommen und das Problem der Superindividuals daher kontrolliert werden kann. Das ist so jedoch nicht völlig korrekt, da unter den r zu verändernden Chromosomen beliebig viele Vertreter des gleichen Chromosoms sein dürfen. Treffen zwei gleiche Chromosomen aufeinander, so werden letztendlich zwei Kopien desselben Chromosoms in die neue Population übernommen. Hat ein Superindividual eine sehr hohe Auswahlwahrscheinlichkeit, so kann das in der Folge dazu führen, dass es in der nächsten Generation schwierig wird, $popsize - r$ verschiedene Chromosomen auszuwählen. Das Problem der Superindividuals ist also keineswegs gelöst, sondern lediglich abgemildert.

7.4.3 GENOCOP

Dieser Abschnitt befasst sich mit einem speziellem, von Michalewicz et. al. entwickeltem Programmpaket: *GENOCOP I* (GEnetic algorithms for Numerical Optimization of COnstrained Problems). Bei GENOCOP handelt es sich um eine Algorithmenform, die speziell auf die Optimierung reeller Funktionen mit l zu optimierenden Parametern unter Berücksichtigung von Nebenbedingungen zugeschnitten ist (siehe auch Abschnitt 6.3.6).

Bei den Nebenbedingungen muss es sich um ein System von *linearen* Gleichungen und Ungleichungen handeln. Um darüber hinaus *nichtlineare Ungleichungen* behandeln zu können, wurde GENOCOP I auf eine co-evolutionäre Algorithmenform erweitert (GENOCOP III). Zur zusätzlichen Einbeziehung *nichtlinearer Gleichungen* wurde GENOCOP II entwickelt, bei dem es sich allerdings weniger um eine Erweiterung des

[1] fitnessabhängig

evolutionären Algorithmus handelt, sondern um einen veränderten Aufbau des Optimierungstools, in das der evolutionäre Algorithmus eingebettet wurde. In diesem Abschnitt sollen alle drei Algorithmenvarianten vorgestellt werden.

GENOCOP I

Die von GENOCOP I bearbeitbare Problemklasse wird aus Optimierungsaufgaben von Funktionen $f : \mathbb{R}^l \rightarrow \mathbb{R}$ mit $f(x_1 \ldots, x_l) \stackrel{!}{=} opt$ gebildet, wobei die Lösung $\mathbf{x} = (x_1, \ldots, x_l)^T$ den Ungleichungen $g_i(\mathbf{x}) \geq 0,\ i \in \{1, \ldots, q\}$ und den Gleichungen $h_j(\mathbf{x}) = 0,\ j \in \{q+1, \ldots, m\}$ für $m \geq 0$ genügen muss. Zusätzlich müssen sich die Variablen in den Grenzen $u_i \leq x_i \leq o_i$ $(i = 1, \ldots, l)$ bewegen. Diese Randbedingungen lassen sich auch in Matrizenform darstellen:

(i) Gleichungen: $A\mathbf{x} = b$

(ii) Ungleichungen: $C\mathbf{x} \leq d$

(iii) Bereichsgrenzen: Geschrieben als Vektoren $\mathbf{u}$ und $\mathbf{o}$: $\mathbf{u} \leq \mathbf{x} \leq \mathbf{o}$

A und C sind hierbei Matrizen mit jeweils l Spalten.

Bei GENOCOP I wird so vorgegangen, dass das vorhandene System von Gleichungen und Ungleichungen umstrukturiert und um einige Variablen verringert wird, so dass sich alle zulässigen Lösungen in einer konvexen Menge begrenzt durch eine Menge von Ungleichungen befinden. Für die folgende Optimierung sind zwei Charakteristiken konvexer Mengen von besonderer Bedeutung:

1. Für zwei beliebige Punkte $\mathbf{x_1}$ und $\mathbf{x_2}$ aus einer konvexe Menge K liegt auch die Linearkombination $a \cdot \mathbf{x_1} + (1-a) \cdot \mathbf{x_2}$ mit $a \in [0, \ldots, 1]$ wieder in K.

2. Für jeden Punkt $\mathbf{x_1}$ im Inneren von K und jede durch diesen Punkt laufende Linie $\mathbf{p}$ gilt, dass $\mathbf{p}$ die Grenzen der konvexen Menge an genau zwei Punkten $\mathbf{s_1}$ und $\mathbf{s_2}$ schneidet.

Die erste Bedingung wird u.a. zur Definition von binären Operatoren verwendet, um zu verhindern, dass die neu erzeugten Chromosomen außerhalb des Suchraums liegen. Betrachtet man nur eine Koordinate eines l-dimensionalen Punktes und hält die übrigen fest, so wird klar, dass die Schnittpunkte $\mathbf{s_1}$ und $\mathbf{s_2}$ von den übrigen Koordinaten abhängen.

Die festen Grenzen ug_i und og_i für jede Variable x_i $(i \in \{1, \ldots, l\})$ gehen also über in dynamische Grenzen $left(i)$ und $right(i)$ in Abhängigkeit von den dafür festgehaltenen Werten aller Variablen x_j mit $j \neq i$, $j \in \{1, \ldots, l\}$. Es gilt also für ein $\mathbf{x} \in \mathcal{S}$ und ein $i \in \{1, \ldots, l\}$:

$$left(\mathbf{x}, i) := \min\{r \in \mathbb{R} | (x_1, \ldots, x_{i-1}, r, x_{i+1}, \ldots, x_n) \in \mathcal{S}\}$$

und

$$right(\mathbf{x}, i) := \max\{r \in \mathbb{R} | (x_1, \ldots, x_{i-1}, r, x_{i+1}, \ldots, x_n) \in \mathcal{S}\}$$

Um die Eigenschaften konvexer Mengen optimal nutzen zu können, ist es notwendig, die Gleichungen zu eliminieren, sie also in die Ungleichungen mit einzubeziehen. Für die Matrixschreibweise ist das gleichbedeutend damit, dass der Rang der Matrix A (rgA) berechnet wird. Daraus ergibt sich automatisch eine Anzahl von rgA eliminierbaren Variablen, die dann in den Ungleichungen durch eine Linearkombination der übrigen $l - rgA$ Variablen ersetzt werden können. Dabei wird wie folgt vorgegangen (siehe [Mic96], [MJ94]):

1. Berechnen des Ranges von A.
2. Darstellen der Variablen x_{i_1} bis x_{i_r} als Linearkombination der übrigen Variablen.
3. Aufteilen der Matrix A in zwei Matrizen A_1 und A_2 in der Form, dass A_1 genau dann die Spalte j aus A enthält, wenn gilt: $j \in \{i_1, \ldots, i_r\}$. Ansonsten gehört die Spalte j zur Matrix A_2. Das Verfahren wird für C analog durchgeführt.
4. Umsortieren der zugehörigen Variablen und Bereichsgrenzen in den Vektoren $\mathbf{x}$, $\mathbf{u}$ und $\mathbf{o}$ (beispielsweise $(x_1, \ldots, x_l) \rightarrow (x_{i_1}, \ldots, x_{i_l})$). Aufteilen in jeweils $\hat{\mathbf{x}}_\mathbf{1}$, $\hat{\mathbf{x}}_\mathbf{2}$, $\hat{\mathbf{u}}_\mathbf{1}$, $\hat{\mathbf{u}}_\mathbf{2}$, $\hat{\mathbf{o}}_\mathbf{1}$ und $\hat{\mathbf{o}}_\mathbf{2}$.

Dann lassen sich die Randbedingungen in der Form $A_1\hat{\mathbf{x}}_\mathbf{1} + A_2\hat{\mathbf{x}}_\mathbf{2} = \mathbf{b}$ schreiben (für C analog), und aufgrund der linearen Unabhängigkeit der Zeilen von A_1 existiert die Matrix A_1^{-1}. Auflösen dieser Gleichung nach $\mathbf{x}_\mathbf{1}$ ergibt:

$$\hat{\mathbf{x}}_\mathbf{1} = A_1^{-1}\mathbf{b} - A_1^{-1}A_2\hat{\mathbf{x}}_\mathbf{2}$$

Die neue Darstellung der Variablen aus dem Vektor $\hat{\mathbf{x}}_\mathbf{1}$ lässt sich nun problemlos in die Bereichsbegrenzungen und die Ungleichungen einsetzen, so dass sich das folgende System von Ungleichungen ergibt:

1. $\hat{\mathbf{u}}_\mathbf{1} \leq A_1^{-1}\mathbf{b} - A_1^{-1}A_2\hat{\mathbf{x}}_\mathbf{2} \leq \hat{\mathbf{o}}_\mathbf{1}$,
2. $\hat{\mathbf{u}}_\mathbf{2} \leq \hat{\mathbf{x}}_\mathbf{2} \leq \hat{\mathbf{o}}_\mathbf{2}$, und
3. $(C_2 - C_1A_1^{-1}A_2)\hat{\mathbf{x}}_\mathbf{2} \leq \mathbf{d} - C_1A_1^{-1}\mathbf{b}$.

Durch diese Umformulierung des Optimierungsproblems kann insbesondere der Suchraum beträchtlich eingeschränkt werden, ebenso wie die Anzahl der zu verwaltenden Variablen.

Beispiel 7.6 Es soll eine Funktion $f(x_1, x_2, x_3, x_4, x_5, x_6)$ mit den Gleichungen

$$\begin{aligned} h_1(\mathbf{x}) &:= x_1 + 3x_3 - 2x_4 = 6 \\ h_2(\mathbf{x}) &:= x_2 - x_4 + 3x_5 + 2x_6 = 10 \\ h_3(\mathbf{x}) &:= x_1 - x_5 = -4\,, \end{aligned}$$

und der Ungleichung

$$u_4(\mathbf{x}) := 2x_1 + x_4 \leq 13\,,$$

und den Bereichsgrenzen

$$\forall\, i \in \{1,\ldots,6\} \;:\; 0 \leq x_i \leq 10$$

minimiert werden.

In Matrixschreibweise erhält man daraus:

$$A \cdot \mathbf{x} = \mathbf{b}$$

$$\Leftrightarrow \begin{pmatrix} 1 & 0 & 3 & -2 & 0 & 0 \\ 0 & 1 & 0 & -1 & 3 & 2 \\ 1 & 0 & 0 & 0 & -1 & 0 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{pmatrix} = \begin{pmatrix} 6 \\ 10 \\ -4 \end{pmatrix}$$

$$\Leftrightarrow \begin{pmatrix} 1 & 0 & 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 & 3 & 2 \\ 0 & 0 & 1 & -2/3 & 1/3 & 0 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{pmatrix} = \begin{pmatrix} -4 \\ 10 \\ 10/3 \end{pmatrix}, \quad \text{also}$$

$$A_1 \cdot \hat{\mathbf{x}}_1 + A_2 \cdot \hat{\mathbf{x}}_2 = \mathbf{b}$$

$$\Leftrightarrow \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} + \begin{pmatrix} 0 & -1 & 0 \\ -1 & 3 & 2 \\ -2/3 & 1/3 & 0 \end{pmatrix} \cdot \begin{pmatrix} x_4 \\ x_5 \\ x_6 \end{pmatrix} = \begin{pmatrix} -4 \\ 10/3 \\ 10 \end{pmatrix}$$

Damit lassen sich die Linearkombinationen für x_1, x_2 und x_3 direkt ablesen und in Gleichungen und Ungleichungen einsetzen. Es ergibt sich damit für u_4, sowie die Unter- und Obergrenzen:

$$\begin{aligned} u_4(\mathbf{x}) &= 2(-4+x_5)+x_4 \leq 13 \Leftrightarrow u_4(\mathbf{x}) = 2x_5 + x_4 \leq 21 \\ 4 &\leq x_5 \overset{(\hat{u}_1)}{\leq} 10 \\ -10/3 &\leq 2/3x_4 - 1/3x_5 \\ -10 &\leq x_4 - 3x_5 - 2x_6 \leq 0 \\ 0 &\leq x_4 \leq 10 \\ 0 &\leq x_6 \leq 10 \end{aligned}$$

Durch das oben berechnete System von Ungleichungen lassen sich nun für jede Variable unter Beibehaltung der Werte für die übrigen Variablen die dynamischen Grenzen berechnen. So sind die Grenzen für die Variable x_4 im Punkt $p = (x_4, x_5, x_6) = (9, 6, 0)$ bei $left(4) = 8$ und $right(4) = 9$. Der Punkt $p = (*, 6, 0)$ ist also nur dann eine gültige Lösung, wenn x_4 aus dem Intervall $[8, 9]$ stammt.

Nachdem die Umformulierung des Problems abgeschlossen wurde, muss die Startpopulation aufgebaut werden. Sie besteht aus einer festgelegten Anzahl gültiger Lösungspunkte aus $\mathcal{S}$, bzw. der entsprechenden Anzahl von Kopien eines gültigen Punktes. Als Kodierung wird ein String der Länge $m-rgA$ verwendet, der für jede übriggebliebene Variable ein reellwertiges Gen aufweist, in dem die Werte der Variablen direkt eingetragen werden. Die Auswahl für die nächste Population erfolgt analog zum normalen genetischen Algorithmus fitnessabhängig, während die genetischen Operatoren so an die Problemstellung angepasst werden, dass ihre Anwendung wieder zu gültigen Lösungen führt. Wie bei einem normalem genetischem Algorithmus lassen sich die Operatoren in die Gruppen Mutation und Crossover einteilen, wobei es sowohl drei verschieden Mutations- als auch Crossoververfahren gibt.

Mutationsoperatoren:
Die folgenden Verfahren werden auf einen Vektor $\hat{\mathbf{x}} \in \mathcal{S}$ angewendet, wobei jeweils eine Variable $\hat{x}_i$ mit $i \in \{1, \ldots, n\}$ zufällig gewählt wird.

Uniform Mutation: Bei der Uniform Mutation wird für die Variable $\hat{x}_i$ zufällig nach einer Gleichverteilung auf dem Intervall $[left(\hat{\mathbf{x}}, i), \ldots, right(\hat{\mathbf{x}}, i)]$ ein neuer Wert $\hat{x_i}$ bestimmt. Die Hauptaufgabe dieses Operators ist insbesondere in der Startphase gegeben, wenn eine Startpopulation von mehrheitlich identischen Chromosomen so verändert werden soll, dass der Suchraum $\mathcal{S}$ gründlich durchforscht wird.

Grenzmutation: Unter einer Grenzmutation wird das Ersetzen der ausgewählten Koordinate $\hat{x}_i$ durch einen der durch die restlichen Koordinaten des Punktes bestimmten Randpunkt $left(\hat{\mathbf{x}}, i)$ oder $right(\hat{\mathbf{x}}, i)$ jeweils mit einer Wahrscheinlichkeit von $\frac{1}{2}$ verstanden. Dieser Operator gewinnt immer dann sehr stark an Bedeutung, wenn man im voraus weiß, dass die gesuchte Lösung auf dem Randbereich des Lösungsraumes liegt, wie es beispielsweise bei linearen Programmen der Fall ist (Optimierung einer Zielfunktion unter Berücksichtigung von durch linearen Ungleichungen gegebenen Randbedingungen).

Non Uniform Mutation: Die Non Uniform Mutation entspricht vollständig der bereits im Abschnitt 6.6 vorgestellten Non Uniform Mutation, wobei für die in der Formel enthaltene Funktion $\Delta(t, y)$ die folgende Struktur verwendet wurde:

$$\Delta(t, y) = y \cdot rand \cdot (1 - \frac{t}{T})^b$$

Hierbei ist $rand$ eine Zufallszahl nach einer Gleichverteilung auf $[0, 1]$, t die aktuelle Generation, bei T handelt es sich um die maximale Anzahl von Generationen und b gibt den Grad der „Nicht-Uniformität" an. y wird in Abhängigkeit von einer binären Zufallszahl durch $(right(\hat{\mathbf{x}}, k) - \hat{x}_k)$ oder $(\hat{x}_k - left(\hat{\mathbf{x}}, k))$

Crossover-Operatoren: Wie bei Crossover-Operatoren üblich, werden alle folgenden Operatoren auf zwei zuvor zufällig gewählte Chromosomen $x, y \in \mathcal{S}$ angewendet.

Einfaches Crossover: Für das einfache Crossover wird zufällig ein Crossoverpunkt $\nu \in \{1, \ldots, l-1\}$ gewählt und die darauf folgenden Allele werden in Form einer Linearkombination zwischen den Chromosomen ausgetauscht. Um sicherzustellen, dass die beiden neuen Chromosomen wiederum im Suchraum $\mathcal{S}$ liegen, wird die Linearkombination in der Form $a \cdot \mathbf{y} + (1-a) \cdot \mathbf{x}$, bzw. $a \cdot \mathbf{x} + (1-a) \cdot \mathbf{y}$ mit einem $a \in [0,1]$ verwendet, das der Konvexitätsbedingung genügt. Hierbei wird a so bestimmt, dass ein maximaler Austausch an genetischem Material stattfindet. Also:

$$a = \max\{\alpha \in [0,1] | (x_1, \ldots, x_\nu, \alpha \cdot y_{\nu+1} + (1-\alpha) \cdot x_{\nu+1}, \ldots, \alpha \cdot y_n + (1-\alpha) \cdot x_n) \in \mathcal{S}\}$$

Damit ergeben sich nach der Durchführung des Crossovers die folgenden zwei Chromosomen:

$$\begin{aligned} \mathbf{x}_{neu} &:= (x_1, \ldots, x_\nu, a \cdot y_{\nu+1} + (1-a) \cdot x_{\nu+1}, \ldots, a \cdot y_n + (1-a) \cdot x_n) \\ \mathbf{y}_{neu} &:= (y_1, \ldots, y_\nu, a \cdot x_{\nu+1} + (1-a) \cdot y_{\nu+1}, \ldots, a \cdot x_n + (1-a) \cdot y_n) \end{aligned}$$

Arithmetisches Crossover: Das arithmetische Crossover arbeitet nach demselben Prinzip wie das einfache Crossover. Es wird wieder auf die Konvexität des Suchraumes zurückgegriffen, indem wie zuvor ein $a \in [0,1]$ zufällig gewählt wird. Der Unterschied beim Operator liegt darin, dass in diesem Fall kein Crossoverpunkt notwendig ist, sondern statt dessen die Linearkombinationen $\mathbf{x}_{neu} := a \cdot \mathbf{x} + (1-a) \cdot \mathbf{y} \in \mathcal{S}$, und $\mathbf{y}_{neu} := a \cdot \mathbf{y} + (1-a) \cdot \mathbf{x} \in \mathcal{S}$ über die gesamten Vektoren $\mathbf{x}$ und $\mathbf{y}$ gebildet werden.

Heuristisches Crossover Das besondere am heuristischen Crossover ist, dass dabei höchstens ein neues Chromosom erzeugt wird, in dem die Eigenschaften beider Elternchromosomen vereint werden. Wie schon beim arithmetischen Crossover wird wieder eine Linearkombination $\mathbf{x}_{neu} := rand \cdot (\mathbf{x} - \mathbf{y}) + \mathbf{y}$ gebildet. Hierbei handelt es sich bei $rand$ um eine Zufallszahl aus dem Intervall $[0,1]$. Des weiteren muss gelten, dass die Bewertung $fit(\mathbf{y})$ höher ist als die Bewertung $fit(\mathbf{x})$ des Chromosoms $\mathbf{x}$. Durch diese Konstruktion ist nicht automatisch garantiert, dass das neue Chromosom ein Element des Suchraumes ist, so dass gegebenenfalls eine durch eine Maximalzahl begrenzte Wiederholung mit neuen Zufallszahlen durchgeführt werden muss. Durch den obigen Operator wird die Suche automatisch in Richtung des vielversprechenderen Chromosoms gelenkt, ohne die im anderen Chromosom enthaltenen Informationen völlig zu vernachlässigen.

Michalewicz testete das GENOCOP System an einer Gruppe von acht verschiedenen Testfunktionen mit unterschiedlichen Charakteristiken, bei denen unter anderem quadratische, nichtlineare und unstetige Funktion vorkamen. Diese Tests verliefen so erfolgreich, dass das GENOCOP I System als Grundlage zur Entwicklung von evolutionären Algorithmen zur Optimierung von Funktionen mit nichtlinearen Nebenbedingungen verwendet wurde.

Beispiel 7.7 (Transportproblem) Ein sehr wichtiges Anwendungsgebiet für die Optimierung von Funktionen unter Berücksichtigung von Nebenbedingungen ist der Bereich der Transportprobleme. Dabei geht es um die Minimierung einer Kostenfunktion, die

die entstehenden Transportkosten zwischen den q Quellen (Produzenten, Lieferanten oder Lager) und den s Senken (Verbraucher oder Kunden) berechnet. Hierbei gibt $Q(i)$ den „Lagerbestand" der Quelle i an, entsprechend $S(i)$ den Bedarf der Senke i. Die zu optimierende Funktion f hat als Definitionsbereich eine $q \times s$-Matrix, in der für jede Kombination von Quelle und Senke die transportierte Menge festgehalten wird:

$$f \quad : \mathbb{R}^{q \times s} \to \mathbb{R}^{+}$$

Gesamttransportkosten $\qquad (x_{1,1}, \ldots, x_{q,s}) \mapsto f(x_{1,1}, \ldots, x_{q,s})$

$\forall\, i \in \{1, \ldots, q\} \forall j \in \{1, \ldots, s\} \quad x_{i,j}$ Transportmenge von Quelle i zur Senke j

Oft wird f direkt als Summe über die Distanzen zwischen Quellen und Senken definiert:

$$f(x_{1,1}, \ldots, x_{q,s}) \;=\; \sum_{i=1}^{q} \sum_{j=1}^{s} distance(i,j) \cdot x_{ij}.$$

Die Randbedingungen beziehen sich auf die lieferbaren Mengen und den Mindestbedarf der Senken. Es gilt dabei:

- Der Lieferant kann insgesamt nicht mehr als seinen Bestand liefern:

$$\sum_{j=1}^{s} x_{ij} \leq Q(i) \quad (i = 1, \ldots, q).$$

- Der Bedarf der Senken muss mindestens gedeckt werden:

$$\forall j \in \{1, \ldots, q\} : \sum_{i=1}^{q} x_{ij} \geq S(j) \quad \Leftrightarrow \quad \forall j \in \{1, \ldots, q\} : \sum_{i=1}^{q} -x_{ij} \leq -S(j).$$

- Nur nicht-negative Mengen können transportiert werden:

$$x_{ij} \geq 0 \quad (i = 1, \ldots, q; j = 1, \ldots, s).$$

In Matrizenform ergibt sich:

$$\begin{pmatrix} 1 & 0 & \ldots & & & 0 \\ -1 & 0 & \ldots & & & 0 \\ 0 & 1 & 0 & \ldots & & 0 \\ 0 & -1 & 0 & \ldots & & 0 \\ 0 & 0 & 1 & 0 & \ldots & 0 \\ 0 & 0 & -1 & 0 & \ldots & 0 \\ \vdots & & & \ddots & & \vdots \\ 0 & \ldots & & & 0 & 1 \\ 0 & \ldots & & & 0 & -1 \end{pmatrix} \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} \leq \begin{pmatrix} u_1 \\ -l_1 \\ u_2 \\ -l_2 \\ \vdots \\ u_n \\ -l_n \end{pmatrix}$$

Durch die Randbedingungen $Cx \leq b$ ergibt sich für die Optimierung ein konvexer Suchraum

$$\mathcal{S} \;=\; \{x \in R^n | Cx^T \leq b\}.$$

GENOCOP II

Wie bereits zuvor erwähnt baut GENOCOP II auf dem System GENOCOP I auf, wurde jedoch so um eine Methodik aus dem Bereich der nichtlinearen Optimierung ergänzt, dass auch Funktionen mit nichtlinearen Bedingungen optimiert werden können. Zu diesem Zweck wurden die Grundlagen der Methode der sequentiellen quadratischen Straffunktionen übernommen [BA83, BN88]. Dabei wird die ursprünglich zu optimierende Funktion f durch die Funktion

$$F(\mathbf{x}, r) = f(\mathbf{x}) + \frac{1}{2\tau}\overline{A}^T\overline{A}, \qquad (\tau > 0)$$

ersetzt, wobei es sich bei $\overline{A}$ um einen Vektor bzw. eine Matrix handelt, in der eine Anzahl von a aktiver Bedingungen enthalten sind.

Zur Anwendung dieses Verfahrens wird die Menge der Randbedingungen in drei Gruppen aufgeteilt, nämlich in die linearen Randbedingungen L, die nichtlinearen Gleichungen N_e, und in die nichtlinearen Ungleichungen N_i. Die beiden letztgenannten Gruppen werden nach und nach in die Matrix A eingefügt, wobei wie folgt vorgegangen wird.

1. Auswahl eines Startpunktes $\mathbf{x}$.
2. Setzen der Matrix A, so dass sowohl N_e, als auch alle bei Verwendung von $\mathbf{x}$ verletzten Bedingungen $V \subseteq N_i$ darin enthalten sind. Hierbei gilt mit einem Schwellwert $\delta > 0$: $c_j \in N_i$ verletzt $\Leftrightarrow$ $c_j(\mathbf{x}) > \delta$.
3. Setzen des Strafwertes $\tau' = g(\tau, t)$, wobei g eine mit t fallende Funktion ist.
4. Optimieren der Funktion $F(\mathbf{x})$ mit GENOCOP I unter Berücksichtigung der linearen Gleichungen L.
5. Übernahme der besten Lösung x^* als neuen Startpunkt und Anpassen von A durch Entfernen der für x^* erfüllten bzw. Hinzufügen der verletzten Bedingungen aus N_i.
6. Durchführen der Schritte 3 bis 5 bis zur Erfüllung einer Abbruchbedingung.

Die von Michalewicz durchgeführten Tests [Mic96] verliefen generell positiv, was bedeutet, dass Lösungen gefunden wurden, die sehr nah am Optimum lagen. Allerdings muss darauf hingewiesen werden, dass nur in einem von fünf Testfällen keine Nebenbedingungen von den gefundenen Lösungen verletzt wurden.

GENOCOP III

Auch dieser Ansatz baut wieder auf dem ursprünglichen GENOCOP I Algorithmus auf, erweitert ihn jedoch um eine co-evolutionäre Population . Hierbei beinhaltet die erste Population P_S Suchpunkte, die den linearen Randbedingungen genügen, während die zweite Population P_r aus Referenzpunkten besteht, die allen Randbedingungen genügen. Gegebenenfalls muss wenigstens ein Referenzpunkt durch den Nutzer vorgegeben werden, der dann mit entsprechender Häufigkeit in der Population vertreten ist.

Die Bewertung neuer Punkte wird dann bei Chromosomen aus P_r direkt durchgeführt, was bei Punkten aus P_s nicht möglich ist, da es sich nicht um voll gültige Lösungen handelt. Um sie zu bewertbaren Lösungen zu machen, greift Michalewizc auf die Konvexitätseigenschaft zurück. Er wählt zur Bewertung eines Punktes $\mathbf{x}_s \in P_s$ fitnessabhängig einen Punkt $\mathbf{x}_r \in P_r$ und eine Zufallszahl $a \in (0,1)$, die dann in die Gleichung $\mathbf{x}_z = a \cdot \mathbf{x}_s + (1-a) \cdot \mathbf{x}_r$ eingesetzt werden. Das Auswürfeln einer Zufallszahl a wird solange wiederholt, bis $\mathbf{x}_z$ eine gültige Lösung darstellt. Nach der Berechnung von $\mathbf{x}_z$ wird die Fitness von $\mathbf{x}_s$ übernommen: $fit(\mathbf{x}_s) = fit(\mathbf{x}_z) = f(\mathbf{x}_z)$. Sollte der neu gefunden Referenzpunkt $\mathbf{x}_z$ besser sein als der ursprüngliche Referenzpunkt $\mathbf{x}_r$, so wird der ursprüngliche Punkt durch den neuen Referenzpunkt ersetzt. Das gleiche ist mit einer Wahrscheinlichkeit p_r für die Ersetzung des Suchpunktes $\mathbf{x}_s$ durch $\mathbf{x}_z$ der Fall.

7.5 Parallele und hybride Ansätze

Evolutionäre Algorithmen erfordern allein dadurch einen hohen Rechenaufwand, dass innerhalb einer Population und über viele Generationen eine große Anzahl von Lösungen evaluiert werden muss. Aus diesem Grund ist es wünschenswert, eine hohe *Parallelisierbarkeit* evolutionärer Algorithmen zu erreichen, um dadurch, falls erforderlich, die benötigte Zeit zur Berechnung einer guten Lösung zu verringern. Es gibt bei der Parallelisierung evolutionärer Algorithmen zwei Aspekte, die von Interesse sind:

- Welche Schritte eines evolutionären Algorithmus lassen sich wie möglichst effizient parallelisieren?
- Wenn ohnehin parallelisiert wird, ist es eventuell möglich, zusätzliche, im Sinne des Evolutionsprozesses vorteilhafte Techniken bei der Parallelisierung mit einzubeziehen?

Zunächst soll die erste Frage nach der grundsätzlichen Parallelisierbarkeit beantwortet werden. Wir betrachten dazu die wesentlichen Einzelschritte evolutionärer Algorithmen.

Erzeugen der Startpopulation: Diese anfängliche Schritt lässt sich sehr einfach parallelisieren, da in den meisten Fällen die Chromosomen der Startpopulation zufällig erzeugt werden. Auch Zusatzbedingungen wie bei der genetischen Programmierung, wo in der Startpopulation eine Verteilung über die Tiefen der die Chromosomen repräsentierenden Bäume festgelegt wird, lassen sich problemlos bei der Parallelisierung berücksichtigen. Fordert man, dass in der Startpopulation keine doppelten Chromosome vorkommen, so behindert das die Parallelisierbarkeit, da alle Chromosomen miteinander verglichen werden müssen. Insgesamt spielt die Erzeugung der Startpopulation bei der Parallelisierung eines evolutionären Algorithmus eine geringere Rolle, da die Startpopulation einmalig erzeugt wird, während alle anderen Schritte in jeder Generation wiederholt ausgeführt werden müssen.

Bewertung der Chromosomen: Da die Chromosomen unabhängig voneinander bewertet werden, eignet sich diese Aufgabe hervorragend für die Parallelisierung.

Berechnung der Fitnesswerte: Bei Evolutionsstrategien genügt die Bewertung der Chromosomen als Fitness aus. Sollen jedoch Selektionsmechanismen wie Rouletterad-Selektion oder Selektion nach dem Erwartungswertmodell angewendet werden, muss die Summe über die Bewertungen aller Chromosomen gebildet werden, was sich weniger gut parallelisieren lässt, da alle – üblicherweise verteilt berechneten Bewertungen – irgendwo zusammengeführt werden müssen.

Selektion: Die Parallelisierbarkeit der Selektion hängt sehr stark vom gewählten Selektionsmechanismus ab. Strategien wie das Eliteprinzip, Ranking oder das Erwartungswertmodell erfordern eine globale Betrachtung der Population und sind daher weniger geeignet für die Parallelisierung. Rouletterad-Selektion bereitet dagegen keine Probleme, da die Chromosomen unabhängig allein aufgrund ihrer eigenen Fitness ausgewählt werden. Für die Selektion ist allerdings – wie bereits bei der Fitnessberechnung erwähnt – die Bestimmung der Gesamtfitness der Population erforderlich, die sich der Parallelisierung weniger erschließt. Ideal ist die Wettkampf- und speziell die Zweikampf-Selektion. Hierbei müssen jeweils nur wenige – bei der Zweikampfselektion nur zwei – Chromosomen gegeneinander antreten und die Entscheidung über die Selektion kann allein aufgrund der Bewertungen der Chromosomen stattfinden, so dass keine Berechnung der Fitnesswerte erforderlich ist.

Die Zweikampf-Selektion lässt sich günstig auf $\frac{popsize}{2}$ Prozessoren implementieren (vorausgesetzt, $popsize$ ist ein Vielfaches von 2), indem pro Prozessor 2 Chromosomen bearbeitet werden. Jeder Prozessor bewertet die Chromosomen und wählt zwei weitere Chromosomen (anderer Prozessoren) aus, mit denen jeweils eines seiner beiden Chromosomen einen Zweikampf durchführt. Der Prozessor behält die beiden Siegerchromosomen.

Im Extremfall kann man bei der Zweikampf-Selektion sogar eine asynchrone Generationenfolge erzeugen. So wie die Bewertung für ein Chromosom durchgeführt wurde, sucht es sich zufällig einen „Gegner", der auf einem anderen Rechner oder Prozessor bewertet wurde und tritt gegen ihn an. Der Sieger überlebt und generiert außerdem noch einen mutierten Nachkommen.

Crossover: Da beim Crossover jeweils nur zwei Chromosomen beteiligt sind, ist auch hier eine sehr gute Parallelisierbarkeit gegeben.

Mutation: Durch die Anwendung der Mutation auf einzelne Chromosomen funktioniert die Parallelisierung reibungslos.

Abbruchbedingung: Die statische Bedingung, dass der Evolutionsprozess nach einer festen Anzahl von Generationen beendet wird, ist einfach zu handhaben. Andere Abbruchkriterien – z.B. „Das beste Chromosom muss eine gewisse Mindestgüte aufweisen" oder „Es ist über t Generationen keine Verbesserung des besten Chromosoms eingetreten" – sind weniger geeignet für die Parallelisierung.

Im Folgenden sollen einige evolutionäre Prinzipien angesprochen werden, die sich bei der Parallelisierung evolutionärer Algorithmen zusätzlich integrieren lassen.

7.5.1 Inselmodell und Migration

Beim *Inselmodell ohne Migration* wird ein und derselbe evolutionäre Algorithmus mit verschiedenen Populationen – und eventuell verschiedenen Parameterkonstellationen wie Crossover- oder Mutationsrate – parallel berechnet. Jeder evolutionäre Algorithmus repräsentiert eine „Insel", zwischen denen kein Austausch stattfindet. Dieses Vorgehen liefert dasselbe Ergebnis wie eine serielle entsprechend oft wiederholte Simulation des evolutionären Algorithmus.

Migration bezeichnet den Austausch einzelner Chromosomen zwischen den „Inseln" in gewissen Zeitabständen. Die Populationen entwickeln sich wie beim Inselmodell über mehrere Generationen parallel und unabhängig. Zu bestimmten Zeitpunkten – z.B. nach einer festen Anzahl von Generationen oder wenn sich über mehrere Generationen keine Verbesserung des besten Chromosoms ergeben hat – wandern einzelne Chromosomen von einer Insel zu einer anderen.

Beim Inselmodell findet keine direkte Rekombination zwischen Chromosomen verschiedener Inseln statt. Rekombination zwischen Chromosomen, die von verschiedenen Inseln stammen, kann nur vorkommen, wenn ein Chromosom durch die Migration von einer zu einer anderen Insel gewandert ist.

Im *Netzwerkmodell* oder *„stepping stone model"* werden die Inseln in Form eines Graphen angeordnet. Ein Austausch von Chromosomen ist nur zwischen benachbarten Inseln möglich.

Wettbewerb zwischen den Inseln (mit oder ohne Migration geschieht mit einer festen Gesamtpopulationsgröße, jedoch mit variabler Populationsgröße der Inseln und unterschiedlichen evolutionären Algorithmen auf den Inseln. Verschiedene Modelle der evolutionären Algorithmen werden durch Variation der Parameter wie Mutations- oder Crossoverwahrscheinlichkeit oder z.B. durch unterschiedliche genetische Operatoren realisiert.

Jeweils nach einer festen Anzahl von Generationen wird für jede Insel die durchschnittliche Fitness ihrer Chromosomen berechnet. Die Populationsgröße der einzelnen Inseln wird dann entsprechend ihrer durchschnittlichen Fitness erhöht oder verringert. Um das Untergehen einer Insel zu verhindern kann eine Mindestpopulationsgröße pro Insel festgelegt werden.

Neben der großen Flexibilität und der guten Parallelisierbarkeit bietet das Inselmodell mit Wettbewerb außerdem die Möglichkeit, Informationen über geeignete Parameter und genetische Operatoren bzw. über die für den jeweiligen evolutionären Algorithmus am besten passenden Chromosomen zu gewinnen. In diesem Sinne ergänzt dieses Modell die Adaptionsverfahren aus Abschnitt 6.7.2.

7.5.2 Zellulare genetische Algorithmen

Bei diesem Ansatz, auch *„isolation by distance"* genannt, werden die Rechner oder Prozessoren, die für die Parallelisierung verwendet werden, als Gitter auf einem Torus arrangiert wie in Abbildung 7.13 dargestellt.

Selektion und Crossover werden auf benachbarte Prozessoren begrenzt. Wenn jeder Prozessor beispielsweise ein Chromosom verwaltet, wählt der jeweilige Prozessor für sein

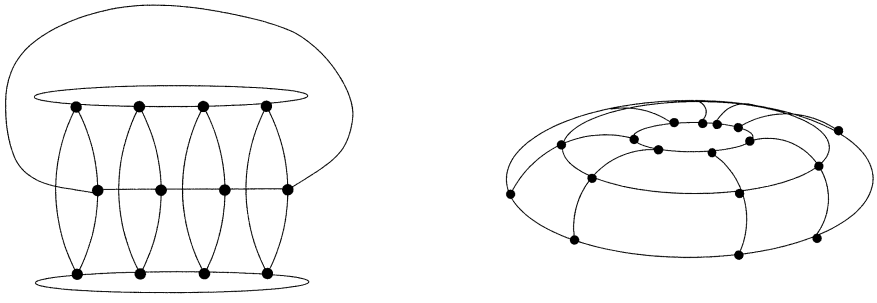


Abbildung 7.13 Anordnung der Prozessoren bei zellularen genetische Algorithmen als Torus

Chromosom das beste Chromosom der vier Nachbarprozessoren oder wählt eines der vier zufällig nach Fitness aus. Dann führt der Prozessor mit seinem und dem gewählten Nachbarchromosom Crossover durch. Er behält das bessere der beiden Chromosomen.

Mutation wird ebenfalls auf jedem Prozessor durchgeführt. Das bessere der beiden Chromosomen – das mutierte oder das ursprüngliche – wird behalten.

Es bilden sich „Inseln" als Gruppen von benachbarten Prozessoren mit ähnlichen Chromosomen heraus. Benachbarte Prozessoren besitzen nach einiger Zeit ähnliche Chromosomen, wodurch die teilweise zerstörerische Wirkung des Crossover-Operators abgemildert wird.

7.5.3 Mühlenbeins Paralleler Genetischer Algorithmus (PGA)

Beim herkömmlichen genetischen Algorithmus wird auf den passiven Chromosomen operiert. Die Idee bei *Mühlenbeins parallelen genetischen Algorithmus* [Müh91, Müh92b] ist nun, den Suchraum $\mathcal{S}$ mit einer Metrik oder Nachbarschaft zu versehen und jedes Chromosom aktiv nach dem Optimum in seiner Nähe suchen zu lassen. Außerdem tauschen nahe beieinander liegende Chromosomen Informationen aus.

Dieses Vorgehen ist vergleichbar mit einer Anzahl von Geologen, die in einer nebligen Berglandschaft den höchsten Punkt suchen und nur durch Zurufe kommunizieren können.

Die Vorgehensweise bei Mühlenbeins parallelen genetischen Algorithmus sieht folgendermaßen aus:

(1) Erzeuge eine Startpopulation

(2) Jedes Individuum führt lokales Hillclimbing durch. Dies entspricht der Mutation bei herkömmlichen evolutionären Algorithmen.

(3) Jedes Chromosom sucht einen Partner in seiner Nachbarschaft.

(4) Durch Crossover mit einem benachbarten Chromosom erzeugt jedes Chromosom einen Nachkommen.

(5) Die Nachkommen führen lokales Hillclimbing durch.

(6) Selektion wird zwischen den Elternchromosomen und den Nachkommen anhand der Fitness nach dem Eliteprinzip vollzogen. Das bessere von beiden Chromosomen überlebt jeweils.

(7) Falls das Abbruchkriterium nicht erfüllt ist, gehe zu Schritt (3).

Die lokalen Hillclimbing-Strategien dürfen für jedes Individuum verschieden sein. In diesem Sinn kann – ähnlich wie bei den Evolutionsstrategien die Mutationsrate – die geeigneten Hillclimbing-Strategie lokal für das Optimierungsproblem adaptiert werden.

Mühlenbeins paralleler genetischer Algorithmus läuft im Allgemeinen asynchron ab, d.h., es erfolgt keine strikte Trennung der Generationen . Nach Möglichkeit sollte insbesondere die Selektion asynchron und damit lokal durchgeführt werden. Auch bei Inselmodellen und auf Nachbarschaftsstrukturen basierenden Modellen lassen sich die meisten Operationen asynchron ausführen.

Einen ausführlichen Überblick mit umfangreicher Literatur zur Parallelisierung evolutionärer Algorithmen wird in [AT02] gegeben.

7.6 Lernende Classifier Systeme

Lernende Classifier Systeme wurden ursprünglich von Holland [HR78, HHNT86] eingeführt. Die Classifier Systeme selbst sind keine evolutionären Algorithmen. Allerdings basiert die Lernkomponente der lernenden Classifier Systeme auf genetischen Algorithmen.

