

**Studienarbeit:**  
Lösung von Optimierungsproblemen  
mittels  
Genetischer Algorithmen und  
Evolutionstrategien

Gerd Weckenmann  
Matr.Nr. 1605967  
cand. Elektrotechnik  
Institut für Theorie der Elektrotechnik

Stuttgart, den 20. Mai 1996<sup>1</sup>

<sup>1</sup> April 1995 – Mai 1996 (Unterbrechungen von Juli '95 – Oktober '95 und März '96 – April '96)

## **Zusammenfassung**

Mit steigenden Rechnerkapazitäten wird die Simulation naturanaloger Verfahren immer interessanter. Gerade die Parallelisierbarkeit dieser Verfahren macht sie für den Einsatz auf Rechnern sinnvoll.

In dieser Arbeit werden die Grundlagen der Evolutionsalgorithmen erarbeitet und dargestellt. Es werden die beiden Hauptzweige der Evolutionstheorie, die Evolutionsstrategien und die Genetischen Algorithmen vorgestellt. Außerdem wird anhand grundlegender Algorithmen die Effektivität dieser Algorithmen untersucht und mit anderen Optimierungsverfahren verglichen.

# Inhaltsverzeichnis

<b>1</b>	<b>Einleitung</b>	<b>1</b>
1.1	Vorbild Natur . . . . .	1
<b>2</b>	<b>Biologische Evolution</b>	<b>2</b>
2.1	Geschichte der Evolution . . . . .	2
2.2	Biologische Evolution . . . . .	4
2.2.1	Genotyp und Phänotyp . . . . .	5
2.2.2	Rekombination und Mutation . . . . .	6
2.2.3	Populationsgröße und Selektion . . . . .	8
2.3	Evolution als Optimierungsprozeß . . . . .	9
2.4	Evolutionalgorithmen . . . . .	10
<b>3</b>	<b>Evolutionstrategien</b>	<b>11</b>
3.1	Rechenbergs Spielkartennotation . . . . .	11
3.2	Grundlegende Evolutionstrategien . . . . .	13
3.2.1	$(\mu + \lambda)$ -Evolutionstrategie . . . . .	13
3.2.2	$(\mu, \lambda)$ -Evolutionstrategie . . . . .	15
3.2.3	$(\mu/\rho\#\lambda)$ -Evolutionstrategie . . . . .	16
3.2.4	Evolutionstrategien mit mehreren Populationen . . . . .	16
3.2.5	Evolutionstrategien mit zeitweise isolierten Populationen . . . . .	17
3.2.6	Standard-Evolutionstrategie . . . . .	18
3.2.7	Schwächen des Rechenberg-Schwefel-Formalismus . . . . .	18
3.2.8	Optimale Wahl der Parameter . . . . .	18
3.3	Codierung, Mutations- und Rekombinationsoperatoren . . . . .	19

3.3.1	Codierung . . . . .	19
3.3.2	Mutationsoperatoren . . . . .	20
3.3.3	Rekombinationsoperatoren . . . . .	21
3.4	Mathematische Theorie der Evolutionsstrategie . . . . .	21
3.4.1	Grundlagen . . . . .	22
3.4.2	Korridormodell . . . . .	24
3.4.3	Kugelmodell . . . . .	25
3.4.4	Ergebnis . . . . .	27
3.4.5	Evolutionsfenster . . . . .	27
3.4.6	Vergleich des Mutations–Selektions–Verfahrens mit der Gradientenstrategie . . . . .	28
3.5	Selbstorganisierte Evolutionsstrategien . . . . .	29
3.5.1	Mutation . . . . .	30
3.5.2	Rekombination . . . . .	30
3.5.3	Qualitätsfunktion . . . . .	30
3.5.4	Initialisierung . . . . .	30
<b>4</b>	<b>Genetische Algorithmen</b>	<b>31</b>
4.1	Grundlagen der Genetischen Algorithmen . . . . .	31
4.1.1	Allgemeiner Pseudocode eines Genetischen Algorithmus . . . . .	32
4.1.2	Codierung . . . . .	32
4.1.3	Fitneßfunktion . . . . .	33
4.1.4	Fortpflanzung . . . . .	33
4.1.5	Konvergenz . . . . .	34
4.2	Funktionsprinzipien . . . . .	34
4.2.1	Das Schemata–Theorem und Implizite Parallelität . . . . .	35
4.2.2	Die Building–Block Hypothese . . . . .	36
4.2.3	Ausbreitung und Kombination . . . . .	39
4.3	Praktische Aspekte . . . . .	39
4.3.1	Fitneßfunktion . . . . .	39
4.3.2	Streuung der Fitneß und Konvergenzverhalten . . . . .	41
4.3.3	Auswahl der Eltern . . . . .	42

4.3.4	Ersetzungsschema . . . . .	42
4.3.5	Populationsgröße . . . . .	43
<b>5</b>	<b>Andere Optimierungsprozesse</b>	<b>45</b>
5.1	Strategische Verfahren . . . . .	45
5.1.1	Gradientenstrategie . . . . .	45
5.2	Stochastische Optimierungsprozesse . . . . .	47
5.2.1	Zufallssuche . . . . .	47
5.2.2	Monte-Carlo Verfahren . . . . .	47
5.2.3	Simulated Annealing . . . . .	48
5.3	Gemischte Verfahren . . . . .	48
5.3.1	Iterative Verfahren . . . . .	48
<b>6</b>	<b>Vergleich verschiedener Evolutionsalgorithmen</b>	<b>49</b>
6.1	2-Dimensionale Testfunktionen . . . . .	49
6.1.1	Quadratische Funktion . . . . .	50
6.1.2	Treppenfunktion . . . . .	51
6.1.3	Lineare cos Überlagerung mit Nebenmaxima . . . . .	52
6.2	Beschreibung der getesteten Algorithmen . . . . .	53
6.3	Simulationsergebnisse . . . . .	53
6.3.1	Quadratische Funktion . . . . .	54
6.3.2	Treppenfunktion . . . . .	57
6.3.3	Lineare cos Überlagerung mit Nebenminima . . . . .	60
6.4	Zusammenfassung der Ergebnisse . . . . .	62
6.5	Das Travelling-Salesman-Problem . . . . .	62
<b>7</b>	<b>EVDice</b>	<b>66</b>
7.1	Überblick . . . . .	66
7.1.1	Algorithmen . . . . .	66
7.1.2	Informationsausgabe . . . . .	67
7.1.3	Problemspezifisches . . . . .	67
7.2	Bedienung und Konfiguration . . . . .	67
7.2.1	Bedienung . . . . .	67

7.2.2	Konfigurationsdatei und Tagkonzept . . . . .	68
7.2.3	Ausgabe . . . . .	68
7.2.4	Auswertung und Outputfile . . . . .	69
7.2.5	Zufallszahlengenerator . . . . .	70
7.2.6	Algorithmen . . . . .	70
7.3	Operatoren . . . . .	72
7.3.1	Mutationsoperatoren . . . . .	72
7.3.2	Rekombinationsoperatoren . . . . .	73
7.3.3	Selektionsstrategien . . . . .	73
7.4	Problemorientierte Module . . . . .	73
7.4.1	Parametrisierbare Probleme . . . . .	73
7.4.2	Das Traveling-Salesman Problem . . . . .	74
7.5	Konzept und Source . . . . .	75
7.6	Erweiterungsmöglichkeiten . . . . .	76
7.7	Bekannte Fehler . . . . .	76
<b>8</b>	<b>Simulationsmodell</b> . . . . .	<b>77</b>
8.1	Die Idee . . . . .	77
8.2	Der Lebenskreislauf . . . . .	77
8.3	Objekte und Operatoren . . . . .	78
8.4	Übersicht der Module . . . . .	79
8.5	Beschreibung der einzelnen Module . . . . .	79
<b>9</b>	<b>Zusammenfassung</b> . . . . .	<b>81</b>
9.1	Ergebnis . . . . .	81
9.2	Ausblick und Schlußbemerkung . . . . .	82
<b>A</b>	<b>Weitere Informationsquellen</b> . . . . .	<b>i</b>
A.1	Vorlesungen . . . . .	i
A.2	Internet . . . . .	ii
A.2.1	Encore – the Evolutionary Computation Repository Network . . . . .	ii
A.2.2	Newsgroups . . . . .	ii