Classifier Systeme sind Systeme von Wenn-Dann-Regeln, die man als *Classifier* bezeichnet. Im einfachsten Fall besteht die Prämisse einer Regel aus einer festen Anzahl von binären Bedingungen, die jeweils erfüllt oder nicht erfüllt sind. Der Konklusionsteil einer Regel enthält eine ebenfalls feste Anzahl binär kodierter Aktionen, die auszuführen sind, wenn die Bedingungen der Prämisse erfüllt sind. Üblicherweise versteht man Classifier Systeme als Sensor-Aktor-Systeme. Das bedeutet, dass die Sensoren überprüfen, welche Bedingungen erfüllt sind, und die Aktoren – entsprechend den Regeln – Aktionen ausführen.

Beispiel 7.8 Wir betrachten einen sehr einfachen Roboter, der mit vier einfachen Sensoren ausgestattet ist, die die Umgebung nach vorne, hinten, links und rechts untersuchen können. Jeder der vier Sensoren erzeugt zwei binäre Meldungen:

1. Wenn sich in der Richtung des Sensor ein Hindernis befindet, gibt der Sensor eine 1 ansonsten eine 0 als erste Meldung aus.

2. Wenn sich in der Richtung des Sensor ein bestimmtes Zielobjekt (beispielsweise im Falle eines Infrarotsensors ein Objekt mit einer bestimmten Temperatur) befindet, gibt der Sensor eine 1 ansonsten eine 0 als zweite Meldung aus.

In diesem Fall bestünde der Bedingungsteil einer Regel aus insgesamt acht binären Eingaben.

Als Aktionen stehen dem Roboter vorwärts fahren, um 90° nach links und um 90° nach rechts drehen zur Verfügung. Damit besteht der Konklusionsteil aus drei Bit, je ein Bit für eine der drei Aktionen. Eine 1 zeigt an, dass die entsprechende Aktion ausgeführt werden soll, eine 0 bedeutet, dass die Aktion nicht ausgeführt wird. Das wenig sinnvolle, gleichzeitige Ausführen der Drehung nach links und nach rechts soll hier zu keiner Aktion führen.

Eine Regel eines entsprechenden Classifier Systems für diesen Roboter könnte folgendermaßen lauten:

wenn:	Hindernis vorn:	1
	Hindernis hinten:	1
	Hindernis links:	0
	Hindernis rechts:	0
	Zielobjekt vorn:	0
	Zielobjekt hinten:	0
	Zielobjekt links:	0
	Zielobjekt rechts:	1
dann:	vorwärts:	0
	nach links:	0
	nach rechts:	1

Diese Regel besagt, dass der Roboter sich nach rechts drehen soll, wenn sich vor und hinter ihm ein Hindernis und nur rechts von ihm das Zielobjekt befindet.

Die Regeln eines Classifier Systems werden häufig in der Kurzform als Sequenzen von Nullen und Einsen geschrieben, eventuell mit einem Doppelpunkt zwischen Bedingungs- und Konklusionsteil. Die Regel aus Beispiel 7.8 würde somit

$$11000001 : 001$$

lauten.

Es ist nicht unbedingt sinnvoll, von einer Regel zu verlangen, dass alle Bedingungen spezifiziert werden müssen. Wenn das primäre Ziel im Beispiel 7.8 in der Verfolgung des Zielobjektes besteht, dann ist die einzig wichtigen Bedingung in der Beispielregel, dass sich das Zielobjekt auf der rechten Seite befindet und dass dort kein Hindernis vorhanden ist. Aus diesem Grund lässt man im Bedingungsteil einer Regel auch das Platzhaltersymbol # zu. Das bedeutet, dass die entsprechende Bedingung für diese Regel irrelevant ist. Man würde daher die Regel aus Beispiel 7.8 durch die allgemeinere Regel

$$\#\#\#0\#\#\#1 : 001$$

ersetzen.

Die Bedingungen in der Prämisse und die Aktionen im Konklusionsteil müssen nicht unbedingt binär sein. Es genügt, wenn sie binär kodiert sind. Würde man in Beispiel 7.8 nicht die binären Bedingungen verwenden, ob ein Hindernis in der jeweiligen Richtung vorhanden ist, sondern den Abstand ganzzahlig in Metern angeben, könnte man jeweils mehrere Bit zu Kodierung verwenden. Wenn die Reichweite des Sensors oder der Kamera sieben Meter beträgt, könnte man die Entfernung in drei Bit als Binärzahl kodieren. Auch hier kann die Verwendung des Platzhaltersymbols hilfreich sein. Will man beispielsweise eine Ausweichaktion durchführen, wenn ein Hindernis in höchstens drei Metern Entfernung auftaucht, wäre die entsprechende Kodierung der drei Bit, d.h. der Binärzahl 0##. Generell können bei einem Classifier System beliebige Eingaben zugelassen werden. Sie müssen sich nur geeignet binär kodieren lassen.

Durch die Verwendung des Platzhaltersymbols # kann es speziellere und allgemeinere Regeln geben. Im obigen Beispiel stimmen die auszuführenden Aktionen der spezielleren Regel 11000001:001 mit denen der allgemeineren Regel ###0###1:001 überein. Es kann durchaus sinnvoll sein, dass die speziellere Regel andere Aktionen verlangt. In diesem Fall bezeichnet man die allgemeinere Regel als *Default-Regel* und die speziellere Regel als *Ausnahmeregel*.

Classifier Systeme werden – wie bereits erwähnt – vorwiegend im Sinne von Sensor-Aktor-Systemen eingesetzt. Typische Anwendungen sind daher steuerungs- oder regelungstechnische Probleme sowie autonome Agenten oder Roboter. Classifier Systeme können aber auch als Klassifikatoren verwendet werden. Bei der Klassifikation muss aus mehreren Attributen ein weiteres diskretes Attribut vorhergesagt werden. Beispielsweise soll anhand von Zustands- und Fehlermeldungen verschiedener Steuergeräte in einem Auto auf die Fehlerursache geschlossen werden, bei der es sich daher das vorherzusagende Attribut handelt. Somit wird der Konklusionsteil einer Regel nicht als Aktion, sondern als Fehlerklasse interpretiert. Für jede Fehlerursache könnte ein Bit im Konklusionsteil reserviert werden. Eine Eins deutet darauf hin, dass die entsprechende Fehlerursache vorliegt, eine Null besagt das Gegenteil. Diese Interpretation eines Classifiers System ist allerdings nur bedingt sinnvoll, da üblicherweise beim Lernen ein dynamisches System vorausgesetzt wird, bei dem die Aktionen in einer zeitlichen Abfolge betrachtet werden.

Während der Anwendungsphase („*Performance Phase*") arbeitet ein Classifier System, indem es Eingaben erhält, mit denen die Bedingungen der Regeln überprüft werden. Die Regeln, bei denen die Bedingungen erfüllt sind, werden in das „Match Set" aufgenommen. Die Regeln des Match Set können unter Umständen widersprüchliche Aktionen in den Konklusionsteilen enthalten. Es muss daher eine Aktion bzw. eine Bitfolge ausgewählt werden, die eine oder mehrere Aktionen kodiert. Wir werden im Folgenden immer von einer auszuführenden Aktion sprechen, auch wenn diese sich aus mehreren Bit und eventuell mehreren gleichzeitig auszuführenden Teilaktionen zusammensetzt. Die Auswahl der Aktion geschieht üblicherweise nach dem Schema der Rouletterad-Selektion. Dazu muss jeder Regel ein Nützlichkeitswert („strength") zugewiesen werden. Wie diese Nützlichkeitswerte zu berechnen sind, werden wir später noch erläutern.

Eine Aktion erhält als „Fitnesswert" die Summe aller Nützlichkeitswerte der Regeln des Match Set, in denen sie als Konklusionsteil Set auftritt. Die auszuführende Aktion wird nun entsprechend dieser Fitnesswerte mittels einmaliger Rouletterad-Selektion ausgewählt.

Die reine Anwendungsphase verändert das Classifier System nicht. Es kann nicht an die „Umgebung" adaptiert werden. Für ein *lernendes Classifier System* wird daher noch eine Lern- oder *Reinforcement*-Phase benötigt. Unter „Reinforcement Learning" oder verstärkendem Lernen versteht man einen Lernprozess, bei dem eine Antwort nur als gut oder schlecht bewertet wird. Tendiert die Antwort in die richtige Richtung, wird die Regel oder der entsprechende Teil des lernenden Systems „belohnt" und „verstärkt".

Bevor die Reinforcement-Phase eines lernenden Classifier ausgeführt werden kann, muss am Ende der Anwendungsphase eine Bewertung der ausgeführten Aktion vorgenommen werden. Alle Classifier (Regeln), die sich in der Anwendungsphase im Match Set befanden und deren Aktion mit der ausgeführten Aktion übereinstimmt, werden in das aktuelle „Action Set" $A(t)$ aufgenommen. Die ausgeführte Aktion wird bewertet. Nur wenn sie als positiv anzusehen war, wird eine Belohnung b verteilt. Diese Belohnung wird unter den Classifiern in $A(t)$ aufgeteilt. Von jedem Classifier in $A(t)$ wird zunächst ein prozentualer Anteil a $(0 < a < 100)$ abgezogen und in einem gemeinsamen „Topf" $T(t)$ abgelegt. Danach wird der Nützlichkeitswert jedes Classifiers aus $A(t)$ um

$$\frac{b \cdot \frac{a}{100}}{\mathsf{card}(A(t))}$$

erhöht. Außerdem werden die Classifier des vorhergehenden Action Set $A(t-1)$ belohnt. Ihre Nützlichkeitswerte werden jeweils um

$$\frac{b \cdot \tilde{a}}{\mathsf{card}(A(t))} \qquad (0 < \tilde{a} < 1)$$

erhöht. Dieses Verfahren zur Erhöhung der Nützlichkeitswerte wird auch als „Implict Bucket Brigade" Algorithmus bezeichnet.

Mit dem Bucket Brigade Algorithmus können zwar die Nützlichkeitskeitswerte der einzelnen Classifier verändert werden, die Menge der Regeln bleibt dadurch aber unverändert. Man müsste daher am Anfang mit einer extrem großen Regelmenge starten, damit (fast) alle möglichen Regeln im Classifier System enthalten sind. Statt mit einer riesigen Regelmenge zu arbeiten, setzt man genetische Algorithmen ein, um die Regelmenge von Zeit zu Zeit zu modifizieren. Der genetische Algorithmus wird in bestimmten Zeitabständen oder mit einer gewissen Wahrscheinlichkeit nach der Beendigung einer Reinforcement-Phase aufgerufen. Für den genetischen Algorithmus dienen die Nützlichkeitswerte der Classifier als Fitnesswerte. Die möglichen Allele für die Gene in den Bedingungsteil der Classifier sind 0,1 und #, während für die Gene im Aktionsteil nur 0 und 1 zugelassen sind. Es empfiehlt sich, Nischentechniken zu verwenden, um nützliche Regeln, deren Bedingungsteil aber seltener auftritt, nicht auszurotten. Die Anzahl der durch den genetischen Algorithmus geänderten Classifier sollte recht klein – maximal bei 5% – gehalten werden. Im Rahmen der Ausführung des genetischen Algorithmus sollten außerdem Maßnahmen ergriffen werden, um Regeln für Fälle zu erzeugen, bei denen bisher keine Regel gegriffen hat, d.h. für Fälle, wo das Action Set leer war.

Es gibt eine Reihe von Weiterentwicklungen der lernenden Classifier Systeme, unter denen insbesondere die XCS Classifier Systeme zu nennen sind [Wil95], bei denen unter Anderem eine Schätzung der Belohnung in die Classifier mit einbezogen wird und die

dadurch eine bessere Leistungsfähigkeit als andere lernende Classifier Systeme zeigen [BKLW04].

Die oben beschriebene Art lernender Classifier Systeme wird auch als *Michigan-Ansatz* bezeichnet [HR78, Gre94]. Das Charakteristische beim Michigan Ansatz besteht darin, dass die gesamte Population die Regelbasis zur Lösung des Problems bildet.

Dagegen verwendet der *Pittsburgh-Ansatz* eine Population von Regelbasen. Beim Michigan-Ansatz entsprechen die Chromosomen den Regeln, während beim Pittsburgh-Ansatz ein einzelnes Chromosom eine ganze Regelbasis kodiert [Smi80].

Das *iterative Regellernen* ist verwandt mit dem Michigan-Ansatz. Es wird dabei nur eine Population von Regeln benutzt. Die eigentlich Regelbasis wird jedoch schrittweise konstruiert. Aus der Population von Regeln wird jeweils die beste entnommen und in die endgültige zu lernende Regelbasis eingefügt. Damit nicht immer wieder dieselbe Regel eingefügt wird, entfernt man sie aus der Population oder versieht sie zumindest mit einem Strafterm [Ven93].

8 Testumgebungen

Will man die Leistungsfähigkeit eines evolutionären Algorithmus für ein spezielles Problem oder eine Problemklasse beurteilen, sollten geeignete Tests durchgeführt. In diesem Kapitel sollen die Vorgehensweise bei derartigen Tests sowie einige Testfunktionen vorgestellt werden.

8.1 Aufbau einer Testumgebung

Um einen evolutionären Algorithmus in einer korrekten Umgebung testen zu können, muss genau festgelegt sein, welche Art von Test durchgeführt werden soll. Generell lassen sich die notwendigen Tests in verschiedene Klassen einteilen:

1. Test mit bis auf Parametereinstellungen identischen Algorithmen (Parameteradaption).
2. Test auf Leistungfähigkeit sowohl speziell auf einen Aufgabentyp beschränkt, als auch generell für unterschiedliche Aufgabentypen, je nachdem, ob es sich um die Lösung einer bestimmten Aufgabe oder um die Entwicklung eines neuen Algorithmentyps handelt.
3. Vergleich unterschiedlicher Algorithmenarten (z.B. Genetischer Algorithmus versus Hillclimbing).

Allen Tests gemeinsam ist die Notwendigkeit, eine im statistischen Sinne ausreichende Anzahl von Simulationsläufen durchzuführen und den Mittelwert über die Ergebnisse zu berechnen. Um fundierte Aussagen zu treffen, bietet es sich an, nicht nur die Mittelwerte (z.B. der besten erzielten Fitness jedes Simulationslaufes) zu bestimmen, sondern auch ein Konfidenzintervall zu bestimmen, so dass das entsprechende Intervall um den Mittelwert mit genügend hoher Wahrscheinlichkeit (95% oder mehr) überdeckt. Beim Vergleich der Leistungsfähigkeit verschiedener Algorithmen können statistisch fundierte Aussagen mittels Hypothesentests getroffen werden.

Die erste der oben aufgeführten Testvarianten sollte eigentlich bei jedem evolutionären Algorithmus durchgeführt werden, da die Parameter wie Populationsgröße, Mutations- und Crossoverraten stark mit der untersuchten Aufgabenstellung variieren können. Das betrifft allerdings nur nicht-adaptive Algorithmen (siehe hierzu Abschnitt 6.7.2), da dort zumindest die beiden letztgenannten Parameter häufig vom Algorithmus selbst angepasst werden. Die Ausrichtung des Algorithmus auf ein spezielles Problem entscheidet dann über die Anzahl der zu verwendenden Testfunktionen. In diesem Sinne kann es zu Überschneidungen mit der zweiten Variante kommen, da auch dort Leistungsfähigkeit und Vielseitigkeit (bei fest gewählten Parametern) nur durch die Verwendung einer größeren Anzahl von Aufgabenstellungen garantiert werden kann. Dem

entsprechend wurden frühzeitig beispielsweise im Bereich der Funktionsoptimierung feste Gruppen von Testfunktionen zusammengestellt, die einen möglichst großen Bereich an unterschiedlichen Eigenschaften abdecken. De Jong [DJ75] stellte in seiner Doktorarbeit fünf Funktionen zusammen, die den folgenden Eigenschaften abdecken:

- stetig / nicht stetig
- konvex / nicht konvex
- unimodal / multimodal
- niedrigdimensional / hochdimensional
- deterministisch / stochastisch

Diese Funktionen bilden bis heute die Grundlage der meisten Testverfahren, wenn sie auch zum Teil kleineren Änderungen unterzogen wurden. Dem entsprechend wird im folgenden Abschnitt näher auf sie eingegangen. Gerade weil diese Funktionen sehr häufig verwendet werden, sind sie für der Test neuer Verfahren besonders geeignet, da entsprechend viel Vergleichsmaterial vorliegt.

Der Vergleich mit anderen Algorithmenarten (Standardalgorithmen), wie sie beispielsweise in Kapitel 3 vorgestellt wurden, ist gerade bei der Verwendung neuer Verfahren aus dem Bereich der evolutionären Algorithmen angeraten, da diese Verfahren ihre Tauglichkeit gegenüber den anerkannten Standardverfahren erst unter Beweis stellen müssen. Insbesondere muss bei der Auswahl der Vergleichsverfahren die Anwendbarkeit auf die spezielle Aufgabenstellung berücksichtigt werden, da die meisten dieser Verfahren sehr stark spezialisiert sind (Beispiel: Tabu-Suche in der kombinatorischen Optimierung).

Ebenfalls problematisch ist die Schaffung einer fairen Testumgebung für zu vergleichende Algorithmen, um für alle Verfahren die gleichen Voraussetzungen zu schaffen. So steht einer Population von $popsize$ Chromosomen beim Vergleich mit einem Hillclimbing-Algorithmus lediglich ein Chromosom gegenüber, was es notwendig macht, das Hillclimbing nacheinander auf $popsize$ zufällig konstruierte Chromosomen anzuwenden und über die Ergebnisse zu mitteln. Selbst beim Vergleich verschiedener Typen von evolutionären Algorithmen ist eine sorgfältige Anpassung der Parameter notwendig, da z.B. beim modGA (siehe Abschnitt 7.4.2) keine Mutationwahrscheinlichkeit p_m existiert, sondern jeweils ein Gen bei einer fest gewählten Anzahl von Chromosomen mutiert wird. Die Mutationswahrscheinlichkeit der Vergleichsalgorithmen muss also so gewählt werden, dass im Mittel dieselbe Anzahl von Mutationen pro Generation stattfindet. Dasselbe gilt für Crossover-Operatoren.

Ebenfalls problematisch sind Tests mit aufeinander aufbauenden Teilproblemen, wie sie beispielsweise bei der Optimierung von nacheinander auf einen Flughafen anfliegenden Flugzeugen auftreten können (Flugroutenoptimierung). Eine gute Lösung für das zuerst optimierte Teilproblem kann verhindern, dass für nachfolgende Teilprobleme noch gute Lösungen gefunden werden können (umgekehrt für schlechte Lösungen). Daher ist es notwendig, selbst wenn ein Gesamtproblem optimiert werden muss, für jedes Teilproblem nicht die zuvor gefundene, sondern eine vorbereitete Startkonstellation zu verwenden.

8.2 Funktionen

8.2.1 Standardfunktionen

Die Optimierung von Funktionen gehört mit zu den klassischen Aufgaben evolutionärer Algorithmen, so dass De Jong es für notwendig befand, explizit darauf hinzuweisen, dass sie mehr sind als nur das (De Jong: Genetic Algorithms are NOT function optimizers [DJ93]). Gerade von De Jong stammt jedoch – wie oben erwähnt – eine der am häufigsten verwendeten Testgruppen von Funktionen [DJ75]. Sie besteht aus den folgenden Funktionen:

F1.) *Parabel-Funktion*:

$$f_1(\mathbf{x}) = \sum_{i=1}^{3} x_i^2 \qquad (\forall\, i \in \{1,2,3\} : -5.12 < x_i < 5.12)$$

$$\min(f_1) = f_1(0,\ldots,0) = 0$$

F2.) *Rosenbrock's Sattel*:

$$f_2(\mathbf{x}) = 100 \cdot (x_1^2 - x_2)^2 + (1 - x_1)^2 \qquad (\forall\, i \in \{1,2\} : -2.048 < x_i < 2.048)$$

$$\min(f_2) = f_2(1,1) = 0$$

F3.) *Stufenfunktion*:

$$f_3(\mathbf{x}) = \sum_{i=1}^{5} integer(x_i) \qquad (\forall\, i \in \{1,\ldots,5\} : -5.12 < x_i < 5.12)$$

$$\min(f_3) = f_3(-5.12,\ldots,5.12) = -12$$

F4.) x_4 *mit Rauschen* ($\sigma = 1$):

$$f_4(\mathbf{x}) = \sum_{i=1}^{30} i \cdot x_i^4 + N(0,1) \qquad (\forall\, i \in \{1,\ldots,30\} : -1.28 < x_i < 1.28)$$

$$\min(f_4) = f_4(0,\ldots,0) = 0$$

Dabei steht der Ausdruck $N(0,1)$ für eine nach einer Standardnormalverteilung erzeugten (Pseudo-)Zufallszahl.

F5.) *Sheckel's Fuchsbauten*:

$$f_5(\mathbf{x}) = 0.002 + \sum_{j=1}^{25} \frac{1}{j + \sum_{i=1}^{2} (x_i - a_{ij})^6}$$

$$(\forall\, j \in \{1,\ldots,25\}\ \forall\, i \in \{1,2\} : -65.536 < x_i < 65.536)$$

$$\min(f_5) = f_5(-32,-32) \approx 1$$

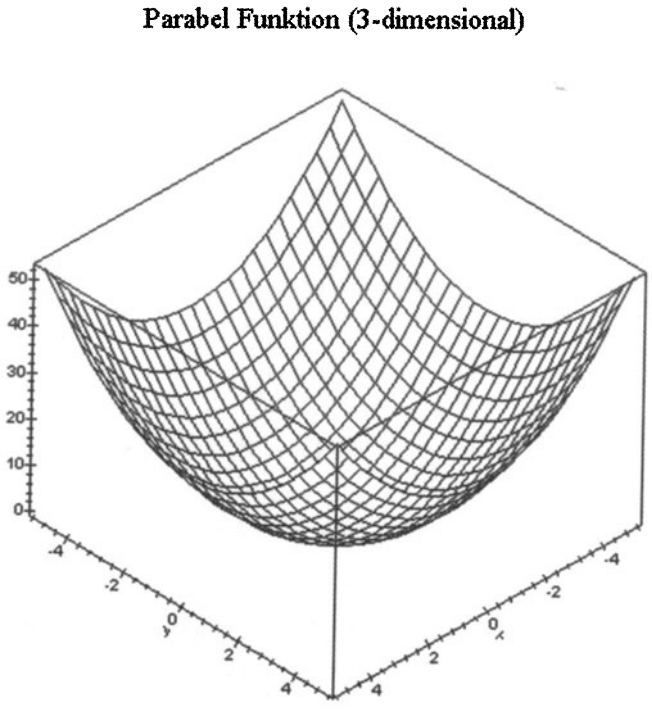


Abbildung 8.1 Parabel Funktion

Da einige dieser Funktionen eine hohe Dimension aufweisen, lassen sie sich nur schlecht grafisch darstellen. Nichtsdestotrotz vermitteln die Bilder 8.1 bis 8.3 einen Eindruck von der Aufgabenstellung.

Für die Stufenfunktion wurde von Bäck et.al. [Bäc96] eine anspruchsvollere Variante entwickelt, die die Funktion

$$f_6(\mathbf{x}) = \sum_{i=1}^{n} \lfloor x_i + 0.5 \rfloor^2$$

verwendet (siehe Bild 8.4). Darüber hinaus haben sich auch die so genannte „*Rastrigin*"- (siehe Bild 2.1 Seite 10) und die „*Ackley*"-Funktion (Bild 8.5) neben anderen in der Praxis bewährt (siehe auch [PDJ94]), da sie alle multimodal und daher insbesondere auch als Test-Funktionen für co-evolutionäre Algorithmen geeignet sind.

Die Stufenfunktion wurde entworfen, um Algorithmen auf eine einfache Funktion mit einer Vielzahl lokaler Minima (jedes Plateau) anwenden zu können. Jedes Plateau bildet dabei einen Bereich, in dem keinerlei Fortschritt zu erwarten ist. Insofern stellt die Suche nach der nächsten Verbesserung eine schwierige Aufgabe dar. Interessant

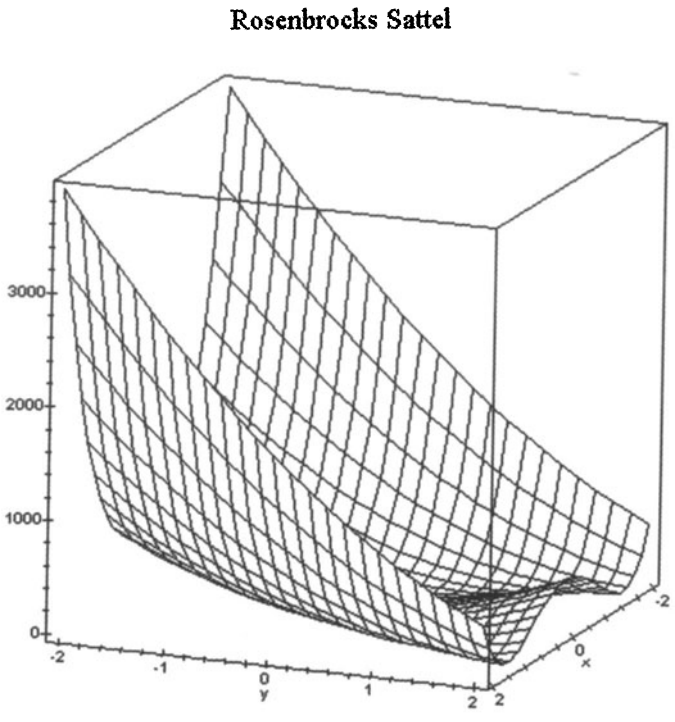


Abbildung 8.2 Rosenbrock's Sattel

ist die Funktion insbesondere dann, wenn beim darauf angewendeten Algorithmus die Mutationsrate adaptiv festgelegt wird, da bei einer größeren Schrittweite auch die Verbesserungswahrscheinlichkeit deutlich steigt.

Die Ackley-Funktion [Ack87] setzt sich ebenfalls aus einer Vielzahl lokaler Minima zusammen, wobei erschwerend hinzukommt, dass sie jeweils von lokalen Maxima umgeben sind. Das bedeutet, dass effektive Verschlechterungen zur Erreichung des Minimums in Kauf genommen werden müssen, was bei der Stufenfunktion nicht der Fall war.

Es werden auch immer wieder Testprobleme verwendet, die nicht auf einer Funktionsoptimierung beruhen. Zu diesen Problemen gehören die so genannten *Long Path Problems* von Horn, Goldberg und Deb [HGD94], sowie die *Royal Roads* von Mitchel et. al. [Mit96]. Da die Long Path Probleme hauptsächlich entwickelt wurden, um die Überlegenheit von genetischen Algorithmen gegenüber der Klasse der Hillclimber zu beweisen, wird auf dieses Problem nur kurz eingegangen. Die Grundlage der Long Path Probleme bildet eine Binärkodierung, und die Aufgabe besteht darin, eine reellwertige, auf binären Chromosomen der Länge l definierte Fitnessfunktion zu maximieren. Hierbei wird das Maximum über einen extrem engen und sehr langen Pfad erreicht,

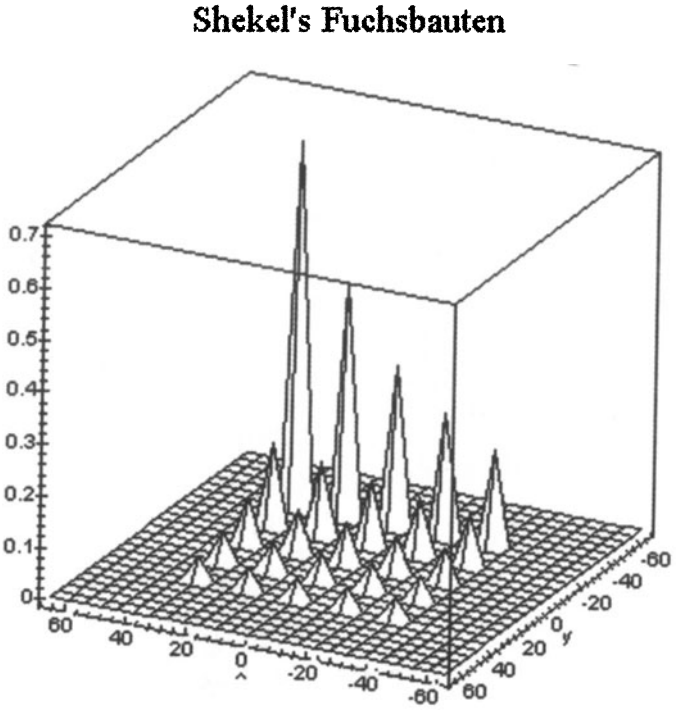


Abbildung 8.3 Shekel's Fuchsbauten

dessen Fitnesswerte über denen der Umgebung liegen. Als Vergleichskriterium zwischen den Algorithmen diente dann die Anzahl der notwendigen Schritte bis zum Optimum, also letztendlich die Fähigkeit, Abkürzungen zu suchen. Erwartungsgemäß erreichten genetische Algorithmen mit Mutation und Crossover ihr Ziel wesentlich schneller als die Hillclimber.

Die Royal Road Funktionen wurden entwickelt, um die Arbeitsweise der Building-Block-Hypothese (siehe Seite 55) an einem Beispiel zu verdeutlichen. Dem entsprechend ging es darum, aus binären Blöcken einer vorher festgelegten Länge b einen String nur bestehend aus Einsen zusammenzusetzen. Die Fitnessfunktionen wurden so konstruiert, dass es nur für die Zugehörigkeit zu vorher festgelegten Schemata Punkte gab. Diese Schemata wurden für zwei unterschiedliche Funktionen R_1 und R_2 festgelegt, wobei die im folgenden dargestellten Schemata von R_1 eine Teilmenge der Schemata von R_2 [FM93] bildeten:

$$s_1 = 11111111**********...**********$$

$$s_2 = ********11111111**********...**********$$

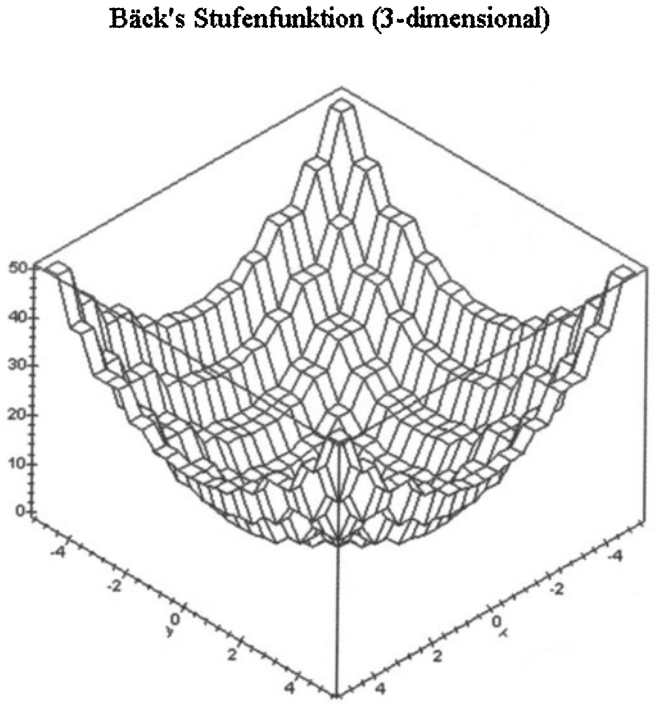


Abbildung 8.4 Stufenfunktion $f_6 = \sum_{i=1}^{n} \lfloor x_i + 0.5 \rfloor$, $x_i \in [0, 1]$

s_3 = **********11111111**********

s_4 = **********11111111**********

s_5 = **********11111111**********

s_6 = **********11111111**********

s_7 = **********11111111********

s_8 = **********11111111

s_{opt} = 11111111111111111111111111111111111111111111111111111111

Jedes der oben aufgeführten Schemata s_1 bis s_8 trägt zur Bewertung eines Chromosoms 8 Punkte bei, wenn es im Chromosom enthalten ist. Für s_{opt} gibt es 64 Punkte. Für die Funktion R_2 wurden zusätzlich noch die Schemata

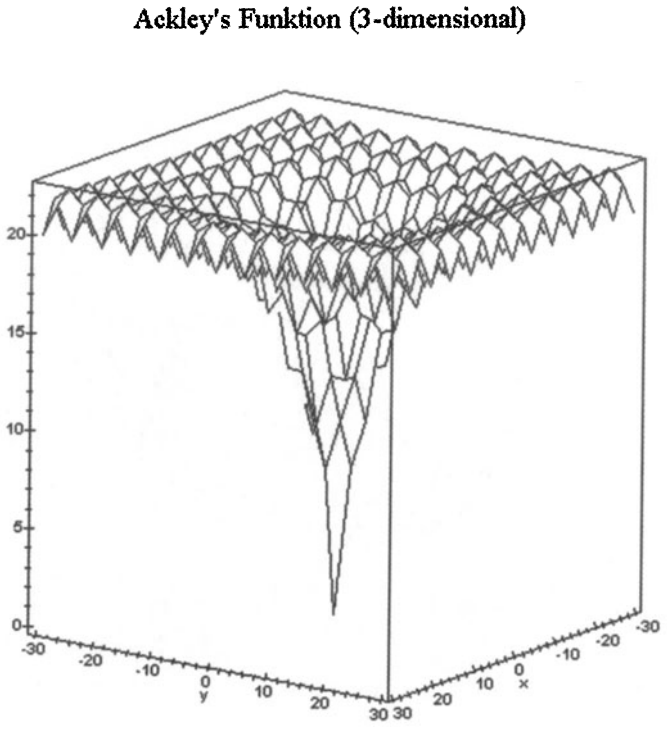


Abbildung 8.5 Ackley's-Funktion $f_7 = 20 + e - 20\exp(-0.2\sqrt{1/n\sum_{i=1}^{n} x_i^2}) - \exp(1/n\sqrt{\sum_{i=1}^{n} cos(2\pi x_i)}\ ,\ x_i \in [-30, 30]$

s_9 = 1111111111111111**********

s_{10} = **********1111111111111111**********

s_{11} = **********1111111111111111**********

s_{12} = **********111111111111111

s_{13} = 11111111111111111111111111111111**********

s_{14} = **********11111111111111111111111111111111

mit Punkten bewertet. Die Punktezahl für alle Chromosomen entspricht dabei der Summe der Ordnungen aller in ihnen vertretenen Schemata. Ein Chromosom, das das Schema S_9 enthält, und damit auch gleichzeitig s_1 und s_2, erhält daher 32 Punkte. Diese Schemata sollen nun die Building-Blocks zur Entwicklung von s_{opt} bilden, wobei sich die kleineren Schemata zu den größeren vereinigen sollten. Dem entsprechend erhoffte sich Mitchell eine Beschleunigung der Optimierung bei der Funktion R_2, wobei jedoch

das Gegenteil der Fall war. Diese Tatsache entspricht der Kritik an der Building-Block-Hypothese insofern, als dass sich bei näherer Betrachtung der Schemataentwicklungen herausstellte, dass Schemata mit höherer Bewertung sich so stark in der Population ausbreiteten, dass niedriger bewertete Bulding-Blocks komplett verdrängt wurden. Es kam also zur Behinderung der Entwicklung einzelner Schemata durch die ungünstigen Gene insgesamt besser bewerteter Schemata. Auch die übrigen in Abschnitt 5.1 auf Seite 55 beschriebenen Probleme wie „Ungleiche Startverteilung der Schemata", „Endliche Population" usw. haben zu diesem Ergebnis beigetragen.

8.2.2 „EA-Schwierige" Probleme

Bis vor einiger Zeit ging man davon aus, dass alle Probleme, die für kanonische genetische Algorithmen (binär, One-Point-Crossover...) schwierig zu lösen sind, unter Zuhilfenahme der Building-Block-Hypothese entwickelt werden können. Dabei wird so vorgegangen, dass die Bewertung für einige Schemata mit zum Optimum gegensätzlichen Genen so hoch sind, dass sie sehr häufig in der Population zu finden sind. Zusätzlich darf sich durch Kombination dieser lokalen Optima nicht das globale Optimum ergeben. Darüber hinaus werden die Bewertungen für Schemata, die zum globalen Optimum führen, niedrig gehalten. Das Ergebnis dieser Konstruktion ist eine Funktion, die die Population absichtlich *täuscht* und in eine falsche Richtung lockt („*Deception*"). Allerdings sind diese künstlich erzeugten Probleme nicht von großer Relevanz im Anwendungsbereich, zumal immer häufiger von der Binärkodierung abgewichen wird. Auch die ursprünglich unter Anderem von Witley [Whi91] aufgestellte Behauptung, dass alle GA-schwierigen Probleme „deceptive" seien, wurde von Grefenstette [Gre93] eingeschränkt. Auch durch die in die Kritik geratene Building-Block-Hypothese haben die Deceptive Functions an Wichtigkeit verloren.

Beispiel für eine Täuschungsfunktion der Ordnung 3(Whitley):
Der Begriff der „Ordnung 3" bezieht sich auf die Ordnung (siehe Definition 5.4) des betrachteten Schemas. Whitley definiert die Bewertungsfunktion für die gesamten Chromosomen folgendermaßen:

$$\begin{aligned} fit(000) &= 25 \qquad fit(001) = 26 \\ fit(010) &= 22 \qquad fit(100) = 14 \\ fit(110) &= 0 \qquad fit(011) = 0 \\ fit(101) &= 0 \qquad fit(111) = 30 \end{aligned}$$

Für die Schemata der Ordnung 2 wurden in Bezug auf die mittleren Erwartungswerte für die zu den Schemata passenden Chromosomen die folgenden Beziehungen festgelegt:

$$\overline{fit(0**)} > \overline{fit(1**)} \qquad \overline{fit(00*)} > \overline{fit(11*)}\ ,\ \overline{fit(01*)} > \overline{f(10*)}$$
$$\overline{fit(*0*)} > \overline{fit(*1*)} \qquad \overline{fit(0*0)} > \overline{fit(1*1)}\ ,\ \overline{fit(0*1)} > \overline{fit(1*0)}$$
$$\overline{fit(**0)} > \overline{fit(**1)} \qquad \overline{fit(*00)} > \overline{f(*11)}\ ,\ \overline{fit(*01)} > \overline{fit(*10)}$$

Beide Festlegungen zusammen führen für einen auf das Problem angesetzten Algorithmus dazu, alle Schemavarianten, die Nullen beinhalten, zu bevorzugen. Dadurch, dass alle Chromosomen, die zwei Einsen enthalten, eine Bewertung von Null zugewiesen bekamen, besteht nicht die geringste Chance, sie zur Konstruktion des optimalen Chromosoms zu verwenden. Das wäre aber notwendig gewesen. Ebenfalls dadurch wird verhindert, dass das Optimum durch Anwendung des Mutationsoperators entstehen kann.

Da Grefenstette widerlegt hat, dass es sich bei Täuschungsfunktionen um die einzige Form der für GA schwierigen Funktionen handelt, wurde die Frage nach neuen Kriterien zur Bestimmung des Schwierigkeitsgrades aufgeworfen. Innerhalb der letzten Jahre wurden verschiedene Ideen aufgebracht und im Allgemeinen auch zügig durch die Konstruktion eines Gegenbeispiels widerlegt. Beispiele für diese Kriterien sind

- Zerklüftung der Fitnesslandschaft (Beziehung zwischen Eltern und Nachkommen),
- Große Entfernungen zwischen lokalen und globalen Optima,
- Multimodalität,
- Epistasie (Der Wert eines Gens beeinflusst oder unterdrückt den Wert eines anderen Gens),
- Isolierung (Nadel im Heuhaufen).

Zu den erfolgreicheren Modellen gehört die so genannte *Fitness-Distanz-Korrelierung* (FDC), bei der die Hammingdistanz zwischen jedem Chromosom und dem nächstgelegenen Optimum mit der Fitness in Beziehung gesetzt wird. Es wird ein Korrelationskoeffizient r berechnet, der Werte zwischen 1 und -1 annimmt, wobei sich -1 für einfache Probleme und 1 für schwierige Probleme ergeben soll. In Fällen, in denen kein klares Ergebnis vorliegt, wird der so genannte „Scatter Plot" verwendet, bei dem Fitness und Hammingdistanz in einem zweidimensionalen Koordinatensystem aufgetragen werden und so einen genaueren Eindruck der Fitnesslandschaft ergibt. Die Intention dieses Ansatzes war die Annahme, dass es für einen Algorithmus umso schwieriger ist, das Optimum zu finden, je mehr lokale Optima sich in weit entfernten Bereichen befinden. Entwickelt wurde er von Jones und Forrest [JF95] und ergab für die meisten Standardprobleme (De Jong Funktionen, Whitleys Deceptive Functions usw.) Korrelationskoeffizienten, die zu den mit den Funktionen gemachten Erfahrungswerten passten. Altenberg [Alt97] konstruierte 1997 ein Gegenbeispiel zum FDC mit Hammingdistanz. Er favorisierte stattdessen die Crossoverdistanz (Anzahl notwendiger Crossover-Schritte zum Erreichen des nächsten Optimums) im Verhältniss zur Fitness, bei deren Verwendung sich eine korrekte Einschätzung des Schwierigkeitsgrades ergab.