<b>B Maschinell erzeugte Zufallszahlen</b>	<b>iii</b>
B.1 Linearverteilung . . . . .	iii
B.2 Gaußverteilung . . . . .	iii
<b>C Begriffe und Abkürzungen</b>	<b>v</b>
<b>4 Source</b>	<b>vii</b>

# Abbildungsverzeichnis

3.1	$(1 + 1)$ -Evolutionstrategie . . . . .	14
3.2	$(\mu + \lambda)$ -Evolutionstrategie . . . . .	15
3.3	$(\mu, \lambda)$ -Evolutionstrategie . . . . .	16
3.4	$(\mu/\rho\#\lambda)$ -Evolutionstrategie . . . . .	17
3.5	Bitmutation . . . . .	20
3.6	Einfache Rekombination reelwertiger Vektoren . . . . .	21
3.7	Qualitätsdichteverteilung beim Korridormodell . . . . .	25
3.8	Qualitätsdichteverteilung beim Kugelmodell . . . . .	26
3.9	Evolutionfenster . . . . .	28
3.10	Vergleich zwischen Gradienten- und Evolutionstrategie . . . . .	29
4.1	Allgemeiner Pseudocode eines Genetischen Algorithmus . . . . .	32
4.2	single point crossover . . . . .	34
4.3	Fitneß der Eltern und Nachkommen nach crossover . . . . .	35
4.4	Mutation durch Bitinversion . . . . .	36
4.5	Konvergenzverhalten eines Genetischen Algorithmus . . . . .	37
5.1	Allgemeiner Pseudocode einer Gradientenstrategie . . . . .	46
5.2	Das „Hillclimbing“-Problem . . . . .	46
6.1	Quadratische Funktion . . . . .	50
6.2	Treppenfunktion . . . . .	51
6.3	Lineare cos Überlagerung mit Nebenmaxima . . . . .	52
6.4	Konvergenzverhalten einer $(1 + 1)$ -ES . . . . .	56
6.5	Spur für das Konvergenzverhalten einer $(1 + 1)$ -ES . . . . .	56

6.6	Konvergenzverhalten einer $(1 + 1)$ -ES . . . . .	59
6.7	Spur für das Konvergenzverhalten einer $(1, 5)$ -ES . . . . .	59
6.8	Konvergenzverhalten einer $(5, 16)$ -ES . . . . .	61
6.9	Spur für das Konvergenzverhalten einer $(1 + 1)$ -ES . . . . .	61
6.10	Wahrscheinlichkeitsverteilung beim TSP . . . . .	64
6.11	Optimaler Weg des gewählten Beispiels . . . . .	64
6.12	Typisches Konvergenzverhalten beim TSP mit einer $(1 + 4)$ -ES . . . . .	65
8.1	Lebenskreislauf . . . . .	78
B.1	Gaußnormalverteilung nach einem Algorithmus aus [PTVF92] . . . . .	iv

# Tabellenverzeichnis

2.1	Wichtige Persönlichkeiten in der Geschichte der Evolution . . . . .	4
3.1	Wahl des Selektionsdruckes $\frac{\mu}{\lambda}$ . . . . .	19
4.1	Details des „single point crossover“ . . . . .	34
6.1	Absolutes und lokale Maxima der Funktion 6.3 . . . . .	52
6.2	Parametereinstellungen der getesteten Algorithmen . . . . .	53
8.1	Übersicht der Module . . . . .	79

# Kapitel 1

## Einleitung und Motivation

### 1.1 Die Natur zum Vorbild

Die Natur birgt oftmals erstaunliche Lösungen für komplexe Probleme. Viele dieser Lösungen werden vom Menschen abgeschaut und in der Technik kopiert. Diese Wissenschaft der Nachahmung der Natur nennt man Bionik. Sie gibt es etwa seit 1960 als eigenständigen Wissenschaftszweig.

Die Forschungen der Bionik beruhen auf der Erkenntnis, daß die heute existierenden Lebewesen in der Natur das Ergebnis eines über drei Milliarden Jahre andauernden Evolutionsprozesses sind. In diesem Langzeitversuch hat die natürliche Auslese alles Unpassende eliminiert und zurück blieben optimal an ihre Umwelt angepaßte Lebensformen. Durch diese Erkenntnis scheint es nur vernünftig, die im Verlauf der Evolution gesammelten Experimentierergebnisse auszuwerten und sich zunutze zu machen.

Einige beispielhaft aufgeführte Forschungsthemen der Bionik sind:

- die datenverarbeitende Funktion des Neurons (Stichwort: Neuronale Netze)
- die Sinnesorgane von Lebewesen als Modelle für die Meßgerätetechnik
- die Stofftrennungseigenschaften biologischer Membrane
- die widerstandsvermindernde Elastizität der Delphinhaut

Angesichts solcher Leistungen fragt man sich, welches Optimierungsverfahren solche Lösungen zustande bringen kann. Die Antwort auf diese Frage ist die Evolution. Wenn es aber die Evolution ist, die solche Ergebnisse hervorbringt, dann lohnt sich eine nähere Betrachtung dieser Methode und deren eventuelle Anwendung auf technische Probleme.

# Kapitel 2

## Die biologische Evolution

### 2.1 Die Geschichte des Evolutionsgedankens<sup>1</sup>

So natürlich uns der Gedanke an eine Evolution der Arten heute ist, vor noch nicht einmal 2 Jahrhunderten war dieser Gedanke völlig unmöglich.

Die Entwicklung des Gedankens an eine Evolution des Lebens ist geprägt durch einige wichtige Persönlichkeiten (siehe Tabelle 2.1).

Wir beginnen unsere Geschichte des Evolutionsgedankens mit dem italienischen Universalgenie *Leonardo da Vinci (1452 – 1519)*. In seinem umfassenden und bunten Lebenswerk machte er sich unter anderem Gedanken über die Bedeutung der Fossilien und erahnte ihre wahre historische Bedeutung. Damals wurden diese Fundstücke für Launen der Natur gehalten und ihre Ähnlichkeit mit Knochen dem Zufall zugeschrieben. Auch waren damals noch kaum Fossilien bekannt. So wurden z.B. erst 1860 Fossilien des vor ca. 150 Millionen Jahren lebenden Urvogels *Archaeopteryx* gefunden. Leider geriet das Wissen um die Fossilien vorerst wieder in Vergessenheit.

Einen weiteren Meilenstein der Geschichte der Evolution legte der schwedische Naturforscher *Carl von Linné (1707 – 1778)*. Er unternahm seit etwa 1740 den Versuch einer systematischen Katalogisierung der damals bekannten Tier- und Pflanzenarten. In seinem Werk „systema naturae“ führt der Arzt ca. 4.000 Tier- und ungefähr 14.000 Pflanzenarten auf.

In Anbetracht der Schätzung, daß ca. 500 Millionen Arten seit der Entstehung des Lebens auf der Erde gelebt haben erscheint die Zahl der heute ca. 2 Millionen lebenden Arten als verschwindend klein.

Linné erstellte die Theorie der Konstanz der Arten, d.h. alle Arten wurden bei der Schöpfung erschaffen und seither sind keine Arten mehr entstanden oder ausgestorben.

Diese Theorie scheint uns heute, da wir uns an den Gedanken einer fortlaufenden Evolution gewöhnt haben, als eher einfältig. Man muß Linnés Theorie der Konstanz der Arten aber aus der Perspektive seiner Zeit betrachten. Die bib-

---

<sup>1</sup>Die Informationen dieses Kapitels sind weitestgehend aus [SHF94] entnommen.

liche Schöpfungsgeschichte geht von einem einmaligen Schöpfungsakt aus und die Wissenschaft Linnés Zeit hatte noch keine Beweise zur Widerlegung dieser glaubensbedingten Theorie. Selbst die genaue Beobachtung der Natur legte eine solche Theorie nahe. Alle Lebewesen erzeugen fast ausschließlich gleichartige Nachkommen. Selbst wenn einmal eine Veränderung eintritt gilt dieses Individuum meist als krank oder entartet. Die Evolution höherer Lebewesen spielt sich eben nicht in Maßstäben eines Menschenlebens oder weniger Generationen ab, sondern bedarf der Zeitspanne von Jahrtausenden oder Jahrmillionen, so daß unserer jungen kulturellen Geschichte diese Veränderungen nicht auffallen.