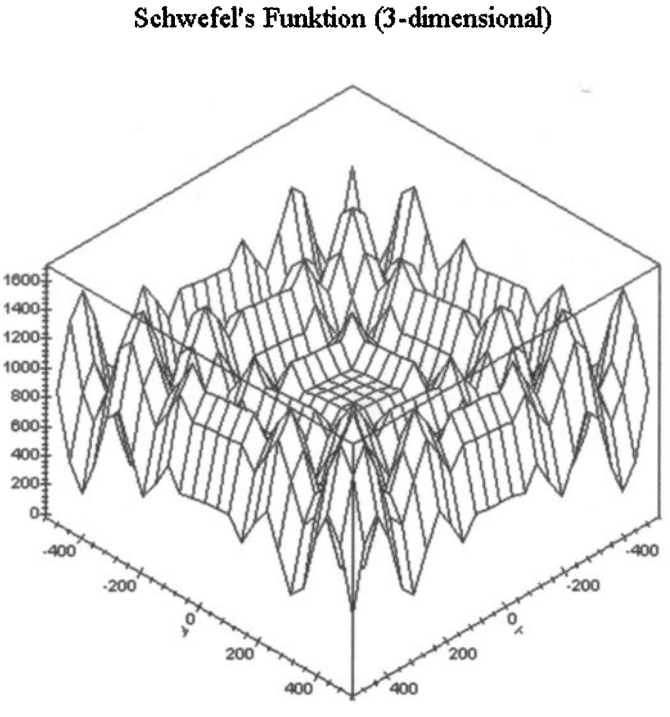


Abbildung 8.6 Schwefel-Funktion $f_8 = 418.9829 \cdot n + \sum_{i=1}^{n} x_i \sin(\sqrt{|x_i|})\ ,\ x_i \in [-500, 500]$

Ein Beispiel für eine Funktion mit einer sehr zerklüfteten Fitnesslandschaft und weit auseinanderliegenden lokalen und globalen Optima ist die „*Schwefel*"-Funktion, die in Bild 8.6 dargestellt ist. Bei dieser Funktion liegt das Minimum bei

$$\min(f_8) \;=\; f_8(420.9687, \ldots, 420.9687) \;=\; 0,$$

während das zweitbeste Minimum weit weg liegt. Weitere GA-schwierige Funktionen finden sich bei Ackley [Ack87] und Liepins und Vose [LV91]. Ob eine Funktion für einen evolutionären Algorithmus leicht oder schwer ist, hängt nicht zuletzt von der Form der Kodierung ab. Lokale und globale Optima können beispielsweise bei Verwendung einer Binärkodierung weit (im Sinne der Hammingdistanz) auseinanderliegen, während dieselben Werte bei Nutzung einer Gray-Kodierung nahe beieinander liegen (siehe Abschnitt 6.2). In Tabelle 6.1 auf Seite 66 wurden Binär- und Gray-Kodierungen gegenübergestellt, so dass sofort zu sehen ist, dass die Kodierungen für 0 und 7 bei einer Binärkodierung eine Hammingdistanz von 3, bei einer Gray-Kodierung aber nur eine Distanz von 1 haben.

8.3 Kombinatorische Optimierungsprobleme

In die Kategorie der kombinatorischen Optimierungsprobleme fallen auch zwei Probleme, die bereits in vorhergehenden Kapiteln vorgestellt wurden. Dabei handelt es sich um das Travelling Salesman Problem (TSP) (siehe Beispiel 2.1) und das Rucksackproblem (siehe Beispiel 3.1). Insbesondere das TSP wird sehr häufig zur Bewertung einzelner oder zum Vergleich mehrerer Algorithmen herangezogen, da Vergleichsdaten für die meisten darauf ansetzbaren Algorithmen vorliegen.

Bin-Packing-Problem
Ein Problem, das dem Rucksackproblem relativ ähnlich ist, ist das so genannte „*Bin-Packing-Problem*". Dabei geht es darum, eine vorgegebene Menge von Objekten

$$\{o_1, \ldots, o_m\}$$

mit festgelegtem Größen $V = \{v_1, \ldots, v_m\}$ in möglichst wenigen Behältern b gleicher Größe v_b unterzubringen. Gesucht ist also

$$\begin{aligned} k \quad = \quad & \min_{n \in \mathbb{N}} \text{card}\,(\{b_1, \ldots, b_n\}) \qquad \text{mit} \\ & \forall\, i\{1, \ldots, m\} \exists j \in \{1, \ldots, n\} \,:\, v_i \in b_j \qquad \text{und} \\ & \forall j \in \{1, \ldots, n\} \,:\, \sum_{\substack{i=1 \\ o_i \in b_j}}^{n} v_i \leq v_b \end{aligned}$$

Diese Art von Problem tritt sowohl im Transportbereich (LKW-Beladung), als auch sehr häufig im Bereich der Verschnittoptimierung auf (z.B. Zuschnitt von Kleidungsstücken oder Stanzen von Formen aus einer Metallplatte).

Job-Scheduling-Problem
Das „*Job-Shop-Scheduling-Problem*" ist ein Optimierungsproblem, das sehr häufig im Bereich der Maschinenbelegung auftritt. Die Aufgabe besteht darin, n Jobs auf m Maschinen zu verteilen, wobei sich jeder Job aus m Teilarbeiten (Tasks) zusammensetzt, die jeweils auf einer der m Maschinen durchgeführt werden müssen. Die interne Reihenfolge der Maschinen und die Belegungszeiten variieren dabei von Job zu Job. Die Reihenfolge der Abarbeitung der Tasks innerhalb der n Jobs ist fest vorgegeben. Bezeichnet man mit B die Menge aller möglichen Belegungspläne und mit t_k den Endzeitpunkt jedes Belegunsplanes $b_k \in B$, dann wird gesucht nach

$$t_{min} = \min_{b_k \in B} \{t_k\}.$$

Quadratic-Assignment-Problem
Ebenfalls mit zu den wichtigen Standardproblemen gehören die „*Quadratic Assignment Problems*", bei denen es darum geht, n Objekte möglichst günstig auf n Orte zu verteilen, wobei sowohl die „Transportkosten" je Streckeneinheit, als auch die Transportstrecke zu berücksichtigen sind. Die Aufgabe besteht darin, die Summe der Gesamtkosten zu minimieren. Ein Beispiel dafür ist die Zuweisung von n Fabriken zu n Standorten, wobei eine Matrix $C = (c_{lk})$ die Kosten pro Streckeneinheit für den Transport der benötigten Waren von Fabrik k zu Fabrik l und $D = (d_{ij})$ die Distanzen zwischen den Standorten i und j enthält $(i, j, k, l \in \{1, \ldots, n\})$. Gesucht ist also eine Permutation π aus der Menge Π aller möglichen Permutationen der Zahlen 1 bis n, bei deren Anwendung auf die Fabriknummern die gesamten Transportkosten minimal werden:

$$\min \stackrel{!}{=} \sum_{i=1}^{n} \sum_{j=1}^{n} d_{ij} c_{\pi(i)\pi(j)}$$

8.4 Strategieentwicklung

Der Bereich der Strategieentwicklung wird nur selten als Anwendungsgebiet für evolutionäre Algorithmen verwendet. Am häufigsten wird dann genetische Programmierung verwendet, da sich dort am ehesten Informationen aus dem Umfeld verarbeiten und entsprechende Rückschlüsse ziehen lassen (z.B. über `IF THEN ELSE` Abfragen).

Prisoners Dilemma
Eines der bekanntesten Beispiele für eine Strategieentwicklung unter Verwendung eines kanonischen genetischen Algorithmus ist das *Gefangenendilemma* („*Prisoners Dilemma*"). Dabei sitzen zwei Gefangene in zwei verschiedenen Zellen und haben keinerlei Kontakt zueinander. Sie sind wegen eines Verbrechens angeklagt, die Beweislage selbst lässt jedoch keine Verurteilung zu. Es ist also notwendig, dass zumindest einer der beiden Gefangenen als Zeuge aussagt, den anderen Gefangenen also betrügt, wofür er eine Belohnung angeboten bekommt. Jeder der beiden kann wählen zwischen:

(D) Verrat des anderen (Betrug des anderen)

(C) kein Verrat des anderen (Kooperation)

Entscheiden sich beide für die Kooperation, so wird keiner der beiden bestraft, sondern sogar geringfügig belohnt. Verraten sich beide gegenseitig, so werden beide hart bestraft. Verrät nur einer den anderen und der andere schweigt weiter, so wird der Verräter belohnt und der andere bestraft, jedoch nicht so hart, als würden sich beide gegenseitig beschuldigen. Beim Spiel Prisoners Dilemma wird nun so vorgegangen, dass entwickelte Strategien gegeneinander antreten und für eine vorgegebene Anzahl von Zweikämpfen je nach Konstellation der Aktionen der beiden Spieler ({D,D}, {D,C}...) Punkte erhalten. Eine solche Punkteliste wurde von Axelrod [Axe87] aufgestellt (G_i bezeichnet den i-ten Gefangen, $i \in \{1, 2\}$):

Aktion		Bonus	
G_1	G_2	G_1	G_2
D	D	1	1
D	C	5	0
C	D	0	5
C	C	3	3

Eine Strategie lässt sich bei diesem Problem dadurch erzeugen, dass das Verhalten des Gegners in den letzten drei Zweikämpfen berücksichtigt wird. Dabei gibt es $4 \times 4 \times 4 = 64$ mögliche Spielausgänge für die letzten 3 Spiele. Eine solche einfache Strategie kann daher in 64 Bit kodiert werden. Jedes Bit schreibt eine der 64 möglichen Vorgeschichten die Handlung (C) oder (D) vor. Es werden zwei Bit pro Spiel benötigt. Insgesamt also sechs; damit ist die Kodierung einer einfachen Strategie mit 70 Bit möglich. Drei hypothetische Spiele dienen am Anfang als künstliche Historie. Axelrod führte zwei Arten von Tests durch, wobei er einmal acht fest vorgegebene Spielstrategien gegen die vom genetischen Algorithmus erzeugten Strategien antreten ließ, beim anderen Mal alle erzeugten Strategien untereinander spielen ließ.

Erwartungsgemäß entwickelten sich im ersten Fall Strategien, die sich die Schwächen ihrer Gegner zunutze machten, während im zweiten Fall Varianten der äußerst erfolgreichen „TIT FOR TAT"-Strategie (Reagiere im nächsten Zug genauso, wie der Gegner im letzten Zug) entstanden.

REVERSI

Ein weiteres Strategie-Spiel, auf das sich die evolutionäre Algorithmen anwenden lassen, ist eine vereinfachte Variante des Spiels Reversi. Es wird von zwei Spielern auf einem 8x8-Feld mit Spielchips gespielt, die auf der einen Seite schwarz und auf der anderen Seite weiß sind. Die Startkonstellation sieht dabei aus wie in Bild 8.7.

Danach legen die Spieler abwechselnd einen Chip mit ihrer Farbe so auf das Spielfeld, dass sich zumindest ein gegnerischer Stein zwischen dem neu gesetztem und wenigstens einem bereits vorhandenem eigenem Chip befindet. All diese „eingeklemmten" Spielmarken werden umgedreht auf die eigene Farbe. Bild 8.7 zeigt die möglichen Züge für Schwarz durch kleine Kreise an. Das Einklemmen ist sowohl horizontal, als auch vertikal und diagonal erlaubt. Hat ein Spieler keine Möglichkeit mehr, Spielmarken zu legen, so muss er aussetzen. Das Spiel ist zuende, wenn alle Felder besetzt sind. Gewonnen hat derjenige, der am Ende die meisten Spielmarken mit seiner Farbe auf dem Feld hat.

Das Interessante an diesem Spiel ist, dass man mit Strategien, die einfach von Anfang an möglichst viele gegnerische Steine umzudrehen versuchen, meist nicht gewinnt, da man nach relativ kurzer Zeit nur noch wenige Züge machen kann. So kann man vom Gegner dazu gezwungen werden, Marken an Positionen zu legen, die nur dem Gegner Vorteile bieten. Solche Positionen sind im Bild durch Sterne bzw. das Plus-Zeichen gekennzeichnet.

Zu den wichtigsten Feldern gehören die Ecken, da es sich bei ihnen um so genannte stabile Steine handelt. Sie können mit Sicherheit nicht mehr umgedreht werden. Das erklärt auch die Wichtigkeit der gekennzeichneten Felder, da über diese Felder die Ecken

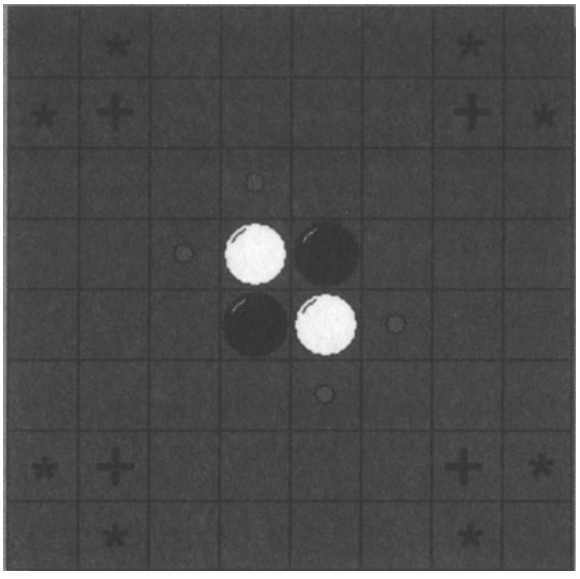


Abbildung 8.7 Reversibrett mit Startkonstellation und kritischen Feldern

erreichen werden können, sofern der Gegner gezwungen ist, sie zu besetzen. Randsteine sind schon relativ stabil, sollten jedoch keine ungerade Anzahl von Lücken zwischen den eigenen Steinen aufweisen, da der Gegner sonst die Reihe leicht übernehmen kann.

All diese Überlegungen lassen sich nicht so ohne weiteres in einem evolutionären Algorithmus nachbilden. Eine Möglichkeit wäre die Verwendung eines Chromosoms, das für jedes Feld des Spielbretts einen Wert mitführt, der ein Maß für die Wichtigkeit ist, das Feld selbst zu besetzen. Negative Werte stehen dabei für eine notwendige Vermeidung des Feldes. Zusätzlich können verschiedene Strategien durch jeweils ein Feld, das den Berücksichtigungsgrad enthält, repräsentiert werden. Beispiele für Strategien sind:

- Maximiere die Zahl Deiner Spielmarken.
- Maximiere die Anzahl Deiner Möglichkeiten zum Setzen von Spielmarken.
- Minimiere die Anzahl der Möglichkeiten zum Setzen von Spielmarken für den Gegner.
- Schaffe stabile Steine in der Mitte des Spielfeldes (umgeben von Steinen der anderen Farbe).
- Besetze Ecken.
- Besetze Randpositionen.

Um die beste Positionierung der nächsten Spielmarke zu ermitteln, wird für jedes mögliche Feld ausgerechnet, welche Bewertung alle danach durch den Gegner belegbaren Felder erhalten (Summe). Je niedriger die Zahl, desto besser. Außerdem wird ein Wert dafür ausgerechnet, inwieweit die Platzierung den Strategien entspricht. Eine geeignete Kombination aus beiden Berechnungsverfahren legt die Bewertung für jede mögliche Platzierung fest. Zur Bewertung der durch ein Chromosom festgelegten Strategie werden alle in einer Population vorhandenen Strategien gegeneinander getestet.

Die Türme von Hanoi

Das Problem der *Türme von Hanoi* setzt sich aus einem Spielfeld mit drei nebeneinanderliegenden Stangen und einer Anzahl unterschiedlich großer gelochter Scheiben zusammen (siehe Abbildung 8.8). Die Scheiben befinden sich der Größe nach sortiert (die größte Scheibe liegt unten) auf der ersten Stange. Die Aufgabe besteht darin, sie unter Zuhilfenahme der zweiten Stange in der ursprünglichen Reihenfolge auf die dritte Stange zu transportieren. Hierbei darf niemals eine kleinere Scheibe unter einer größeren Scheibe liegen. Es gibt in jedem Schritt maximal sechs Züge ($1 \rightarrow 2, 1 \rightarrow 3, 2 \rightarrow 1, 2 \rightarrow 3, 3 \rightarrow 1, 3 \rightarrow 2$).

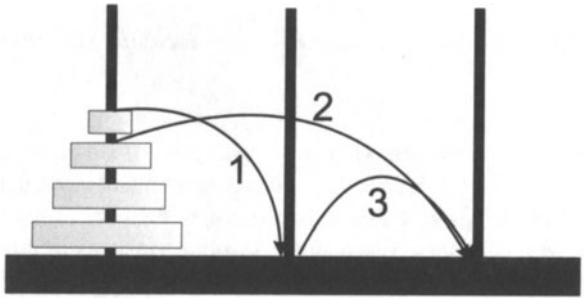


Abbildung 8.8 Türme von Hanoi mit Zugfolge zum Verschieben der obersten beiden Scheiben auf Stange 3.

Die Chromosomen bestehen aus einer variablen Anzahl von Genen, wobei jedes Gen einen der möglichen Züge enthält. Bewertet wird die Anzahl der korrekt sortierten Scheiben auf der dritten Stange, wobei die Maximalzahl der auszuführenden Züge begrenzt ist. Falsche Züge werden gezählt, aber nicht ausgeführt. Angeline und Pollack [AP94] verwendeten dieses Problem zum Test einer neuen Repräsentationsform, bei der mit einer festen Anzahl möglicher Züge gestartet wird, aus denen dann Module entwickelt werden, die ihrerseits ganze Abfolgen von Zügen enthalten. Diese Module wurden der Menge der möglichen Züge hinzugefügt, also wie ein einzelner Zug behandelt. Es wurde versucht, Module zu entwickeln, die sinnvolle Aktionen enthalten, die immer wieder durchgeführt werden müssen (z. B. Verschieben von zwei Scheiben von einer Stange auf eine andere). Die verwendete Algorithmenart entwickelte also selbst eine Art Sprache zum Aufbau nützlicher Zugsequenzen.

9 Fuzzy-Systeme

Dieses Kapitel beinhaltet eine kurze Einführung in die Thematik der Fuzzy-Systeme. Wir beschränken uns auf die wesentlichen Definitionen und Konzepte, die für das Verständnis des Kapitels 10 über Kombinationen von evolutionären Algorithmen mit Fuzzy-Systemen erforderlich sind. [KGK94] gibt eine ausführliche Einführung in die Grundlagen der Fuzzy-Systeme, [MKKN02] behandelt vor allem die regelungstechnischen Aspekte der Fuzzy-Systeme, deren Konzepte auch häufig im Zusammenhang mit evolutionären Algorithmen verwendet werden.

9.1 Grundprinzipien

Der Begriff der Fuzzy-Menge wurde 1965 von Zadeh geprägt [Zad65]. Die wesentliche Motivation bestand damals darin, vor allem im Bereich der Regelungstechnik ein neues Modellierungsparadigma einzuführen.

Die Regelungstechnik befasst sich mit der Beeinflussung von Systemen, um wichtigen Ausgangsgrößen einen bestimmten zeitlichen Verlauf aufzuprägen. Wir haben bereits auf Seite 6 mit dem Problem der Leerlaufdrehzahlregelung ein Beispiel für eine regelungstechnische Aufgabe kennen gelernt. Die Ausgangsgröße ist in diesem Fall die Motordrehzahl (im Leerlauf). Bei einer Klimaanlage wäre die Ausgangsgröße die Temperatur und eventuell auch die Luftfeuchtigkeit, wenn es sich um eine komplexere Anlage handelt, die in einem Museum mit wertvollen antiken Kunstwerken ein Klima garantieren soll, in dem die Kunstwerke möglichst keinen Schaden nehmen. Auch ein Kraftwerk mit der abgegebenen elektrischen Leistung als Ausgangsgröße kann als regelungstechnisches System betrachtet werden. Die zu regelnden Systeme müssen nicht notwendigerweise technischer Natur sein. Ein Bioreaktor, in dem Bakterien ein bestimmtes Antibiotikum produzieren, stellt genauso ein regelungstechnisches Problem dar wie das System Volkswirtschaft, bei dem als Ausgangsgröße beispielsweise die Inflationsrate angesehen werden könnte. Ein weiteres Beispiel für ein regelungstechnisches System ist ein evolutionärer Algorithmus, dessen Parameter – z.B. Mutations- und Crossover-Rate oder die Populationsgröße adaptiv angepasst werden sollen, um für die Verbesserungsrate des besten Chromosoms als Ausgangsgröße einen möglichst großen Wert zu erzielen.

Der klassische Ansatz zur Lösung eines regelungstechnischen Problems besteht darin, zunächst ein mathematisches Modell des zu regelnden Systems – der so genannten Strecke – meist in Form von Differential- oder Differenzengleichungen aufzustellen. Der eigentliche Regler wird dann anhand dieses mathematischen Modells entworfen. Diese Vorgehensweise hat den Vorteil, dass man, sofern das mathematische Modell nicht zu komplex ist, den Regler bezüglich bestimmter Kriterien optimal auslegen kann und wichtige Eigenschaften wie Stabilität nachweisen kann. Die Voraussetzung dafür ist jedoch, dass ein genügend genaues mathematisches Modell der Strecke aufgestellt werden

kann. Diese Voraussetzung ist zum einen nicht immer gegeben, zum anderen können Menschen viele regelungstechnische Aufgaben hervorragend bewältigen, ohne auch nur eine einzige mathematische Gleichung aufzustellen oder zu lösen. Die menschliche Regelungsstrategie besteht eher aus relativ einfachen Regeln, etwa in der Form

> Wenn das Wasser im Kochtopf sehr stark kocht/sprudelt,
> dann reduziere die Temperatur der Herdplatte ein wenig.

Charakteristisch an derartigen Regeln ist, dass darin keine quantitativen, sondern qualitative Angaben wie „Reduziere die Temperatur *ein wenig*" auftreten. Im Gegensatz zu technischen Systemen und Computern verwenden Menschen eher diese Art von Informationen und können damit hervorragend umgehen. Niemand würde eine Aussage wie „Drehe den Temperaturregler für die Herdplatte um 17° entgegen den Uhrzeigersinn" machen oder sie als hilfreich empfinden. Menschen können mit qualitativen Aussagen flexibel und fehlertolerant umgehen. Dies war die Motivation zur Einführung des Begriffs der Fuzzy-Menge, mit der qualitative oder impräzise Angaben modellierbar sein sollen. Zadeh prägte für dieses Konzept später den Begriff „Computing with words" [Zad96].

Als die Popularität von Fuzzy-Systemen Anfang der neunziger Jahre stark gestiegen war und viele erfolgreiche Anwendungen vorzuweisen waren, stellte man jedoch immer mehr fest, dass die Spezifikation von qualitativen Angaben und ihre intuitive Umsetzung in Fuzzy-Mengen für viele Aufgaben nicht ausreichte, sondern ein nachträglicher, oft sehr langwieriger und mühsamer Fine-Tuning-Prozess durchgeführt werden musste, um befriedigende Ergebnisse zu erhalten. Aus diesem Grund wurden Ansätze entwickelt, die die Automatisierung dieses Fine-Tuning- oder Optimierungsprozesses unterstützen. Neben neuronalen Netzen oder Neuro-Fuzzy-Systemen, die ausführlich in [BKKN03] behandelt werden, eignen sich für diesen Zweck vor allem auch evolutionäre Algorithmen.

Nachdem entsprechende Optimierungstechniken für Fuzzy-Systeme entwickelt wurden, ist man inzwischen dazu übergegangen, sie nicht nur zum Fine-Tuning zu verwenden, sondern sie auch zum Erlernen von Fuzzy-Regeln aus Daten einzusetzen. Dabei müssen die Daten nicht unbedingt einem regelungstechnischen Prozess entstammen, bei dem aus Eingangs- oder Messgrößen Stellwerte berechnet werden, mit denen die Ausgangsgröße beeinflusst wird. Fuzzy-Systeme eignen sich grundsätzlich zur Beschreibung eines Ein-/Ausgabeverhaltens in Form von (unscharfen) Regeln, beispielsweise auch für Regressionsprobleme, die schon mehrfach in diesem Buch angesprochen wurden.

9.2 Fuzzy-Mengen

Zur Modellierung qualitativer oder unscharfer Angaben werden bei Fuzzy-Systemen Fuzzy-Mengen verwendet. Anstatt exakter Quantifizierungen eines präzisen Wertes wie

> „Er ist 205cm groß."

oder eines Intervalls

> „Seine Größe liegt zwischen 200cm und 210cm ."

sollen in Fuzzy-Systemen unscharfe Angaben wie

„Er ist sehr groß."

modelliert werden. Während die ersten beiden Aussagen durch eine exakte Menge von Werten, nämlich $\{205\}$ bzw. $[200, 210]$ eindeutig definierbar sind, ist es nicht sinnvoll, die dritte ebenfalls durch eine exakte Menge oder ein exaktes Intervall zu beschreiben. Kaum jemand lässt sich bei qualitativen oder unscharfen Angaben auf eindeutige Grenzen festlegen, also z.B. ab wann eine Person als sehr groß bezeichnet wird. Es gibt einen Bereich, bei dem man mit Sicherheit zustimmt, dass die entsprechenden Körpergrößen als sehr groß zu bezeichnen sind. Genauso gibt es einen Bereich von Körpergrößen, die man mit Sicherheit nicht als sehr groß bezeichnen würde. Zwischen diesen beiden Bereichen liegen Körpergrößen, die man mehr oder weniger als sehr groß bezeichnen würde. Deswegen erweitert man bei Fuzzy-Systemen das Konzept der Menge, indem man nicht nur eine binäre Zugehörigkeiten – ein Element gehört entweder zu einer Menge oder es gehört nicht dazu – sondern graduelle Zugehörigkeiten zu einer Menge zulässt.

Definition 9.1 (Fuzzy-Menge) *Eine* Fuzzy-Menge μ *über einer Grundmenge* X *ist eine Abbildung von* X *in das Einheitsintervall, d.h.*

$$\mu : X \to [0, 1].$$

$\mu(x)$ *wird als* Zugehörigkeitsgrad *des Elementes* $x \in X$ *zur Fuzzy-Menge* μ *bezeichnet.*

In der Literatur wird teilweise zwischen der Fuzzy-Menge und der Zugehörigkeitsfunktion μ unterschieden. Als Fuzzy-Menge wird ein abstraktes Konzept wie „Groß" bezeichnet. Die Zugehörigkeitsfunktion $\mu_{\text{Groß}}$ wird dann als konkrete Modellierung der Fuzzy-Menge „Groß" verstanden. Da in diesem Fall die Fuzzy-Mengen letztendlich nur als Namen für die Zugehörigkeitsfunktionen fungieren, verzichten wir auf diese Unterscheidung.

Man kann Fuzzy-Mengen als verallgemeinerte charakteristische Funktionen von Mengen auffassen. Die *charakteristische Funktion* oder *Indikatorfunktion* einer Menge $M \subseteq X$ über einer Grundmenge X ordnet jedem Element von M den Wert 1 und allen anderen Elementen den Wert 0 zu:

$$I_M : X \to \{0, 1\}, \quad x \mapsto \begin{cases} 1 & \text{falls } x \in M \\ 0 & \text{falls } x \notin M \end{cases}$$

Abbildung 9.1 zeigt links die charakteristische Funktion der Menge $M = [-2, 2] \subset \mathbb{R}$. Bei Fuzzy-Mengen wird der Wertebereich der charakteristischen Funktion von $\{0, 1\}$ auf das Einheitsintervall ausgedehnt. In Abbildung 9.1 ist rechts eine Fuzzy-Menge dargestellt, die als Modellierung der qualitativen Angabe „ungefähr Null" dienen könnte.

Die Interpretation einer Fuzzy-Menge im Sinne einer verallgemeinerten charakteristischen Funktion ist zwar intuitiv, aber lässt oft zu viel Freiraum. Für das Verständnis der Anwendungen in diesem Buch reicht diese Sichtweise jedoch völlig aus. Eine Diskussion verschiedener Interpretationen des Konzeptes der Zugehörigkeitsgrade findet sich in [MKKN02].

Typischerweise treten in Anwendungen nicht beliebige Zugehörigkeitsfunktionen auf, sondern primär *Dreiecksfunktionen* der Form

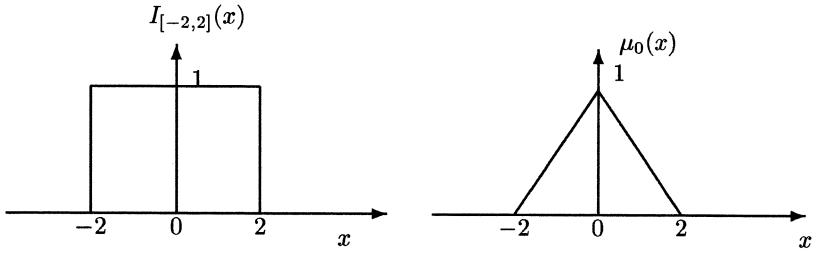


Abbildung 9.1 Die charakteristische Funktion der Menge $[-2, 2]$ und eine Fuzzy-Menge für den unscharfen Wert „ungefähr Null"

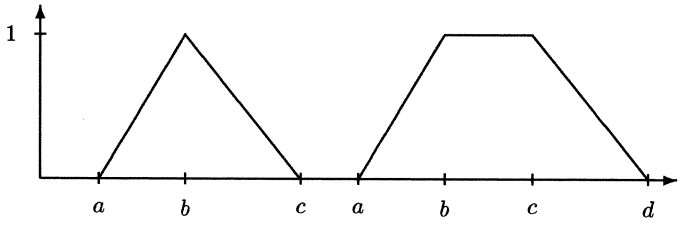


Abbildung 9.2 Eine Dreiecks- und eine Trapezfunktion

$$\mu(x) = \begin{cases} \frac{x-a}{b-a} & \text{falls } a \leq x \leq b \\ \frac{c-x}{c-b} & \text{falls } b \leq x \leq c \\ 0 & \text{sonst} \end{cases}$$

mit $a \leq b \leq c$ oder *Trapezfunktionen*

$$\mu(x) = \begin{cases} \frac{x-a}{b-a} & \text{falls } a \leq x \leq b \\ 1 & \text{falls } b \leq x \leq c \\ \frac{d-x}{d-c} & \text{falls } c \leq x \leq d \\ 0 & \text{sonst} \end{cases}$$

mit $a \leq b \leq c \leq d$ sowie auch *Gaußfunkfionen*

$$\mu(x) = \exp(-b(x-a)^2)$$

mit $a > 0$.

Abbildung 9.2 veranschaulicht die Dreiecks- und die Trapezfunktion. Bei der Gaußfunktion bestimmt der Parameter a, wo das Maximum von eins angenommen wird. Die Breite unter der Glockenkurve wird vom Parameter b gesteuert. Je größer b ist, desto schmaler wird die Kurve. In Abbildung 9.3 ist rechts eine Gaußfunktion mit $a = 2$ und $b = 0.5$ zu sehen.

Der Vorteil der Verwendung dieser Fuzzy-Mengen besteht in ihrer einfachen Parametrisierung, so dass beispielsweise bei einer Optimierung der Fuzzy-Mengen mittels eines evolutionären Algorithmus nur die entsprechenden Parameter angepasst werden müssen.

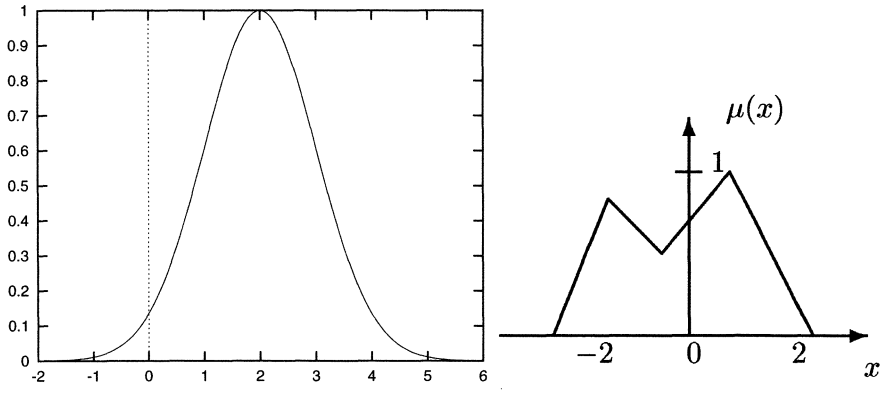


Abbildung 9.3 Eine Gaußfunktion mit $a = 2$ und $b = 0.5$ und eine stückweise lineare Fuzzy-Menge

In seltenen Fällen werden auch allgemeinere Parametrisierungen von Fuzzy-Mengen zugelassen. Eine Möglichkeit besteht darin, eine Fuzzy-Menge über eine Liste von Punktepaaren $(x_i, \mu(x_i))$ zu parametrisieren und zwischen den Werten linear zu interpolieren. Auf diese Weise erhält man als Verallgemeinerung der Dreiecks- und Trapezfunktionen beliebige stückweise lineare Fuzzy-Mengen. Abbildung 9.3 zeigt im rechten Teil eine solche Fuzzy-Menge, die sich mit fünf Punktepaaren definieren lässt. Diese Art von Fuzzy-Mengen bringen allerdings eine Reihe von Problemen mit sich. Sofern man nicht zusätzliche Bedingungen an die definierenden Punktepaare stellt, kann sich eine multimodale Zugehörigkeitsfunktion wie in Abbildung 9.3 rechts ergeben, die sich schwer interpretieren lässt, zumindest nicht mehr als unscharfer Wert oder Bereich. Hinzu kommt, dass bei einer variablen Länge der Punktepaarliste die Kodierung für evolutionäre Algorithmen komplizierter wird, da man mit Chromosomen variabler Länge arbeiten müsste.

9.3 Fuzzy-Regler

Eines der größten Anwendungsgebiete der Fuzzy-Systeme sind die Fuzzy-Regler. Übliche Fuzzy-Regler sind Kennfeldregler, d.h. sie beschreiben eine Funktion, die Werte von Eingangsgrößen auf die Werte einer Ausgangsgröße abbildet, die zur Steuerung oder Regelung des Systems dient. Da es sich bei den Fuzzy-Regler somit im Wesentlichen um die Definition von Funktionen handelt, werden die Konzepte der Fuzzy-Regelung genauso in anderen Gebieten eingesetzt, z.B. für die wissensbasierte Regression.

Um zu erläutern, wie Fuzzy-Regler arbeiten, betrachten wir ein einfaches Beispiel: das *Stabbalanceproblem*, das auch als *inverses Pendel* bezeichnet wird. Es handelt sich dabei um die Aufgabe, ein Pendel in die aufrechte Position zu bringen bzw. es dort zu halten. Das Pendel befindet sich auf einem Wagen, der mit einer Kraft F nach links oder

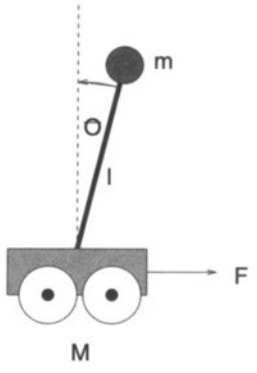


Abbildung 9.4 Inverses Pendel

rechts bewegt werden kann. Das Pendel ist an einem entsprechenden Gelenk befestigt, so dass es sich ebenfalls nur nach links und rechts, aber nicht nach vorn oder hinten frei bewegen kann. Abbildung 9.4 veranschaulicht die Aufgabe. Am Ende des Pendels der Länge l befindet sich eine punktförmige Masse m. Der Wagen, auf dem das Pendel befestigt ist, besitzt die Masse M. Unter Vernachlässigung von Reibung lässt sich diese Regelungsstrecke durch die folgende Differentialgleichung beschreiben:

$$(M+m)(\sin(\Theta))^2 l\ddot{\Theta} + ml\sin(\Theta)\cos(\Theta)(\dot{\Theta})^2 - (m+M)g\sin(\Theta) \;=\; -F\cos\Theta \quad (9.1)$$

g ist dabei die Gravitationskonstante. Θ ist die Abweichung des Pendels von der vertikalen Achse.

Für den Regler verwenden wir als Eingangsgrößen den Winkel Θ und die Winkelgeschwindigkeit $\dot{\Theta}$ und als Ausgabe- oder Stellgröße die Kraft F. Die zu regelnde Ausgangsgröße ist der Winkel Θ, der auf den Wert Null gebracht und gehalten werden soll. Wie bei vielen regelungstechnischen Anwendungen üblich verwenden wir hier den Fehler Θ und die Änderung des Fehlers $\dot{\Theta}$ als Eingangsgrößen. Klassische PD-Regler arbeiten nach diesem Konzept. Dabei steht „P" für einen proportionalen Anteil des Fehlers und „D" entsprechend für einen Anteil der Fehleränderung. Um konstante Störungen ausgleichen zu können, werden keine reinen PD-Regler, sondern PID-Regler eingesetzt. Bei diesen Reglern wird nicht der Wert der Stellgröße direkt ausgegeben, sondern nur seine Änderung. Den Stellwert erhält man somit durch die Integration – deshalb das „I" – über seine Änderungen.

Der Fuzzy-Regler soll aus Regeln der Form

> Wenn θ etwa Null ist und $\dot{\theta}$ ungefähr Null ist,
> dann ist auch F ungefähr Null.

bestehen. Um solche Regeln modellieren zu können, müssen zunächst geeignete Fuzzy-Mengen für die verwendeten Variablen, den Winkel, die Winkelgeschwindigkeit und die Kraft, definiert werden. Eine Festlegung von Fuzzy-Mengen über dem Definitionsbereich einer Variablen wird meistens als Fuzzy-Partition bezeichnet. Abbildung 9.5 zeigt eine

θ \ $\dot{\theta}$	ng	nm	eN	pm	pg
ng	nm	ng	eN	nm	
nm	ng	nm	nm	eN	nm
eN	ng	nm	eN	pm	pg
pm	nm	eN	pm	pm	pg
pg		pm	pg	pg	pm

Tabelle 9.1 Die Regelbasis für das inverse Pendel

mögliche Partition für den Definitionsbereich des Winkels. Die Fuzzy-Mengen repräsentieren die Ausdrücke „negativ groß" (ng), „negativ mittel" (nm), „etwa Null" (eN), „positiv mittel" (pm) und „positiv groß" (pg).

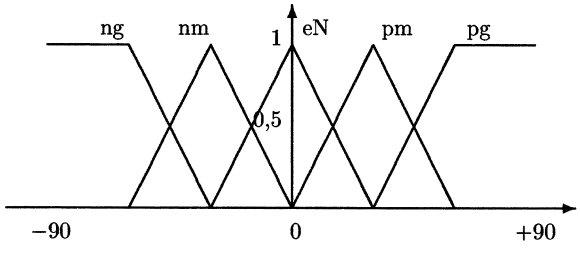


Abbildung 9.5 Die Fuzzy-Partition für den Winkel

Für die Definitonsbereiche der Winkelgeschwindigkeit und der Kraft werden analoge Fuzzy-Partitionen definiert, für die ebenfalls die obigen Ausdrücke verwendet werden, auch wenn die Fuzzy-Mengen nicht dieselben Zugehörigkeitsfunktionen auf den unterschiedlichen Definitionsbereichen repräsentieren.

Mit diesen Fuzzy-Mengen lassen sich nun die Regeln für das Balancieren des inversen Pendels formulieren. Tabelle 9.1 zeigt eine mögliche Regelbasis für das Stabbalanceproblem. Beispielsweise ist der Eintrag in Zeile vier und Spalte zwei als folgende Regel zu lesen:

$$\text{Wenn } \theta\text{=pm und } \dot{\theta}\text{=nm, dann } F\text{=eN}$$

Der Fuzzy-Regler besteht im Wesentlichen aus dieser „Wissensbasis" und den dazugehörigen Fuzzy-Mengen. Liegt eine konkrete, aktuelle Eingabe in Form eines Wertes für $\theta = \theta_0$ und $\dot{\theta} = \dot{\theta}_0$ vor, wird zunächst für jede Regel ihr Erfüllungsgrad bestimmt. Dazu ermittelt man für die in der entsprechenden Regel auftretenden Fuzzy-Mengen für θ und $\dot{\theta}$ den Zugehörigkeitsgrad für den Wert θ_0 bzw. $\dot{\theta}_0$. Diese beiden Zugehörigkeitsgrade müssen nun geeignet verknüpft werden und ergeben so den Erfüllungsgrad der Regel. Im einfachsten Fall wählt man als Erfüllungsgrad das Minimum der beiden Zugehörigkeitsgrade, d.h. den schlechteren der beiden Werte. Die Fuzzy-Menge im Konklusionsteil der Regel wird in der Höhe des Erfüllungsgrades der Regel abgeschnitten. Ist

die Regel zu 100% erfüllt, ergibt sich die ursprüngliche Ausgabe-Fuzzy-Menge. Trifft die Regel überhaupt nicht zu, liefert die Regel keinen Beitrag. Abbildung 9.6 verdeutlicht diese Vorgehensweise anhand zweier Regeln. Für den Winkel wird dort ein Eingangswert von $\theta = 20°$ angenommen, für die Winkelgeschwindigkeit $\dot{\theta} = 0.13$. Die beiden betrachteten Regeln lauten

> Wenn θ etwa Null und $\dot{\theta}$ etwa Null ist,
> dann ist F etwa Null

bzw.

> Wenn θ positiv mittel und $\dot{\theta}$ etwa Null ist,
> dann ist F positiv mittel.

Für die erste Regel ergibt sich ein Erfüllungsgrad von 0.33, da man für die vorliegenden Werte von θ und $\dot{\theta}$ Zugehörigkeitsgrade von 0.33 bzw. 0.75 zu den entsprechenden Fuzzy-Mengen erhält. Der Erfüllungsgrad, das Minimum von 0.33 und 0.75 wird dazu verwendet, um die Ausgabe-Fuzzy-Menge in dieser Höhe „abzuschneiden". Mit der zweiten Regel wird analog verfahren.

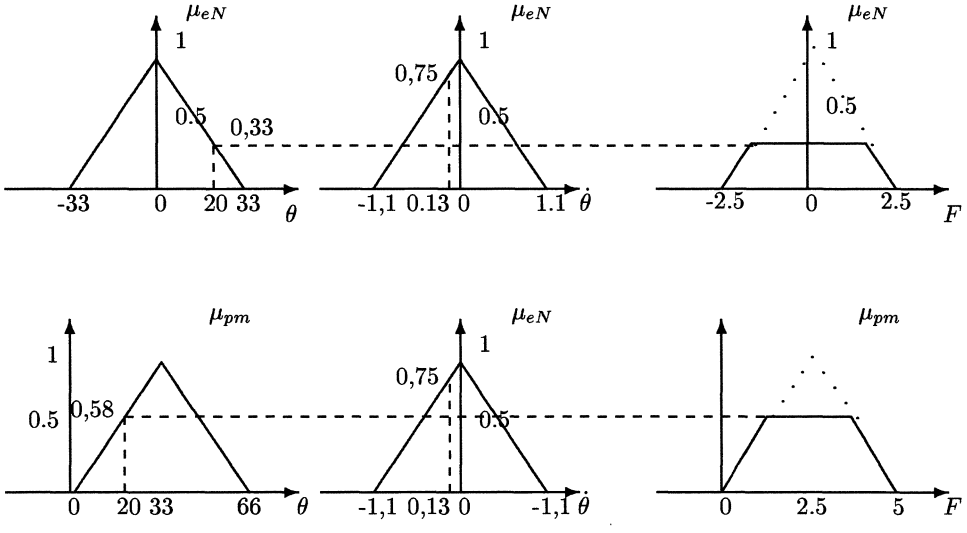


Abbildung 9.6 Die Auswertung zweier Regeln

Im nächsten Schritten müssen die so berechneten Ausgabe-Fuzzy-Mengen der einzelnen Regeln zu einer Fuzzy-Menge zusammengefasst werden. Hierfür bietet sich an, punktweise das Maximum über die Zugehörigkeitsgrade zu bilden. Auf diese Weise ergeben sich hohe Zugehörigkeitsgrade in Bereichen, wo Regeln mit einem hohen Erfüllungsgrad feuern. Abbildung 9.7 zeigt exemplarisch, wie die Ausgabe-Fuzzy-Mengen zweier Regeln auf diese Art zusammengefügt werden.

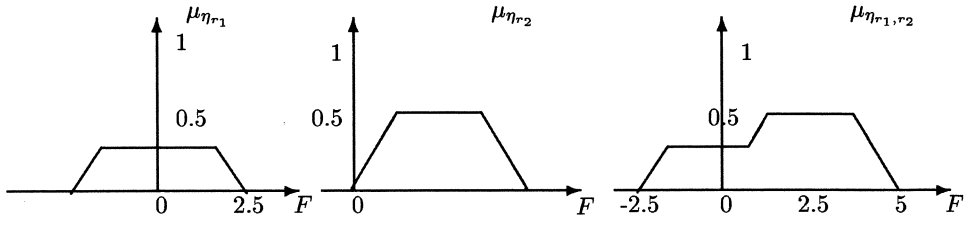


Abbildung 9.7 Die Verknüpfung der Ausgaben zweier Regeln

Das Ergebnis beschreibt den aus den Eingabewerten und den Regeln gewonnenen Stellwert in unscharfer Form als Fuzzy-Menge. Aus dieser Gesamtausgabe-Fuzzy-Menge muss schließlich ein konkreter Stellwert gewonnen werden. Diesen Vorgang nennt man *Defuzzifizierung*. Dies kann z.B. durch Berechnung des Schwerpunktes der Fläche unter der Kurve geschehen. Dieser Schwerpunkt entspricht dem mit den Zugehörigkeitsgraden der Gesamtausgabe-Fuzzy-Menge gewichteten Mittel über alle Stellwerte. Abbildung 9.8 zeigt die Defuzzifizierung.

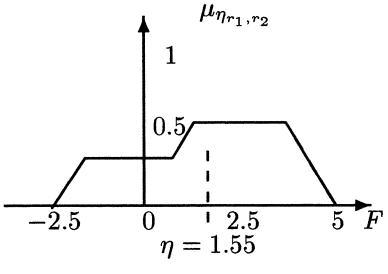


Abbildung 9.8 Defuzzifizierung

Diese intuitive Vorgehensweise der Verknüpfung der Regeln und der Berechnung des Stellwertes lassen sich auch aus theoretischer Sicht rechtfertigen, wenn die Regeln nicht als logische Implikationen, sondern als stückweise Definition einer unscharfen Funktion aufgefasst werden [KK93, KK95, Kla00].

Zur Berechnung des Erfüllungsgrades einer Regel aus den Zugehörigkeitsgraden der Eingangswerte zu den entsprechenden Fuzzy-Mengen wie in Abbildung 9.6 werden auch andere Operatoren anstelle des Minimums verwendet, beispielsweise das Produkt.

Ebenso wird das Maximum bei der Verknüpfung der Ausgabe-Fuzzy-Mengen wie in Abbildung 9.7 zum Teil auch mit anderen Operationen wie etwa der „Bounded Sum" berechnet. Die „Bounded Sum" summiert die Zugehörigkeitsgrade der Fuzzy-Mengen punktweise auf und begrenzt dabei die Summe jeweils auf Eins, falls Eins überschritten wird.

Auch für die Defuzzifizierung werden teilweise andere Strategien als die Schwerpunktberechnung verwendet.

Unabhängig davon, wie die oben beschriebenen Schritte im Einzelnen berechnet werden, nennt man diesen Typ eines Fuzzy-Reglers *Mamdani-Regler*. In der einfachsten Form verwendet der Mamdani-Regler keine Fuzzy-Mengen in den Konklusionsteilen der Regeln, sondern konstante Werte. Die Defuzzifizierung vereinfacht sich dann zum mit den Erfüllungsgraden der Regeln gewichteten Mittel dieser Konstanten. Diese Form des Fuzzy-Reglers ist ebenfalls ein Spezialfall eines anderen Fuzzy-Regler-Typs, dem *Tagaki-Sugeno-Kang-* oder nur *Tagaki-Sugeno-Regler*, der auch als *TSK-Modell* bezeichnet wird [SY93, TS85]. Das TSK-Modell unterscheidet sich vom Mamdani-Regler dadurch, dass in den Konklusionsteilen der Regeln keine Fuzzy-Mengen, sondern Funktionen stehen, die von den Eingangsgrößen abhängen. Eine Regel kann dann folgendermaßen aussehen:

$$\text{Wenn } \theta\text{=pm und } \dot{\theta}\text{=nm, dann } F = f(\theta, \dot{\theta}) = -0.1\theta + 0.05\dot{\theta}.$$

Selbstverständlich müssten in diesem Fall die physikalischen Einheiten noch angepasst werden.

Analog zum Fall, dass in den Konklusionsteilen der Regeln konstante Werte stehen, wird als Stellwert das mit den Erfüllungsgraden der entsprechenden Regeln gewichtete Mittel der berechneten Funktionswerte ausgegeben. Liegen die Eingangswerte θ_0 für den Winkel und $\dot{\theta}_0$ für die Winkelgeschwindigkeit vor und bezeichnet μ_R den Erfüllungsgrad der Regel R mit der Funktion f_R im Konklusionsteil, so ergibt sich als Stellwert

$$F = \frac{\sum_R \mu_R \cdot f_R(\theta_0, \dot{\theta}_0)}{\sum_R \mu_R}.$$

Üblicherweise verwendet man lineare Funktionen f_R in den Konklusionsteilen der Regeln. Der Gültigkeitsbereich einer Regel, der durch die Fuzzy-Mengen im Prämissenteil festgelegt wird, stellt einen unscharfen Bereich dar, in dem die zugehörige Funktion im Konklusionsteil der Regel als lokales Modell in diesem Bereich interpretiert werden kann. Die unscharfen Gültigkeitsbereiche der einzelnen Regeln überlappen sich, so dass an ihren Rändern durch die Berechnungsvorschrift für den Stellwert eine Interpolation zwischen den lokalen Modellen stattfindet. Man sollte allerdings bei einem TSK-Modell darauf achten, dass sich die Fuzzy-Mengen weniger stark überlappen als bei den üblichen Mamdani-Reglern. Ansonsten haben die lokalen Modelle nirgendwo ihre Gültigkeit, weil an allen Stellen zwischen ihnen interpoliert wird. Das führt dazu, dass der Regler zwar immer noch ein gutes Regelungsverhalten aufweisen kann, die Interpretierbarkeit der Regeln aber völlig verloren geht.

9.4 Fuzzy-Klassifikatoren

Betrachtet man einen Fuzzy-Regler als wissensbasierte Beschreibung für eine Funktionen, deren Wertebereich – die Werte der Stellgröße – reell ist, so dienen Fuzzy-Klassifikatoren dazu, Funktionen zu beschreiben, deren Wertebereich aus einer endlichen Menge von Klassen besteht. Bei einem Klassifikationsproblem soll wie bei einem Regressionsproblem der Wert einer Variablen aus der Kenntnis anderer vorhergesagt werden.

Allerdings nimmt die vorherzusagende Variable, auch Attribut genannt, keine reellen Werte wie bei der Regression an, sondern nur endlich viele, üblicherweise wohlunterscheidbare Klassen. Ein Beispiel für eine derartige Klassifikation können im Rahmen der Qualitätssicherung die Klassen *defekt/nicht defekt* für durch eine Maschine produzierte Teile sein. Es Klassen könnten auch *Glas, Papier, Plastik,...* bei einer automatischen Müllsortierungsanlage sein oder *kreditwürdig/nicht kreditwürdig* bei Kunden einer Bank.

Ein wesentlicher konzeptioneller Unterschied zu den Fuzzy-Reglern besteht bei den Fuzzy-Klassifikatoren darin, dass es keinen Sinn macht, zwischen den Ausgaben unterschiedlicher Regeln zu interpolieren. Feuern bei einem Fuzzy-Regler zwei Regeln, bei dem die eine mit einem schwachen Erfüllungsgrad für einen sehr großen Stellwert plädiert, während die andere mit einem höheren Erfüllungsgrad für einen mittleren Stellwert spricht, wird man wahrscheinlich einen etwas größeren als einen mittleren Wert nehmen. Bei einem Klassifikator, der einen Bankkunden als halb kreditwürdig und halb nicht kreditwürdig einstuft, macht es im Allgemeinen keinen Sinn, den halben Kredit zu gewähren. Man gibt ihm entweder den Kredit oder verweigert ihn. Eine Zwischenstufe ist nicht möglich.

Wenn die Möglichkeit zur Interpolation bei Fuzzy-Klassifikatoren nicht gegeben ist, stellt sich die Frage, ob es überhaupt einen Vorteil bringt, Fuzzy-Regeln zu verwenden, oder ob man mit scharfen Regeln nicht genauso viel erreichen kann. In [KK97] wurde gezeigt, dass man durch die Verwendung von Fuzzy-Systemen im Vergleich zu einfachen scharfen Regeln Klassifikationsprobleme zum einen mit weniger Regeln lösen kann und zum anderen Klassentrennungen beschreiben kann, die scharfe Regeln nur approximieren können. Dabei kommt es allerdings auf die Art der verwendeten Operationen z.B. für die Berechnung der Erfüllungsgrade der Regeln an. Das Minimum hat in diesem Fall stark einschränkende Eigenschaften [vSK99].

Es gibt eine Reihe verschiedener Ansätze für Fuzzy-Klassifikatoren. Einen sehr guten Überblick verschafft [Kun00]. Wir beschränken uns hier auf einen sehr einfachen Typ. Die Regeln, die der Fuzzy-Klassifikator verwendet, gleichen im Prämissenteil denen der Mamdani-Regler und der TSK-Modelle. Der Konklusionsteil besteht jedoch nur in der Angabe einer Klasse. Eine entsprechende Fuzzy-Klassifikationsregel könnte z.B. lauten:

Wenn Einkommen=hoch und Firmenzugehörigkeit=lang, dann Klasse=kreditwürdig.

Die Ausdrücke „hoch" bezogen auf das Einkommen eines Kunden und „lang" bezogen die Zugehörigkeit zu der Firma, in der der Kunde arbeitet, werden durch Fuzzy-Mengen modelliert. Der Ausdruck „kreditwürdig" wird nicht durch eine Fuzzy-Menge repräsentiert. Er sagt aus, dass dem Kunden ein Kredit gewährt werden sollte.

9.5 Fuzzy-Clusteranalyse

Die Clusteranalyse hat zum Ziel, einen Datensatz in endlich viele Gruppen oder Cluster ähnlicher Daten aufzuteilen. Clusteranalyse zählt somit auch zu den Klassifikationsverfahren. Der Unterschied zu Klassifikatoren, wie wir sie im vorhergehenden Abschnitt besprochen haben, besteht darin, dass bei einer Clusteranalyse die Klassen und ihre Labels oder Bezeichnungen vorher nicht bekannt sind. Ist für einen Datensatz die Klasse

jedes Elementes gegeben und soll daraus ein Klassifikator gelernt werden, gehört dies zum Bereich des überwachten Lernens. Die Clusteranalyse zählt zu den nicht-überwachten Lernverfahren.

Abbildung 9.9 zeigt einen Datensatz, der einer Clusteranalyse unterzogen werden soll. Nehmen wir einmal an, der Datensatz enthielte Daten von Lehrlingen einer Firma. Auf der x-Achse ist für jeden Lehrling die Zeit aufgetragen, die er für die Erstellung eines Werkstücks benötigte. Auf der y-Achse wird angegeben, wie viel Zeit ein Meister benötigte, um das Werkstück des Lehrlings zu korrigieren, so dass es verkauft werden konnte.

Zunächst einmal stellt man fest, dass die Daten nicht irgendwie verteilt sind, sondern mehr oder weniger in zwei Gruppen oder Cluster zerfallen. Man kann diese beiden Cluster sogar interpretieren. Das Cluster unten rechts repräsentiert offenbar die Lehrlinge, die langsam und sorgfältig arbeiten. Die Nachbearbeitungszeit für den Meister ist dort eher klein. Das Cluster oben links enthält die „schnellen Pfuscher", die ihre Arbeit eher rasch erledigen, bei denen aber ein höher Nachbearbeitungsbedarf besteht. Aus diesr Analyse kann man nun geeignete Maßnahmen und Konsequenzen für die Lehrlingsausbildung ableiten.

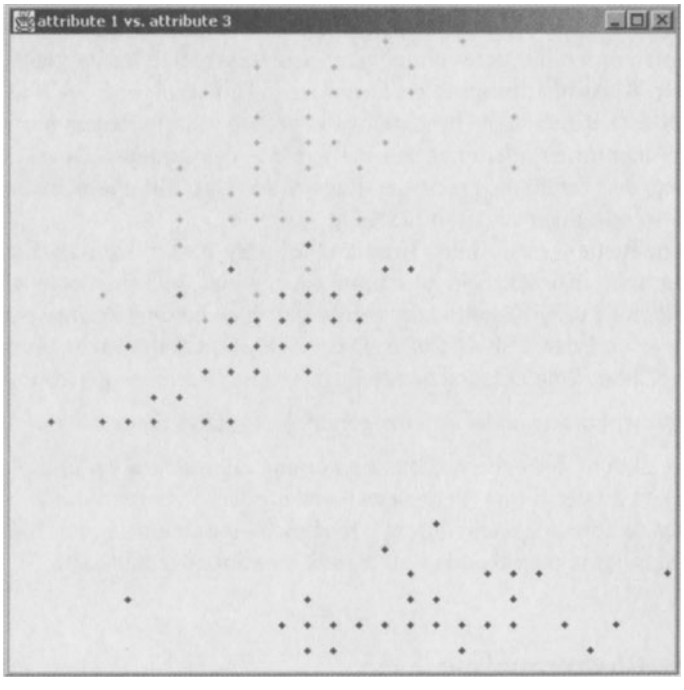


Abbildung 9.9 Ein zu clusternder Datensatz

Dieses hypothetische Beispiel veranschaulicht auf sehr einfache Weise, was man unter Clusteranalyse versteht und was man damit erreichen kann. Ein anderes Beispiel ist die Kundensegmentierung, bei der es darum geht, anstelle der Lehrlinge die Kunden

eines Unternehmens zu clustern um festzustellen, ob bzw. welche unterschiedlichen Kundenprofile vorliegen. Darauf aufbauend kann man wiederum Marketing-Kampagnen starten oder Produkte entwerfen, die speziell auf bestimmte Kundenprofile zugeschnitten sind, entweder um neue Kunden zu werben, deren Profil noch nicht abgedeckt wird, oder um ein noch besseres Angebot für die bereits bestehende Kundschaft zu haben.

Abbildung 9.9 veranschaulicht zwar die grundsätzliche Problematik der Clusteranalyse. In der Praxis hat man es aber selten mit Datensätzen zu tun, die nur aus zwei Dimensionen oder Attributen bestehen. Für höherdimensionale Daten ist es nicht möglich, die Clusterstruktur visuell zu erfassen. Daher werden Verfahren benötigt, die diese Aufgabe automatisieren und nicht in der Dimension der Daten eingeschränkt sind. Es gibt eine Reihe verschiedener Ansätze zur Clusteranalyse. Wir wollen uns hier auf die zielfunktionsbasierten Verfahren beschränken, da bei ihnen die Clusteranalyse durch die Minimierung einer Zielfunktion vollzogen wird und sich somit eine offensichtliche Verwandtschaft zu den evolutionären Algorithmen als Optimierungsverfahren ergibt.

Wir gehen davon aus, dass ein Datensatz $X = \{x_1, \ldots, x_n\} \subset \mathbb{R}^p$, bestehend aus n Datentupeln mit jeweils p reellen Komponenten, in c Cluster eingeteilt werden soll. In vielen Fällen ist es schwierig, die Anzahl c der Cluster geeignet zu wählen. Hierfür gibt es spezielle Verfahren, die diese Anzahl automatisch bestimmen. Eine Diskussion dieser Verfahren würde aber im Rahmen dieses Buches zu weit führen, so dass wir auf die Literatur [HKKR99] verweisen.

Bei einem vorgegebenen Datensatz und festgelegter Anzahl von Clustern wird häufig die folgende Zielfunktion für die Clusteranalyse verwendet:

$$f = \sum_{i=1}^{c} \sum_{j=1}^{n} u_{ij}^m d_{ij} \tag{9.2}$$

d_{ij} ist ein Distanzmaß, das angibt, wie weit das Datenobjekt x_j vom Cluster i entfernt ist. Im einfachsten Fall wird ein Cluster wie die Daten durch einen Punkt, dem Clusterprototypen $v_i \in \mathbb{R}^p$, repräsentiert und man definiert als Abstand des Datenobjektes x_j zum Cluster i den quadratischen euklidischen Abstand zwischen x_j und v_i, d.h.

$$d_{ij} = \| x_j - v_i \|^2 . \tag{9.3}$$

Der Wert u_{ij} gibt den Zugehörigkeitsgrad des Datentupels x_j zum Cluster i an. Sollen die Daten eindeutig Clustern zugeordnet werden, verlangt man $u_{ij} \in \{0, 1\}$. Bei der Fuzzy-Clusteranalyse wird $u_{ij} \in [0, 1]$ zugelassen, d.h. ein Datentupel kann zu mehreren Clustern mit unterschiedlichen Zugehörigkeitsgraden gehören. Der reelle Parameter $m > 1$ wird *Fuzzifier* genannt. Mit m kann gesteuert werden, wie stark sich Cluster überlappen dürfen. Für $m \to 1$ ergeben sich scharfe Cluster, d.h. $u_{ij} \to 0$ oder $u_{ij} \to 1$.

Die Optimierungsaufgabe besteht darin, die Clusterprototypen v_i und die Zugehörigkeitsgrade u_{ij} so zu bestimmen, dass die Zielfunktion einen möglichst kleinen Wert annimmt, d.h., dass die Daten eine möglichst kleine Distanz zu einem Cluster aufweisen sollten, dem sie zugeordnet sind. So formuliert, ist die Optimierungsaufgabe trivial zu lösen: Man wählt $u_{ij} = 0$ für alle $i \in \{1, \ldots, c\}$ und alle $j \in \{1, \ldots, n\}$. Das bedeutet, dass kein Datentupel irgendeinem Cluster zugeordnet wird. Um diese triviale Lösung auszuschließen, verlangt man die Nebenbedingungen

$$\sum_{i=1}^{c} u_{ij} = 1 \tag{9.4}$$

für alle $j \in \{1, \ldots, n\}$. Das bedeutet, dass jedes Datentupel eine Gesamtzugehörigkeit von Eins haben muss, diese aber auf verschiedene Cluster aufgeteilt werden kann.

Die Lösung dieses nichtlinearen Optimierungsproblems geschieht üblicherweise mittels einer *alternierenden Optimierung*. Dabei hält man abwechselnd einen der beiden Parametersätze – entweder die Zugehörigkeitsgrade oder die Clusterprototypen – fest, während der andere Parametersatz optimiert wird. Durch das Festhalten der Zugehörigkeitsgrade ergibt sich durch Ableiten die folgende analytische Lösung für die Clusterprototypen, die sich als Schwerpunkte der mit den Zugehörigkeitsgraden gewichteten Daten berechnen lassen:

$$v_i = \frac{\sum_{j=1}^{n} u_{ij}^m x_j}{\sum_{j=1}^{n} u_{ij}^m} \tag{9.5}$$

Hält man die Clusterprototypen fest und optimiert die Zugehörigkeitsgrade u_{ij} unter Berücksichtigung der Nebenbedingungen (9.4) mittels Lagrange'scher Multiplikatoren, ergibt sich

$$u_{ij} = \frac{1}{\sum_{k=1}^{c} \left(\frac{d_{ij}}{d_{kj}}\right)^{\frac{1}{m-1}}}. \tag{9.6}$$

Falls $d_{kj} = 0$ für mindestens ein Cluster k gilt, so muss von dieser Formel abgewichen werden und für eines dieser Cluster mit Distanz Null der Zugehörigkeitsgrad für das Datentupel x_j auf Eins und für alle anderen Cluster auf Null gesetzt werden.

Die Clusteranalyse wird durchgeführt, indem zufällig c Prototypen v_i initialisiert werden und dann abwechselnd die Zugehörigkeitsgrade mit Gleichung (9.6) und die Clusterprototypen mit Gleichung (9.5) aktualisiert werden, bis der Prozess konvergiert. Dieses Clustering-Verfahren wird *Fuzzy-c-Means-Algorithmus* (*FCM*) genannt. Lässt man nur scharfe Zugehörigkeiten zu, d.h. $u_{ij} \in \{0, 1\}$, wird anstelle der Gleichung (9.6) für die Zugehörigkeitsgrade für jedes Datentupel x_j der Zugehörigkeitsgrad für das Cluster i_0 auf Eins gesetzt, zu dem x_j den geringsten Abstand hat. Alle anderen u_{ij} werden auf Null gesetzt. Diese Form des Algorithmus heißt *harter c-Means-* (*HCM*) oder einfach nur *c-Means-Algorithmus*.

Einfache Clusterprototypen, die lediglich das Clusterzentrum repräsentieren, können nur sehr begrenzte Formen von Clustern erkennen. Es gibt eine Reihe von komplexeren Prototypen, die sich beispielsweise besser an die Clusterform [GK79] oder das Clustervolumen [KK03] anpassen können. Für solche Verfahren werden zusätzliche Parameter bei den Clusterprototypen aufgenommen und die Distanzfunktion (9.3) entsprechend modifiziert. Bei komplexeren Prototypen und Distanzfunktionen kann die Optimierung der Prototypen häufig nicht mehr analytisch durchgeführt werden. Besonders für diese Fälle sind evolutionäre Algorithmen interessant.

Die so genannten *Shell-Clustering-Verfahren* suchen nicht nach wolken- oder haufenförmigen Clustern, sondern nach Clustern in Form von Rändern geometrischer Formen wie Kreisen oder Ellipsen. Ein Vertreter dieser Klasse ist der *Fuzzy-c-Shells-Algorithmus* (*FCS*), der als Clusterprototypen zusätzlich zum Zentrum v_i einen Clusterradius r_i verwendet [Dav90]. Die Distanz eines Datentupels zum Cluster wird definiert als

$$d_{ij} = (\| x_j - v_i \| - r_i)^2 . \tag{9.7}$$

Dadurch hat ein Datentupel x_j genau dann die Distanz Null zu dem Cluster, wenn es sich auf dem Kreis (oder der Kugel, je nach Dimension der Daten) mit Mittelpunkt v_i und Radius r_i befindet. Für den FCS lassen sich keine expliziten Gleichungen für die Berechnung der Clusterprototypen angeben. Sie werden daher üblicherweise mit numerischen Verfahren, z.B. mit dem Newton-Verfahren, optimiert. Im Abschnitt 10.3 wird die Anwendung evolutionärer Algorithmen auf diese Problematik genauer diskutiert.

10 Kombinationen evolutionärer Algorithmen mit Fuzzy-Systemen

In diesem abschließenden Kapitel sollen die wichtigsten Kombinationen von evolutionären Algorithmen mit Fuzzy-Systemen aufgezeigt werden. Es gibt zu diesem Thema inzwischen so umfangreiche Literatur, dass nur ein Teil der Beiträge in das Literaturverzeichnis aufgenommen werden konnte. Allein das bereits 1997 erschienene Buch [Ped97] über „Fuzzy Evolutionary Computing" enthält 299 Literaturverweise.

Dieses Kapitel zeigt zunächst im Abschnitt 10.1 an einem ausführlicheren Beispiel mit grundsätzlichen Überlegungen auf, wie man einen Fuzzy-Regler mit evolutionären Algorithmen optimieren kann. Danach geben wir im Abschnitt 10.2 einen Überblick über unterschiedliche Ansätze, Anwendungen und Methoden zum Erlernen und Optimieren von regelbasierten Fuzzy-Systemen. Welche Vorteile und Grenzen der Einsatz evolutionärer Algorithmen für die (Fuzzy-)Clusteranalyse hat, wird im Abschnitt 10.3 behandelt. Alle diese Themen gehören zu dem Komplex der Optimierung von Fuzzy-Systemen mittels evolutionärer Algorithmen. Hierzu geben auch die Beiträge [HM97, CHHM01, CGH+04] einen guten Überblick.

Eine andere Kombinationsmöglichkeit von evolutionären Algorithmen mit Fuzzy-Systemen besteht darin, die Steuerung und Adaption des evolutionären Algorithmus mit Fuzzy-Regeln vorzunehmen. Diese Techniken sind Thema des Abschnitts 10.4.

10.1 Fuzzy-Regler-Optimierung: Ein ausführliches Beispiel

In diesem Abschnitt folgen wir dem Entwurf eines Mamdani-Reglers wie er in [KKK94b, KKK94a] vorgeschlagen wurde. Bei einem Mamdani-Regler können folgende Aspekte bei der Optimierung berücksichtigt werden:

- Die Fuzzy-Mengen
- Die Regelbasis
- Zusätzliche Parameter, falls beispielsweise die Verknüpfung von Zugehörigkeits- bzw. Erfüllungsgraden bei der Auswertung der Regeln nicht nur über das Minimum und das Maximum erfolgen soll, sondern alternativ parametrische Verknüpfungsoperatoren verwendet werden sollen. Das Gleiche gilt für die Defuzzifizierung. Hier gibt es andere – zum Teil parametrisierte Techniken – als die Schwerpunktmethode.
- Die Auswahl der relevanten Eingangsgrößen. In den meisten Anwendungen ist bekannt, welche Eingangsgrößen für die Regelung erforderlich sind. Bei einigen komplexen Systemen kann es aber vorkommen, dass eine größere Anzahl von zum

Teil redundanten Größen bekannt ist, die für die Regelung wichtig sind. In diesem Fall muss zuerst eine geeignete Auswahl der Größen getroffen werden, die der Regler verwenden soll.

Im Folgenden werden wir uns auf die beiden ersten Aspekte konzentrieren. Hier bieten sich zunächst drei mögliche Vorgehensweisen an.

1. Die Fuzzy-Mengen und die Regelbasis werden gleichzeitig optimiert.
2. Die Regelbasis wird zunächst bei fest vorgegebenen Fuzzy-Mengen optimiert. Anschließend werden die Fuzzy-Mengen bei Beibehaltung der vorher gefundenen Regelbasis angepasst.
3. Die Fuzzy-Mengen werden zunächst bei einer fest vorgegebenen Regelbasis optimiert. Anschließend wird die Regelbasis unter Verwendung der vorher konstruierten Fuzzy-Mengen optimiert.

Die dritte Möglichkeit ist nur in dem Falle realisierbar, wenn bereits Expertenwissen in Form von entsprechenden Regeln über die Regelung der Strecke vorliegt. Mit einer zufälligen Regelbasis zu starten, wird im Allgemeinen dazu führen, dass durch die Anpassung der Fuzzy-Mengen keine annähernd gute Regelung möglich ist. Im schlimmsten Fall müssen sich die Fuzzy-Mengen an falsche Regeln anpassen. Wenn das erfolgt, führt dies wiederum dazu, dass durch das anschließende Optimieren der Regelbasis keine sinnvolle Verbesserung erreicht werden kann.

Die erste Möglichkeit bedeutet, dass man gleichzeitig eine sehr große Anzahl von Parametern oder Freiheitsgraden zu berücksichtigen hat. Wenn sich eine Fuzzy-Menge gerade optimal für eine Regel eingestellt hat und die Regel anschließend durch genetische Operatoren geändert wird, muss der Anpassungsprozess wieder völlig von vorne beginnen. Deswegen verzichten wir auch auf die Anwendung dieser Strategie.

Die zweite Vorgehensweise ist sehr naheliegend. Man kann zunächst mit über die jeweiligen Wertebereiche ungefähr gleichmäßig verteilten Fuzzy-Mengen – wie z.B. in Abbildung 9.5 auf Seite 185 – zunächst die Regelbasis erlernen. Danach wird die Regelbasis festgehalten und die Fuzzy-Mengen werden optimiert. Die Änderung einer Regel, d.h. das Ersetzen einer Fuzzy-Menge in der Regel durch eine andere, bewirkt automatisch größere Änderungen im Reglerverhalten. Mit einer minimalen Veränderung einer Fuzzy-Menge wird sich auch das Reglerverhalten nur geringfügig ändern. Auch aus diesem Grund ist das stufenweise Vorgehen gerechtfertigt, zuerst die Regelbasis und dann die Fuzzy-Mengen zu optimieren. Mit der Regelbasis wird eine erste Grobanpassung vorgenommen. Die Fuzzy-Mengen dienen dann zur Feinabstimmung.

10.1.1 Generierung der Regelbasis

Betrachtet man die Regelbasis in Tabelle 9.1 auf Seite 185 für das inverse Pendel, so besteht die Aufgabe der Festlegung der Regelbasis im Ausfüllen einer Tabelle. Da bei dem inversen Pendel nur zwei Eingangsgrößen für den Regler verwendet wurden, handelt es sich um eine Tabelle im üblichen Sinn, deren Zeilen und Spalten durch die auf

den Eingangsgrößen definierten Fuzzy-Mengen indiziert werden. Die Tabelleneinträge spezifizieren die Ausgabe-Fuzzy-Menge der entsprechenden Regel. Bei mehr als zwei Eingangsgrößen entspricht die „Tabelle" einem mehrdimensionalen Array. Auch hier erfolgt die Indizierung über die Fuzzy-Mengen, die über den Eingangsgrößen definiert sind.

Dem Ausfüllen der Tabelle liegt bereits ein enorm großer Suchraum zugrunde. Bei zwei Eingangsgrößen und jeweils fünf Fuzzy-Mengen auf ihren Wertebereichen und dem Wertebereich der Stellgröße besteht der Suchraum aus $5^{25} \approx 2.98 \cdot 10^{17}$ möglichen Regeltabellen.

In der Praxis wird bei Fuzzy-Reglern die Regeltabelle in den seltensten Fällen vollständig ausgefüllt. Zum einen überlappen sich die unscharfen Gültigkeitsbereiche der Regeln, so dass, selbst wenn einzelne Einträge fehlen, immer noch ein Stellwert berechnet werden kann. Zum anderen gibt es Konfigurationen, in die die Regelstrecke niemals – oder nur durch extreme Eingriffe – geraten kann. Beispielsweise wird es beim inversen Pendel nicht vorkommen, dass der Stab fast waagerecht liegt, d.h. $\theta \approx 90°$, und sich gleichzeitig mit enorm großer Winkelgeschwindigkeit $\dot{\theta}$ aufrichtet. Die Reglerausgabe kann für solche Situationen undefiniert bleiben oder zu Null definiert werden. Aus diesem Grund wird zusätzlich zu den festgelegten Fuzzy-Mengen für die Stellgröße der leere Eintrag `NIL` zugelassen.

Im Beispiel des inversen Pendels, wie es im Abschnitt 9.3 eingeführt wurde, hatten wir für jede der beiden Eingangsgrößen und die Stellgröße jeweils fünf Fuzzy-Mengen vorgesehen. Um die Regeltabelle auszufüllen, müssen somit 25 Einträge definiert werden. Für jeden dieser Einträge gibt es sechs Möglichkeiten: Eine der fünf Fuzzy-Mengen der Stellgröße oder den Wert `NIL`. Es liegt also nahe, für die Regelbasis ein Chromosom mit 25 Genen zu verwenden, bei dem jedes Gen sechs mögliche Werte annehmen kann.

Üblicherweise werden Chromosomen als Sequenz von Genen aufgefasst. Eine kanonische Kodierung der Tabelle zeigt Abbildung 10.1. Die Zeilen der Tabelle werden aneinander gehängt und bilden das Chromosom.

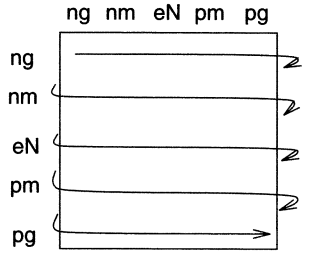


Abbildung 10.1 Linearisierung der Regeltabelle als Chromosom oder Gensequenz

Auch wenn die Bedeutung des Schema-Theorems und der Building-Block-Hypothese umstritten ist, so sollte berücksichtigt werden, was bei einer Mutation und insbesondere beim Crossover passiert. Bei einem Optimierungsproblem, bei dem die einzelnen Parameter, d.h. die Gene eines evolutionären Algorithmus, einzeln und völlig unabhängig

voneinander optimiert werden können, kann Crossover sehr vorteilhaft sein. Chromosomen, die einzelne Gene sehr gut optimiert haben, können diese Gene durch Crossover anderen Chromosomen zur Verfügung stellen. Der recht triviale Fall, dass die Parameter oder Gene unabhängig optimiert werden können, liegt in der Praxis jedoch eher selten vor. Es existieren mehr oder weniger starke Abhängigkeiten zwischen den einzelnen Parametern. Hängt die Wahl der Ausprägung zweier Gene sehr stark von einander ab und werden die beiden Gene beim Crossover auseinander gerissen, so trägt dies im Allgemeinen negativ zur Gesamtfitness der Nachkommen bei, wenn die Elternchromosomen zwar jeweils gute, aber unterschiedliche Kombinationen der beiden Gene besitzen. Für Crossover ist es daher vorteilhaft, wenn die Wahrscheinlichkeit, dass stark von einander abhängige Gene auseinander gerissen werden, klein ist.

Bei der Regelbasis eines Fuzzy-Reglers lassen sich die Abhängigkeiten zwischen den Parametern, d.h. den Tabelleneinträgen, relativ gut erkennen. Benachbarte Einträge wirken zum Teil gleichzeitig, da sich die unscharfen Gültigkeitsbereiche der zugehörigen Regeln überlappen. Einträge, die in der Tabelle weit von einander entfernt sind, beeinflussen sich praktisch überhaupt nicht, da die entsprechenden Regeln in völlig unterschiedlichen Situationen greifen. Will man diese Abhängigkeits- oder Nachbarschaftsstruktur beim Crossover so weit wie möglich erhalten, so wird dies durch die in Abbildung 10.1 veranschaulichte Kodierung der Regeltabelle als Chromosom nicht unterstützt. Abbildung 10.2 verdeutlicht, wie die in der Tabelle benachbarten, umrahmten Einträge bei der linearisierten Kodierung ihre Nachbarschaftsbeziehung verlieren.

	1	2	3	4	5
1	*	nm	nb	nm	nb
2	nb	nm	nm	*	*
3	*	*	*	*	*
4	*	*	*	*	*

1,1	1,2	1,3	1,4	1,5	2,1	2,2	2,3	2,4	
*	nm	nb	nm	nb	nb	nm	nm	*	···

Abbildung 10.2 Nachbarschaftsbeziehungen werden bei linearen Chromosomen aufgebrochen

Aus diesem Grund betrachten wir die Chromosomen nicht als lineare, sondern bei zwei Eingangsgrößen als planare Struktur. Bei drei oder mehr Eingangsgrößen liegt die Struktur in Form eines Quaders bzw. Hyperquaders vor.

Das One-Point-Crossover für gewöhnliche „lineare" Chromosomen wird durchgeführt, indem ein Gen ausgewählt wird und ab diesem Gen ein Austausch aller Gene mit dem Partnerchromosom stattfindet. In unserem Fall der planaren oder (hyper-)quaderförmigen Struktur wählen wir ebenfalls ein Gen $G_1 : (c_1^1, \ldots, c_n^1)$ als Crossoverpunkt aus und tauschen dann alle Gene aus, deren Koordinaten mindestens so groß sind wie die des Crossoverpunktes:

$$G : (x_1, \ldots, x_n);\ x_1 \geq c_1^1, \ldots, x_n \geq c_n^1$$

Abbildung 10.3 veranschaulicht diese Vorgehensweise. Das eingerahmte Gen wurde als

Crossoverpunkt ausgewählt und die darunter liegenden Teilrechtecke der beiden Chromosomen ausgetauscht.

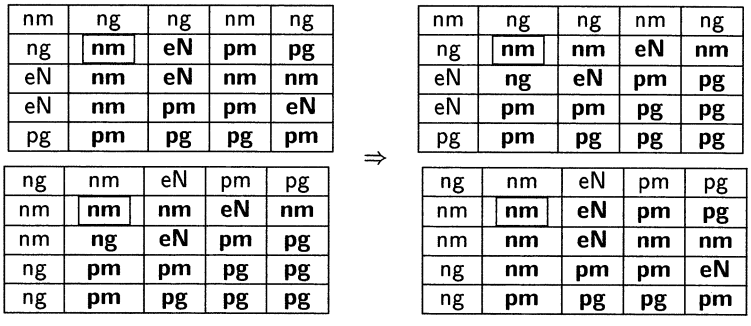


Abbildung 10.3 One-Point-Crossover für planare Chromosomen

Das Two-Point-Crossover lässt sich analog verallgemeinern, indem zwei Crossoverpunkte $G_1 : (c_1^1, \ldots, c_n^1)$ und $G_2 : (c_1^2, \ldots, c_n^2)$ gewählt werden. Die Koordinaten des zweiten Punktes müssen dabei in allen Koordinaten mindestens so groß sein, wie die des ersten Punktes. Es werden dann alle Gene ausgetauscht, deren Koordinaten in dem durch die beiden gewählten Gene begrenzten Rechteck, Quader bzw. Hyperquader liegen:

$$G : (x_1, \ldots, x_n);\ c_1^1 \leq x_1 \leq c_1^2, \ldots, c_n^1 \leq x_n \leq c_n^2$$

In Abbildung 10.4 ist dieser Vorgang noch einmal dargestellt.

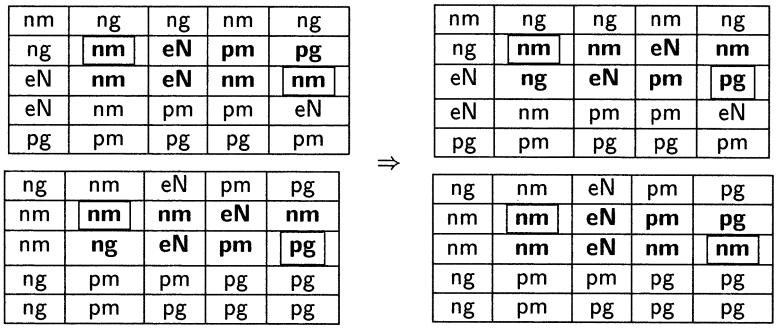


Abbildung 10.4 Two-Point-Crossover für planare Chromosomen

Da sich bei herkömmlichen linearen Chromosomen aus praktischer und theoretischer Sicht das Two-Point-Crossover dem One-Point-Crossover als überlegen erwiesen hat und sich die Argumentation dafür auch auf unsere allgemeineren Strukturen übertragen lassen, verwenden wir die eben vorgestellte Form des Two-Point-Crossovers für die Erzeugung der Regelbasis.