Durch Untersuchungen von Fossilien widerlegte der französische Naturforscher *Georges Baron de Cuvier (1769 - 1832)* die Theorie der Konstanz der Arten. Der Begründer der Paläontologie und der vergleichenden Anatomie stellte die Theorie auf, daß die Arten, durch Naturkatastrophen, aussterben und neue an deren Stelle entstehen. Cuvier ging bei seiner Theorie davon aus, daß sich die Arten zwischen den Katastrophen nicht verändern oder gar weiterentwickeln.

Erst der französische Naturforscher *Jean Baptiste de Lamarck (1744 - 1832)* erkannte durch systematische Studien eine Vielzahl abgestufter Ähnlichkeiten zwischen verschiedenen Pflanzen und Tierarten und folgerte daraus, daß es eine Entwicklung der Arten geben müsse.

Lamarck interpretierte die erkennbaren Ähnlichkeiten zwischen den Arten als Verwandtschaftsstufen sich auseinanderentwickelnder Arten. Die Ursache dieser Auseinanderentwicklung sah Lamarck in dem Bedürfnis der Lebewesen, sich möglichst gut an ihre Umwelt anzupassen. Dies geschieht seiner Meinung nach durch Vererbung erworbener Eigenschaften, die durch Umwelteinflüsse verstärkt, sich über die Generationen herauskristallisieren.

Wesentliche Züge des Evolutionsgedankens erkannten schon vor Darwin der Engländer *R. Chambers (1802 - 1871)* und *Alfred Russell Wallace (1823 - 1913)*.

Erst *Charles Darwin (1809 - 1882)* blieb es aber vergönnt, mit seiner „Theorie des Überlebens des Stärksten“ auf ein breites Publikum zu stoßen. Darwin traf damit den Nerv des aufstrebenden und liberal eingestellten Bürgertums Englands. Nicht die aristokratische Geburt rechtfertigt Erfolg, sondern der, der am geschicktesten ist und sich selbst zu helfen weiß, wird auf lange Sicht diesen ernten.

Darwin hatte erkannt, daß in der Natur ein potentieller Überschuß an Nachkommen erzeugt wird. Trotzdem bleibt die Populationsgröße meist relativ konstant. Demzufolge mußte der größte Teil der Lebewesen sterben, bevor sie selbst Nachkommen erzeugen können.

Diese Grundthese wurde durch das mathematische Modell, der Entwicklung und Ausbreitung von Populationen, durch den Mathematiker *Malthus (1766 - 1834)* gestützt. Malthus zeigte in diesem Modell, daß sich häufig das Bevölkerungswachstum von der Entwicklung der Nahrungsmittel unterscheidet.

Dadurch, daß sich die Bevölkerung in einem exponentiellen, die der Nahrungsmittel aber meist nur in einem linearen Verhältnis entwickeln, entsteht ein „Selektionsdruck“, der letztendlich für eine Abnahme der Bevölkerung und somit zu einer Stabilisierung führt.

Eine weitere wichtige Einsicht Darwins ist, daß sich Lebewesen einer Art zwar

Name	Jahr	Theorie
Leonardo da Vinci	1452 – 1519	Eraht die Bedeutung der Fossilien
Carl von Linné	1707 – 1778	Konstanz der Arten
Georges Baron de Curvier	1769 – 1832	Theorie, daß Arten entstehen und aussterben
Jean Baptiste de Lamarck	1744 – 1832	Entwicklung der Arten
Charles Darwin	1809 – 1882	Begründer der modernen Evolutionstheorie
Gregor Johann Mendel	1822 – 1884	Vererbungsgesetze

Tabelle 2.1: Wichtige Persönlichkeiten in der Geschichte der Evolution

ähnlich, aber nicht vollkommen identisch sind. Jede Art mußte daher mit einer mehr oder weniger starken Variationsbreite an Erbgut ausgestattet sein.

Weiterhin erkannte Darwin, daß sich erbliche Variationen einer Art, die sich im Überlebenskampf bewährt haben, in der Folgegeneration verstärkt wiederfinden. Dadurch können sich kleine Variationen der Individuen einer Art quasi addieren und führen so zur Optimierung einer Art an seine Umwelt.

Aus diesen Grundannahmen formulierte er die Theorie, daß der Kampf ums Überleben zu einer Zuchtwahl, also zu einer natürlichen Auslese (*Selektion*) führt, wodurch die tauglichsten Individuen überleben (*survival of the fittest*) und die besten Chancen erhalten, ihre Eigenschaften an die nachfolgende Generation zu vererben, was langfristig zu einer Optimierung der Art an ihre Umwelt führt.

Der Botaniker und Augustinerabt *Gregor Johann Mendel (1822 – 1884)* entdeckte wichtige Gesetzmäßigkeiten, die hinter der Weitergabe von Eigenschaften von einer Generation zur anderen stehen. Er entwickelte um das Jahr 1865 durch Versuche an Pflanzenhybriden die nach ihm benannten Vererbungsgesetze.

Nachdem wir nun einen Streifzug durch die Entwicklung des Evolutionsgedankens gemacht haben, wollen wir uns der Betrachtung der Evolution als Optimierungsprozeß widmen.

## 2.2 Die biologische Evolution

Die Leistung der biologischen Evolution besteht darin, das Leben immer besser an seine Umwelt anzupassen und weiterzuentwickeln, aber immer noch flexibel genug zu sein, um auf Veränderungen dieser zu reagieren.

In Anbetracht dessen, daß sich diese Strategie seit über 3,5 Milliarden Jahren bewährt hat, liegt die Schlußfolgerung nahe, daß es sich bei der Evolution um einen sehr flexiblen Optimierungsprozeß handeln muß. Um diese Strategie zu verstehen wollen wir uns zuerst die wesentlichen Elemente vor Augen führen und diese dann aus unserem Vorbild, der Natur, extrahieren.

Jedes Lebewesen gehört zu einer Gattung, einer Spezies von ähnlichen Lebewesen, die sich nur in kleinen Details unterscheiden. Innerhalb seiner Spezies

fungiert ein einzelnes Lebewesen als Träger und sogleich Tester einer möglichen Ausprägung seiner Gattung. Erreicht ein Lebewesen sein fortpflanzungsfähiges Alter und findet die zur Fortpflanzung geeigneten Randbedingungen (evtl. Geschlechtspartner, ...) vor, wird es seine spezielle Ausprägung an seine Nachkommen weitergeben können.

Bei dieser Weitergabe können, wie bei jeder Übertragung von Informationen, Fehler entstehen, so daß die Nachkommen eines Lebewesens nie exakt ihren Erzeugern entsprechen. Noch gravierender wird dies bei der zweigeschlechtlichen Fortpflanzung deutlich, bei der die Nachkommen eine Mischung der Eltern darstellen.

Über mehrere Generationen hinweg werden sich gewisse Merkmale, die zu einer höheren Fortpflanzungswahrscheinlichkeit führen, in der gesamten Gattung durchsetzen.

Gleichzeitig führt der Fehler bei der Erzeugung der Nachkommen dazu, daß immer wieder neue Merkmale entstehen, die sich entweder in der Gattung ausbreiten oder wieder verschwinden.

Je größer dieser Fehler ist, je langsamer werden sich Merkmale in der Gattung ausbreiten können.

Man kann den Evolutionsprozeß zusammengefaßt folgendermaßen beschreiben: Ein Lebewesen wird geboren, durch seine Umwelt auf Tauglichkeit geprüft (*Selektion*) und erzeugt gegebenenfalls eine Anzahl Nachkommen (*Rekombination, Mutation*), die den Prozeß von neuem durchlaufen. Da dies innerhalb einer Gattung viele tausendmal gleichzeitig geschieht, ergibt sich eine sehr hohe Parallelität, genauer eine Parallelität in der Größenordnung der aktuellen Anzahl an Mitgliedern einer Gattung.

Nachfolgend setzen wir uns detaillierter mit den oben eingeführten Begriffen und ihrer Rolle im Evolutionsprozeß auseinander.