Um mit der Mutation keine extremen, sondern eher kleine Änderungen hervorzurufen, wird bei einem Gen, das mutiert werden soll, das vorhandene Allel nicht durch ein beliebiges anderes ersetzt. Das vorhandene Allele, das eine Fuzzy-Menge über dem Wertebereich des Stellwertes repräsentiert, wird zufällig durch eine benachbarte Fuzzy-Menge oder den Wert `NIL` ersetzt. Das Allel `eN` darf somit nur zu `pm`, `nm` oder `NIL` mutiert werden.

Man könnte die Erzeugung der Regelbasen der Startpopulation völlig zufällig vornehmen. Die Chromosom bestehen dann aus 25 Genen mit jeweils sechs möglichen Allelen, fünf Fuzzy-Mengen und dem Wert `NIL`. Damit hat der Suchraum eine Größe von $6^{25} \approx 2.84 \cdot 10^{19}$ Elementen. Davon sind die meisten Lösungen völlig unbrauchbar. Wie bereits beim Crossover und bei der Mutation sollte man auch hier vorhandenes Wissen über das Problem nutzen und einbringen.

Im Fall des Stabbalanceproblems könnte das wie folgt aussehen: Wenn das inverse Pendel aufrecht steht ($\theta \approx 0$) und sich nicht bewegt ($\dot{\theta} \approx 0$), muss der Regler auch nicht eingreifen. Allgemeiner gilt üblicherweise, dass keine Aktion eines Reglers erforderlich ist, wenn keine Abweichung vom Sollwert vorliegt und auch die Fehleränderung bei Null liegt. Diese Regel besagt, dass in der Regeltabelle an der Stelle (`eN`,`eN`) der Eintrag `eN` erfolgen sollte.

Eine weitere heuristische Regel besagt, dass sich benachbarte Einträge nicht extrem unterscheiden sollten. Auch dies sollte bei der Erzeugung der Chromosomen der Startpopulation berücksichtigt werden, beispielsweise indem man verlangt, dass der Abstand in der Reihenfolge der Fuzzy-Mengen für benachbarte Einträge niemals größer als zwei sein darf. Benachbarte Einträge von `pm` dürfen daher nur aus der Menge

$$\{\texttt{nm}, \texttt{eN}, \texttt{pm}, \texttt{pg}\}$$

stammen. In [SKK96] werden diese Konzepte verwendet, um eine allgemeine Reglerstruktur zu entwerfen, die mit evolutionären Algorithmen optimiert werden kann.

10.1.2 Die Fitnessfunktion

Es gibt eine Reihe von Kriterien, die ein guter Regler erfüllen sollte. Der Regler sollte beispielsweise aus jeder denkbaren Anfangskonstellation die Strecke möglichst schnell und mit wenig Aufwand auf den Sollwert führen.

Um einen Regler zu bewerten, wird das Verhalten der Regelstrecke, ausgehend von verschiedenen Ausgangssituationen oder Testfällen, über einen bestimmten Zeitraum hinweg mit Hilfe eines mathematischen Modells simuliert. Im Wesentlichen wird dazu die Differentialgleichung 9.1, die das inverse Pendel beschreibt, jeweils numerisch gelöst unter den aktuell gegebenen Anfangsbedingungen $\theta = \theta_0$ und $\dot{\theta} = \dot{\theta}_0$ und der durch den Regler vorgegebenen Kraft $F = F_0$ als Stellwert.

Um das Ziel, dass der Regler die Strecke möglichst schnell auf den Sollwert führt, zu bewerten, wird die Abweichung des Istwertes vom Sollwert verfolgt. Außerdem soll der Regler den Sollwert mit möglichst wenig Aufwand erreichen. Dazu werden seine Stellwertausgaben protokolliert und betragsmäßig aufintegriert.

Bei der Durchführung der Testfälle für einen Regler wird folgendermaßen vorgegangen. Weicht der Istwert bei einer Simulation vom Sollwert um einen nicht mehr

akzeptablen Wert ab, wird die Simulation abgebrochen und der Regler erhält keine (positiven) Bewertungspunkte für diesen Testfall. Hierzu muss eine Schranke festgelegt werden, die der Istwert zu keinem Zeitpunkt der Simulation überschreiten darf. Beim inversen Pendel muss der Winkel im Bereich zwischen $-90°$ und $90°$ bleiben.

Diese generelle Abweichung entspricht einer Situation, in die der Prozess auf keinen Fall geraten sollte. Zur Bewertung, in wie weit der Regler den Sollwert einstellen kann, wird ab einer bestimmten Dauer der Simulation überprüft, ob sich der Istwert in der Nähe des Sollwertes befindet und dort verbleibt. Ist dies nicht gegeben, wird die Simulation ebenfalls abgebrochen und der Regler erhält keine Bewertungspunkte für diesen Testfall. Wählt man die Toleranzschwelle um den Sollwert statisch für alle Generationen, ergibt sich folgendes Problem. Bei einem sehr kleinen Toleranzwert werden die Regelbasen der Startpopulation wahrscheinlich keinen der Testfälle erfolgreich beenden können. Alle Chromosomen erhalten eine Fitness von Null und es findet eine reine Zufallssuche statt, bis schließlich ein Chromosom wenigstens einen Testfall bewältigen kann. Dieses Chromosom wird dann die Nachfolgegeneration dominieren. Entscheidet man sich für einen großzügigen Toleranzwert besteht in späteren Generationen kein Anreiz für den Regler, den Sollwert möglichst genau einzustellen.

Aus diesem Grund wird in Anlehnung an [HK94] ein variabler Toleranzwert verwendet. Man startet mit einem großen Toleranzwert der schrittweise, mit den sich verbessernden Chromosomen immer kleiner gewählt wird. Auf diese Weise wird der Selektionsdruck im Laufe der Generationen erhöht.

Am Anfang starten wir mit einem Toleranzwert von $\pm 90°$, d.h., für ein Chromosom der Startpopulation genügt es, nur zu verhindern, dass das inverse Pendel nicht umfällt. Das Balancieren in der Vertikalen spielt noch keine Rolle. Immer wenn mindestens ein Chromosom einer Generation in der Lage war, alle Testfälle erfolgreich zu bestehen, wird der Toleranzwert um einen Faktor verringert, beispielsweise auf 70% des bisherigen Wertes gesetzt. Dies wird solange fortgesetzt, bis der Toleranzwert weniger als $\pm 3°$ beträgt.

Bei den bisherigen Überlegungen wurde der Energieaufwand, den ein Regler benötigt, noch nicht berücksichtigt. Durch große, sehr schnell das Vorzeichen wechselnde Werte der Kraft kann der Regler in der Art eines „Dead-Beat-Controllers" das inverse Pendel innerhalb eines sehr engen Toleranzbereiches um den Sollwert halten. Um dieses Verhalten nicht zu fördern, werden bei Testfällen, die ein Chromosom erfolgreich bewältigt hat, die Stellwerte aufintegriert bzw. aufsummiert. Je geringer diese Summe ist, desto besser wird das Chromosom bewertet.

Selektion erfolgt nach dem Remainder Stochastic Sampling Prinzip. Eine Kopie des besten Chromosoms wird in jedem Fall in die nächste Generation übernommen, ohne dass es durch Crossover oder Mutation verändert wird.

10.1.3 Optimierung der Fuzzy-Mengen

Nachdem im vorhergehenden Abschnitt beschrieben wurde, wie eine Regelbasis eines Fuzzy-Reglers bei fest gewählten und zunächst nicht veränderbaren Fuzzy-Mengen mittels eines evolutionären Algorithmus erzeugt werden kann, sollen nun die Fuzzy-Mengen

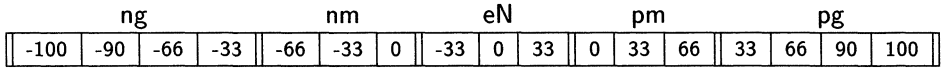


Abbildung 10.5 Eine mögliche Kodierung einer Fuzzy-Partition

modifiziert werden, um das Reglerverhalten noch weiter zu verbessern. Dazu muss zunächst eine geeignete Kodierung der Fuzzy-Mengen festgelegt werden. Verwendet man dreieckige oder trapezförmige Fuzzy-Mengen, könnte man für jeden Wertebereich jede Fuzzy-Menge durch drei bzw. vier Werte kodieren. Eine derartige Kodierung mit binären bzw. reellen Genen wurde in [Kar91, Kar93, LT93b] vorgenommen. Abbildung 10.1.3 zeigt die Kodierung der in Abbildung 9.5 auf Seite 185 dargestellten Fuzzy-Partition.

Bei dieser Form der Kodierung muss ggf. durch geeignete Reparaturmechanismen sicher gestellt werden, dass die drei oder vier Parameter, die eine Dreiecks- bzw. Trapezfunktion definieren, jeweils in aufsteigender Reihenfolge sortiert sind. Ein weiterer unerwünschter Effekt sind mögliche „Überholvorgänge" bei den Fuzzy-Mengen. Die Kodierung schließt nicht aus, dass die Fuzzy-Menge `negativ mittel` durch Mutation und Crossover nicht irgendwann rechts von der Fuzzy-Menge `positiv mittel` liegt. Ein solches Ergebnis würde die Regelbasis ad absurdum führen.

Eine Möglichkeit, dieses Problem zu lösen, besteht darin, nicht die Koordinaten direkt anzugeben, sondern jeweils den Abstand zur vorhergehenden Koordinate und nur positive Werte zuzulassen [LT93b]. Trotzdem bleibt diese Art der Kodierung relativ starr, da man vorher durch die Anzahl der Parameter festlegen muss, welche Fuzzy-Mengen als Dreiecks- bzw. Trapezfunktionen ausgelegt werden sollen.

Ein wesentlicher Nachteil zeigt sich jedoch, wenn man die Effekte des Crossover-Operators betrachtet. Auch wenn durch eine geeignete Kodierung Überholvorgänge vermieden werden, können die einem Ausdruck wie `etwa Null` in zwei unterschiedlichen Chromosomen zugeordneten Fuzzy-Mengen an ganz unterschiedlichen Stellen des Wertebereichs der entsprechenden Größe ihren Wirkungsbereich haben. Bei der Durchführung von Crossover sollten eher Fuzzy-Mengen ausgetauscht werden, die einen ähnlichen Wirkungsbereich haben als Fuzzy-Mengen, die zufällig die gleichen Namen haben.

Aus diesem Grund soll hier die folgende Strategie realisiert werden. Die Zugehörigkeitsgrade der Fuzzy-Mengen werden an n Stellen des Wertebereichs einer Größe abgetastet. Zwischen zwei Abtaststellen werden die Zugehörigkeitsgrade linear interpoliert. Werden n_d Fuzzy-Mengen für die Fuzzy-Partition einer Größe d verwendet, deren Wertebereich an k Stellen abgetastet wird, enthält jedes der k Gene n_d Werte zwischen Null und Eins, die die Zugehörigkeitsgrade der Fuzzy-Mengen an der entsprechenden Abtaststelle, die dem Gen zugeordnet ist, wiedergeben. Abbildung 10.6 veranschaulicht diese Kodierung.

Stellt man die in Abbildung 9.5 auf Seite 185 dargestellte Fuzzy-Partition in dieser Form mit $k = 13$ über den Wertebereich gleichmäßig verteilten Abtaststellen

$$\{-90, -75, 60, \ldots, 60, 75, 90\}$$

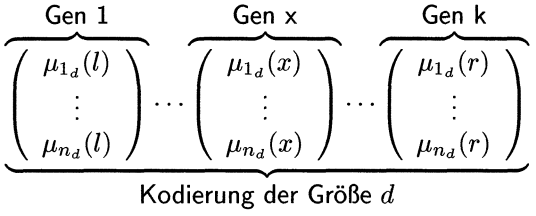


Abbildung 10.6 Kodierung der Größe d

dar, ergibt sich die Kodierung aus Tabelle 10.1.

ng	1	1	1	0.5	0	0	0	0	0	0	0	0	0
nm	0	0	0	0.5	1	0.5	0	0	0	0	0	0	0
eN	0	0	0	0	0	0.5	1	0.5	0	0	0	0	0
pm	0	0	0	0	0	0	0	0.5	1	0.5	0	0	0
pg	0	0	0	0	0	0	0	0	0	0.5	1	1	1
	-90	-75	-60	-45	-30	-15	0	15	30	45	60	75	90

Tabelle 10.1 Beispiel einer Kodierung

Mit dieser Kodierung lassen sich wesentlich allgemeinere Fuzzy-Mengen als nur Dreiecks- oder Trapezfunktionen repräsentieren. Insbesondere können auch multimodale Fuzzy-Mengen auftreten, die die Interpretierbarkeit des Fuzzy-Reglers erschweren. Aus diesem Grund müssen beim Crossover und der Mutation geeignete Reparaturmechanismen angewendet werden.

Wir betrachten zunächst den Crossover-Operator. Es soll Two-Point-Crossover auf Gene in der Form aus Abbildung 10.6 angewendet werden. Es kann, wie in Abbildung 10.7 zu sehen ist, dazu kommen, dass Fuzzy-Mengen beim Crossover aus einer Fuzzy-Partition verschwinden. Dies kann nachteilig, unter Umständen aber auch vorteilhaft sein, wenn diese Fuzzy-Mengen für die Regelung nicht erforderlich sind.

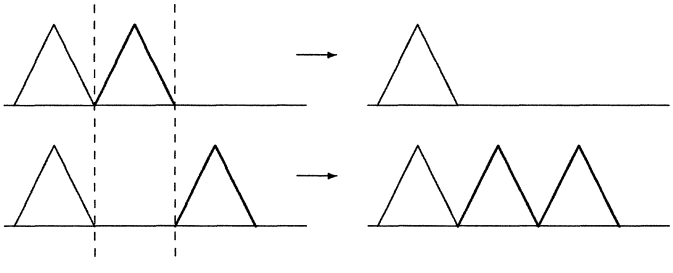


Abbildung 10.7 Auslöschen von Fuzzy-Mengen beim Crossover

Es kann auch vorkommen, dass sich multimodale Fuzzy-Mengen nach dem Crossover ergeben. In diesem Fall sollten Reparaturmechanismen angewendet werden. Abbildung 10.8 zeigt drei Möglichkeiten, eine bimodale Fuzzy-Mengen mittels eines Reparaturmechanismus in eine unimodale Fuzzy-Menge umzuwandeln. Bei der zweiten und dritten Methode kann es vorkommen, dass die Fuzzy-Menge auch im Maximum nicht den Zugehörigkeitsgrad Eins annimmt. In diesem Fall müsste ein weiterer Reparaturmechanismus greifen, der das Maximum ggf. auf Eins anhebt.

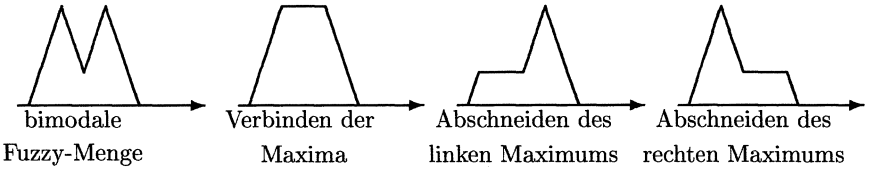


Abbildung 10.8 Auslöschen von Fuzzy-Mengen beim Crossover

Diese Reparaturmechanismen müssen auch bei der Mutation angewendet werden. Um durch die Mutation nur kleine Änderungen hervorzurufen, wird ein Wert u, der mutiert werden soll, durch einen Zufallswert aus dem Intervall

$$\left[\frac{\text{fit}}{\text{maxfitness}} \cdot u, 1 - (1-u) \cdot \frac{\text{fit}}{\text{maxfitness}}\right]$$

ersetzt. fit ist dabei die Fitness des Chromosoms, zu dem u gehört, maxfitness die maximal erreichbare Fitness.

Für die Startpopulation wird bei der Optimierung der Fuzzy-Mengen das Chromosom verwendet, das die Fuzzy-Partitionen repräsentiert, die für die Optimierung der Regelbasis verwendet wurden. Dieses Chromosom selbst und eine entsprechende Anzahl durch Mutation dieses Chromosoms erzeugter Chromosomen bilden die Startpopulation.

Während es beim Erzeugen der Regelbasis primär darum ging, überhaupt einen funktionierenden Fuzzy-Regler zu erzeugen, dient die Anpassung der Fuzzy-Mengen dazu, das Verhalten dieses Reglers möglichst optimal zu gestalten. Daher sollte hier eine andere Fitnessfunktion als bei der Regelbasis verwendet werden. Die Fitnessfunktion sollte die gewünschten Aspekte des Reglers stärker berücksichtigen, da man davon ausgehen kann, dass das alleinige Bewältigen der Testfälle kein Problem mehr darstellt. Je nachdem, welche Ziele mit dem Regler verfolgt werden, kann die Fitnessfunktion angepasst werden. Derartige Ziele – einzeln oder in Kombination – könnten z.B. sein:

- eine noch genauere Einhaltung des Sollwertes
- ein möglichst schnelles Erreichen des (ungefähren) Sollwertes
- ein möglichst geringer Energieaufwand

θ \ $\dot{\theta}$	ng	nm	eN	pm	pg
ng	pm	nm	ng	pg	eN
nm	ng	ng	nm	pg	pg
eN	nm	ng	eN	pg	pg
pm	eN	pm	pm	pg	pg
pg	eN	nm	pm	pm	eN

Tabelle 10.2 Die beste erzeugte Regelbasis

10.1.4 Ergebnisse

Abbildung 10.9 zeigt die Entwicklung der maximalen und der durchschnittlichen Fitness für das inverse Pendel bei der Erzeugung der Regelbasis, angefangen bei der Startpopulation (Generation 0) bis zur letzten Generation (Generation 15).

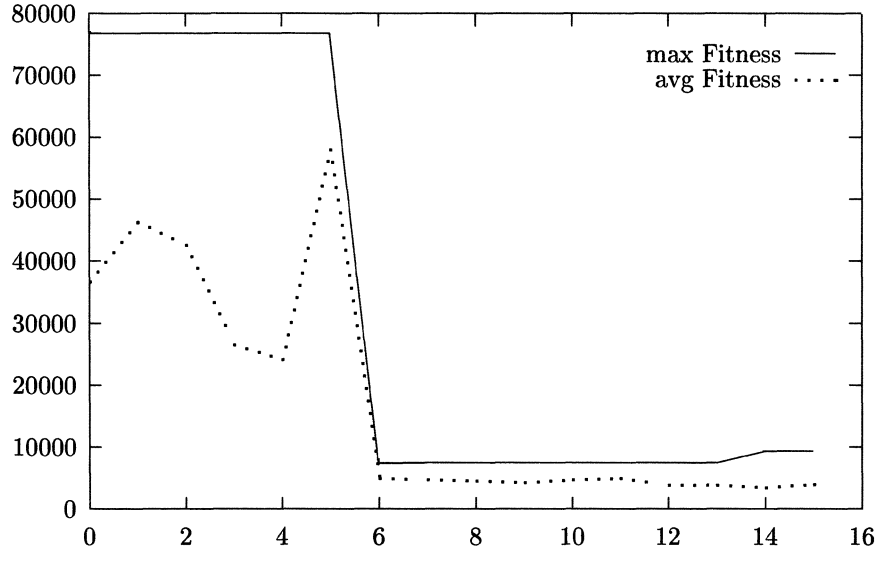


Abbildung 10.9 Fitnessverlauf beim Erzeugen der Regelbasis

Der nach der fünften Generation auftretende starke Abfall der Fitness liegt darin begründet, dass der Toleranzbereich um den Sollwert entsprechend klein geworden ist. Jedes Mal, wenn ein Chromosom alle Testfälle bewältigen konnte wurde der Toleranzwert halbiert. In der sechsten Generation ist dieser Wert von anfangs 90° auf weniger als 3° abgefallen. Die beste, nach 15 Generationen bei einer Populationsgröße von 150 Chromosomen erzeugte Regelbasis ist in Tabelle 10.1.4 beschrieben.

Mit dieser Regelbasis wurden anschließend die Fuzzy-Mengen optimiert. Dabei wurde ein vereinfachter Mamdani-Regler mit konstanten Ausgabewerten in den Konklusionsteilen der Regeln verwendet, so dass nur die Fuzzy-Partitionen für die Eingangsgrößen θ und $\dot{\theta}$ angepasst werden mussten. Den entsprechenden Verlauf der Fitness zeigt Abbildung 10.10.

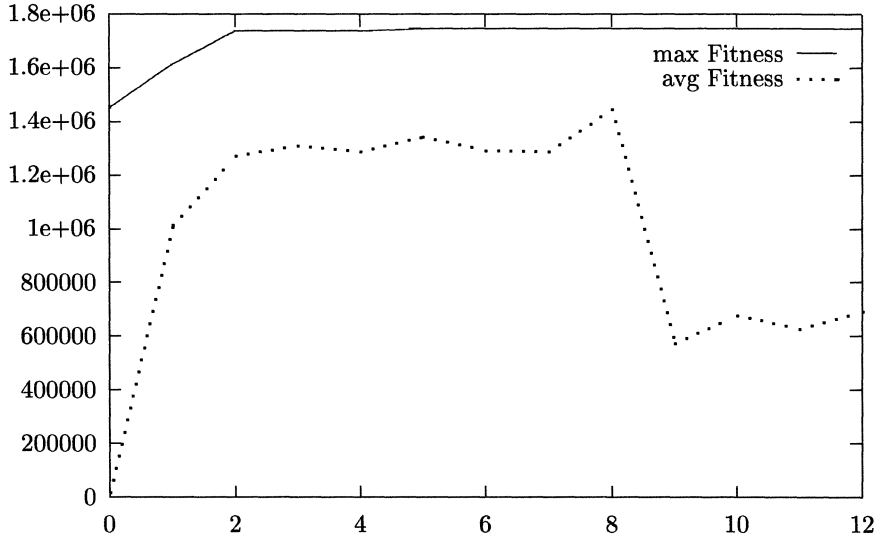


Abbildung 10.10 Fitnessverlauf beim Optimieren der Fuzzy-Mengen

Die optimierten Fuzzy-Mengen in Abbildung 10.11 sehen zunächst etwas überraschend aus. Es wurde nur jeweils die Fuzzy-Menge für `eN` modifiziert und nach rechts verschoben bzw. ausgedehnt. Dies liegt daran, dass wir in unserer Fitnessfunktion dem geringen Energieverbrauch ein großes Gewicht gegeben haben. Durch die Ausdehnung der Fuzzy-Menge, die primär an Regeln mit Ausgabe Null oder zumindest kleiner Ausgabe beteiligt ist, kann die Gesamtenergie, die der Regler benötigt, verringert werden.

Als abschließendes Beispiels dieses Abschnitts sollen noch die Ergebnisse zur Drehzahlregelung eines Gleichstrommotors unter Last betrachtet werden. In Abbildung 10.12 sind die Verhaltensweise klassischer PI- und PID-Regler beim Einregeln auf den Sollwert Eins zu sehen. Beide Regler zeigen ein Überschwingverhalten. Der PID-Regler erreicht den Sollwert relativ schnell. Die Regler wurden nicht auf Energieaufwandsminimierung ausgelegt.

Dem gegenüber zeigt der Fuzzy-Regler, bei dem allerdings der Energieaufwand mit berücksicht wurde, ein etwas anderes Verhalten, das in Abbildung 10.13 für die reine Erzeugung der Regelbasis und der anschließenden Optimierung der Fuzzy-Mengen dargestellt ist. Er zeigt kein Überschwingverhalten, benötigt dafür aber etwas mehr Zeit als der PID-Regler um den Sollwert zu erreichen.

Der Fuzzy-Regler mit nachträglich optimierten Fuzzy-Mengen zeigt ein verbessertes Regelverhalten gegenüber dem Regler, bei dem nur die Regelbasis optimiert wurde. Bei

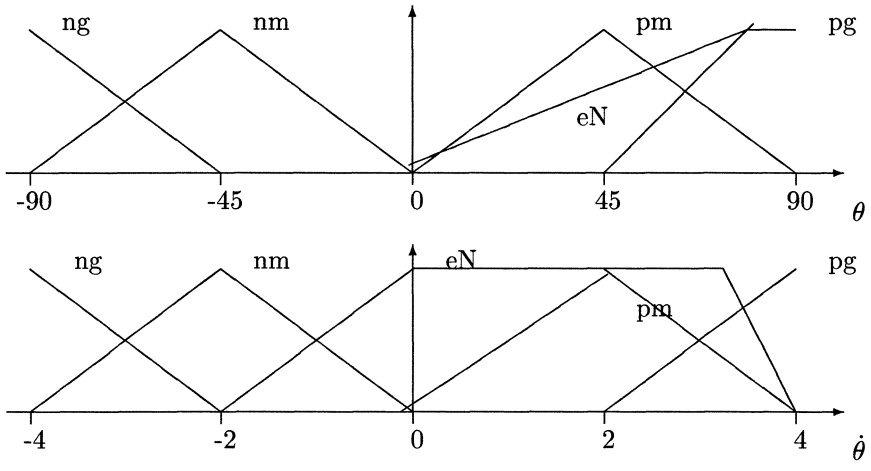


Abbildung 10.11 Die optimierten Fuzzy-Partitionen

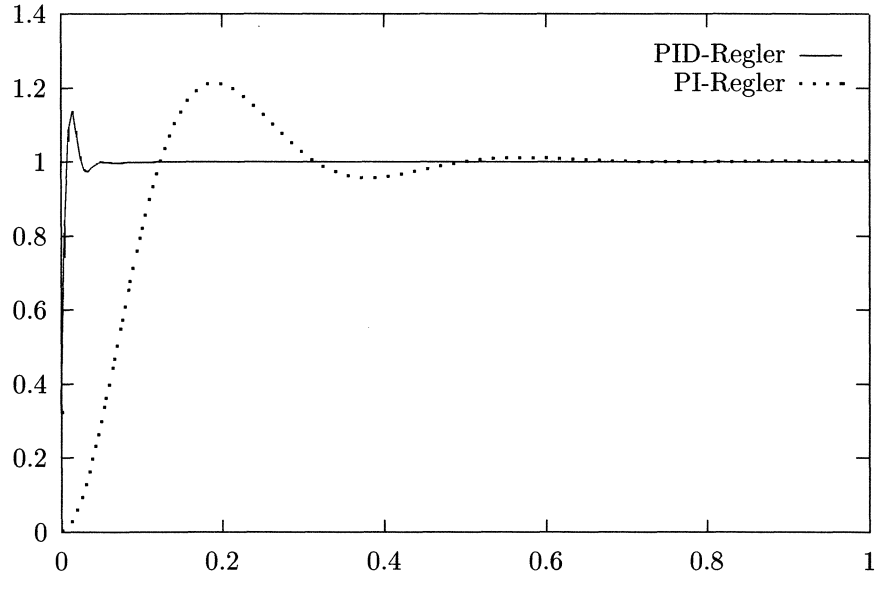


Abbildung 10.12 Regelung eines Gleichstrommotors mit einem PI- und einem PID-Regler

der Anpassung tritt ein ähnlicher, allerdings noch stärkerer Effekt auf wie bereits beim inversen Pendel. Die Fuzzy-Mengen, die in Abbildung 10.14 zu sehen sind, verlieren hier ihre Interpretierbarkeit fast vollständig.

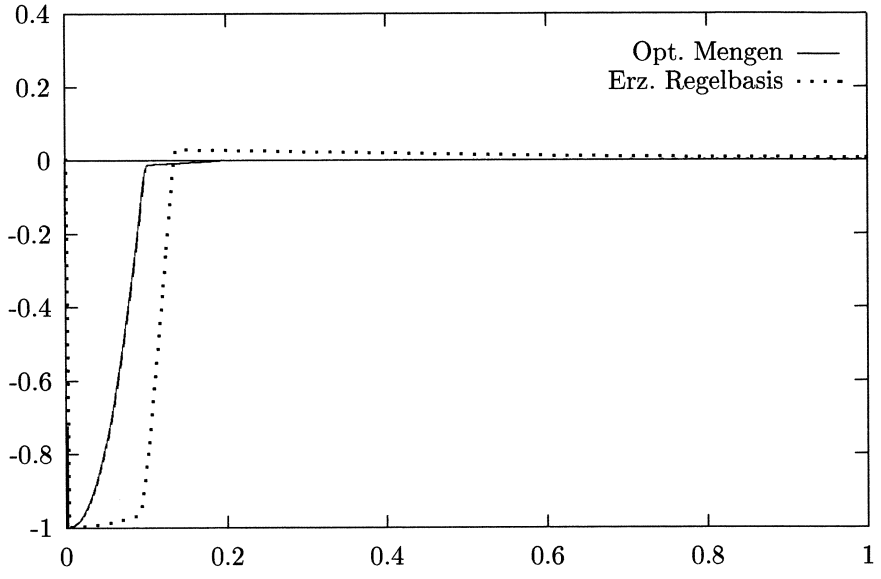


Abbildung 10.13 Regelung eines Gleichstrommotors mit einem Fuzzy-Regler

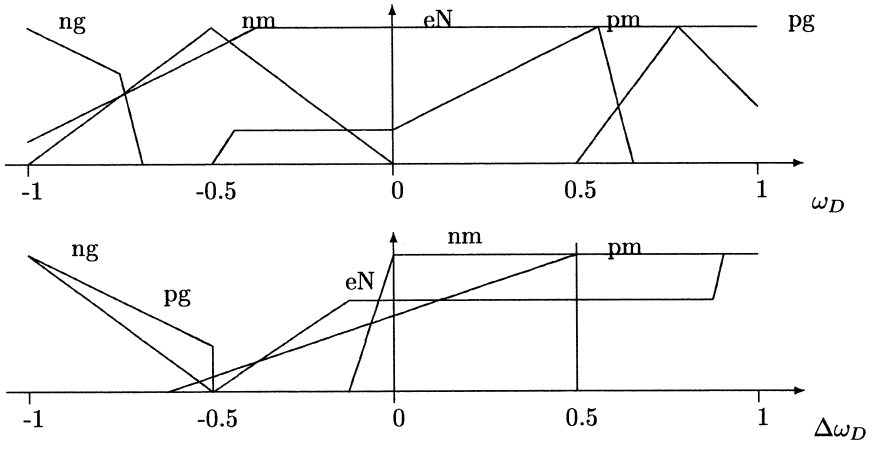


Abbildung 10.14 Die optimierten Fuzzy-Partitionen für die Gleichstrommotorregelung

Dieses Problem des Verlusts der Interpretierbarkeit tritt häufig bei Fuzzy-Systemen auf, wenn der Optimierung freien Lauf gelassen wird, ohne die Parameter stärker einzuschränken. Im Folgenden werden wir weitere Ansätze zur Optimierung von regelbasierten Fuzzy-Systemen diskutieren, die zum Teil auch auf die Problematik der Interpretierbarkeit eingehen.

10.2 Optimierung von Fuzzy-Systemen mit evolutionären Algorithmen

Allein die Zitate der gesamten Literatur zur Optimierung von Fuzzy-Systemen mit evolutionären Algorithmen würden dieses Buch füllen. Deswegen beschränken wir uns hier auf eine Diskussion der grundsätzlichen Ansätze und Prinzipien mit Verweisen auf ausgewählte Literatur.

Wir konzentrieren uns in diesem Abschnitt primär auf Fuzzy-Regler oder allgemeiner auf Systeme, die eine Regelbasis mit Fuzzy-Mengen verwenden, um einen reellen Ausgabewert, bei Fuzzy-Reglern den Stellwert, zu bestimmen.

Ein erster Ansatzpunkt für evolutionäre Algorithmen besteht in einer Vorverarbeitung der Eingaben eines Fuzzy-Reglers in Form von Skalierungen. Anstelle der Eingabe x_i wird die transformierte Eingabe $f_i(x_i)$ verwendet. Dabei ist $f_i : \mathbb{R} \to \mathbb{R}$ eine geeignete, üblicherweise streng monoton wachsende Transformation. In [NL94] wird ein einfacher Faktor zur Skalierung verwendet, [MM97] skaliert linear auf ein beliebiges Intervall. Nichtlineare Skalierungen finden sich in [Mag97, GGP98].

Man kann Skalierungsfunktionen als Transformationen der Eingangswerte sehen. Man kann sie aber genauso gut als Transformationen der Fuzzy-Mengen verstehen [KK94]. Die transformierten Fuzzy-Mengen $\tilde{\mu}$ ergeben sich aus den ursprünglichen Fuzzy-Mengen μ durch $\tilde{\mu}(x) = \mu(f(x))$, wobei f die Transformation für die Größe x ist.

Zur Optimierung der Fuzzy-Mengen wird üblicherweise eine einfach parametrisierbare Form der Fuzzy-Mengen vorausgesetzt, d.h. Dreiecks- (z.B. in [CH97] oder [CH01], wo auch eine Nischentechnik zur Anwendung kommt), Trapez- (z.B. in [Kar93, HLV95]) oder Gaußfunktionen (z.B. in [HS96, Gur99]).

Das Erlernen der Regelbasis erfolgt meistens im Sinne des Pittsburgh-Ansatzes. Das bedeutet, dass jedes Chromosom eine vollständige Regelbasis kodiert [PK91, Thr91, HK94, KKK94b, HP97]. Der Michigan-Ansatz findet sich in [VR91, Bon96, INM99, JLL00]. Iteratives Regellernen wird in [CH97, CdJHL99, GP01, Hof04] verwendet.

Bei TSK-Modellen oder Mamdani-Reglern mit konstanten Werten in den Konklusionsteilen der Regeln wird häufig eine Kombination evolutionärer und klassischer Regressionstechniken angewendet. Die Fuzzy-Mengen und die Regelbasis werden mittels evolutionärer Algorithmen optimiert, während die linear parametrisierten Funktionen in den Konklusionsteilen der Regeln mit der üblichen „Kleinste-Quadrate-Methode" angepasst werden [Rus00, WC00].

In den meisten Fällen werden die Regelbasis und die Fuzzy-Mengen separat optimiert und nicht wie in [CFM96], wo alle Parameter gleichzeitig adaptiert werden. Zum Teil werden auch weitere Parameter des Fuzzy-Systems wie Aggregationsoperatoren oder Defuzzifizierungsmethoden betrachtet [RDVZG01].

Verschiedene regelungstechnisch motivierte Fitnessfunktionen, die beispielsweise das Überschwingverhalten des Fuzzy-Reglers bewerten, werden in [Hof97] diskutiert.

Wie bereits im vorhergehenden Abschnitt erwähnt, spielt die Interpretierbarkeit eines durch einen evolutionären Algorithmus erzeugten oder optimierten Fuzzy-Systems

eine wichtige Rolle. Hierzu wurden Ansätze entwickelt, die verschiedene Kriterien berücksichtigen. In [CGP01] wird versucht, die Anzahl der Variablen, die im Prämissenteil einer Fuzzy-Regel verwendet werden, zu minimieren. Kriterien wie die Kompaktheit (geringe Anzahl von Regeln), die Vollständigkeit (Abdeckung aller relevanten Bereiche des Eingaberaums), die Konsistenz (keine Regeln mit nahezu identischen Prämissen und völlig unterschiedlichen Konklusionsteilen) einer Regelbasis sowie Beschränkungen an die Fuzzy-Partition (z.B. an die Überlappung der Fuzzy-Mengen) finden sich in [IMT97, INM01, Jin00, JvSS99, CL00].

In einigen Ansätzen werden Fuzzy-Regeln mittels Techniken der genetischen Programmierung gelernt. Dazu werden die Regeln in Form von Bäumen wie in Abbildung 10.15 kodiert. Der dargestellte Baum entspricht der Regel

```
If X1 is small and X2 is big then Y is large.
```

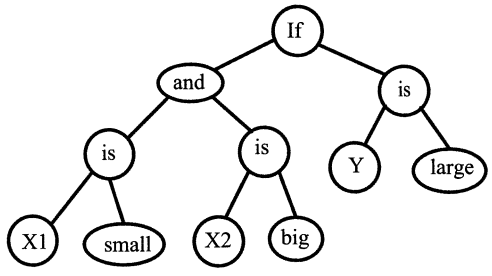


Abbildung 10.15 Darstellung einer Fuzzy-Regel für die genetische Programmierung

Solche Konzepte wurden in [ACT99, Bas94, HN01, CLH02] umgesetzt. In [GS95] werden Sprachgrammatiken und Classifier-Systeme zum Erlernen von Fuzzy-Regeln verbunden.

Die meisten zitierten Ansätze beziehen sich vorwiegend auf das Erlernen oder Optimieren von Fuzzy-Reglern anhand von Simulationen oder im Sinne eines Regressionsproblems bei vorliegenden Daten. Mit Fuzzy-Klassifikation beschäftigen sich [IMT97, INM99, CLH02, Hof04].

10.3 Fuzzy-Clustering mit evolutionären Algorithmen

Im Abschnitt 9.5 wurde das Prinzip der zielfunktionsbasierten Fuzzy-Clusteranalyse vorgestellt, bei der die Zielfunktion mittels eines alternierenden Optimierungsschemas minimiert wurde. Das entsprechende Schema ist auf die Clusteranalyse zugeschnitten und führt die Minimierung der Zielfunktion relativ effizient durch. Da evolutionäre Algorithmen als universelle Optimierungsverfahren nicht in dem Maße spezialisiert und zielgerichtet sind, werden sie die Optimierung kaum schneller bewerkstelligen können. Es bieten sich für die Clusteranalyse zwei Einsatzmöglichkeiten für evolutionäre Algorithmen.

- Auch wenn sie nicht so effizient wie speziell zugeschnittene alternierende Optimierungsstrategien auf ein Minimum zusteuern, tasten sie den Suchraum breiter ab und unterliegen weniger der Gefahr, in einem schlechten lokalen Minimum stehen zu bleiben. Das alternierende Optimierungsschema hängt von der Initialisierung ab und konvergiert bei einer schlechten Initialisierung in einem lokalen Minimum.

- Ein anderer Ansatzpunkt als die Vermeidung lokaler Minima mittels evolutionärer Algorithmen bietet sich, wenn dem alternierenden Optimierungsschema wie beim FCS in einem Schritt keine analytische Lösung für die entsprechenden Parameter zur Verfügung steht. In diesem Fall muss ein numerisches Iterationsverfahren angewendet werden, das sowohl den Rechenaufwand als auch die Gefahr lokaler Minima erhöht.

Wir betrachten zunächst den ersten Punkt, evolutionäre Algorithmen als konkurrierende Techniken zur alternierenden Optimierung zu verwenden. Ansätze in dieser Richtung mit unterschiedlicher Kodierung, die auf die Zielfunktion des HCM und des FCM angewendet wurden, finden sich in [BBHB94, VL95]. Erweiterungen, bei denen gleichzeitig versucht wird, die Anzahl der Cluster zu bestimmen, beschreiben [NMP97] und [PBP+94], wo auch ellipsoide Clusterformen zugelassen werden.

Entscheidend sind aber die Feststellungen in [HOB99], die sich aus Untersuchungen der alternierenden im Vergleich zur evolutionären Optimierung für den HCM und den FCM ergeben haben. Der evolutionäre Ansatz wurde als GGA („genetically guided algorithm") bezeichnet.

- Der GGA hat in keinem Fall ein besseres Optimum als der (mit verschiedenen Initialisierungen gestartete) HCM bzw. FCM.

- Wenn dem GGA eine genügend große Population und hinreichend viele Generationen zugestanden werden, findet er bei einer einigermaßen vernünftigen Wahl der Parameter ein genauso gutes Optimum wie der HCM bzw. FCM.

- Für die meisten Datensätze benötigte der GGA eine um zwei Größenordnungen höhere Rechenzeit als der HCM bzw. FCM.

Das Fazit, das in [HOB99] gezogen wird, besagt, dass sich ein Einsatz evolutionärer Algorithmen für die (Fuzzy-)Clusteranalyse wahrscheinlich nur dann lohnt, wenn der oben besprochene Anwendungsfall vorliegt, bei dem die Clusterprototypen nicht analytisch in der alternierenden Optimierung bestimmt werden können. Die folgenden Untersuchungen zu diesem Thema stammen aus [KK98].

Wir betrachten zunächst den FCS, der nach Clustern in Form von Kreisrändern sucht. Abbildung 10.16 zeigt oben einen Beispieldatensatz mit fünf Kreisen, der beispielsweise aus einem Bild resultieren könnte, auf das ein Kantenextraktionsalgorithmus angewendet wurde. Ebenfalls in der Abbildung sind unten zwei typische Resultate gezeigt, die sich ergeben, wenn man die Zielfunktion des FCS mit einem evolutionären Algorithmus optimiert. Im einen Fall wurde kein Kreis korrekt erkannt, im anderen einer von fünf. Dieses negative Resultat konnte weder durch Variationen der Mutations-

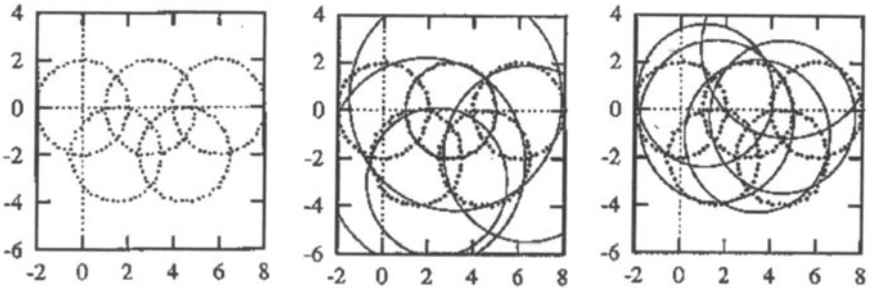


Abbildung 10.16 Schlechte Ergebnisse beim FCS

oder Crossoverraten noch durch andere Techniken wie z.B. Fitnessskalierung verbessert werden.