### 2.2.1 Genotyp und Phänotyp

Jede Zelle eines Lebewesens, bis auf wenige Ausnahmen wie Bakterien und Blaualgen, tragen in ihrem Zellkern (Nukleus) den Bauplan des gesamten Lebewesens. Dieser Bauplan oder Erbgut, da es an die Nachkommen vererbt wird, ist in seinen Genen codiert. Als Träger der Gene dienen die Chromosomen. Die Chromosomen bestehen aus Nukleinsäuren und Proteinen. Der wichtigste Bestandteil der Chromosomen ist die Desoxyribonukleinsäure (DNS). Am Aufbau der DNS sind vier Basen beteiligt: Adenin (A), Guanin (G), Cytosin (C) und Thymin (T). Diese Basen stellen eine Art Alphabet eines Schriftsystems dar. Sequenzen von Basen codieren die Bausteine von Eiweißen (Sätze), die Aminosäuren (Worte). Die Eiweiße sind sogleich Botenstoffe und Baumaterial eines Lebewesens.

Durch einen komplizierten Prozeß wird mit Hilfe der Erbinformationen ein neues Lebewesen geschaffen. Genauer wird dies in [SHF94], Seite 46 bis 73 und [WHR<sup>+</sup>87] beschrieben.

Die Gesamtheit der genetischen Informationen eines Lebewesens nennt man Genotyp. Leider codiert ein Gen meist nicht allein ein Merkmal, so daß man die Merkmale eines Individuums nicht einer speziellen Gensequenz zuordnen kann.

Die Ausprägung von Merkmalen ist oft über mehrere Gene verstreut codiert, wobei sich meist auch Merkmale überlappen, d.h. eine Gensequenz ist Teil der Beschreibung mehrerer Merkmale. Auch beinhaltet der Genotyp oft mehrmalig die Beschreibung desselben Merkmales, wobei eines davon dominant ist.

Die Gesamtheit der Merkmalsausprägungen eines Individuums nennt man Phänotyp. Der Phänotyp ist somit die Summe aller beobachtbaren Eigenschaften eines Lebewesens.

## 2.2.2 Rekombination und Mutation

Bei der Zellteilung unterscheidet man zwei Arten des Kernteilungsprozesses: die Mitose und die Meiose.

Die Mitose ist eine erbgleiche Zell- und Kernteilung d.h. es wird identisches Erbgut an die Tochterzelle weitergegeben. Diese tritt bei der Entwicklung vielzelliger Organismen aus einer befruchteten Eizelle, dem Regenerationswachstum und der bei Pflanzen oder niederen Tieren häufig auftretenden ungeschlechtlichen Fortpflanzung auf.

Bei der Meiose werden nach der Verdoppelung der Chromosomen diese an einzelnen Stellen aufgetrennt und zufällig neu rekombiniert. Diesen Vorgang nennt man *crossing-over*.

Eine weitere wichtige Unterscheidung der Meiose von der Mitose ist die Reduzierung der Chromosomen auf einen einfachen Chromosomensatz. Alle Körperzellen und die befruchtete Eizelle besitzen einen doppelten Chromosomensatz, sie sind diploid. Würde nun eine unbefruchtete Eizelle und eine Spermazelle je einen doppelten Chromosomensatz besitzen, würde sich die Anzahl der Chromosomen und damit das Erbgut bei jeder sexuellen Fortpflanzung verdoppeln. Deswegen ist die Meiose die Voraussetzung für die Erzeugung von Geschlechtszellen und damit der sexuellen Fortpflanzung.

Beim sogenannten *crossing-over* brechen zwei Chromosomen an einer Stelle auf und binden sich an das jeweilige gegenseitige Chromosomenstück wieder an. Dadurch werden Stücke der Erbinformation zwischen Chromosomen ausgetauscht. Dieses Aufbrechen geschieht nicht an allen Stellen des Chromosoms mit der selben Häufigkeit. In Abbildung 4.2 wird dieser Vorgang nochmals verdeutlicht.

Die Rekombination des Erbgutes bei der geschlechtlichen Fortpflanzung ist im Gegensatz zur Mutation ein gewollter Vorgang. Sie dient dazu vorhandenes Genmaterial neu anzuordnen und Merkmale zu addieren. Im Gegensatz dazu steht die Mutation des Erbgutes, sie ist rein zufällig und erzeugt meist einen Defekt des Genmaterials, bringt aber zugleich neue Informationen in die Spielweise des Erbgutes und erhöht dessen Varianz.

Man unterscheidet drei Arten der Mutation: Die Chromosomenmutation, die Genommutationen und die eigentlichen Genmutationen.

Zuerst wollen wir die Chromosomenmutationen näher betrachten.

Durch äußere Einwirkungen wie Strahlen oder UV-Licht können Chromosomen verändert werden. Diese Mutationen nennt man Chromosomenaberration. Dabei entstehen Brüche in den Chromosomen, die zu Verlusten der Endstücke (Defi-

zienz), Ausbrechen von Zwischenstücken (Deletion), Inversionen von Teilen des Chromosoms (Inversion), Verlängerungen oder Vertauschungen von Teilstücken zwischen Chromosomen unterschiedlichen Types führen. Als Folge der Aberration können sich somit die Anzahl, die Reihenfolge und die Gene eines Chromosoms verändern. Meist hat dies eher schwerwiegende nachteilige Auswirkungen auf das betroffene Lebewesen.

Dennoch sollte man die Rolle dieser Mutationen und ihre eventuellen positiven Effekte nicht unterschätzen.

Zum Beispiel liefert die Chromosomenverlängerung und damit die Verdopplung von Genen der Evolution neue Spielmasse, da die eigentliche Information erhalten bleibt und auf den duplizierten Genen zusätzliche Varianten ausprobiert werden können.

Während der Meiose oder der Mitose können die Chromosomen fehlerhaft auf die Tochterzellen verteilt werden, so daß eine Zelle zuviele oder zuwenige Chromosomen erhält. Normalerweise ist die Anzahl der Chromosomen innerhalb einer Art konstant.

Diese Art der Mutation nennt man Genommutation. Meist führt sie zu einem erheblichen Defekt des betroffenen Lebewesens.

Die Genmutation stellt eine direkte Veränderung der Basensequenzen der DNS dar. Dabei werden durch die Einwirkung von UV-Licht, Strahlen oder chemischen Substanzen einzelne Basen verändert, eingefügt oder herausgerissen.

Das Austauschen einer einzelnen Base durch eine andere hat meist keine gravierenden Auswirkungen, da der Genetische Code hochgradig redundant ist. Durch vier Basen werden in einem drei Basen umfassenden Wort nur 20 Aminosäuren codiert. Dabei kann immer die letzte Base durch eine andere ersetzt werden, ohne daß eine andere Aminosäure codiert wird. Die übrigen Codes, sogenannte Nonsense Codes, dienen dazu einzelne Sequenzen von codierten Aminosäuren von anderen zu trennen.

Erheblicher wirken sich Einfügungen oder Entfernungen einzelner Basen aus einer Sequenz von Basen aus. Dadurch wird die gesamte Sequenz beeinflusst.

Einzelne Gene mutieren nicht mit derselben Wahrscheinlichkeit. Es gibt stabilere und labilere Gene. Ausserdem gibt es Gene, die die Mutationsneigung anderer Gene beeinflussen, sogenannte Mutatorgene.

Die Wahrscheinlichkeit, daß ein Gen mutiert liegt, in etwa bei  $10^{-5}$ . Da höhere Lebewesen jedoch sehr viele Gene besitzen können, ist die Wahrscheinlichkeit, daß eines oder mehrere der Gene eines Genotyps mutiert auftreten können, jedoch relativ hoch. Beim Menschen z.B. enthalten sogar bis zu 40% aller Keimzellen einer Generation ein mutiertes Gen. Die Mutationen wirken sich jedoch nur selten phänotypisch aus.

Ein Grund dafür ist die schon früher erwähnte Verteilung eines Merkmales über mehrere Gene, sogenannte Genwirkketten.

Ein weiterer Grund ist, daß neuentwickelte Ausprägungen eines Genes in der Regel zuerst rezessiv auftreten. Erst wenn sich diese Ausprägung in einer Population verbreitet werden diese Mutationen dominant. Es wurden auch schon Gene nachgewiesen, die dieses dominant werden von Mutanten bewirken, sogenannte Modifikatorgene.

Die Rückmutation eines Genes kann auch zur Behebung einer Mutation führen.

Der wichtigste Grund für die seltene Auswirkung von Mutationen auf den Phänotyp eines Lebewesens dürfte wohl die Selbstreperatur der DNS sein. Dabei werden zwei Arten unterschieden: Die Photoreparatur, die Schäden, die von UV-Licht bestimmter Wellenlänge hervorgerufen werden durch UV-Licht einer anderen Wellenlänge wieder aufhebt, und die Dunkelreparatur, die durch einen trickreichen Gebrauch der Doppelstrangstruktur herrührt und durch mehrere Enzyme ausgeführt wird.

### 2.2.3 Populationsgröße und Selektion

Die Selektion ist für die eigentliche Steuerung der Evolution verantwortlich. Sie bestimmt die Richtung, in der sich der Genpool der Population verändert. Durch die Bevorzugung oder die Benachteiligung einzelner Individuen bei der Fortpflanzung gehen dessen Erbanlagen mehr oder weniger in die Nachfolgeneration ein.

Nach dieser Beschreibung müsste man annehmen die Selektion wäre eine deterministische Komponente. Die Selektion ist aber keine konstante Größe, sondern wird durch Störungen verschiedenster Art beeinflusst. Einerseits können durch zufällige Ereignisse, wie dem frühzeitigen Tod eines optimal angepassten Vertreters seiner Gattung vor der Fortpflanzung durch einen Unfall wichtige Erbinformationen verlorengehen, andererseits verändert sich die Umwelt und damit eine der wichtigsten Steuergrößen der Selektion ständig. Diese zufälligen Größen stören den deterministischen Charakter der Selektion.

Die Selektion darf man nicht als Operator auf ein Individuum betrachten, vielmehr operiert sie auf dem gesamten Genpool einer Population. Eine Population ist die Gesamtheit aller theoretisch miteinander rekombinierbaren Individuen. Die Grenze wird sowohl durch die sexuelle Fortpflanzungsfähigkeit untereinander als auch durch die räumliche Trennung gezogen.

Je größer die Population ist, je breiter ist das Erbgut innerhalb dieser Population gestreut. Wenn eine Population über einen großen Genpool verfügt, kann sie besser auf eine Änderung der Umwelt und damit des Selektionsoperators reagieren. Damit wird dem nichtdeterministischen Charakter der Selektion entgegengewirkt.

In einer großen Population mit einem breitgestreutem Genpool herrscht ein großer Mutationsdruck. Es sind vielfältige Rekombinationsmöglichkeiten vorhanden und es können viele dieser Möglichkeiten ausprobiert werden.

In einer kleineren, homogenen Population wirkt sich die Selektion ungleich stärker aus. Eine zufällige Mutation kann sich innerhalb der Population viel schneller ausbreiten als in einer größeren. Der Gendrift in einer so beschaffenen Population ist viel größer. Unter Gendrift versteht man die auf Zufall beruhende signifikante Veränderung von Allelfrequenzen (Merkmalsausbildungen der gesamten Population). In einer kleinen Population ist folglich der Selektionsdruck größer.

Die Zahl der Mitglieder einer Population ist auch nicht immer konstant. Durch besonders günstige Umweltbedingungen kann es zu einer enormen Vergrößerung der Populationsgröße kommen. Andererseits kann eine Verschlechterung der

Umweltbedingungen zu einer Verringerung der Größe einer Population führen. Populationen können sich durch Nischenbildung trennen und so eigene Gattungen bilden.

Durch die Einflußnahme der Population auf die Umwelt kann es zu einem Rückkopplungsprozess zwischen der Population und der Selektion kommen.

Zusammenfassend kann man den Zusammenhang zwischen Populationsgröße und Selektion so beschreiben: Je kleiner eine Population ist, je größer ist der Selektionsdruck und je kleiner der Mutationsdruck.

## 2.3 Die Evolution als Optimierungsprozeß

Die Evolution ist eine Art *Suchprozeß* innerhalb des genetischen Informationsraums. Seine wichtigsten Operatoren sind die Rekombination, die Mutation und die Selektion.

Betrachtet man das Erbgut eines Individuums als Molekül eines Gases, stellt eine Population eine Wolke dieser Gase da.

Die Rekombination und die Mutation führen quasi zu einer Aufheizung der Wolke, so daß diese sich in alle Richtungen ausdehnen würde. Die Selektion begrenzt nun diese Ausdehnung durch eine unterschiedliche Abkühlung an verschiedenen Stellen. Dadurch wirbelt diese Wolke durch den Raum der möglichen Genkombinationen, gesteuert durch die Selektion wie eine Herde Schafe geleitet durch einen Schäfer und dessen Schäferhunde, der die jeweilig beste Weide für seine Schützlinge sucht.

Die Rekombination und die Mutation dient folglich zu einer Verbreiterung der Varianz des Genpools, während die Selektion eine Verringerung der Varianz bewirkt. Durch diesen Steuermechanismus wird der Genpool an gewissen Stellen ausgedünnt, während an anderen Stellen seine Ausbreitung gefördert wird.

Die Evolution gleicht, um in unserer Allegorie zu sprechen, einem blindem Schäfer, der nur am blöken seiner Schafe feststellen kann, ob sie eine gute Stelle zum Grasens gefunden haben. Er kombiniert aus dem Blökkonzert Stellen (Rekombination), die eine reiche Weide versprechen und treibt seine Herde mittels seiner Hunde (Selektion) zu diesen Stellen. Die Schafe folgen den Hunden aber nicht immer und geraten ab und an durch Zufall vom Weg ab (Mutation) und finden so vielleicht bessere Stellen zum Grasens.

Die Evolution ist eine Kombination aus gerichteten und ungerichteten Suchprozessen.

Die Mutation übernimmt dabei unter anderem die Aufgabe lokale Optima zu überwinden. Sie steuert der Homogenisierung des Erbgutes einer Population entgegen, indem sie eine zu schnelle Einpendelung auf ein Optimum verhindert. Gleichzeitig dient sie als eine Art Späher in weiter abgelegene Regionen, indem sie vorher nicht vorhandenes Erbgut erzeugt und neue Kombinationen erzeugen kann, die durch Rekombination nicht entstehen könnten.

Die Rekombination mischt vorhandenes Erbgut, dabei werden zusammengehörige Nukleotidketten seltener getrennt als nicht zusammengehörige. Dadurch be-

wirkt die Rekombination zwar ein zufälliges Mischen des Erbgutes, befolgt dabei aber gewisse statistische Gesetzmäßigkeiten (Mendelsche Gesetze). Somit steht die Rekombination hinsichtlich der Zielfindung zwischen der Mutation und der Selektion.

Die Selektion ist die eigentlich steuernde Größe im ganzen Evolutionsprozeß. Sie bestimmt die Richtlinien und Grenzen der anderen Operatoren (Mutation und Rekombination) durch die Auswahl der zur Fortpflanzung bestimmten Individuen und deren Vermehrungsrate. Indirekt bestimmt sie auch die Populationsgröße und reguliert damit die Gewichtung von Mutation und Rekombination.

## 2.4 Evolutionsalgorithmen

Bei der Nachahmung der biologischen Evolution durch mathematische und algorithmische Modelle sind im wesentlichen zwei Modelle hervorzuheben. Zum einen die Evolutionsstrategien (ES), die in den sechziger Jahren von Ingo Rechenberg an der Technischen Universität in Berlin entwickelt wurden. Zum anderen die Genetischen Algorithmen (GA), die von John Holland durch sein Hauptwerk *Adaption in Natural and Artificial Systems* 1975 begründet wurden.

## Kapitel 3

# Evolutionstrategien

Ingo Rechenberg erkannte den potentiellen Nutzen, der hinter der Adaption der Methoden der biologischen Evolution auf die Lösung technischer Probleme steht. Deshalb wählte er von Anfang an eher einen Ansatz, der von der praktischen Seite her kam. In seinem Standardwerk *Evolutionstrategie. Optimierung technischer Systeme nach dem Prinzip der biologischen Evolution*, das im Jahre 1973 erschienen ist, beschreibt er mehrere Möglichkeiten der Nutzung der biologischen Evolutionsmethoden zur Lösung technischer Probleme. Rechenberg versucht dabei die wesentlichen Elemente der biologischen Evolution zu extrahieren und diese zur Lösungsfindung technischer Optimierungsprobleme einzusetzen. Er betont dabei, daß nicht unbedingt die höchste Nachahmungsstufe zum größten Erfolg führen muß, sondern daß eine Anpassung auf die speziellen Gegebenheiten technischer Probleme von Nöten sei.