Verwendet man das klassische, nicht-evolutionäre Verfahren für den FCS, ergibt sich bei einer vernünftigen Initialisierung das korrekte Resultat. Die Zielfunktion des FCS neigt dazu, eher etwas zu große als zu kleine Kreise zu bevorzugen. Deswegen wurde der FCS mit evolutionärer Optimierung noch einmal mit einem sehr einfachen Datensatz mit nur einem Kreis mit Mittelpunkt im Ursprung (0,0) und Radius 2 getestet. In einem Fall wurde der Suchraum für die Koordinaten des Zentrums und den Radius auf das Intervall $[-2.2, 2.2]$ bzw. $[0, 2]$ beschränkt, im anderen auf das wesentlich größere Intervall $[-22, 22]$ bzw. $[0, 22]$. Im ersten Fall konnte der evolutionäre Algorithmus nach ca. 30 Generationen den Kreis mit Mittelpunkt und Radius in allen Testläufen korrekte erkennen. Im zweiten Fall, wo ein größerer Suchraum zugrunde gelegt wurde, konnte der evolutionäre Algorithmus den Kreis nur in ca. 60% der Testläufe nach jeweils ungefähr 125 Generationen korrekt erkennen.

Dieses wenig zufrieden stellende Ergebnis lässt sich erklären, wenn man die Zielfunktion etwas genauer analysiert. Dazu betrachten wir noch einmal den Datensatz, der aus nur einem Kreis besteht mit Mittelpunkt im Koordinatenursprung und Radius 2. Abbildung 10.17 zeigt den Wert der Zielfunktion, wenn das Chromosom einen Kreis mit Radius Mittelpunkt im Punkt $(x, 0)$ $(0 \leq x \leq 22)$ für verschiedene Radiuswerte. Das mittlere und untere Diagramm zeigen lediglich Vergrößerungen des oberen Diagramms im Bereich von $[0, 13]$ bzw. $[0, 7]$.

Der korrekte Radius 2 erhält nur dann eine gute Bewertung, wenn auch die x-Koordinate nahe dem korrekten Wert Null liegt. Die Kurve, die unten links im unteren Diagramm startet, gehört zum Radius zwei. Ab einer x-Koordinate von 2 wird sie bereits von Radien, die sich um einen ganzzahligen Wert vom korrekten Radius unterscheiden, überholt. Diese Situation verschlimmert sich noch deutlich, wenn wir die y-Koordinate ebenfalls variieren. Der zu kleine Radius 1 liefert immer sehr schlechte Werte. Um so weiter der Mittelpunkt des Kreises, den ein Chromosom repräsentiert, vom Koordinatenursprung entfernt ist, desto besser ist es für das Chromosom einen größeren Wert für den Radius anzunehmen. Daher haben in frühen Generationen, wenn die zufällig erzeugten Mittelpunkte nicht in der Nähe des Ursprungs liegen, Chromosomen mit ei-

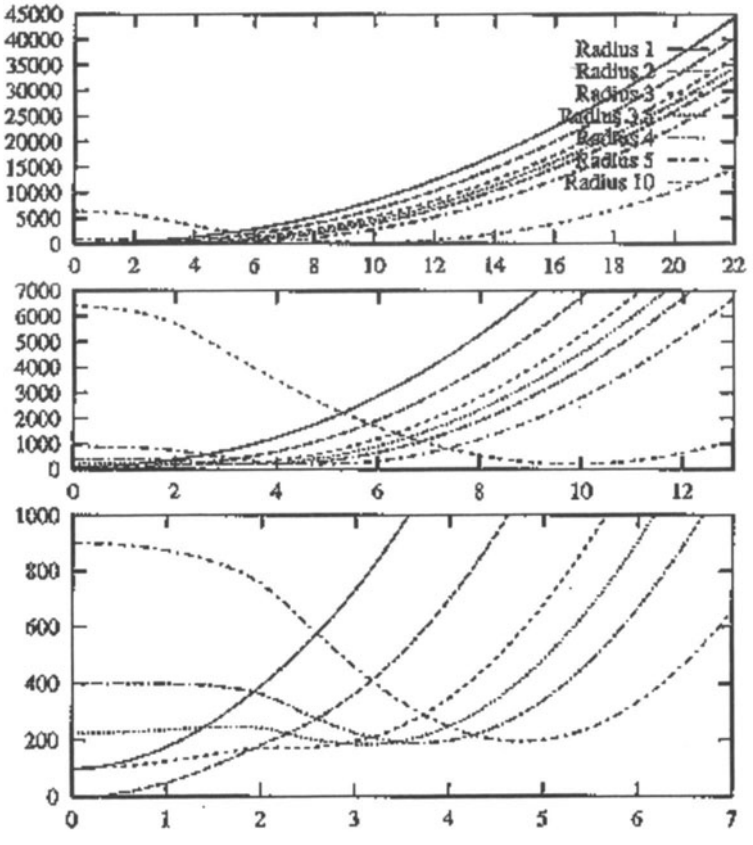


Abbildung 10.17 Probleme der Zielfunktion beim FCS

nem kleinen Radiuswert kaum Überlebenschancen. Es erfordert daher eine relative große zufällige gleichzeitige Mutation des Radius und des Mittelpunktes, um in die Nähe des korrekten Wertes zu gelangen. Die Parameter, d.h. die Gene des Chromosoms weisen bei diesem Problem eine so starke Abhängigkeit auf, dass es für den evolutionären Algorithmus sehr schwierig ist, das globale Minimum der Zielfunktion zu finden. Wenn man den Suchraum für den Radius auf das Intervall $[0, 2.2]$ einschränkt, schaffte es der evolutionäre Algorithmus wieder den Kreis in jedem Testlauf nach ca. 80 Generationen zu erkennen.

Beim FCS und allgemein beim Shell-Clustering – Cluster in Form von geometrischen Konturen – besitzt die Zielfunktion sehr viele lokale Minima. Das globale Minimum entspricht häufig einem sehr spitzen und engem Tal, so dass ein evolutionärer Algorithmus leicht fehlgeleitet werden kann. Für ein anderes Clusteringproblem ließen sich evolutionäre Algorithmen jedoch erfolgreich einsetzen. In einigen Fällen – z.B. wenn Fuzzy-Cluster später als Fuzzy-Regeln interpretiert werden sollen – ist es sinnvoll, nach

Clustern in Form von achsenparallelen Rechtecken, Quadern bzw. Hyperquadern zu suchen. Um derartige Cluster zu finden, verwendet man bei p-dimensionalen Daten als Clusterparameter das Clusterzentrum $z_i \in \mathbb{R}^p$ und eine Diagonalmatrix B_i. Der Abstand eines Datentupels zu dem Cluster i wird definiert als

$$\begin{aligned} d^2(x_k, (z_i, B_i)) &= \| B_i(z_i - x_k) \|_\infty^2 \\ &= \left(\max_{1 \le j \le p} \{ b_{ij} \cdot |z_{ij} - x_{kj}| \} \right)^2, \end{aligned}$$

wobei $\| \cdot \|_\infty$ für die Supremumsnorm steht.

Diese Distanzfunktion ersetzt die quadratische euklidische Distanz beim FCM. Um die triviale Lösung $b_{ij} = 0$ für alle i, j auszuschließen, verlangen wir zusätzlich $\det(B_i) = 1$.

Mit diesem Ansatz konnten in [KK98] gute Clusteringergebnisse erzielt werden, insbesondere um die Cluster später als Fuzzy-Regeln zu interpretieren.

Bei der Optimierung der Zielfunktion beim Fuzzy-Clustering werden – unabhängig welche Art von Clusterformen man voraussetzt – durch einen evolutionären Algorithmus in den meisten Ansätzen nur die Clusterparameter optimiert. Die Zugehörigkeitsgrade werden direkt mittels der für das Fuzzy-Clustering hergeleiteten Gleichung (9.6) berechnet. Zum einen würden Zugehörigkeitsgrade die Anzahl der Parameter enorm erhöhen. Zum anderen sollte man nützliches Wissen, das zur Verfügung steht, auch für einen evolutionären Algorithmus nutzen. Da die optimalen Werte für die Zugehörigkeitsgrade bei festen Clusterparametern bekannt sind, sollte man diese Information nicht ignorieren und es dem evolutionären Algorithmus nicht unnötig schwer machen. Das Gleiche gilt auch für den Fall des harten Clustering. Hier wird jedes Datentupel dem Cluster zugeordnet, zu dem es den geringsten Abstand hat.

Fuzzy-Clustering gilt allerdings als robuster, da die Zielfunktion im Fuzzy-Fall anscheinend wenige lokale Minima aufweist als im Fall scharfer Zugehörigkeitsgrade. Auch wenn diese Aussage nicht generell gilt, so kann es in manchen Fällen bei der Optimierung günstiger sein, ein Problem, das eigentlich binäre Parameter hat, d.h. jeweils die Werte Null und Eins, künstlich zu „fuzzifizieren", also in der Zielfunktion Werte zwischen Null und Eins zuzulassen. Ein positiver Effekt zeigt sich nicht nur beim Fuzzy-Clustering, sondern zum Beispiel auch bei der Lösung von logischen Erfüllbarkeitsproblemen. Ein entsprechender Ansatz, der Erfüllbarkeitsprobleme auf diese Weise in ein Optimierungsproblem mit reellen Parametern umkodiert und für die Optimierung evolutionäre Algorithmen verwendet, wird in [PSS02] beschrieben.

10.4 Steuerung evolutionärer Algorithmen mit Fuzzy-Regeln

Bisher haben wir den Einsatz von evolutionären Algorithmen zum Erlernen, Optimieren oder zur Feinabstimmung von Fuzzy-Systemen betrachtet. In diesem Abschnitt soll es umgekehrt darum gehen, Fuzzy-Regeln dafür einzusetzen, um das Verhalten evolutionärer Algorithmen zu verbessern.

Die Grundidee besteht dabei darin, Parameter wie die Mutations- oder die Crossoverrate während der Evolution zu adaptieren. In [HLHB00] werden beispielsweise Regeln der Form

> Wenn die Population nahe der optimalen Lösung ist,
> wende häufiger Crossover an.

bzw.

> Wenn die Population noch weit von der optimalen Lösung entfernt ist,
> wende Crossover seltener an.

verwendet. Dabei wird die Entfernung zur optimalen Lösung über ein heuristisches Verfahren geschätzt.

Soll allgemein das Konzept der Fuzzy-Regeln zur Steuerung eines evolutionären Algorithmus eingesetzt werden, so sind folgende Schritte durchzuführen [HL96]:

1. Es müssen die Eingangs- und die Ausgangsgrößen definiert werden. Die Eingaben sollten das Verhalten und den aktuellen Erfolg des evolutionären Algorithmus beinhalten. Vorschläge für mögliche Eingaben sind beste, durchschnittliche und minimale Fitness sowie Maße für die Diversität einer Population [Tet95]. Außerdem können die aktuellen Steuerungsparameter des evolutionären Algorithmus ebenfalls bei den Eingaben berücksichtigt werden [LT93a, XVII94].

   Die Ausgangsgrößen des Fuzzy-Reglers sind Parameter, die den evolutionären Algorithmus steuern. Dazu gehören unter Anderem Mutationsrate und -art, Crossoverrate und -art, Populationsgröße sowie Selektionsdruck.

2. Für jede Eingangsgröße müssen geeignete Fuzzy-Partitionen, d.h. Fuzzy-Mengen, definiert werden.

3. Eine Regelbasis muss spezifiziert werden.

In [XVII94] sollen die Fuzzy-Regeln dafür sorgen, die Probleme der vorzeitigen Konvergenz und eines zu langsamen Fortschreitens des evolutionären Algorithmus zu lösen. Die Steuerung von Crossoverparametern und des Selektionsdrucks mit Fuzzy-Regeln werden in [HL96] beschrieben. In [Vog96] entscheiden Fuzzy-Regeln in jeder Generation darüber, wie bei einer Multikriteriennoptimierung die einzelnen Fitnessfunktionen zu einer Gesamtfitnessfunktion zusammengesetzt werden. Eine Erweiterung dieses Verfahrens in [Vog98] versucht, die pareto-optimalen Lösungen möglichst gleichmäßig abzudecken.

Grundsätzlich können Adaptionsmechanismen für evolutionäre Algorithmen auf drei Ebenen verwendet werden [Ang95, SF97]:

- Auf der Populationsebene werden Parameter adaptiert, die sich auf die gesamte Population beziehen. Beispiele sind die allgemeine Mutations- oder Crossoverrate.

- Auf der Individuen- oder Chromosomenebene werden Parameter für einzelne Chromosomen eingestellt, z.B. eine individuelle Crossoverwahrscheinlichkeit für jedes Chromosom.

- Auf der Komponenten- oder Genebene werden Parameter wie Mutationswahrscheinlichkeiten oder - schrittweiten individuell für jedes Gen adaptiert.

Die meisten adaptiven Ansätze, die auf Fuzzy-Regeln basieren, arbeiten auf der Populationsebene. Die Fuzzy-Regeln können entweder vorher allgemein festgelegt werden oder wie in [LT93a, LT94] während des eigentlichen evolutionären Algorithmus durch einen weiteren evolutionären Algorithmus selbst angepasst werden, eine Art Meta-GA-Strategie wie sie auch Grefenstette [Gre86] verwendet hat. Der Meta-GA bestimmt mit Testfunktionen, welche Fuzzy-Regelbasis für den eigentlichen evolutionären Algorithmus verwendet werden soll.

[HL01] beschreibt einen alternativen, co-evolutionären Ansatz. Dabei werden die Fuzzy-Regler während des eigentlichen Evolutionsprozesses mit optimiert. Man verwaltet dazu zwei Populationen. Die eine Population enthält die Chromosomen zur Lösung des eigentlichen Problems. Die andere Population besteht aus einer Menge von Fuzzy-Reglern. Die zweite Population sollte deutlich kleiner sein als die erste. Die Fuzzy-Regler steuern beispielsweise Mutation und Crossover. Jeder Fuzzy-Regler wird einer festen Anzahl von Paaren aus der ersten Population zugeordnet. Auf diese Paare wird Mutation und Crossover angewendet, so wie es der entsprechende Fuzzy-Regler vorschlägt. Nachdem die Chromosomen der ersten Population bewertet wurden, wird daraus die Nützlichkeit der einzelnen Fuzzy-Regler bewertet, beispielsweise die durchschnittliche oder beste Fitness, die mit ihm erzeugt wurde.

Auf diese Weise wird über mehrere Generationen der Population fortgefahren. Nach jeweils einer festen Anzahl von Generationen werden für jeden Fuzzy-Regler die Bewertungen der Nützlichkeit über die vergangenen Generationen zu einem Fitnesswert aufaddiert. Damit lßst sich eine neue Generation von Fuzzy-Reglern erzeugen, die wieder zufällig Chromosomen der ersten Population zugeordnet werden und deren Nützlichkeiten über einige Generationen der ersten Population bewertet werden, um dann wiederum neue Fuzzy-Regler zu erzeugen.

Der Algorithmus ist schematisch auf der folgenden Seite dargestellt. Dabei bezeichnet $P(t)$ die t-te Generation der Population mit den Chromosomen für das eigentliche Optimierungsproblem und $P_F(t_F)$ die t_F-te Generation der Population der Fuzzy-Regler.

```
Procedure FGA
    begin
        t := 0, t_F := 0
        Initialisiere P(t) und P_F(t_F)
        Bewerte P(t)
        while ( nicht Abbruchkriterium ) do
        begin
            FOR (τ = 0; τ < τ_max; τ++)
                t := t + 1
                Selektiere P(t) aus P(t − 1)
                Ordne Chromosomen aus P_F(t_F)
                zufällig Chromosomen bzw. Paaren
                von Chromosomen aus P(t) zu
                Berechne, welche genetischen Operatoren
                gemäß den zugeordneten Fuzzy-Regeln
                auf die (Paare von) Chromosomen angewendet
                werden sollen, und wende diese Operatoren an
                Bewerte P(t)
                Sammle die Nützlichkeitswerte für die
                Chromosomen aus P_F(t_F)
            Bewerte P_F(t) gemäß den gesammelten Bewertungen
            t_F := t_F + 1
            Selektiere P_F(t_F) aus P_F(t_F − 1)
            Wende Crossover und Mutation auf P_F(t_F) an
        end
    end
```

A Anhang: Biologische Evolution und evolutionäre Algorithmen

Der Ursprung der Evolutionären Algorithmen liegt in der Theorie der biologischen Evolution, wie sie von Charles Darwin (1809 - 1882) und Alfred Wallace (1823 - 1913) vor knapp 150 Jahren zum ersten Mal vorgestellt wurde. Die dahinter stehende Idee knüpft an die Fähigkeit in der Natur an, sich durch Adaption an eine verändernde Umwelt anzupassen. Sie ermöglicht es Lebewesen, durch Innovationen die vorhandenen Ressourcen ihrer Umgebung besser zu nutzen und sich zu reproduzieren, wobei sie ihre Fähigkeiten zum Überleben an ihre Nachkommen weitergeben. Um den Grundgedanken evolutionärer Algorithmen zu verstehen ist es notwendig, sich mit ihrer biologischen Bedeutung vertraut zu machen. Im folgenden Abschnitt werden deshalb die wichtigsten Grundlagen der Genetik erläutert und den entsprechenden Begriffen der „künstlichen" Evolution gegenübergestellt.

A.1 Die biologische Evolution

Die bekannten Mechanismen der biologischen Evolution sind die natürliche Selektion, Rekombination und Mutation, Populationswellen sowie unvorhersehbare Zufallsereignisse. Unter Selektion wird allgemein die „natürliche Auslese" verstanden, die es nur den erfolgreichsten Lebewesen gestattet, zu überleben und sich zu vermehren. Obwohl die Begriffe „Evolution" und „surviving of the fittest" erst viel später von Herbert Spencer (1820 - 1903) eingeführt wurden, prägten Darwin und Wallace 1858 ihren Grundgedanken, der im Laufe der folgenden Jahrzehnte das bis dahin gültige biologische Weltbild bis in seine Grundfesten erschüttern sollte. Darwin und Wallace stützten ihre Ideen auf drei Beobachtungen und leiteten daraus mehrere Annahmen ab: Die Natur produziert einen potentiellen Überschuss an Lebewesen, doch die Populationsgrößen bleiben dabei trotzdem relativ konstant. Daraus folgerten sie, dass ein Großteil der Lebewesen stirbt, bevor er sich fortpflanzen kann. Sie entdeckten einen Mechanismus, der später unter dem Begriff „Selektionsdruck" bekannt wurde: Es gibt immer bestimmte Individuen einer Art, die aufgrund ihrer oftmals nur sehr geringfügigen genetischen Variation besser an ihre Umwelt angepasst sind als andere Individuen der gleichen Art. Diese genetisch unterschiedlichen Individuen besitzen eine größere Überlebenswahrscheinlichkeit, da sie sich an Faktoren wie Klima, Nahrungsangebot oder Feinde besser anpassen können. Allerdings wird der Begriff „Anpassung" heute teilweise anders definiert: Nicht der Druck von äußeren Einflüssen ist der Motor der genetischen Variationen, sondern der Schutz vor Parasiten und Krankheiten fordert die Anpassung heraus. Für unsere weiteren Betrachtungen soll uns diese Unterscheidung jedoch nur am Rande interessieren.

A.1.1 Zellen und Genetik

Der zentrale Punkt bei der Vererbung und der Evolution ist die Molekular- und Evolutionsgenetik, die auf den Zellen, den Grundbausteinen des Lebens, basiert. Zellen werden in der Biologie als kleinste noch selbständige Funktionseinheit alles Lebenden definiert. Das bedeutet, dass sie für die Erzeugung von Nachkommen (Reproduktion) keine weiteren lebenden Substanzen benötigen. Man unterscheidet zwei Arten von Zellen: Zum einen die prokaryotischen Zellen, die keinen Zellkern (Nukleus) besitzen und zum anderen eukaryotische Zellen, die mit einem Zellkern ausgestattet sind und die überwiegende Mehrheit aller vorkommenden Zellen bilden. Der Zellkern enthält die Chromosomen als den eigentlichen Träger der Erbsubstanz. In ihnen sind der Bauplan und bestimmte Eigenschaften eines Lebewesens festgelegt. Erstaunlicherweise hat die Anzahl der Chromosomen jedoch nichts mit der Komplexität des Lebewesens zu tun. So besitzt der Mensch 46, Schimpansen und Kartoffeln 48 und ein Karpfen sogar 104 Chromosomen. Die Hausmaus hingegen besitzt 40 und Mais sogar nur 20 Chromosomen. Chromosomen sind die Träger des eigentlichen Erbgutes, den Genen. Bei ihnen handelt es sich um einzelne Teilstücke eines Chromosoms, die eine Teileigenschaft wie beispielsweise die Augenfarbe festlegen. Der Ort, an dem sich ein entsprechendes Gen auf einem Chromosom befindet, wird Locus genannt. Gene können nicht beliebige Zustände annehmen, sondern besitzen festgelegte Ausprägungen, die Allele. So haben die meisten Menschen eine braune, grüne oder blaue Augenfarbe, da kein Allel für gelbe oder violette Farbpigmente existiert. In den meisten Fällen wirken allerdings mehrere Gene bei der Ausbildung eines Merkmales zusammen, so dass von einer Polygenie der Phäne gesprochen wird. Es wird vermutet, dass beispielsweise die Hautfarbe von zehn Genen gesteuert wird. Bei der Polyphänie der Gene hingegen kann ein Gen auch mehrere Merkmale des Lebewesens beeinflussen. Die Gesamtheit aller Gene einer Population wird als Genom bezeichnet, der Genpool ist die Gesamtzahl aller Allele des Genoms. Jeder Zellkern enthält die Chromosomen in zweifacher Ausführung, die sie jeweils einmal vom Vater und von der Mutter geerbt haben. Die Gene dieser diploiden Chromosomen werden als homozygot bezeichnet, wenn zwei entsprechende Allele, beispielsweise für die Augenfarbe, dieselbe Ausprägung besitzen. Haben Vater und Mutter ihrem Nachwuchs hingegen unterschiedliche Augenfarben vererbt, so spricht man von heterozygoten Allelen. Das Ergebnis ist allerdings nicht immer eine Mischung, sondern wird mit Hilfe einer Art Rangordnung geregelt: Die Natur unterscheidet rezessive und dominante Ausprägungen. Treffen ein dominantes Allel wie die braune Augenfarbe und ein rezessives Allel für blaue Augenfarbe zusammen, wird der Nachwuchs braune Augen haben. Darüberhinaus gibt es aber auch intermediäre Erbgänge. Bei ihnen steuert auch das rezessive Allel einen Teil zur Merkmalsausprägung bei.

A.1.2 Aufbau der Gene

Die Gene selbst bestehen aus Nukleinsäuren und Proteinen. Die wichtigste Nukleinsäure unseres Erbmaterials ist die Desoxyribonukleinsäure, abgekürzt DNS. Sie trägt die erbrelevanten Informationen und wird deshalb auch als „Bauplan des Lebens“ bezeichnet. Die DNS setzt sich aus den 4 Basen Adenin, Guanin, Cytosin und Thymin zusammen.

Diese Basen stellen eine Art Alphabet dar, mit denen die Chromosomen in bestimmte Abschnitte unterteilt werden. Dabei handelt es sich um eine Dreiercodierung (Tripletts) der 4 Basen, so dass $4^3 = 64$ Worte daraus gebildet werden können. 61 von ihnen werden zur Beschreibung von Aminosäuren, die drei übrigen als Steuerzeichen verwendet. In einem DNS-Molekül mit n Nukleotiden sind dementsprechend 4^n verschiedene Sequenzen möglich. Bei einem mittleren Molekulargewicht eines Gens von 10^6 sind 4^{1500} verschiedene Gene möglich, alle jeweils aus den vier genetischen Buchstaben aufgebaut. Das reicht aus, um allen existierenden Lebewesen, einschließlich jedes einzelnen Bakteriums, individuelle Eigenschaften zu verleihen. Der Gen-Code ist in seinem Aufbau hochgradig redundant, da jede Aminosäure durch mehrere unterschiedliche Tripletts beschrieben werden kann. Der genetische Code ist die Basis allen Lebens auf der Erde. Dies beweist die Tatsache, dass alle Organismen den Code auf dieselbe Weise übersetzen.

Bei den höher entwickelten Lebewesen, bei denen sich die Gene innerhalb des Zellkerns befinden, werden die Erbinformationen in Exons und Introns unterteilt. Bei den Exons handelt es sich um die Informationen, die für den Aufbau des Organismus auch tatsächlich verwendet werden. Die Introns hingegen besitzen keinen nachweisbaren Einfluss auf das jeweilige Lebewesen, obwohl sie manchmal über die Hälfte des genetischen Codes ausmachen. Bei der Zellteilung werden Exons und Introns zunächst gemeinsam kopiert, doch anschließend werden die Introns wieder herausgeschnitten und ignoriert. Eine Theorie besagt, dass es sich bei den Introns um Fähigkeiten oder Eigenschaften handelt, die ein Lebewesen früher einmal besessen hatte. Im Laufe der Zeit jedoch, in der sich die Umweltbedingungen änderten, wurden diese nicht mehr benötigt und vergessen. Kommt es wiederum zu einer Änderung der Umgebung, können sich die Organismen mit dem „versteckten" Wissen möglicherweise schneller und effektiver diesen neuen Umständen anpassen. Ähnliches wurde bei den Allelen beobachtet: Es gibt Ausprägungen von bestimmten Genen, die von den Lebewesen allerdings nie verwendet werden. Ändern sich die äußeren Umstände, werden sie plötzlich wichtig und verhelfen dem Organismus zum entscheidenden Vorsprung gegenüber Artgenossen und Konkurrenten.

A.1.3 Chromosomensätze und Kopierfehler

Die Gesamtzahl der Chromosomen eines Lebewesens wird Chromosomensatz genannt und ist bei jeder Art unterschiedlich. Der Mensch besitzt 46 Chromosomen, die sich normalerweise als eine Art Knäuel im Zellkern befindet. Kommt es zur Zellteilung und damit zu einer Kopie der Chromosomen, werden diese zunächst zu einer wesentlich kürzeren und stabileren Doppelhelix verdrillt. Sie gewährleistet eine möglichst exakte Aufteilung der Chromosomen bei der Zellteilung und verringert die Gefahr eines unkontrollierten Auseinanderreißens. Trotzdem kann es bei der Zellteilung zu Fehlern kommen. Bei der Verdopplung der Erbinformationen entstehen manchmal „Kopierfehler", die Chromosomenaberration genannt werden. Es handelt sich dabei um Mutationen und Crossing-over. Die Ursachen dafür sind beispielsweise radioaktive Strahlung, UV-Licht oder Chemikalien, die einen Einfluss auf die Zellteilung ausüben. Es gibt aber auch Untersuchungen, bei denen Mutator-Gene gefunden wurden, die solche Mutationen gezielt

auslösen können. Bei den Mutationen werden verschiedene Chromosomenaberationen unterschieden. Bei der Deletion wird ein Teil des Chromosoms zerstört oder geht anderweitig verloren. Dieser Verlust endet für das betroffene Lebewesen meist tödlich. Tauschen hingegen zwei oder mehr Gene ihre Plätze, nennt man den Vorgang Translokation. Sind nach der Zellteilung Gene auf einem Chromosom doppelt vorhanden, so handelt es sich um eine Duplikation. Oftmals fehlt das entsprechende Gen dafür auf einem anderen Chromosom. Geht nur das Endstück eines Chromosoms verloren, handelt es sich um Defizienz. Bei der Inversion hingegen geht keine Information verloren, sondern die Reihenfolge der Chromosomen wird teilweise vertauscht. Auch eine Veränderung der Chromosomenzahl (Genommutation) kann bei der Zellteilung vorkommen, die besonders beim Menschen oft mit negativen Folgen verbunden ist. Hat ein Mensch 47 anstatt 46 Chromosomen, so handelt es sich um eine Trisomie, bei der ein Chromosom nicht zweimal, sondern dreimal vorhanden ist. Am häufigsten passiert dies beim Menschen mit dem 21. Chromosom. Die Krankheit wird deshalb auch Trisomie 21 genannt, besser bekannt unter den Namen Down Syndrom oder Mongolismus. Bei der Genmutation kommt es zu einer Veränderung innerhalb der Basensequenzen eines DNS-Stranges, manchmal sogar nur innerhalb eines einzelnen Basenpaares. Die meisten Mutationen an Allelen werden jedoch zunächst noch nicht bemerkt, da sie meistens eine rezessive Ausprägung besitzen. Tritt eine bestimmte Mutation innerhalb einer Population jedoch häufiger auf, so kann sie dominant werden. Bei einer Mutation können auch einzelne Moleküle innerhalb des DNS-Stranges durch andere, sehr ähnliche Moleküle ausgetauscht werden, wodurch Informationen verändert oder gar ganz gelöscht werden. Eine der schlimmsten Formen ist die Schubmutation. Bei ihr werden einzelne Basen aus der DNS entfernt oder eingefügt. Dadurch kommt es zu einer Reihenfolgenänderung innerhalb der Triplets und damit auch innerhalb des gesamten DNS-Stranges. Er wird damit als ganzes unbrauchbar. Nicht jede Mutation ist auch nach außen sichtbar. Im Gegenteil, die meisten Veränderungen wirken sich nur genotypisch, also ausschließlich auf den Aufbau der Chromosomen, aus. Nur ein kleiner Teil der Veränderungen ist phänotypisch und damit äußerlich an einem Lebewesen zu erkennen.

Der Einfluss der Aberration auf die Evolution ist heute noch umstritten, da die Folgen für das einzelne Lebewesen oftmals sehr gravierend sind. Auf der einen Seite haben sie damit einen eher hemmenden als fördernden Einfluss auf die Weiterentwicklung. Andererseits ist der Informationsgehalt der DNS mit seiner Länge korreliert, so dass sich eine Translokation und eine Chromosomenverlängerung durchaus positiv für ein Lebewesen auswirken können. Denn liegen Gene doppelt vor, kann die Natur mit der einen Hälfte experimentieren, ohne das die ursprüngliche Erbinformation verloren geht.

Doch nicht jede Zellteilung gibt der Natur die Möglichkeit, am genetischen Bauplan eines Lebewesens Verbesserungen durchzuführen. Bei der ungeschlechtlichen Zellteilung (Mitose) kommt es zu einer erbgleichen Zell- und Kernteilung. Die Chromosomen werden bei diesem Vorgang der Länge nach in zwei Chromatiden aufgespalten und die jeweils fehlende Hälfte mit Hilfe der DNS-Polymerase wieder ergänzt. Bei der geschlechtlichen Zellteilung (Meiose) hingegen kommt es zu einer Mischung von zwei unterschiedlichen Erbmaterialien, bei der es zu Rekombinationen, dem sogenannten Crossing-over, mit neuen Eigenschaften kommen kann. Thomas Morgen (1866 - 1945) entdeckte bei Kreuzungsversuchen mit der Fruchtfliege Drosophila, dass sich während der Meiose die

Chromosomen bei der Reife paaren und umeinander schlingen. Dabei kann es an einer oder mehreren Stellen zu einem Bruch von Nicht-Schwesterchromatiden kommen, deren Bruchstücke sich anschließend über Kreuz verknüpfen. Anschließend werden die auf den abgetrennten Teilstücken liegenden Gene gegeneinander ausgetauscht. Dieser als Crossing-over bezeichnete Tauschvorgang kommt nur an bestimmten Sollbruchstellen (Chromomaren) vor, die sich bei der Spiralisierung herausbilden. Damit stellt die Natur sicher, dass bei diesem auch Kopplungsbruch genannten Vorgang nur sehr selten lebensunfähige Kombinationen entstehen. Als Austauschwert wird der Prozentsatz bezeichnet, der die Häufigkeit einer Überkreuzung an einer bestimmten Stelle beschreibt. Durch Versuche konnte gezeigt werden, dass bestimmte Merkmalskombinationen immer zusammen auftreten und bei einem Crossing-over nicht getrennt werden. Daraus kann man schließen, dass diese immer zusammen auftretenden Merkmale auf einem Gen liegen. Beim Crossing-over handelt es sich also nicht um einen spontanen Unfall mit ausschließlich negativen Folgen für das betroffene Lebewesen, sondern um ein von der Evolution bewusst eingesetztes Mittel, um Anpassungen und Entwicklung überhaupt zu ermöglichen oder einfach zu beschleunigen. Anders hingegen sieht es bei der Mutation aus, bei der es sich um eine spontane und ungerichtete Veränderung der Gene handelt. Damit besitzt die Natur einen Mechanismus für die Zuchtwahl (Selektion), deren Nützlichkeit für die Weiterentwicklung von Lebewesen jedoch umstritten ist. Allerdings verhindert die Mutation das Festfahren in eine bestimmte Entwicklungsrichtung, wenn ein lokales Optimum der Anpassung erreicht wurde. Sie bringt von Zeit zu Zeit neue Allele hervor, so dass bereits vorhandene Fähigkeiten eines Lebewesens ergänzt und damit seine Entwicklung in eine neue Richtung geführt werden kann. Die Möglichkeit, ein absolutes oder globales Optimum zu erreichen, gibt es in der Natur allerdings nicht.

A.1.4 Anpassungsgeschwindigkeit an Umweltbedingungen

Ein weiteres Kriterium für die Geschwindigkeit, mit der sich Lebewesen ihrer Umgebung anpassen können, ist die Mutationshäufigkeit. Sie ist bei den unterschiedlichen Lebewesen nicht gleich, sondern schwankt innerhalb eines Bereiches von mehreren Zehnerpotenzen. Bei höheren Lebewesen wurde eine Mutation pro Zellteilung auf 10^5 bis 10^9 Gene beobachtet, bei niederen Lebewesen ist sie für ein einzelnes Gen geringer. Einen Mittelwert zu nennen ist allerdings sehr schwer, da die Mutationsrate auch für einzelne Gene unterschiedlich ist. Durch die große Anzahl an Genen enthalten allerdings bis zu 40 % aller menschlichen Keimzellen eine Mutation, die sich aber nur in den seltensten Fällen phänotypisch auswirkt. In ganz seltenen Fällen kann es auch zu einer Rückmutation kommen. Dann wird eine durch eine vorhergehende Mutation veränderte DNS durch eine erneute Veränderung des Erbgutes wieder in ihren alten Zustand zurückversetzt. Häufiger dagegen tritt eine Supressormutation ein: Durch sie wird eine vorhergehende Mutation an einem anderen Ort teilweise oder ganz ausgeglichen.

Das Erbgut ist den zufälligen Mutationen durch äußere Einflüsse jedoch nicht vollkommen ausgeliefert, sondern besitzt ein eigenes Reparatursystem. Mit diesem ist es möglich, zumindest kleinere Schäden zu beseitigen und die Informationen im Erbgut wieder herzustellen. Die Natur hat zwei Möglichkeiten entwickelt, diese Reparaturen

durchzuführen: Photoreparatur wird das Verfahren genannt, bei dem UV-Licht im Bereich von 3100 bis 4400 Ångström die Bildung von Enzymen bewirkt, die wiederum das Reparatursystem steuern. Das Gegenstück ist die Dunkelreparatur, die ohne Licht auskommt. Auch hier steuern Enzyme die Behebung von Schäden an den Genen. Dazu werden die defekten Stränge herausgeschnitten und mit Hilfe der DNS-Polymerase und den Informationen vom zweiten DNS-Strang ergänzt. Anschließend verknüpft das Enzym Ligase den neu erstellten Strang mit der ursprünglichen DNS. Dieses Verfahren funktioniert selbstverständlich nur so lange, wie die genetischen Informationen zumindest auf einem der beiden DNS-Stränge noch vorhanden ist. Werden beide Stränge beschädigt, so ist die auf ihnen liegende Information für diese Zelle verloren.

Im Gegensatz zum Selektionsdruck, bei dem die natürliche Auslese untaugliche Mutationen aus einer Population entfernt, gibt es auch einen Mutationsdruck, der die Summe aller in einer Population auftretenden Mutationen beschreibt. Je höher sich eine Population entwickelt hat, desto schwieriger wird es für Mutationen, sich durchzusetzen, da sie immer öfter negative Folgen für die Individuen haben werden.

A.1.5 Zufallseinflüsse

Trotz aller Regelmäßigkeiten, die bei der Entwicklung von Lebewesen beobachtet werden, spielt der Zufall doch die bedeutendste Rolle. Ob es sich um Veränderungen innerhalb des Erbgutes handelt oder um sich ändernde Umwelteinflüsse, die schließlich die Art und Intensität des Selektionsdruckes ausmachen - statistische Ereignisse lenken die Entwicklung immer wieder in nicht vorhersehbare Richtungen. Betrachten wir beispielsweise eine frisch geborene Maus in der freien Natur. Der Zufall wollte es, dass sie durch eine bis dahin nicht vorgekommene Mutation ein verändertes Enzym produziert, durch das ihr Alterungsprozess verlangsamt wird. Die Vorteile, die die Maus dadurch hat, werden jedoch erst im Laufe ihres Lebens relevant, da sie durch ihr höheres Alter mehr Nachkommen zeugen kann als es ihren Artgenossinnen möglich ist. Doch bis dahin kann sie bereits als Jungtier durch Zufall an einer Krankheit gestorben oder verhungert sein, einem Predator zum Opfer gefallen oder von ihren Artgenossen getrennt worden sein. Alle diese zufälligen Ereignisse würden einer Mutation, die sich zum Vorteil einer ganzen Art erweisen könnte, jede Chance auf eine Verbreitung nehmen. Auch die Auswahl der Crossing-over Punkte zwischen zwei homogenen Chromosomen wird wahrscheinlich ebenso vom Zufall bestimmt sowie die Aufteilung der väterlichen und mütterlichen Chromosomen auf die vier haploiden Geschlechtszellen bei der Fortpflanzung.

A.1.6 Das Ziel der Optimierung

Eine zentrale Frage, die auch von Biologen noch nicht endgültig beantwortet wurde, ist, was bei der Evolution eigentlich optimiert wird. Zum einen könnten es die Gene und Chromosomen an sich sein, zum anderen aber die Individuen. In den letzten Jahren deuten jedoch die Untersuchungen vermehrt darauf, dass im Laufe der Evolution eine Art als ganzes beziehungsweise ihr Genpool verbessert wird. Damit steht also nicht das Individuum im Mittelpunkt des Interesses, sondern die gesamte Population, der das einzelne

Lebewesen angehört. In der Kalahari-Wüste im Südwesten Afrikas beispielsweise leben Geparden, die bis zu 110 km/h schnell werden können und sich dort überwiegend von Springböcken ernähren. Die Landschaft hat nur eine sehr spärliche Vegetation, so dass sie den Jägern nur wenig Deckung und den Gejagten kaum Versteckmöglichkeiten bietet. Die Geparden gleichen diese Erschwernis durch eine kurzzeitige hohe Geschwindigkeit aus, mit der sie die Distanz zwischen sich und ihrer potentiellen Nahrung überbrücken. Die geringe Vegetationsdichte erlaubt es ihnen, mit einer so großen Geschwindigkeit zu laufen. Springböcke sind zwar im Laufe der Evolution nicht so schnell geworden, besitzen dafür aber eine größere Ausdauer und Wendigkeit, die es ihnen ermöglicht, ihren Jägern meistens zu entkommen. So werden in der Regel nur ältere und kranke Individuen des südafrikanischen Wappentiers vom schnellsten Landsäugetier der Erde geschlagen.