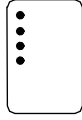
In den nachfolgenden Abschnitten möchte ich einige verschiedene Strategien Rechenbergs beschreiben. Diese werden in der Reihenfolge aufsteigender Nachahmungstiefe aufgeführt.

Außerdem hat Rechenberg ein vereinfachtes mathematisches Modell seiner Evolutionsstrategien entwickelt, welches im Abschnitt 3.4 näher erläutert wird.

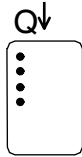
### 3.1 Rechenbergs Spielkartennotation

Zur Veranschaulichung verschiedener Evolutionsalgorithmen vergleicht Rechenberg diese mit einem Kartenspiel.

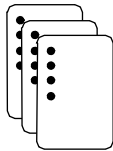
Nachfolgend wird eine leicht variierte Version erläutert, die auf Rechenbergs Notation basiert. Diese Variation soll einer beseren Übersichtlichkeit dienen. Zuerst werden einige Symbole eingeführt, anhand denen die Spielregeln der einzelnen Algorithmen in den nachfolgenden Abschnitten erklärt werden.



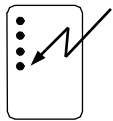
Die Grundlage eines Kartenspieles sind die Spielkarten selbst. Auf jeder Spielkarte werden die Chromosomen als Punkte angedeutet. Jedes Chromosom steht als Repräsentant einer nicht näher spezifizierten Anzahl von Parameterwerten des zu bearbeitenden Problems. Die Chromosomen stellen einen Lösungsvektor dar, der auf eine der möglichen Lösungen in einem Lösungsraum zeigt, der gleich der Dimension, der Anzahl der Chromosomen ist. Die Gesamtheit der Punkte bzw. Chromosomen steht für die Erbinformation des Individuums. Eine Karte repräsentiert also ein Individuum, das als Träger der Erbinformation dient.



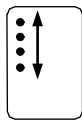
Durch einen Pfeil mit einem Q, der auf die Karte zeigt, wird die Bewertung des Individuums gekennzeichnet. Die Tauglichkeit bzw. Qualität kann auf der Spielkarte notiert werden (z.B.  $Q=9$ ).



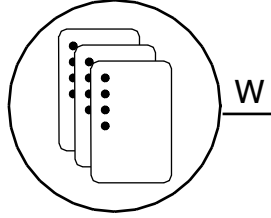
Mehrere hintereinander liegende Karten deuten eine Population an. Die Variable  $P$  unter dem Kartenstapel stellt die Größe der Population dar. Ihr kann auch ein fester Wert zugewiesen werden, was einer feststehenden Populationsgröße entspricht.



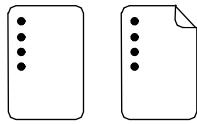
Will man einen Genmutationsoperator auf ein Chromosom anwenden, beschreibt man dies durch einen Blitz, der auf die Chromosomen zeigt. Daneben kann eine formale Beschreibung der Operation stehen. Eine Genmutation beeinflusst nur den Wert eines Chromosoms und nicht dessen Position auf dem Erbträger.



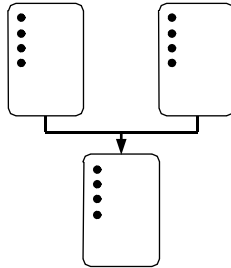
Eine Chromosomenmutation wird mit einem doppelten Pfeil neben den Chromosomen beschrieben. Die Beschreibung der Operation kann wie gehabt formal erfolgen. Eine Chromosomenmutation beeinflusst die Anzahl und Position der Chromosomen auf dem Erbträger.



Wird um eine Population ein Kreis gezogen deutet, man damit eine Auswahl aus der Population an. Die Auswahl einer Karte aus dem Kartenstapel entspricht der Selektion. Der Kreis symbolisiert eine Wahlurne oder ein Vergrößerungsglas. Nach welchen Kriterien die Auswahl getroffen wird, kann man durch einen Pfeil, der auf den Kreis deutet, mit einer vorangestellten Formel oder Symbolisierung beschreiben. Am häufigsten erfolgt die Auswahl anhand der Qualität, was durch ein vorangestelltes  $Q$ , oder durch einen Gleichverteilten Zufall, was mit einem  $W$  angedeutet wird.



Soll von einer Karte eine Kopie angefertigt werden, zeichnet man zwei nebeneinander liegende Karten, wovon die Kopie durch ein Eselsohr gekennzeichnet wird. Dies soll einen Abzug vom Original andeuten.



Eine Rekombination von Chromosomen zwei oder mehrerer Eltern wird durch eine Baumstruktur angedeutet. Das resultierende Individuum steht unter den Eltern. Die Operation, die zur Erzeugung des Nachkommens führt, kann durch eine formale Angabe erfolgen.

## 3.2 Grundlegende Evolutionsstrategien

In diesem Abschnitt werden die grundlegenden Evolutionsstrategien diskutiert. Die Überschriften sind in der formalen Schreibweise, die von Rechenberg und Schwefel entwickelt wurden, gehalten. Zu jeder Grundstrategie wird eine Abbildung in der Rechenberggraphiknotation angegeben.

### 3.2.1 $(\mu + \lambda)$ -Evolutionsstrategie

Die einfachste Form einer Evolutionsstrategie stellt die sogenannte „zweigledrige“  $(1 + 1)$ -Evolutionsstrategie dar. In Abbildung 3.1 wird diese Strategie verbildlicht.

Ausgehend von einem „Ur-Individuum“, das als Elter der nächsten Generation dient, zeugt dieses einen identischen Nachkommen durch Kopieren. Dies simu-

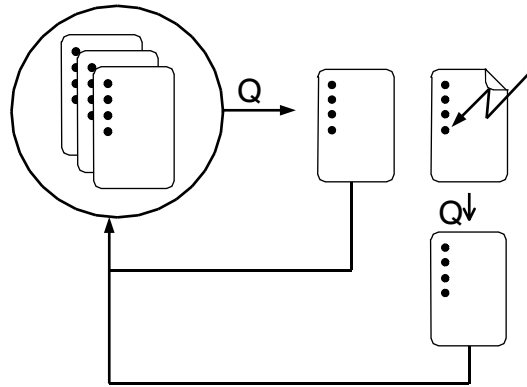


Abbildung 3.1: Die  $(1 + 1)$ -Evolutionstrategie

liert den Prozeß einer einzelligen Zellteilung. Dieser Nachkomme wird durch einen geeigneten Mutationsoperator mutiert. Nun unterscheidet sich der Nachkomme leicht von seinem Zeuger. Danach werden Elter und Kind durch eine Qualitätsfunktion bewertet. Der als besser bewertete geht als Elter der nächsten Generation in den Prozeß ein und der schlechtere wird vergessen. Sind beide gleich bewertet worden, überlebt einer nach dem Zufallsprinzip. Diese Schleife wird durch eine Abbruchbedingung, die sich nach der Anzahl der Generationen, der Rechenzeit oder einer zu erreichenden Qualität richtet, abgebrochen. Welche Abbruchbedingung gewählt wird richtet sich nach dem Problem. Oft ist eine Kombination der Abbruchbedingungen sinnvoll, z.B. gibt man eine maximale Rechenzeit als absolute Grenze vor und bricht die Schleife frühzeitig ab, wenn in einer vorgegebenen Anzahl von Generationen keine Verbesserungen mehr erzielt worden sind.

Rechenberg brachte die  $(1 + 1)$ -Evolutionstrategie auf folgende Kurzformel:

Elter erzeugt mutierten Nachkommen; der bessere überlebt.

Die  $(1 + 1)$ -Evolutionstrategie hat ihren Namen daher, daß Eltern und Nachkommen für die Auswahl des Stammvaters der nächsten Generation herangezogen werden.

Eine naheliegende Erweiterung der  $(1 + 1)$ -ES besteht darin,  $\mu$  Eltern  $\lambda$  Nachkommen erzeugen zu lassen. Dies durchbricht den seriellen Charakter der  $(1 + 1)$ -EV und läßt sich leicht auch als paralleles Verfahren realisieren. Diese Verallgemeinerung wird als  $(\mu + \lambda)$ -Evolutionstrategie bezeichnet.