A.1.7 Genetische Drift und Varianz

Die Auftrittshäufigkeit eines Allels in einer Population wird Allelfrequenz oder auch Allelenspektrum genannt. Verändert sich das Allelenspektrum einer abgetrennten Population, spricht man von einer genetischen Drift. Sie stellt eine auf dem Zufall beruhende signifikante Veränderung der Allelfrequenz dar, die unabhängig von Umwelteinflüssen voranschreitet. Über die Änderung dieser Frequenz im Lauf der Zeit wird versucht, die Evolution einer Art nachzuvollziehen. Dabei spielen schwer bestimmbare Randbedingungen wie der Einfluss des Nahrungsangebotes auf die Populationsgröße eine so entscheidende Rolle, dass sich deren mathematische Modellierung allerdings sehr kompliziert gestaltet. Insbesondere kleine Populationen haben unter der Gendrift zu leiden. Bei ihnen kommen nur wenige Individuen zur Fortpflanzung, so dass die Wahrscheinlichkeit einer durchgängigen Rekombination der Gene im Genpool sinkt. Dadurch wird nur ein geringer Teil des Genpools durch die Natur phänotypisch umgesetzt und ausgetestet. Je kleiner die betroffene Population ist, desto geringer wird deren genetische Varianz, was eine Steigerung der Gleichförmigkeit zur Folge hat. Das Ergebnis davon sind Inzuchteffekte wie Inzuchtdegradation. Wird eine Population durch äußere Einflüsse in zwei getrennte Gruppen geteilt, kommt es durch einen Genverlust zu unterschiedlichen Entwicklungen. Darüber hinaus kann es durch unterschiedliche Umwelteinflüsse zu einer Änderung der Selektion kommen. Große Populationen hingegen tendieren meistens zur Ungleichförmigkeit. Starker Selektionsdruck, der eine vereinheitlichende Wirkung hat, kann aber auch hier eine Gleichförmigkeit bewirken. Die Auswirkungen von Selektion, zufälligen Umwelteinflüssen und Gendrift sind bei kleinen Populationen allgemein gravierender als bei größeren. Ändern sich die Umweltbedingungen wie Nahrungsangebot oder Temperatur, existieren in kleinen gleichförmigen Populationen keine Individuen, die sich durch einen etwas anderen genetischen Aufbau schnell genug anpassen können und damit ein Überleben der Art garantieren. Je perfekter sich Lebewesen an ihre Umwelt angepasst haben, desto schwieriger wird es für sie, auf geänderte Umweltbedingungen zu reagieren. Das Maß für diese Nicht-Spezialisierung bzw. Nicht-Anpassung wird genetische Reaktionsnorm genannt. Arten, die nur eine kurze Generationszeit haben und gleichzeitig in großen Populationen auftreten, kommen am schnellsten mit sich ändernden Umwelteinflüssen zurecht. So können sich Insekten innerhalb kürzester Zeit

anpassen, da sie einem hohen Selektionsdruck ausgesetzt sind und eine breite Varianz an Ausprägungen besitzen. Seltene Allele, die später einmal wichtig werden könnten, gehen in einer kleinen Population schnell verloren. In Großen hingegen wird es immer ein paar Individuen geben, die diese zur Zeit vielleicht noch überflüssigen Ausprägungen besitzen und an nachfolgende Generationen weitergeben können. Sie erlauben es den Nachkommen vielleicht einmal, auf neue Umweltbedingungen geschickter zu reagieren oder sich in neue Gebiete, die aufgrund ihrer Umwelt bisher für diese Art verschlossen waren, auszubreiten. Ändern sich die Umweltbedingungen innerhalb eines sehr kurzen Zeitraumes jedoch radikal, können sich nur ganz wenige Arten den neuen Bedingungen anpassen. Es kommt zu einem Faunen- bzw. Florenschnitt, bei dem sehr viele Arten gleichzeitig aussterben, wie es auf der Erde am Ende der Dinosaurierzeit vor 65 Millionen Jahren passierte. Doch diese katastrophalen Ereignisse sind glücklicher Weise eine relativ seltene Ausnahme, so dass Lebewesen in der Regel mehr Zeit zur Verfügung haben, sich ihrer Umwelt anzupassen. Bei der Anpassungsgeschwindigkeit wird heute zwischen zwei Theorien unterschieden: Die These des Gradualismus besagt, dass die Veränderungen der Lebewesen langsam aber kontinuierlich voranschreiten. Die Anhänger des Saltationismus hingegen behaupten, dass Veränderungen durch größere Änderungen im Erbgut sprunghaft hervorgerufen werden. Am wahrscheinlichsten jedoch ist, dass im Laufe der erdgeschichtlichen Entwicklung beide Arten mehr oder weniger häufig vorkamen.

A.1.8 Natürliche Schwankungen von Populationen

Es gibt aber auch natürliche Schwankungen, die die Populationsgröße in periodischen Abständen stark vergrößern oder verkleinern können: Populationswellen, wie sie bei Heuschrecken und Lemmingen beobachtet werden, können durch wechselnde Umwelteinflüsse wie Futtermangel oder -überfluss, Temperaturänderungen, Beutezyklen oder Niederschlagsschwankungen verursacht werden. Sie können durch die daraus resultierenden Abwanderungen großer Herden zur Aufspaltung einer Population führen, durch die sich im Laufe der Zeit eine neue Art durch adaptive Radiation herausbilden kann. Dieser Begriff wird in der Biogeografie, die sich mit der Verbreitung von Arten beschäftigt, unterschiedlich strikt definiert. Einige Autoren sprechen nur dann von einer adaptiven Radiation, wenn eine Population nur kurzzeitig geografisch in zwei Gruppen aufgeteilt wurde und es anschließend zu keinem genetischen Austausch mehr zwischen ihnen kommt, doch das Zusammenleben der Arten immer noch durch zwischenartliche Konkurrenz, beispielsweise in der Nahrungsart, bestimmt wird. Nach dieser Definition kommt es dann zu einer sympatrischen Artbildung. Andere Autoren hingegen sprechen bereits von adaptiver Radiation, wenn eine Population dauerhaft geografisch aufgespalten wird und sich dadurch zwei unterschiedliche Arten entwickeln. Diese Artbildung ohne Konkurrenz wird allopatrisch genannt.

A.1.9 Ökologische Nischen

Eine ökologische Nische ist die Stellung einer Organismenart innerhalb einer Lebensgemeinschaft. Dadurch sind Nische und Konkurrenz theoretisch miteinander verknüpft.

Da ökologisch identische Arten nicht miteinander in demselben Lebensraum koexistieren können, bildet jede Art eine eigene artspezifische ökologische Nische aus, die sich aus Nahrungsform, Nahrungsaufnahmezeit, Brutzeit, Brutort etc. zusammensetzt. Durch diese Spezialisierung wird das Zusammenleben ähnlicher Arten überhaupt erst ermöglicht. Allgemein wird angenommen, dass Arten, die sich mit den Randbereichen des Ressourcenkontinuums begnügen, einen breiteren Ressourcenabschnitt, also eine breitere ökologische Nische, besetzen, da ihre Ressourcenqualität zumeist geringer ist. Bedingung für eine sympatrische Artbildung ist die Einnischung. Bei ihr passen sich Lebewesen an eine Umwelt so an, dass sie die von anderen nicht genutzten Ressourcen für ihre eigene Fortpflanzung verwenden um einem Konkurrenzdruck der eigenen oder einer fremden Art auszuweichen. Stirbt eine Art aus und hinterlässt sie eine nur von ihr genutzte Ressource, so ist die Wahrscheinlichkeit sehr groß, dass sich eine andere Art in dieser ökologischen Nische ausbreitet. Diese Populationen sind zunächst erst einmal sehr klein, da sie sich nur aus wenigen Gründerindividuen zusammensetzen. Sollte die übernommene Ressource ergiebig genug sein, wird sich die neue Population weiter spezialisieren und ausbreiten. Wird eine Ressource von mehr als einer Art genutzt, kommt es zur Ausbeutungskonkurrenz. Dann ist es für alle Beteiligten günstiger, durch ökologische Differenzierung eine Ressourcenaufteilung vorzunehmen. Sie läuft nach dem gleichen Schema wie die Einnischung ab. Aufgrund dieser und anderer Umweltfaktoren wachsen Populationen auch nicht bis ins Unendliche, sondern streben auf einen mehr oder weniger konstanten Wert zu. Dieser Gleichgewichtszustand hängt unter anderem auch von der Reproduktionsrate ab.

A.1.10 Evolution

Die evolutionäre Entwicklung stellt damit ein Optimierungsverfahren dar, dessen Suchprozess sich im Raum der möglichen Erbanlagen bewegt. Eine Optimierung findet dahingehend statt, dass ein Individuum oder eine Art am besten in seiner Umwelt zurecht kommt. Die Evolution ist dabei sehr effizient: Mutation, Rekombination und Selektion nach Tauglichkeit verknüpfen sich zu einem gerichteten und ungerichteten Suchprozess. Mutationen werden zur Überwindung lokaler Optima gebraucht, die Selektion hingegen stellt die eigentliche Steuerung der Evolution dar, die jedoch in der Natur durch Katastrophen oder Klimaschwankungen einem ständigen Kriterienwandel unterliegt. Das Maß für die Optimierung ist die Zeit: Je schneller es einer Art gelingt, sich den vorherrschenden Umweltbedingungen anzupassen, desto eher kann sie eine ökologische Nische und damit einen eigenen Lebensraum besetzen und gegen Konkurrenten verteidigen. Um dieses Ziel zu erreichen, geizt die Natur nicht mit ihren Individuen, da sie Arten eine kurze Generationsfolge ermöglicht oder sie in jeder Generation mit einer großen Anzahl Nachkommen ausstattet. Evolution ist ein parallel ablaufender Optimierungsprozess, da viele Individuen gleichzeitig existieren und damit unter „lebensnahen Bedingungen" getestet werden. So werden Suboptimale Lösungspfade schnell erkannt und nicht weiter verfolgt. Gleichzeitig handelt es sich um einen Prozess mit Rückkopplung, da die Lebewesen ihre Umwelt verändern, die wiederum einen Einfluss auf die Lebensbedingungen der Arten hat. So haben Pflanzen beispielsweise einen Einfluss auf die Zusammensetzung

der Atmosphäre, und Mäuse, die sich stark vermehren, ermöglichen auch ein Populationswachstum von Mäusebussarden, die sich wiederum stärker von Mäusen ernähren, was anschließend zu einer Verminderung der kleinen Nagetiere führt.

Jüngere Forschungsergebnisse zeigen, dass die Gene nicht nur die eigentlichen Strukturmerkmale der Lebewesen, sondern auch deren rekursiven Erzeugungsfunktionen für den Phänotyp des Individuums beinhalten. Sie stellen also nicht den Bauplan des Lebewesens, sondern eher den Algorithmus des Genregulationssystems dar. So können die Erbinformationen kompakter und zuverlässiger gespeichert werden. Mutationen stellen damit keine unmittelbare Änderung des „genetischen Wissens" dar, sondern viele mehr eine Änderung des Algorithmus des Lebens.

A.2 Terminolgie der evolutionären Algorithmen

Evolutionäre Algorithmen stellen ein Modell der natürlichen Evolution dar, bei der die biologischen Fähigkeiten zur Optimierung eines Prozesses verwendet werden. Dazu werden einige biologische Komponenten und ihre Bedeutungen auf die Syntax einer Programmiersprache abgebildet. Ihre Nomenklatur soll im folgenden kurz erläutert werden.

Die Menge aller Lebewesen stellt die Population dar, die in einem Algorithmus einer Menge oder einer Familie von Chromosomen entsprechen. Die Population zu einem Zeitpunkt wird sowohl in der Biologie als auch bei Evolutionären Algorithmen Generation genannt. Sie Reproduziert sich, indem sie Nachkommen aus einem oder mehreren Lebewesen erzeugen. Analog werden in einem Algorithmus neue Chromosomen aus einem oder mehreren Chromosomen erzeugt. Ihre Fitness und damit ihre Überlebens- bzw. ihre Vermehrungsfähigkeit entspricht der Güte einer durch Chromosomen kodierten Lösung. Diese Chromosomen, die in der natürlichen Evolution den Bauplan und damit die Eigenschaften eines Lebewesens bzw. Teile davon bilden, werden als Zeichenketten oder Strings fester oder variabler Länge dargestellt. Gene, Teilstücke eines Chromosoms, die eine Teileigenschaft oder auch Teil einer komplexeren Eigenschaft beinhalten, können auf Zeichen, Characters oder allgemein auf Variablen abgebildet werden. Das eigentliche Allel, das in der Biologie eine Ausprägung darstellt, ist dann entsprechend der Wert eines einzelnen Zeichens. Die Position dieses Zeichens entspricht dem Ort eines Gens auf dem Chromosom und somit seinem Locus. Der Phänotyp beschreibt in der Biologie das Erscheinungsbild eines Lebewesens, in einem Evolutionären Algorithmus handelt es sich um eine auf einem Chromosom kodierte Problemlösung. Ähnlich beim Genotyp oder Erbbild. Er stellt das Erscheinungsbild eines Chromosoms dar und entspricht der formalen Kodierung einer Lösung. So können unterschiedliche Genotypen den gleichen Phänotyp zur folge haben. Als Beispiel sei hier der Beginn einer Rundreise in einer anderen Stadt beim TSP genannt.

B Anhang: NP-Vollständigkeit

Im Rahmen der Komplexitätstheorie werden für Algorithmen Aufwandsabschätzungen bezüglich der Rechenzeit und des benötigten Speicher vorgenommen. Dieser Anhang soll einige wichtige Begriffe erläutern, die bei der Beurteilung der Komplexität von Algorithmen und Computer-Programmen eine wichtige Rolle spielen. Wir konzentrieren uns hierbei allein auf den Rechenaufwand.

Die benötigte Rechenzeit eines Programms hängt sehr stark von der Geschwindigkeit des Rechners ab. Aus diesem Grund betrachtet man nicht die Rechenzeit direkt, sondern die Anzahl der wesentlichen Rechenschritte, die zur Durchführung eines Programms oder eines Algorithmus erforderlich sind. Die Anzahl der wesentliche Rechenschritte, die ein Programm bei einer bestimmten Eingabe durchführen muss, hängt sehr stark von der Eingabe selbst ab. Betrachten wir ein typisches Problem wie das Sortieren einer Zahlen- oder Namensliste. Es ist offensichtlich, dass der Rechenaufwand im Allgemeinen mit Länge der Liste steigt. Er hängt unter Umständen außerdem von der Vorsortierung der Liste ab. Es gibt Sortierverfahren, die immer eine feste Anzahl von Schritten zur Sortierung benötigen. Es existieren aber auch Sortierverfahren, die sofort abbrechen, wenn die Liste sortiert ist. Um bei den Komplexitätsbetrachtungen von der Eingabe unabhängiger zu werden, stellt man üblicherweis nur Worst-Case Betrachtungen an. Der Rechenaufwand wird – in Abhängigkeit der Länge oder Komplexität der Eingabe – immer für den schlechtesten Fall angegeben. Bei Algorithmen, die sehr häufig angewendet werden, ist es durchaus sinnvoll, anstelle der Worst-Case-Komplexität die Average-Case-Komplexität zu betrachten. Dies ist allerdings wesentlich schwieriger, da man zum einen Verteilungsannahmen über die Eingaben machen muss und zum anderen eine Mittelwertbildung über alle möglichen Eingaben vornehmen muss.

Neben der meist ausschließlcihen Betrachtung der Worst-Case-Analyse besteht eine weitere Einschränkung, die man vornimmt, darin, dass man den Rechenaufwand nur für „große" Eingaben betrachtet. Hierfür wird dann die so genannte O-Notation verwendet. Die Aussagen über die Komplexität gelten formal nur für den Grenzübergang, d.h. im Falle der Sortieralgorithmen für Listen der Länge n mit $n \to \infty$. Übliche Sortierverfahren benötigen einen quadratischen ($O(n^2)$) oder die besseren einen Aufwand von $O(n \log(n))$. Dies bedeutet, dass die Anzahl der wesentlichen Rechenschritte bei diesen Sortierverfahren für große n proportional zu n^2 bzw. zu $n \log(n)$ ist.

Bei Komplexitätsbetrachtungen muss man grundsätzlich unterscheiden zwischen dem Aufwand eines konkreten Algorithmus und dem generellen Aufwand zur Lösung eines Problems. Bei einem vorgegebenen Algorithmus lässt sich der Aufwand in den meisten Fällen relativ einfach bestimmen, indem man Schleifendurchläufe und rekursive Funktionsaufrufe zählt. Der Aufwand, der zur Lösung eines Problems – beispielsweise das bereits erwähnte Sortieren einer Liste mit n Einträgen – benötigt wird, lässt sich wesentlich schwieriger ermitteln. Hat man einen Algorithmus gefunden, der das Problem mit einem Aufwand von $s(n)$ Schritten löst, kann man nur behaupten, dass der Aufwand zur Lösung des Problems höchstens $O(s(n))$ beträgt. Es könnte ja jemand einen effizi-

enteren Algorithmus finden, der das Problem mit wesentlich geringerem Aufwand löst. Zu beweisen, dass es keinen Algorithmus gibt, der das Problem mit weniger als $s(n)$ löst, ist in fast allen Fällen wesentlicher schwieriger als die Anzahl der Rechenschritte für einen konkreten Algorithmus zu berechnen.

Man hat für Probleme Komplexitätsklassen eingeführt, die den maximalen Aufwand zur Lösung des Problems im obigen Sinne angeben. Man beschränkt sich dabei üblicherweise auf Entscheidungsprobleme, d.h., die Eingabe für das Problem hat eine variable Größe n und die Lösung des Problems ist eine einfache Ja/Nein-Antwort.

B.1 Die Klasse $\mathcal{P}$

Die Klasse $\mathcal{P}$ besteht aus allen Problemen, für die es einen Algorithmus gibt, dessen Rechenzeitverhalten höchstens polynomial mit der Größe der Eingabe wächst, d.h., es existiert ein Polynom $p(n)$, so dass der Algorithmus einen Rechenaufwand von $O(p(n))$ besitzt. Zur Klasse $\mathcal{P}$ gehört beispielsweise das Problem festzustellen, ob in einer Liste von n Zahlen zwei identische vorhanden sind. Indem man alle Zahlen paarweise vergleicht, hat man einen Algorithmus, der das Problem in $\frac{n(n-1)}{2}$ Schritten löst. Der Aufwand $\frac{n(n-1)}{2} = \frac{n^2}{2} - \frac{n}{2}$ ist für große n proportional zu n^2, so dass der Aufwand zur Lösung dieses Problems höchstens $O(n^2)$, also quadratisch ist.

Die Probleme der Klasse $\mathcal{P}$ werden im Allgemeinen als einigermaßen handhabbar angesehen, da polynomiales Wachstum im Gegensatz zu exponentiellem noch äußerst moderat ist. Man würde erwarten, dass man als nächstes die Klasse $\mathcal{EXP}$ von Problemen einführt, für die ein höchstens exponentieller Aufwand erforderlich ist. Es hat sich aber als äußerst schwierig herausgestellt zu beweisen, dass ein Problem echt zu dieser Klasse $\mathcal{EXP}$ gehört, d.h., dass es zu $\mathcal{EXP}$, aber nicht zu $\mathcal{P}$ gehört. Die Schwierigkeit besteht darin nachzuweisen, dass es keinen Algorithmus geben kann, der das Problem in polynomialer Zeit löst. Deswegen hat man die Klasse $\mathcal{NP}$ eingeführt.

B.2 Die Klasse $\mathcal{NP}$

Die Klasse $\mathcal{NP}$ enthält alle Probleme, für die es einen nicht-deterministischen Algorithmus gibt, der das Problem in polynomialer Zeit löst. $\mathcal{NP}$ steht nicht für *nicht-polynomial*, sondern für *nicht-deterministisch*. Ein nicht-determistischer Algorithmus ist ein theoretischer Algorithmus, der Verzweigungen haben kann und der, wenn er an eine Verzweigung gelangt, automatisch den richtigen oder optimalen Zweig zur Lösung des Problems verfolgt. Zur Durchführung eines nicht-determistischen Algorithmus müsste man ein Orakel haben, das einem an jeder Verzweigung verrät, welchen Ast man auszuwählen hat. Man kann nicht-determistische Algorithmen mit herkömmlichen Computern simulieren, indem man alle Verzweigungen durchspielt und anschließend die geeignete Verzweigung auswählt. Da Verzweigungen aber auch ineinander geschachtelt oder in rekursiven Aufrufen auftreten können, benötigt man mit dieser Art der Simulation

eines nicht-deterministischen Algorithmus einen exponentiellen Rechenaufwand. Sicherlich gilt, dass die Klasse $\mathcal{P}$ in der Klasse $\mathcal{NP}$ enthalten ist. Man vermutet, dass $\mathcal{P}$ echt in $\mathcal{NP}$ enthalten ist. Einen Beweis dafür hat man aber bisher noch nicht gefunden. Man müsste dafür ein Problem in $\mathcal{NP}$ finden, für das man zeigen kann, dass es keinen (deterministischen) Algorithmus gibt, der es in polynomialer Zeit löst. Es gibt jedoch einige besondere Probleme, die gute Kandidaten dafür sind.

Ein Problem P heißt *NP-hart*, wenn sich jedes Problem aus $\mathcal{NP}$ durch Umkodierung oder Umformulierung auf P zurückführen lässt. NP-harte Probleme in $\mathcal{NP}$ werden *NP-vollständig* genannt. Würde man für ein NP-vollständiges Problem einen Algorithmus finden, der es in polynomialer Zeit löst, so könnte man alle Probleme aus $\mathcal{NP}$ in polynomialer Zeit lösen. Man vermutet, dass ein solcher Algorithmus nicht existiert. Könnte man für ein Problem aus $\mathcal{NP}$ beweisen, dass kein Algorithmus existiert, der es in polynomialer Zeit löst, dann würde dies automatisch auch für alle NP-vollständigen Probleme gelten.

Literaturverzeichnis

[Ack87] D.H. Ackley. *A connectionist machine for genetic hillclimbing*. Kluwer, Bosten, 1987.

[ACT99] E. Alba, C. Cotta, and J.M. Troya. Evolutionary design of fuzzy logic controllers using strongly-typed gp. *Mathware & Soft Computing*, 6:109–124, 1999.

[ADV96] A. Amberg, W. Domschke, and S. Voss. Capacitated minimum spanning trees: Algorithms using intelligent search. *Combinatorial Optimization: Theory and Practice*, 1:9–39, 1996.

[All78] A.O. Allen. *Probability, Statistics, and Queuing Theory with Computer Science Applications*. Computer Science and Applied Mathematics. Academic Press, 1978.

[Alt97] L. Altenberg. Fitness distance correlation analysis: An instructive counterexample. In T. Bäck, editor, *Prooceedings of the Seventh International Conference on Genetic Algorithms*, pages 57 – 64, San Fransisco, CA, 1997. Morgan Kaufmann.

[AMM94] J. Arabas, Z. Michalewicz, and J. Mulawka. GAVaPS - a genetic algorithm with varying population size. In *Proceedings of the First IEEE Conference on Evolutionary Computation*, volume 1, pages 73 – 78, Orlando, 1994.

[And53] R.L. Anderson. Recent advances in finding best operating conditions. *Journ. Amer. Stat. Assoc.*, 48:789–798, 1953.

[Ang95] P.J. Angeline. Adaptive and self-adaptive evolutionary computations. In Y. Attikiouzel, R. Marks, D. Fogel, and T. Fukuda, editors, *Computational Intelligence: A Dynamic Systems Perspective*, pages 152–163, Piscataway, 1995. IEEE Press.

[AP94] P.J. Angeline and J.B. Pollack. Coevolving high-level representations. In C. Langton, editor, *Artifical Life III*, pages 55 – 71. Addison Wesley, Reading, MA, 1994.

[AT02] E. Alba and M. Tomassini. Parallelism and evolutionary algorithms. *IEEE Trans. on Evolutionary Computation*, 6:443–462, 2002.

[Axe87] R. Axelrod. Genetic algorithm for the prisoners dilemma problem. In L. Davis, editor, *Genetic Algorithms and Simulated Annealing*, pages 32 – 41. Morgan Kaufmann, San Mateo, CA, 1987.

[BA83] C.G. Broyden and N.F. Attia. A smooth sequential penalty function method for solving nonlinear programming problem. In *System Modelling and Optimization, Proceedings of the 11 th IFIP Conference*, volume 59 of *Lecture Notes in Control and Information Sciences*. Springer, July 1983.

[Bäc94] T. Bäck. Selective pressure in evolutionary algorithms: A characterization of selection mechanisms. In *Proceedings of The First IEEE Conference on Evolutionary Computation*, volume 1, pages 57 – 62, Orlando, 1994.

[Bäc96] T. Bäck. *Evolutionary Algorithms in Theory and Practise*. Oxford University Press, New York , Oxford, 1996.

[Bak87] J.E. Baker. Reducing bias and inefficiency in the selection algorithm. In J.J. Grefenstette, editor, *Proceedings of the 2nd International Conference on Genetic Algorithms and Their Applications*, pages 14 – 21, Hillsdale, NJ, 1987. Lawrence Erlbaum Associates.

[Bas94] A. Bastian. Identifying fuzzy model utilizing genetic programming. *Fuzzy Sets and Systems*, 113:333–350, 1994.

[BBHB94] J.C. Bezdek, S. Boggavarapu, L.O. Hall, and A. Bensaid. Genetic algorithm guided clustering. In *Proc. of the First IEEE Conference on Evolutionary Computation*, pages 34–39, Piscataway, 1994. IEEE Press.

[Bey93] H.-G. Beyer. Toward a theory of evolution strategies: Some asymptotical results from the (1,+λ)-theory. *Evolutionary Computation*, 1:165–188, 1993.

[Bey94] H.-G. Beyer. Toward a theory of evolution strategies: The $(\mu + \lambda)$-theory. *Evolutionary Computation*, 2:381–407, 1994.

[Bey95] H.-G. Beyer. Toward a theory of evolution strategies: On the benefits of sex – the $(\mu/\mu, \lambda)$-theory. *Evolutionary Computation*, 3:81–111, 1995.

[Bey96] H.-G. Beyer. On the asymptotic behavior of multirecombinant evolution strategies. In H.-M. Voigt, W. Ebeling, I. Rechenberg, and H.-P. Schwefel, editors, *Parallel Problem Solving form Nature*, pages 122–133. Springer, Berlin, 1996.

[Bey00] H.-G. Beyer. *The Theory of Evolutionary Algorithms*. Springer, Berlin, 2000.

[Bez92] J.C. Bezdek. On the relationship between neural networks, pattern recognition and intelligence. *Approximate Reasoning*, 6:85–107, 1992.

[Bez94] J.C. Bezdek. What is computational intelligence? In J.M. Zurada, R.J. Marks II, and C.J. Robinson, editors, *Computational Intelligence: Imitating Life*, pages 1–12. IEEE Press, New York, 1994.

[BGK04] E.K. Burke, S. Gustafson, and G. Kendall. Diversity in genetic programming: An analysis of measures and correlation with fitness. *IEEE Trans. on Evolutionary Computation*, 8:47–61, 2004.

[BH91] T. Bäck and F. Hoffmeister. Extended selection mechanisms in genetic algorithms. In R.K. Belew and L.B. Booker, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, volume 1, pages 92 – 99, San Diego, 1991. Morgan Kaufmann.

[BKKN03] C. Borgelt, F. Klawonn, R. Kruse, and D. Nauck. *Neuro-Fuzzy-Systeme. Von den Grundlagen künstlicher Neuronaler Netze zur Kopplung mit Fuzzy-Systemen.* Vieweg, Wiesbaden, 2003.

[BKLW04] M.V. Butz, T. Kovacs, P.L. Lanzi, and S.W. Wilson. Toward a theory and generalization and learning in XCS. *IEEE Trans. on Evolutionary Computation*, 8:28–46, 2004.

[BN88] C:G: Broyden and N.F. Penalty functions, Newton's method, and quadratic programming. *Journal of Optimization Theory and Applications*, 58(3), 1988.

[Bon96] A. Bonarini. Evolutionary learning of fuzzy rules: Competition and cooperation. In W. Pedrycz, editor, *Fuzzy Modelling: Paradigms and Practice*, pages 265–284. Kluwer, Norwell, 1996.

[Bor78] J. Born. *Evolutionsstrategien zur numerischen Lösung von Adaptionsaufgaben.* PhD thesis, Humboldt-Universität, Berlin, 1978.

[Box57] G.E.P. Box. Evolutionary operation: A method for increasing industrial productivity. *Applied Statistics: A Journ. of the Royal Statistical Society*, 6:81–101, 1957.

[Bre62] H.J. Bremermann. Optimization through evolution and recombination. In M.C. Yovits, G.T. Jacobi, and D.G. Goldstein, editors, *Self-Organizing Systems*, pages 93–106. Spartan, Washington, 1962.

[BS93] T. Bäck and H.-P. Schwefel. An overview of evolutionary algorithms. *Evolutionary Computation*, 1:1–23, 1993.

[Byr95] J. Byrne. User manual for the base of aircraft data (bada). User Manual Revision 2.3, Issue 1, EUROCONTROL Experimental Centre, June 1995.

[CdJHL99] O. Cordón, M.J. del Jesús, F. Herrera, and M. Lozano. MOGUL: A methodology to obtain genetic fuzzy rule-based systems under the iterative rule learning approach. *Intelligent Systems*, 14:1123–1153, 1999.

[CFM96] B. Carse, T.C. Fogarty, and A. Munro. Evolving fuzzy rule based controllers using genetic algorithms. *Fuzzy Sets and Systems*, 80:273–294, 1996.

[CGH+04] O. Cordón, F. Gomide, F. Herrera, F. Hoffmann, and L. Magdalena. Ten years of genetic fuzzy systems: Current framework and new trends. *Fuzzy Sets and Systems*, 141:5–31, 2004.

[CGP01] L. Castillo, A. Gonzáles, and R. Pérez. Including a simplicty criterion in the selection of the best rule in fuzzy genetic learning algorithm. *Fuzzy Sets and Systems*, 120:309–321, 2001.

[CH97] O. Cordón and F. Herrera. A three-stage evolutionary process for learning descriptive and approximate fuzzy logic controller knowledge bases from examples. *Approximate Reasoning*, 17:369–407, 1997.

[CH01] O. Cordón and F. Herrera. Hybridizing genetic algorithms with sharing scheme and evolution strategies for designing apprximate fuzzy rule-based systems. *Fuzzy Sets and Systems*, 118:235–255, 2001.

[CHHM01] O. Cordón, F. Herrera, F. Hoffmann, and L. Magdalena. *Genetic Fuzzy Systems: Evolutionary Tuning and Learning of Fuzzy Knowledge Bases.* World Scientific, Singapore, 2001.

[CHS02] O. Cordón, F. Herrera, and T. Stützle. A review on the ant colony optimization metaheuristics: Basics, models and new trends. *Mathware & Soft Computing*, 9:141–175, 2002.

[CL00] F. Cheong and R. Lai. Constraining the optimization of a fuzzy logic controller using an enhanced genetic algorithm. *IEEE Trans. on Systems, Man, and Cybernetics, Part B*, 30:31–46, 2000.

[CLH02] B.-C. Chien, J.Y. Lin, and T.-P. Hong. Learning discriminant functions with fuzzy attributes for classification using geneteic programming. *Expert Systems Appl.*, 23:31–37, 2002.

[CRF94] D. Corne, P. Ross, and H.-L. Fang. GA research note 7: Fast practical evolutionary timetable. Technical report, University of Edinburgh Department of Artifical Intelligence, 1994.

[Dav85] L. Davis. Applying adaptive algorithms to epistatic domains. In *Proceedings of the International Joint Conference on Artifical Intelligence*, pages 162 – 164, 1985.

[Dav90] R.N. Davé. Fuzzy shell clustering and application to circle detection in digital images. *Intern. Journ. General Systems*, 16:343–355, 1990.

[Dav91] L. Davis. *Handbook of Genetic Algorithms.* Van Nostrand Reinhold, 1991.

[Deb01] K. Deb. *Multi-Objective Optimization Using Evolutionary Algorithms.* Wiley, Chichester, 2001.

[DJ75] K.A. De Jong. *An Analysis of the Behavior of a Class of Genetic Adaptive Systems.* PhD thesis, University of Michigan, Ann Arbor, 1975.

[DJ93] K.A. De Jong. Genetic algorithms are not function optimizers. In L. Darrell Whitley, editor, *Foundations of Genetic Algorithms 2*, pages 5 – 17. Morgan Kaufmann, San Mateo, California, 1993.

[DMC96] M. Dorigo, V. Maniezzo, and A. Colorni. The ant system: Optimization by a colony of cooperating agents. *IEEE Trans. on Systems, Man, and Cybernetics – Part B*, 26:29–41, 1996.

[DS90] G. Duck and T. Scheuer. Threshold accepting: A general purpose optimization algorithm appearing superior to simulated annealing. *Journal of Computational Physics*, 90:161 – 175, 1990.

[DSW93] G. Duck, T. Scheuer, and H.-M. Wallmeier. Toleranzschwelle und Sintflut: Neue Ideen zur Optimierung. *Spektrum der Wissenschaft*, pages 42 – 51, März 1993.

[EAVH91] A.E. Eiben, E.H.L. Aarts, and K.M. Van Hee. Global convergence of genetic algorithms: An infinite markov chain analysis. In H.-P. Schwefel and R. Männer, editors, *Parallel Problem Solving from Nature*, pages 4 – 12. Springer, Berlin, 1991.

[ECS89] L.J. Eshelman, R.A. Caruana, and J.D. Schaffer. Biases in the crossover landscape. In H. D. Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms and Their Applications*, pages 10 – 19, San Mateo, CA, 1989. Morgan Kaufmann.

[EHM99] A.E. Eiben, R. Hinterding, and Z. Michlewicz. Parameter control in evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 3:124 – 141, 1999.

[ES91] L.J. Eshelman and J.D. Schaffer. Preventing premature convergence in genetic algorithm by preventing incest. In R.K. Belew and L.B. Booker, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, volume 1, pages 115 – 122, San Diego, 1991. Morgan Kaufmann.

[Esh91] L.J. Eshelman. The CHC adaptive search algorithm: How to have save search when engaging in nontraditional genetic recombination. In G.J.E. Rawlins, editor, *Foundations of Genetic Algorithms*, pages 265 – 283. Morgan Kaufmann, San Mateo, California, 1991.

[Fel57] W. Feller. *An Introduction to Probability Theory and its Application*, volume 1. John Wiley and Sons, Inc., New York, London, Sydney, 1957. Second Edition.

[FM91] B.R. Fox and M.B. McMahon. Genetic operators for sequencing problems. In G.J.E. Rawlins, editor, *Foundations of Genetic Algorithms*, pages 284 – 300. Morgan Kaufmann, San Mateo, California, 1991.

[FM93] S. Forrest and M. Mitchell. Relative building-block fitness and the building-block hypothesis. In L. Darrel Whitley, editor, *Foundations of Genetic Algorithms 2*, pages 109 – 126. Morgan Kaufmann, San Mateo, California, 1993.

[Fog92] D.B. Fogel. *Evolving Artifical Intelligence.* Ph.D. thesis, University of California, San Diego, 1992.

[For85] S. Forrest. Documentation for prisoner's dilemma and norms programs that use the genetic algorithm. Technical report, University of New Mexico, Albuquerque, NM, 1985.

[FOW66] L.J. Fogel, A.J. Owens, and M.J. Walsh. *Artifical Intelligence through Simulated Evolution.* Wiley, New York, 1966.

[Fra57] A.S. Fraser. Simulation of genetic systems by automatic digital computers. *Australian Journ. of Biol. Science*, 10:484–499, 1957.

[Fra72] D.R. Frantz. *Non-Linearities in Genetic Adaptive Search.* Dissertation, University of Michigan, Dissertation Abstracts International, 33(11), 5240B-5241B (University Microfilms No. 73-11,116), 1972.

[Fri58] R.M. Friedberg. A learning machine, part i. *IBM Journal*, 2:2–13, 1958.

[GB89] J.J. Grefenstette and J.E. Baker. How genetic algorithms work: A critical look at implicit parallelism. In J.D. Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms and Their Applications*, pages 20 – 27, San Mateo, CA, 1989. Morgan Kaufmann.

[GD91] D.E. Goldberg and K. Deb. A comparative analysis of selection shemes used in genetic algorithms. In G.J.E. Rawlins, editor, *Foundations of Genetic Algorithms*, pages 69 – 93. Morgan Kaufmann, San Mateo, California, 1991.

[GDK90] D.E. Goldberg, K. Deb, and B. Korb. Messy genetic algorithms revisited: Studies in mixed size and scale. *Complex Systems*, 4:415 – 444, 1990.

[Ger94] I. Gerdes. Application of genetic algorithms to the problem of free routing for aircraft. In *Proc. of the IEEE World Congress on Computational Intelligence: First IEEE Conference on Evolutionary Computation*, volume 2, pages 536–541, Orlando, 1994.

[Ger95] I. Gerdes. Application of genetic algorithms for solving problems related to free routing for aircraft. In J. Biethan and V. Nissen, editors, *Evolutionary Algorithms in Management Applications*, pages 328–340. Springer, Berlin, 1995.

[Ger97] I. Gerdes. Construction of conflict-free routes for aircraft in case of free-routing with genetic algorithms. In *Proceedings of the Air Traffic Management R&D Seminar*, Sacley, 1997.

[GGP98] R. Gudwin, F. Gomide, and W. Pedrycz. Context adaptation in fuzzy processing and genetic algorithms. *Intelligent Systems*, 13:929–948, 1998.

[GH03] R. Ghanea-Hercock. *Applied Evolutionary Algorithms in Java*. Springer, New York, 2003.

[Gil85] A.M. Gillies. *Machine Learning Procedures for Generating Image Domain Feature Detectors*. PhD thesis, University of Michigan, Michigan, 1985.

[GK79] D.E. Gustafson and W.C. Kessel. Fuzzy clustering with a fuzzy covariance matrix. In *IEEE CDC*, pages 761–766, San Diego, 1979.

[GKD89] D.E. Goldberg, B. Korb, and K. Deb. Messy genetic algorithms: Motivation, analysis and first results. *Complex Systems*, 3:493 – 530, 1989.

[GKKH93] D.E. Goldberg, D. Kalyanmoy, H. Kargupta, and G. Harik. Rapid, accurate optimization of difficult problems using fast messy genetic algorithms. IlliGAL Report 93004, Illinois Genetic Algoritms Laboratory, University of Illinois at Urbana-Champaign, Urbana, IL 61801, Februar 1993.

[GL85] D.E. Goldberg and R. Lingle. Alleles, loci and the TSP. In J.J.. Grefenstette, editor, *Proceedings of the First International Conference on Genetic Algorithms and Their Applications*, pages 154 – 159, Hillsdale, NJ, 1985. Lawrence Erlbaum Associates.

[Glo89] F. Glover. Tabu search - part i. *ORSA Journal on Computing*, 1:190 – 206, 1989.

[Glo90] F. Glover. Tabu search - part ii. *ORSA Journal onComputing*, 2:4 – 32, 1990.

[Gol89] D.E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, Reading, 1989.

[Gol90] D.E. Goldberg. Real-coded genetic algorithms, virtual alphabets, and blocking. technical report 90001, University of Illinois at Urbana-Champaign, September 1990.

[GP01] A. Gonzáles and R. Pérez. SLAVE: A genetic learning system based on an iterative approach. *IEEE Trans. on Fuzzy Systems*, 7:417–425, 2001.

[GR87] D.E. Goldberg and J. Richardson. Genetic algorithms with sharing for multimodal function optimization. In J.J. Grefenstette, editor, *Proceedings of the Second International Conference on Genetic Algorithms and Their Applications*, pages 41 – 49, Hillsdale, NJ, 1987. Lawrence Erlbaum Associates.

[Gre86] J.J. Grefenstette. Optimization of control parameters for genetic algorithms. *IEEE Transaction on Systems, Man and Cybernatics*, 16:122 – 128, 1986.

[Gre93] J.J. Grefenstette. Deception considered harmful. In L. Darrell Whitley, editor, *Foundations of Genetic Algorithms 2*, pages 75 – 91. Morgan Kaufmann, San Mateo, California, 1993.

[Gre94] J.J. Grefenstette, editor. *Genetic Algorithms for Machine Learning*. Kluwer, Dordrecht, 1994.

[GS95] A. Geyer-Schulz. *Fuzzy Rule-Based Expert Systems and Genetic Machine Learning*. Physica, Heidelberg, 1995.

[Gur99] H.B. Gurocak. A genetic-algorithm-based method for tuning fuzzy logic controllers. *Fuzzy Sets and Systems*, 108:39–47, 1999.

[Har97] C. Harris. Strongly typed genetic programming to promote hierarchy through explicit syntactic constraints. In J.R. Koza, editor, *Proc. Late Breaking Papers at the Genetic Programming 1997 Conf.*, pages 72–80, Stanford, 1997.

[Hei94] J. Heistermann. *Genetische Algorithmen*. Teubner, Stuttgart, 1994.

[HGD94] J. Horn, D.E. Goldberg, and K. Deb. Long path problems. In Y. Davidor, H.-P. Schwefel, and R. Mnner, editors, *Parallel Problem Solving From Nature – PPSN III*, volume 866 of *Lecture Notes in Computer Science*, pages 149 – 158. Springer, Berlin, 1994.

[HHNT86] J.H. Holland, K.J. Holyoak, R.E. Nisbett, and P.R. Thagard. *Induction: Processes of Inference, Learning, and Discovery*. MIT Press, Cambridge, MA, 1986.

[Hil92] W.D. Hillis. Co-evolving parasites improve simulated evolution as an optimization procedure. In C.G. Langton, C. Taylor, J.D. Farmer, and S. Rasmussen, editors, *Artifical Life II*, pages 313 – 324. Addison Wesley, 1992.

[Hin94] R. Hinterding. Mapping order-independent genes and the knapsack problem. In *Proceedings of The First IEEE Conference on Evolutionary Computation*, volume 1, pages 13 – 17, Orlando, 1994.

[HK94] J. Hopf and F. Klawonn. Learning the rule base of a fuzzy controller by a genetic algorithm. In R. Kruse, J. Gebhardt, and R. Palm, editors, *Fuzzy Systems in Computer Science*. Vieweg, Braunschweig, 1994.

[HKKR99] F. Höppner, F. Klawonn, R. Kruse, and T. Runkler. *Fuzzy Cluster Analysis*. Wiley, Chichester, 1999.

[HL96] F. Herrera and M. Lozano. Adaptive of genetic algorithm parameters based on fuzzy logic controllers. In F. Herrera and J.L. Verdegay, editors, *Genetic Algorithms and Soft Computing*, pages 95–125. Springer, Berlin, 1996.

[HL01] F. Herrera and M. Lozano. Adaptive genetic operators based on coevolution with fuzzy behavior. *IEEE Trans. on Evolutionary Computation*, 5:149–165, 2001.

[HLG97] G.R. Harik, F.G. Lobo, and D.E. Goldberg. The compact genetic algorithm. IlliGAL Report 97006, University of Illinois at Urbana-Champaign, August 1997.

[HLHB00] S. Hajri, N. Liouane, S. Hammadi, and P. Borne. A controlled genetic algorithm by fuzzy logic and belief functions for job shop scheduling. *IEEE Trans. on Systems, Man, and Cybernetics - Part B: Cybernetics*, 30:812–818, 2000.

[HLV95] F. Herrera, M. Lozano, and J.L. Verdegay. Tuning fuzzy controllers by genetic algorithms. *Approximate Reasoning*, 12:299–315, 1995.

[HM97] F. Herrera and L. Magdalena. Genetic fuzzy systems. *Tatra Mountains Mathematical Publications*, 13:93–121, 1997.

[HN01] F. Hoffmann and O. Nelles. Genetic programming for model selection of tsk fuzzy systems. *Information Sciences*, 136:7–28, 2001.

[HNG94] J. Horn, N. Nafpliotis, and D.E. Goldberg. A niched pareto genetic algorithm for multiobjectiveoptimization. In *Proceedings of The First IEEE Conference on Evolutionary Computation*, volume 1, pages 82 – 87, Orlando, 1994.

[HOB99] L.O. Hall, I.B. Özyurt, and J.C. Bezdek. Clustering with a genetically optimized approach. *IEEE Trans. on Evolutionary Computation*, 3:103–112, 1999.

[Hof97] F. Hoffmann. *Entwurf von Fuzzy-Reglern mit Genetischen Algorithmen*. DeutscherUniversitätsVerlag, Wiesbaden, 1997.

[Hof04] F. Hoffmann. Combining boosting and evolutionary algorithms for learning classification rules. *Fuzzy Sets and Systems*, 141:47–58, 2004.

[Hol75] J.H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, 1975.

[Hol94] J.H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, 2. edition, 1994.

[HP97] F. Hoffmann and G. Pfister. Evolutionary design of a fuzzy knowlegde base for a mobile robot. *Approximate Reasoning*, 17:447–469, 1997.

[HR78] J.H. Holland and J.S. Reitmann. Cognitive systems based on adaptive algorithms. In D.A. Watermann and F. Hayes-Roth, editors, *Pattern-Directed Inference Systems*, pages 313–329. Academic Press, New York, 1978.

[HS96] U.D. Hanebeck and G.K. Schmidt. Genetic optimization of fuzzy networks. *Fuzzy Sets and Systems*, 79:411–415, 1996.

[IMT97] H. Ishibushi, T. Murata, and T. Turksen. Single-objective and two-objective genetic algorithms for selecting linguistic rules for pattern classification problems. *Fuzzy Sets and Systems*, 89:135–150, 1997.

[INM99] H. Ishibushi, T. Nakashima, and T. Murata. Performance evaluation of fuzzy classifier systems for multidimensional pattern classification problems. *IEEE Trans. on Systems, Man, and Cybernetics*, 29:601–618, 1999.

[INM01] H. Ishibushi, T. Nakashima, and T. Murata. Three-objective genetics-based machine learning for linguistic rule extraction. *Information Sciences*, 136:109–133, 2001.

[Ios80] M. Iosifescu. *Finite Markov Processes and their Applications*. Wiley, Chichester, 1980.

[Jen03] M.T. Jensen. Reducing the run-time complexity of multiobjective EA's: The NSGA-II and other algorithms. *IEEE Trans. on Evolutionary Computation*, 7:503–515, 2003.

[JF95] T. Jones and S. Forrest. Fitness distance correlation as a measure of problem difficulty for genetic algorithms. In L.J. Eshelmann, editor, *Proceedings of the six'th International Conference on Genetic Algorithms*, pages 184 – 192, San Fransisco, 1995. Morgan Kaufmann.

[JH94] J.A. Joines and C.R. Houck. On the use of non-stationary penalty functions to solve nonlinear constrained optimization problems with gas. In *Proceedings of The First IEEE Conference on Evolutionary Computation*, volume 1, pages 579 – 584, Orlando, 1994.

[Jin00] Y. Jin. Fuzzy modeling of high-dimensional systems: Complexity reduction and interpretability improvement. *IEEE Trans. on Fuzzy Systems*, 8:212–220, 2000.

[JLL00] C.-F. Juang, J.-Y. Lin, and C.-T. Lin. Genetic reinforcement learning through symbiotic evolution for fuzzy controller design. *IEEE Trans. on Systems, Man, and Cybernetics, Part B*, 30:290–302, 2000.

[JP98] H. Juille and J.B. Pollack. Coevolutionary learning: a case study. In *Proceedings of Fifteenth International Conference on Machine Learning*, San Francisco, CA, 1998. Morgan Kaufmann.

[JvSS99] Y. Jin, W. von Seelen, and B. Sendhoff. On FC^3 fuzzy rule systems from data using evolution strategies. *IEEE Trans. on Systems, Man, and Cybernetics, Part B*, 29:829–845, 1999.

[Kar91] C. Karr. Genetic algorithms for fuzzy controllers. *AI Expert*, 2:27–33, 1991.

[Kar93] C. Karr. Fuzzy control of ph using genetic algorithms. *IEEE Trans. on Fuzzy Systems*, 1:46–53, 1993.

[Kar95] H. Kargupta. *SEARCH, Polynomial Complexity, and the Fast Messy Genetic Algorithms.* PhD thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, October 1995. IlliGAL Report No. 95008.

[KBAK99] J.R. Koza, F.H. Bennett, D. Andre, and M.A. Keane. *Genetic Programming III: Darwinian Invention and Problem Solving.* Morgan Kaufmann, San Francisco, 1999.

[KGK94] R. Kruse, J. Gebhardt, and F. Klawonn. *Foundations of Fuzzy Systems.* Wiley, Chichester, 1994.

[KK93] F. Klawonn and R. Kruse. Equality relations as a basis for fuzzy control. *Fuzzy Sets and Systems*, 54:147–156, 1993.

[KK94] F. Klawonn and R. Kruse. Fuzzy partitions and transformations. In *3rd IEEE Conference on Fuzzy Systems*, pages 1269–1273, Piscataway, 1994. IEEE Press.

[KK95] F. Klawonn and R. Kruse. Fuzzy control on the basis of equality relations – with an example from idle speed control. *IEEE Trans. on Fuzzy Systems*, 3:336–350, 1995.

[KK97] F. Klawonn and E.-P. Klement. Mathematical analysis of fuzzy classifiers. In X. Liu, P. Cohen, and M. Berthold, editors, *Advances in Intelligent Data Analysis*, pages 359–370. Springer, Berlin, 1997.

[KK98] F. Klawonn and A. Keller. Fuzzy clustering with evolutionary algorithms. *Intern. Journ. of Intelligent Systems*, 13:975–991, 1998.

[KK03] A. Keller and F. Klawonn. Adaptation of cluster sizes in objective function based fuzzy clustering. In C.T. Leondes, editor, *Intelligent Systems: Technology and Applications vol. IV: Database and Learning Systems.* CRC Press, Boca Raton, 2003.

[KKK94a] J. Kinzel, F. Klawonn, and R. Kruse. Anpassung Genetischer Algorithmen zum Erlernen und Optimieren von Fuzzy-Reglern. In *Fuzzy Logik*, pages 103–110. Springer, Berlin, 1994.

[KKK94b] J. Kinzel, F. Klawonn, and R. Kruse. Modifications of genetic algorithms for designing and optimizing fuzzy controllers. In *Proc. of the First IEEE Conference on Evolutionary Computation*, pages 28–33, Piscataway, 1994. IEEE Press.

[Kla00] F. Klawonn. Fuzzy points, fuzzy relations and fuzzy functions. In V. Novák and I. Perfilieva, editors, *Discovering the World with Fuzzy Logic*, pages 431–453. Physica, 2000.

[Koz90] J.R. Koza. Genetic programming: A paradigm for genetically breeding populations of computer programs to solve problems. technical report STAN-CS-90-1314, Stanford University Computer Science Department, 1990.

[Koz92] J.R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* MIT Press, Cambridge, MA, 1992.

[Koz94] J.R. Koza. *Genetic Programming II: Automatic Discovery of Reusable Programs.* MIT Press, Cambridge, MA, 1994.

[Kun00] L. Kuncheva. *Fuzzy Classifier Design.* Physica, Heidelberg, 2000.

[Lan98] W.B. Langdon. *Genetic Programming and Data Structures: Genetic Programming + Data Structures = Automatic Programming.* Kluwer, Norwell, 1998.

[LR93] S.J. Louis and G.J.E Rawlins. Syntactic analysis of convergence in genetic algorithms. In L. Darrell Whitley, editor, *Foundations of Genetic Algorithms 2*, pages 141 – 151. Morgan Kaufmann, San Mateo, California, 1993.

[LT93a] M. Lee and H. Takagi. Dynamic control of genetic algorithms using fuzzy logic techniques. In S. Forrest, editor, *Proc. of the 5th International Conference on Genetic Algorithms*, pages 76–83, San Mateo, 1993. Morgan Kaufmann.

[LT93b] M. Lee and H. Takagi. Integrating design stages of fuzzy systems using genetic algorithms. In *Proc. 2nd IEEE International Conference on Fuzzy Systems*, pages 612–617, San Francisco, 1993.

[LT94] M. Lee and H. Takagi. A framework for studying the effects of dynamic crossover, mutation, and population sizing in genetic algorithms. In T. Furuhashi, editor, *Advances in Fuzzy Logic, Neural Networks and Genetic Algorithms*, pages 111–126. Springer, Berlin, 1994.

[LV91] G. Liepins and M.D. Vose. Deceptiveness and genetic algorithm dynamics. In G.J.E. Rawlins, editor, *Foundations of Genetic Algorithms*, volume 1, pages 36 – 50. Morgan Kaufmann, San Mateo, CA, 1991.

[Mag97] L. Magdalena. Adapting the gain of an flc with genetic algorithms. *Approximate Reasoning*, 17:327–349, 1997.

[Mah95a] S.W. Mahfoud. A comparison of parallel and sequential niching methods. In L.J. Eshelmann, editor, *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 136 – 143. Morgan Kaufmann, 1995.

[Mah95b] S.W. Mahfoud. *Niching Methods for Genetic Algorithms.* Illigal report no. 95001, Illinois Genetic Algorithms Laboratory, Illinois, 1995.

[MHR98] N.J. McPhee, N.J. Hopper, and M.L. Reierson. Impact of types on essentially typeless problems in GP. In J.R. Koza, W. Banzhaf, K. Chellapilla, K. Deb, and M. Riolo, editors, *Proc. 3rd Annual Genetic Programming Conf.*, pages 232–240, San Francisco, 1998.

[Mic96] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs.* Springer, Berlin, 2. edition, 1996.

[Mit96] M. Mitchell. *An Introduction to Genetic Algorithms.* MIT Press, Cambridge, Massachusetts and London, England, 1996.

[MJ94] Z. Michalewicz and C.Z. Janikow. Handling constraints in genetic algorithms. In *Proceedings of The First IEEE Conference on Evolutionary Computation*, volume 1, pages 151 – 175, Orlando, 1994.

[MKKN02] K. Michels, F. Klawonn, R. Kruse, and A. Nürnberger. *Fuzzy-Regelung.* Springer, Berlin, 2002.

[MM97] L. Magdalena and F. Monasterio. A fuzzy logic controller with learning through the evolution of its knowledge base. *Approximate Reasoning*, 16:335–358, 1997.

[MN95] Z. Michalewicz and G. Nazhiyath. Genocop III: A co-evolutionary algorithm for numerical optimization problems with nonlinear constraints. In *Proceedings of the Second IEEE Conference on Evolutionary Computation*, volume 2, pages 647 – 651, Piscataway, NJ, 1995. IEEE Press.

[Mon95] D.J. Montana. Strongly typed genetic programming. *Evolutionary Computation*, 3:199–230, 1995.

[MRR+53] M. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller. Equation of state calculations by fast computing machine. *Journal of Chemical Physics*, 21:1087–1092, 1953.

[MS96] B.L. Miller and M.J. Shaw. Genetic algorithms with dynamic niche sharing for multimodal function optimization. In *Third IEEE International Conference on Evolutionary Computation*, pages 786 – 791. IEEE Press, 1996.

[Müh91] H. Mühlenbein. Evolution in time and space – the parallel genetic algorithm. In G. Rawlins, editor, *Foundations of Genetic Algorithms*, pages 316–337. Morgan Kaufmann, San Francisco, 1991.

[Müh92a] H. Mühlenbein. How genetic algorithms really work: Mutation and hillclimbing. In R. Männer and B. Manderick, editors, *Parallel Problem Solving from Nature*, pages 15–26. North-Holland, Amsterdam, 1992.

[Müh92b] H. Mühlenbein. Parallel genetic algorithms in combinatorial optimization. In O. Balci, R. Sharda, and S. Zenios, editors, *Computer Science and Operations Research*, pages 441–456. Pergamon Press, New York, 1992.

[Mus97] I. Muslea. A general-purpose AI planning system based on the genetic programming paradigm. In J.R. Koza, editor, *Proc. Late Breaking Papers at the Genetic Programming 1997 Conf.*, pages 157–164, Stanford, 1997.

[MW94] K.E. Mathias and L.D. Witley. Transforming the search space with Gray-coding. In *Proceedings of The First IEEE Conference on Evolutionary Computation*, volume 1, pages 513 – 518, Orlando, 1994.

[NB95] P. Nordin and W. Banzhaf. Complexity compression and evolution. In L.J. Eshelman, editor, *Proc. of the Sixth Intern. Conf. on Genetic Programming*, pages 310–317, San Francisco, 1995. Morgan Kaufmann.

[Nis97] V. Nissen. *Einführung in Evolutionäre Algorithmen*. Vieweg, Braunschweig, 1997.

[NL94] K.C. Ng and Y. Li. Design of sophisticated fuzzy logic controllers using genetic algorithms. In *3rd IEEE Conference on Fuzzy Systems*, pages 1708–1712, Piscataway, 1994. IEEE Press.

[NMP97] S. Nascimiento and F. Moura-Pires. A genetic approach for fuzzy clustering with a validity measure function. In X. Liu, P. Cohen, and M. Berthold, editors, *Advances in Intelligent Data Analysis*, pages 325–335. Springer, Berlin, 1997.

[OR01] M. O'Neill and C. Ryan. Grammatical evolution. *IEEE Trans. on Evolutionary Computation*, 5:349–358, 2001.

[OSH87] I.M. Oliver, D.J. Smith, and J.R.C. Holland. A study of permutation crossover operators on the traveling salesman problem. In J.J. Grefenstette, editor, *Proceedings of the Second International Conference on Genetic Algorithms and Their Applications*, pages 224 – 230. Lawrence Erlbaum Associates, 1987.

[Pap96] M. Papageorgiou. *Optimierung*. Oldenbourg, München, 2nd edition, 1996.

[Par96] J. Paredis. Coevolutionary algorithms. In T. Bäck, D. Fogel, and Z. Michalewicz, editors, *The Handbook of Evolutionary Computation*. Oxford University Press, New York, 1996.

[PBP+94] F. Petry, B. Buckles, D. Prabhu, R. George, and R. Srikanth. Fuzzy clustering with genetic search. In *Proc. of the First IEEE Conference on Evolutionary Computation*, pages 34–39, Piscataway, 1994. IEEE Press.

[PDJ94] M. Potter and K. De Jong. A cooperative coevolutionary approach to function optimization. In Y. Davidor, H.-P. Schwefel, and R. Männer, editors, *Proceedings of the Third Conference on Parallel Problem Solving from Nature*, volume 866 of *Lecture Notes in Comuter Science*, pages 249 – 257. Springer, 1994.

[PDJG95] M.A. Potter, K.A. De Jong, and J.J. Grefenstette. A coevolutionary approach to learning sequential decision rules. In L.J. Eshelmann, editor, *Proceedings of the Sixth International Conference '95*, pages 366 – 372. Morgan Kaumann, 1995.

[Ped97] W. Pedrycz, editor. *Fuzzy Evolutionary Computation.* Kluwer, Boston, 1997.

[PK91] D.T. Pham and D. Karaboga. Optimum fuzzy of fuzzy logic controllers using genetic algorithms. *Systems Engineering*, 1:114–118, 1991.

[PSS02] W. Pedrycz, G. Succi, and O. Shai. Genetic-fuzzy approach to the boolean satisfyability problem. *IEEE Trans. on Evolutionary Computation*, 6:519–525, 2002.

[RB95] C.D. Rosin and R.K. Belew. Methods for competetive coevolution: Finding opponents worth beating. In L.J. Eshelmann, editor, *Proceedings of the Sixt International Conference on Genetic Algorithms*, 1995.

[RB97] C.D. Rosin and R.K. Belew. New methods for competetive coevolution. In *Proceedings of the Sixth International Conference on Genetic Algorithms*, IEEE Transactions on Evolutionary Computation. IEEE, 1997.

[RDVZG01] M. Regattieri-Delgado, F. Von-Zuben, and F. Gomide. Hierarchical genetic fuzzy systems. *Information Sciences*, 136:29–52, 2001.

[Rec73] I. Rechenberg. *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution.* Frommann-Holzoog, Stuttgart, 1973.

[Rec94] I. Rechenberg. *Evolutionsstrategie'94.* Frommann-Holzboog, Stuttgart, 1994.

[Rud92a] G. Rudolph. On correlated mutations in evolution strategies. In R. Männer and B. Manderick, editors, *Parallel Problem Solving from Nature*, pages 105–114. North-Holland, Amsterdam, 1992.

[Rud92b] G. Rudolph. Parallel approaches to stochastic global optimization. In W. Joosen and E. Milgrom, editors, *Parallel Computing: From Theory to Sound Practice*, Proceedings of the European Workshop on Parallel Computing, pages 256 –267, Amsterdam, 1992. IOS Press.

[Rud94a] G. Rudolph. Convergence analysis of canonical genetic algorithms. *IEEE Transactions on Neural Networks*, 5:96 – 101, 1994.

[Rud94b] G. Rudolph. Convergence of non-elitist strategies. In D.B. Fogel, editor, *ICEC'94: Proceedings of the First IEEE Conference on Evolutionary Computation*, volume 1, pages 63 – 66, Orlando, 1994.

[Rus00] M. Russo. Genetic fuzzy learning. *IEEE Trans. on Evolutionary Computation*, 4:259–273, 2000.

[SB92] N. Schraudoph and R. Belew. Dynamic parameter encoding for genetic algorithms. *Machine Learning*, 9(1):9 – 21, 1992.

[SCED89] J.D. Schaffer, R.A. Caruana, R.J. Eshelmann, and R. Das. A study of control parameters affecting online performance of genetic algorithms for function optimization. In J.D. Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms and Their Applications*, pages 51–60. Morgan Kaufmann, 1989.

[Sch75] H.-P. Schwefel. *Evolutionsstrategie und numerische Optimierung*. PhD thesis, TU Berlin, Berlin, 1975.

[Sch84] J.D. Schaffer. *Some Experiments in Machine Learning using Vector Evaluated Genetic Algorithms*. PhD thesis, Vanderbuilt University, Nashville, 1984.

[Sch87a] C.G. Schaefer. The argot strategy. In J.J. Grefenstette, editor, *Proceedings of the Second International Conference on Genetic Algorithms and Their Applications*, pages 50 – 55, Hillsdale, NJ, 1987. Lawrence Erlbaum Associates.

[Sch87b] H.-P. Schwefel. Collective phenomena in evolutionary systems. In *Preprints of the 31st Annual Meeting of the International Society for General System Research*, volume 2, pages 1025–1033, Budapest, 1987.

[Sch95] H.-P. Schwefel. *Evolution and Optimum Seeking*. Wiley, New York, 1995.

[SDJ91] W.M. Spears and K.A. De Jong. On the virtues of parameterized uniform crossover. In R.K. Belew and L.B. Booker, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 230 – 236, San Mateo, CA, 1991. Morgan Kaufmann.

[Sed95] R. Sedgewick. *Algorithmen*. Addison-Wesley, Deutschland, 1995. 2. korrigierter Nachdruck.

[SF97] J.E. Smith and T.C. Fogarty. Operator and parameter adaptation in genetic algorithms. *Soft Computing*, 1:81–87, 1997.

[SH94] R. Schmitz and K. Hurrass. Bestimmung der Passing Frequencies auf benachbarten Flugflächen im oberen Luftraum der Bundesrepublik. Institutsbericht IB 112-94/38, DLR Braunschweig, Institut für Flugführung, Braunschweig, 1994.

[SHF94] E. Schöneburg, F. Heizmann, and S. Feddersen. *Genetische Algorithmen und Evolutionsstrategien*. Addision-Wesley, Bonn, 1994.

[SKK96] M. Schröder, F. Klawonn, and R. Kruse. Sequential optimization of multidimensional controllers using genetic algorithms and fuzzy situations. In F. Herrera and J.L. Verdegay, editors, *Genetic Algorithms and Soft Computing*, pages 419–444. Physica, Heidelberg, 1996.

[SM87] J.D. Schaffer and A. Morishima. An adaptive crossover distribution mechanism for genetic algorithms. In J.J. Grefenstette, editor, *Proceedings of the Second International Conference on Genetic Algorithms and Their Applications*, pages 36 – 40, Hillsdale, NJ, 1987. Lawrence Erlbaum Associates.

[Smi80] S.F. Smith. *A Learning System Based on Genetic Adaptive Algorithms.* PhD thesis, Department of Computer Science, University of Pittsburgh, 1980.

[SMM+91] T. Starkweather, S. McDaniel, K. Mathias, D. Whitley, and C. Witley. A comparison of genetic sequencing operators. In R.K. Belew and L.B. Booker, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 69 – 76, San Diego, 1991. Morgan Kaufmann.

[Spe93] W.M. Spears. Crossover or mutation? In L. Darrell Whitley, editor, *Foundations of Genetic Algorithms 2*, pages 221 – 237. Morgan Kaufmann, San Mateo, California, 1993.

[Spe95] W.M. Spears. Adapting crossover in evolutionary algorithms. In J.R. McDonnell, R.G. Reynolds, and D.B. Fogel, editors, *Proceedings of the Fourth Annual Conference on Evolutionary Programming*, pages 367 – 384. The MIT Press, 1995.

[SX93] M. Schoenauer and S. Xanthakis. Constrained GA optimization. In S. Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms and Their Applications*, pages 573 – 580, San Mateo, CA, 1993. Morgan Kaufmann.

[SY93] M. Sugeno and T. Yasukawa. A fuzzy-logic-based approach to qualitative modelling. *IEEE Trans. on Fuzzy Systems*, 1:7–31, 1993.

[Sys89] G. Syswerda. Uniform crossover in genetic algorithms. In J.D. Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms and Their Applications*, pages 2 – 9. Morgan Kaufmann, 1989.

[Sys91] G. Syswerda. A study of reproduction in generational and steady state genetic algorithms. In G.J.E. Rawlins, editor, *Foundations of Genetic Algorithms*, pages 94 – 101. Morgan Kaufmann, San Mateo, CA, 1991.

[Tet95] A.G. Tettamanzi. Evolutionary algorithms and fuzzy logic: A two-way integration. In *Proc. 2nd Joint Conf. on Information Sciences*, pages 464–467, Durham, 1995.

[Thr91] P. Thrift. Fuzzy logic synthesis with genetic algorithms. In *Proc. Fourth Intern. Conf. on Genetic Algorithms*, pages 509–513, San Diego, 1991. Morgan Kaufmann.

[TR98] A. Tuson and P. Ross. Adapting operator settings in genetic algorithms. *Evolutionary Computation*, 6:161–184, 1998.

[TS85] T. Takagi and M. Sugeno. Fuzzy identification of systems and its application to modeling and control. *IEEE Trans. on Systems, Man, and cybernetics*, 15:116–132, 1985.

[Ven93] G. Venturini. SIA: A supervised inductive algorithm with genetic search for learning attribute based concepts. In *Proc. European Conf. on Machine Learning*, pages 280–296, Vienna, 1993.

[Vir03] L. Viriakis. Genetica: A computer language that supports general form expressions with evolving data structures. *IEEE Trans. on Evolutionary Computation*, 7:456–481, 2003.

[vL91] G. von Laszewski. Intelligent structural operators for the k-way graph partitioning problem. In R.K. Belew and L.B. Booker, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 45 – 52, San Mateo, CA, 1991. Morgan Kaufmann.

[VL95] T. Van Le. Evolutionary fuzzy clustering. In *Proc. 2nd IEEE Conf. on Evolutionary Computation, Vol. 2*, pages 753–758, Piscataway, 1995. IEEE.

[Vog96] S. Voget. Multiobjective optimization with genetic algorithms and fuzzy control. In *Proc. 4th Europ. Congress on Intelligent Techniques and Soft Computing*, pages 391–394, 1996.

[Vog98] S. Voget. Multi-criteria optimization with a fuzzy genetic algorithms. *Heuristics*, 4:221–244, 1998.

[Vos93] S. Voss. Intelligent search. Vorlesungsskript, TH Darmstadt, Fachbereich Rechts- und Wirtschaftswissenschaften, 1993.

[VR91] M. Valenzuela-Rendón. The fuzzy classifier system: Motivations and first results. In H. Schwefel and R. Männer, editors, *Proc. First Intern. Conf. on Parallel Problem Solving from Nature – PPSN*, pages 330–334, Berlin, 1991. Springer.

[vSK99] B. von Schmidt and F. Klawonn. Fuzzy max-min classifiers decide locally on the basis of two attributes. *Mathware and Soft Computing*, 6:91–108, 1999.

[WC00] C-.C. Wong and C.-C. Chen. A GA-based method for constructing fuzzy systems directly from numerical data. *IEEE Trans. Systems, Man, and Cybernetics, Part B*, 30:904–911, 2000.

[Whi89] D. Whitley. The GENITOR algorithm and selection pressure: Why rank-based allocation of reproductive trials is best. In J.D. Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms and Their Applications*, pages 116 – 121, San Mateo, 1989. Morgan Kaufmann.

[Whi91] D. Whitley. The CHC adaptive search algorithm: How to have save search when engaging in nontraditional genetic recombination. In G.J.E. Rawlins, editor, *Foundations of Genetic Algorithms*, pages 221 – 241. Morgan Kaufmann, San Mateo, California, 1991.

[Wil95] S.W. Wilson. Classifier fitness based on accuracy. *Evolutionary Computation*, 3:149–175, 1995.

[WMC00] J. Werfel, M. Mitchell, and J.P. Crutchfield. Resource sharing and coevolution in evolving cellular automata. *IEEE Trans. on Evolutionary Computation*, 4:388–393, 2000.

[XVII94] H.Y. Xu, G. Vukovich, Y. Ichikawa, and Y Ishii. Fuzzy evolutionary algorithms and automatic robot trajectory generation. In *Proc. of the First IEEE Conference on Evolutionary Computation*, pages 596–600, Piscataway, 1994. IEEE Press.

[YC98a] T. Yu and C. Clack. Polygp: A polymorphic genetic programming system in haskell. In J.R. Koza, W. Banzhaf, K. Chellapilla, K. Deb, and M. Riolo, editors, *Proc. 3rd Annual Genetic Programming Conf.*, pages 416–421, San Francisco, 1998.

[YC98b] T. Yu and C. Clack. Recursion, lambda abstractions and genetic programming. In J.R. Koza, W. Banzhaf, K. Chellapilla, K. Deb, and M. Riolo, editors, *Proc. 3rd Annual Genetic Programming Conf.*, pages 422–430, San Francisco, 1998.

[Zad65] L.A. Zadeh. Fuzzy sets. *Information and Control*, 8:338–353, 1965.

[Zad96] L.A. Zadeh. Fuzzy logic = computing with words. *IEEE Trans. on Fuzzy Systems*, 4:103–111, 1996.

[ZMIR94] J.M. Zurada, R.J. Marks II, and C.J. Robinson, editors. *Computational Intelligence: Imitating Life*. IEEE Press, New York, 1994.

[ZT99] E. Zitzler and L. Thiele. Multiobjective evolutionary algorithms: A comparative study and the strength pareto approach. *IEEE Trans. on Evolutionary Computation*, 3:257–271, 1999.

[ZTL+03] E. Zitzler, L. Thiele, M Laumanns, C.M. Fonseca, and V.G. da Fonseca. Performance assessment of multiobjective optimizers: An analysis and review. *IEEE Trans. on Evolutionary Computation*, 7:117–132, 2003.

Index

ACO, 29
Adaption, 102, 117
— 1-Bit-, 106
— Adaptive, 103
— COBRA, 106
— Compact Genetic Algorithm, 106
— Deterministische, 103
— Dynamic Parameter Encoding, 105
— dynamische, 102
— Feineinteilung, 104
— Grobeinteilung, 103
— Punctuated Crossover, 90
— Selbst-Adaptive, 103
— statische, 102
Algorithmus
— Generational, 85
— genetischer, 36
— — Ablauf, 41
Alphabet, 37
Ameisenkolonieoptimierung, 29
Ant Colony Optimization, 29
Arität
— feste, 128
Ausnahmeregel, 160
Automatisch definierte Funktionen, 140

Beispiel
— ROGENA, 110
— Schnellster Weg, 43, 64
— Travelling Salesman Problem, 42
Bias, 81
Binärdarstellung, 37
Boltzmannkonstante, 26
Building-Block-Hypothese, 47, 50, 55, 97, 143
— statische, 56

card, 78
charakteristische Funktion, 181
Chromosom, 37
— als Ring, 88
— Länge, 68
Classifier, 158
Classifier System, 158
— lernendes, 158, 161
Clusteranalyse, 189
Clusterprototyp, 191
c-Means-Algorithmus, 192
— harter, 192
Co-Evolution, 107
— Competitive Fitness Sharing, 109
— konkurrierend, 108
— kooperativ, 107
— prädativ, 108
— Shared Sampling, 109
Co-evolution
— GENOCOP III, 153
COBRA, 106
Compact Genetic Algorithm, 106
Computational Intelligence, 1
Crossover, 40, 88–97
— k-Point-, 89
— arithmetisch, 151
— Cycle, 92
— Enhanced Edge Rekombination, 94
— heuristisch, 151
— One-Point-, 40
— Order, 93
— Parametrisiertes Uniform, 89
— Partially Matched, 91
— Punctuated, 90
— Segmented, *siehe* k-Point-
— Shuffle, 90
— Two-Point-, 88
— uniformes, 89, 122

Deception, 171
Default-Regel, 160
Defuzzifizierung, 187
dominieren, 9
don't-care-Symbol, 48
Dreiecksfunktion, 181
Dynamic Niche Sharing, 101
Dynamic Parameter Encoding, 105

Editing, 138
Ein-Fünftel-Erfolgsregel, 118
Elite-Prinzip, 56
Eliteprinzip, 116
erschöpfend, 128
ES, 115
evolutionäre Programmierung, 2
evolutionäre Programmierung, 125
evolutionarer Algorithmus, 1
Evolutionary Computation, 1
Evolutionsstrategie, 115

Exploitation, 61
Exploration, 61

FCM, 192
FCS, 192
Fitness
— σ-Skalierung, 70
— adjustierte, 131
— bei Mehrkriterienoptimierung, 73
— Exponentielle Skalierung, 70
— geteilte, 72
— invertiert, 108
— normalisierte, 131
— rohe, 131
— skaliert, 69
— standardisierte, 131
Fitness-Distanz-Korrelierung, 172
Fitnessfunktion, 36, 37, 68–79
— mit Zusatzbedingungen, 76
Funktion
— charakteristische, 47
Fuzzifier, 191
Fuzzy-c-Means-Algorithmus, 192
Fuzzy-c-Shells-Algorithmus, 192
Fuzzy-Clusteranalyse, 191
Fuzzy-Menge, 181

Gaußfunktion, 182
Gen, 37
— parasitär, 107
genetische Programmierung, 127
GENOCOP, 79, 146
Gewichtungsfunktion, 78
Gradienten
— konjugierte, 17
Gradientenverfahren, 16
Gray-Code, 66
Greedy-Heuristiken, 23

Hammingdistanz, 47
— durchschnittliche, 61
HCM, 192
Hillclimbing, 25
Hot Spots, 90

Indikatorfunktion, 181
Inselmodell, 156
— mit Migration, 156
— mit Wettbewerb, 156
— ohne Migration, 156
inverses Pendel, 183
Inversion, 97
iteratives Regellernen, 162

Kapselung, 140
Kodierung, 64, 68
— Alphabet, 64
— Binär, 64
— Gray-Code, 66
— Nachbarschaft, 65, 67
— positionsunabhängige, 67
— reellwertige, 64
— Struktur, 67
— Vergleich, 64
Komma-Strategie, 116
Konvergenz, 56
— Übernahmezeit, 62
— gegen das globale Optimum, 59
— vorzeitige, 37, 42, 61
Kuhn-Tucker-Multiplikatoren, 21

Lagrange-Funktion, 20
Lagrangesche Multiplikatoren, 20
LISP, 128
Liste, 128
Luftraum, 110

Mühlenbeins paralleler genetischer Algorithmus, 157
Mamdani-Regler, 188
Markovprozess, 57
Mehrkriterienoptimierung, 8, 73, 100
Menge
— konvex, 147
Messy GA, 143
— Building-Block-Filtering, 145
— Fast, 144
— probabilistic complete initialization, 144
— Schwellwertakzeptanz, 143
Michigan-Ansatz, 162
Migration, 156
modGA, 145
Mutation, 35, 40, 41, 98, 138
— korrelierte, 120
— Non Uniform, 99, 103
— Struktur-, 100
— Uniform, 150

Nachbarschaft, 21
Netzwerkmodell, 156
Newton-Verfahren, 17
— Quasi, 19

Niched Pareto Genetic Algorithm, 101
Nischentechnik, 86, 100–102
— Deterministisches Crowding, 86
— Dynamic Niche Sharing, 101
— Fitness Sharing, 72
— Niched Pareto Genetic Algorithm, 101
— Preselection, 101
— VEGA, 101
NP, 228
— hart, 229
— Vollständigkeit, 229

Optimierung
— alternierende, 192
— kombinatorische, 11
— mit Zusatzbedingungen, 76, 146
Optimierungsverfahren, 15
— Ameisenkolonie, 29
— Gradientenverfahren, 16
— Greedy-Heuristiken, 23
— Hillclimbing, 25
— mit Nebenbedingungen, 19
— Newton-Verfahren, 17
— Simulated Annealing, 26
— Sintflut-Algorithmus, 28
— Tabu-Search, 21
— Threshold Accepting, 28

Parallelisierbarkeit, 154
Parallelität
— implizit, 48
pareto-optimal, 9, 74, 75
Penalty, 77
Pittsburgh-Ansatz, 162
Plus-Strategie, 116
Population
— Größe, 42
— Konvergenz, 50
Preselection, 101
Problem des Handlungsreisenden, 5
Prozess
— diskret parametrisierter, 57
— kontinuierlich parametrisierter, 57
— stochastischer, 57

reduzierte Surrogate, 48
Regelungstechnik, 179
Regression, 6
— symbolische, 135
Reinforcement, 161
Rekombination
— diskrete, 122
— intermediäre, 122
Reparaturmechanismen, 42
ROGENA, 110

Schema, 47
— definierende Länge, 48
— Fitness, 52
— Ordnung, 48
— Veranschaulichung, 49
— Zugehörigkeit, 50
Schema-Theorem, 50, 54
— Konvergenzaussage, 60
Schlupfvariable, 20
Selektion, 39, 79, 86
— Boltzmann, 83
— Deterministisches Crowding, 86
— diskriminierende/nicht diskriminierende, 80
— Elite/reine, 81
— Komma-Strategie, 116
— Lineares Ranking, 83
— Plus-Strategie, 116
— Remainder Stochastic Sampling, 81
— Rouletterad-, 39
— statische/dynamische, 80
— Steady State, 85
— Stochastic Sampling with Replacement, 80
— Stochastic Universal Sampling, 82
— Tournament, 84
— Wettkampf, 84, 101
— Zweikampf, 85
Sharing-Funktion, 72
Shell-Clustering, 192
Simulated Annealing, 26
Sintflut-Algorithmus, 28
Skalierung
— σ-, 70
Spannweite, 71
Spread, 81
Stabbalanceproblem, 183
Startpopulation, 37, 64
stepping stone model, 156
Straffunktion, 77
Strafwert, 77
Suchraum, 6, 36
— Repräsentation, *siehe* Kodierung
Superindividuals, 69, 146

Täuschung, 171
Tabu-Search, 21

Takagi-Sugeno-Regler, 188
Temperatur
— Simulated Annealing, 26
Testfunktionen, 164
— x^4 mit Rauschen, 165
— Ackley, 166
— Bäck's Stufenfunktion, 166
— Bin-Packing-Problem, 174
— Job-Shop-Scheduling, 174
— Long Path, 167
— Parabel, 165
— Prisoners Dilemma, 175
— Quadratic-Assignment-Problem, 175
— Reversi, 176
— Rosenbrock's Sattel, 165
— Royal Roads, 167
— Rucksackproblem, 24
— Schwefel, 173
— Sheckel's Fuchsbauten, 165
— Stufenfunktion, 165
— Täuschung, 171
— Türme von Hanoi, 178
Testumgebung, 163
Threshold Accepting, 28
Trapezfunktion, 182
Travelling Salesman Problem, 5
TSK-Modell, 188
TSP, 5
Typisierung, 142

Überlebenswahrscheinlichkeit, 38
Übernahmezeit, 62

VEGA, 101
vollständige Definiertheit, 128

Wahrscheinlichkeit
— bedingte, 57

zellularer genetischer Algorithmus, 156
Zielfunktion, 7
Zugehörigkeitsgrad, 181

„Die Vielzahl der auf der Erde existierenden Lebensformen gibt uns eine Vorstellung von der gewaltigen Größe des möglichen Lösungsraumes."
(J. Kinzel, Zeichnung von M. Masuch)

Component Pascal and BlackBox

J. Stanley Warford

Computing Fundamentals

The Theory and Practice of Software Design
with BlackBox Component Builder

2002. xx, 611 pp. with 229 figs. Softc. € 36,90 ISBN 3-528-05828-5

Contents: Theory: Languages and Grammars, Complexity Analysis, Guarded Command Language - Programming Basics: Flow of Control, Functions, Procedures, Recursion - Graphical User Interfaces: Dialog Boxes, Window Input/Output - Algorithms: Searching, Sorting, Random Numbers - Data Structures: Arrays, Linked Lists, Stacks, Queues, Binary Trees - Object Orientation: Class Composition, Inheritance, Polymorphism - Design Patterns: Factory, Iterator, MVC, State - Design Topics: Abstraction, Frameworks, UML Diagrams, Interfaces

The book introduces the reader to computer programming, i.e. algorithms and data structures. It covers many new programming concepts that have emerged in recent years including object-oriented programming and design patterns. The book emphasizes the practical aspects of software construction without neglecting their solid theoretical foundation.

Abraham-Lincoln-Straße 46
65189 Wiesbaden
Fax 0611.7878-400
www.vieweg.de

Prices and other details are subject to change without notice. Please order at your bookstore.

Bestseller aus dem Bereich IT erfolgreich lernen

Dietmar Abts, Wilhelm Mülder

Grundkurs Wirtschaftsinformatik

Eine kompakte und praxisorientierte Einführung

5., überarb. u. erw. Auflage 2004. XIV, 467 S. mit 130 Abb.

Br. € 19,90 ISBN 3-528-45503-9

Inhalt: Grundbegriffe der Wirtschaftsinformatik - Rechnersysteme - Software - Datenübertragung und Netze - Internet - Datenbanken - Enterprise Resource Planning - Managementinformationssysteme - E-Business - Methoden der Software-Entwicklung und Auswahl - Informationsmanagement

Der erfolgreiche "Grundkurs Wirtschaftsinformatik" liegt jetzt in der 5. Auflage vor. Neu hinzu gekommen sind aktuelle Themen wie z.B. LKW-Maut, ITIL, ROI von IT-Systemen, IT-Projektmanagement, Business Intelligence, Mobile Business und E-Government. Die Leserinnen und Leser erhalten ein Gesamtwerk der Wirtschaftsinformatik, das die Grundlagen und aktuellen Trends anschaulich, kompakt und praxisorientiert behandelt.

vieweg

Abraham-Lincoln-Straße 46
65189 Wiesbaden
Fax 0611.7878-400
www.vieweg.de

Stand 1.1.2004. Änderungen vorbehalten.
Erhältlich im Buchhandel oder im Verlag.