In Abbildung 3.2 wird dieser Sachverhalt verdeutlicht. Aus einer Population von  $\mu$  Eltern wird  $\lambda$  mal ein Elter ausgewählt, der einen Nachkommen erzeugt. Dabei sind Mehrfachauswahlen zugelassen und jedes Elter wird mit derselben Wahrscheinlichkeit ausgewählt. Aus den  $\mu + \lambda$  Individuen werden nun die besten  $\mu$  besten ausgewählt und gehen als Eltern in die nächste Generation ein.  $\mu$  und  $\lambda$  repräsentieren beliebige ganze Zahlen, wobei  $\lambda \geq \mu \geq 1$  gelten muß.

Die Kurzformel dazu lautet:



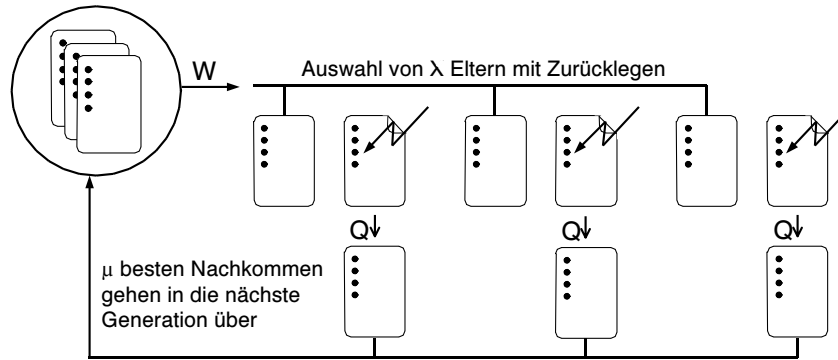


Abbildung 3.3: Die  $(\mu, \lambda)$ -Evolutionstrategie

### 3.2.3 $(\mu/\rho\#\lambda)$ -Evolutionstrategie

Im folgenden steht das # für „+“ oder „-“, da die nachfolgenden Betrachtungen für beide Varianten gelten.

Die bislang diskutierten Strategien machen keinen Gebrauch von der sexuellen Rekombination. Sie standen vielmehr für die Verdoppelungsstrategie von Einzellern.

Bei der nun zu betrachtenden  $(\mu/\rho\#\lambda)$ -ES erzeugen wie bisher  $\mu$  Eltern  $\lambda$  Nachkommen, indem sie zunächst  $\lambda$  Duplikate erzeugen, die dann noch mutiert werden.

Das Erzeugen der  $\lambda$  Duplikate geschieht nun, indem aus einer Gruppe von  $\rho$  Eltern das Erbgut für den Nachkommen rekombiniert wird.

Diese Rekombination kann nach verschiedensten Verfahren geschehen und wird im Abschnitt 3.3.3 näher erläutert.

### 3.2.4 Evolutionstrategien mit mehreren Populationen

So wie die bisher betrachteten Evolutionstrategien auf der Individuen-Ebene operieren, läßt sich auch ein Evolutionsschema angeben, das mit ganzen Populationen arbeitet.

Zum Beispiel könnte man zwei unabhängige Populationen generieren, die jeweils drei neue Populationen erzeugen. Dies geschieht analog zur Erzeugung einzelner Individuen. Jede Population könnte aus drei Individuen bestehen, die ihrerseits vier Nachkommen erzeugen. Aus diesen acht Individuen werden jeweils die vier Besten ausgewählt, die zu drei neuen Populationen zusammengefaßt werden. Aus diesen drei neuen Populationen werden dann die beiden Besten ausgewählt, um den Prozeß von neuem beginnen zu können.

Den Sachverhalt des Beispiels würde man in der Rechenberg-Schwefel-Formulierung in folgender Weise angeben:  $[2, 3(4 + 4)]$ .

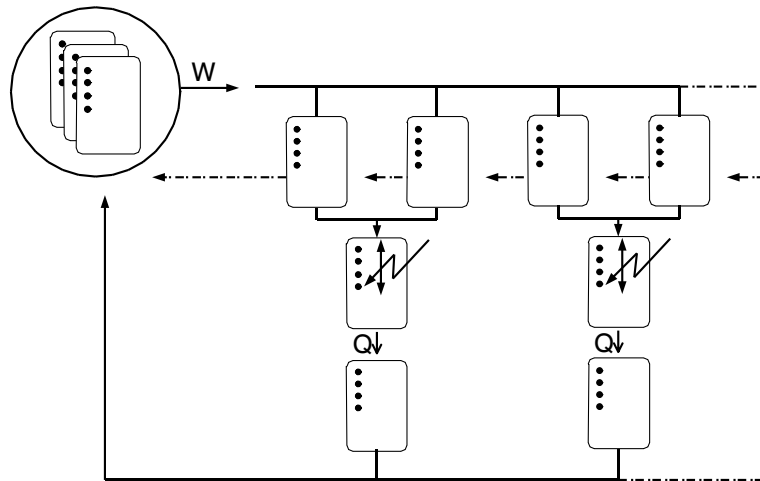


Abbildung 3.4: Die  $(\mu/\rho\#\lambda)$ -Evolutionstrategie

Dabei kennzeichnen die runden Klammern die Vorgehensweise für Individuen und die eckigen Klammern die für ganze Populationen.

Für diese Art der Evolutionstrategie muß man sich noch eine Möglichkeit zur Bewertung ganzer Populationen überlegen.

Eine Möglichkeit ist die durchschnittliche Qualität der Mitglieder der Population, die Standardabweichung oder einfach nur das beste Mitglied als Repräsentant.

### 3.2.5 Evolutionstrategien mit zeitweise isolierten Populationen

Eine weitere Möglichkeit der Erweiterung der Grundevolutionstrategie ist die Simulation zeitweise isolierter Populationen.

So kann man eine Population für eine Anzahl von Generationen isolieren und danach jeweils wie im vorhergehenden Abschnitt beschrieben verfahren. Die Dauer der Isolation wird durch eine hochgestellte Zahl angegeben.

Um das Beispiel des vorhergehenden Abschnitts aufzugreifen, würde dies bei einer jeweils 50 Generationen umfassenden Isolierung so aussehen:  $[2, 3(4+4)^{50}]$ .

Es ist auch möglich, einzelne Populationen komplett zu isolieren und nur einzelnen Individuen zwischen diesen nach bestimmten Vorgehensweisen auszutauschen. Diese Vorgehensweise läßt sich leider nicht in der Rechenber-Schwefel-Formulierung angeben, stellt aber eine wesentlich realitätsnähere Simulation der Natur da, da dort der Austausch zwischen Populationen auch über einzelne zuwandernde Individuen geschieht.

Die Strategie der zeitweisen Isolation von ganzen Populationen eignet sich hervorragend für die Implementierung auf Parallelrechnern, da sich jede Population auf einem Prozessor simulieren läßt und erst zum Zeitpunkt des Ablaufes der

Isolation eine Kommunikation mit den anderen Prozessoren notwendig wird.

Der Effekt der Isolation ist der, daß sich eine Population als eigenständiges Gebilde für eine gewisse Zeit im Suchraum ausbreiten kann und somit parallel zu den anderen neue Erkenntnisse gewinnt. Erst nach Ablauf der Isolation werden Informationen über vielversprechende Stellen im Suchraum ausgetauscht.

### 3.2.6 Standard-Evolutionsstrategie

Aufgrund des bisher beschriebenen Rechenberg-Schwefel-Formalismus läßt sich eine Unzahl von Varianten angeben. So kann man verschiedene Strategien auch schachteln und die  $,$  und  $+$  Operatoren in beliebiger Weise austauschen.

Schöneburg stellt aber in seinem Buch [SHF94] fest, daß es nicht sinnvoll ist eine Schachtelung tiefer als bis zur 2. Stufe vorzunehmen.

Somit lautet also die Standard-Evolutionsstrategie, mit der alle sinnvollen Möglichkeiten abgedeckt sind:  $[u/v\#w(x/y\#z)^n]$ .

### 3.2.7 Schwächen des Rechenberg-Schwefel-Formalismus

Leider kann man mit dem Rechenberg-Schwefel-Formalismus nicht beschreiben, aufgrund welcher Kriterien die Auswahl der einzelnen Individuen erfolgt oder die Bewertung ganzer Populationen geschieht. Auch lassen sich keine Algorithmen angeben, die auf der Basis isolierter Populationen, die einzelne Individuen austauschen basieren. Dadurch ist man gezwungen, diese Einzelheiten gesondert zu beschreiben.

### 3.2.8 Optimale Wahl der Parameter

Bei der im vorhergehenden Abschnitt angedeuteten Vielzahl von Möglichkeiten stellt sich die Frage nach der optimalen Wahl der Parameter  $\mu$ ,  $\lambda$  und  $\rho$ . Dazu läßt sich aber sagen, daß die nachfolgenden Empfehlungen nur für einfache, flache und gutartige Qualitätsfunktionen gültig sind. Bei anderen Qualitätsfunktionen muß man von Fall zu Fall durch Experimente oder Vorabüberlegungen eine günstige Wahl treffen.

In seinem Buch [Rec94] empfiehlt Rechenberg für einfache „flache“ Qualitätsfunktionen die  $(1\#5)$ -Evolutionsstrategie. Wenn man eine gleichmäßige Konvergenz erreichen möchte, empfiehlt er  $\lambda$  auf 10 zu erhöhen (d.h.  $(1\#10)$ ).

Ist eine möglichst gute Zuverlässigkeit für die Konvergenz gewünscht, so ist der Selektionsdruck ausschlaggebend. Als Selektionsdruck bezeichnet man den Quotienten aus  $\mu$  und  $\lambda$  ( $\frac{\mu}{\lambda}$ ). Er ist ein Maß für den Druck, den die Bewertungsfunktion auf die Population ausübt. Bei zu großer Wahl des Selektionsdruckes sammelt sich die Population oft sehr schnell in Nebenmaxima, und bei zu kleiner Wahl erhält man oft keine genügende Konvergenz.

Rechenberg gibt als Empfehlung für die Wahl des Selektionsdruckes Werte zwischen  $\frac{1}{5}$  und  $\frac{1}{3}$  an. Somit lassen sich folgende in Tabelle 3.1 aufgeführten Emp-

$\mu$	1	2	3	4	5	6	7	8	9	10
$\lambda$	5	8	11	13	16	19	22	25	27	30

Tabelle 3.1: Wahl des Selektionsdruckes  $\frac{\mu}{\lambda}$

fehlungen abgeben.

Für das Rekombinationsverhältnis einer  $(\mu/\rho\#\lambda)$ -ES empfiehlt er den Maximalwert  $\rho = \mu$ . Dabei erhält man, wenn  $\mu$  größer als 2 gewählt wird, mehr als zwei Eltern und muß sich dann eine geeignete Rekombinationsstrategie zurechtlegen, die über die einfache sexuelle der Natur hinausgeht (siehe Abschnitt 3.3.3)

### 3.3 Codierung, Mutations- und Rekombinationsoperatoren

Um ein Problem durch einen Evolutionsalgorithmus lösen zu können, muß man dieses erst einmal in einer geeigneten Form codieren.

Rechenberg wählte den aus ingenieurtechnischer Sicht einleuchtensten Weg. Die Parameter, welche für das Problem zu optimieren sind, werden als Vektor reeller Zahlen codiert. Dieser Vektor zeigt auf eine mögliche Lösung des Problems. Man schafft sich gewissermaßen einen Lösungsraum, dessen Dimension gleich der Anzahl der zu optimierenden Parametern ist. Die Chromosomen werden dabei mit den Vektoren identifiziert.

Bei diskreten Optimierungsaufgaben oder bei der Suche nach optimalen Strukturen eignet sich diese Codierung weniger. Hier sind binäre Codierungen geeigneter. Am grundsätzlichen Vorgehen einer Evolutionsstrategie ändert sich dabei jedoch nicht viel. Es müssen lediglich die Mutations- und Rekombinationsoperatoren an die jeweilige Codierung angepaßt werden.

Die frühen Evolutionsstrategen beschäftigten sich vorwiegend mit Optimierungen, bei denen die Codierung als reelle Vektoren verwirklicht wurde. Da dieses Vorgehen die Erbinformation eher auf die meßbaren Ausprägungen verdichtet, spricht man hier von einer *phänotypischen* Codierung.

Bei dieser Art der Codierung reduziert sich die Genmutation auf das Addieren und Subtrahieren kleiner Werte auf einzelne Elemente des Lösungsvektors. Chromosomenmutationen, die nur die Anordnung einzelner Elemente des Vektors verändern, können lediglich dazu beitragen, einzelne Parameter in eine besser zu beeinflussende Position zu bringen, falls die Genmutationsoperation nicht alle Elemente des Vektors mit der gleichen Häufigkeit verändert.

#### 3.3.1 Codierung

Bei der Codierung des Problems ist darauf zu achten, daß man eine Codierung wählt, bei der ähnliche Lösungen auch ähnlich codiert sind.

Diese Ähnlichkeit der Codierung läßt sich bei einer binären Betrachtung als *Hamming-Abstand* messen. Dabei hat z.B. das Byte *00000001* zum Byte *00000000* den Hamming-Abstand eins. Der Hammingabstand mißt sich als Anzahl der zu verändernden Bits um aus einem Codewort das nächste zu generieren.

Der Hamming-Abstand sollte den jeweiligen Mutationsoperatoren angepaßt werden, so daß eine solche Mutation keine allzu großen Sprünge der Qualitätsfunktion hervorruft. Dies hat den Vorteil einer geglätteten Qualitätsfunktion. Bei einer so geglätteten Qualitätsfunktion gibt es weniger Möglichkeiten, in winzigen Nebenmaxima festhängen zu bleiben. Außerdem kann sich der Lösungsalgorithmus schneller durch ein solches Qualitätsgebirge bewegen, d.h. es ist eine höhere Fortschrittsgeschwindigkeit zu erwarten.

### 3.3.2 Mutationsoperatoren

Wie im vorhergehenden Abschnitt erklärt sollte der Mutationsoperator an den jeweils verwendeten Code angepaßt werden.

Bei der Codierung einzelner Symbole als Bitfolgen sei ein einfacher bitbasierender Mutationsoperator angeraten. Dieser kann zum Beispiel so aussehen, daß er einzelne Bits invertiert. Zusätzlich kann man eine Maske für die Mutationswahrscheinlichkeit angeben, die die Häufigkeit einer Mutation jedes Bits des Codes angibt. In Abbildung 3.5 wird dies erläutert.

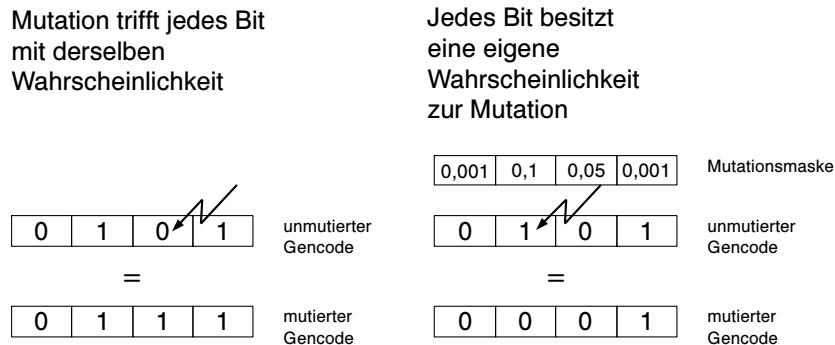


Abbildung 3.5: Links eine Bitmutation ohne und rechts mit Mutationsmaske

Bei einer reelwertigen Codierung der Parameter ist ein Mutationsoperator empfohlen, der eine Zufallsgröße auf diese addiert oder subtrahiert. Diese Zufallsgröße kann normalverteilt oder jeder anderen sinnvollen Verteilung entsprechen. Die Formel 3.1 soll dies exemplarisch für eine Normalverteilung veranschaulichen.

$$\vec{x} = \vec{x} + \vec{z} \quad \text{wobei} \quad w(\vec{z}) = \left( \frac{1}{\sqrt{2\pi}\sigma} \right)^2 \exp \left( -\frac{1}{2\sigma^2} \vec{z}^2 \right) \quad (3.1)$$

### 3.3.3 Rekombinationsoperatoren

Die Evolutionsstrategien gestehen der Rekombination keine so große Bedeutung innerhalb der Evolutionstheorie zu. Für sie dient die Rekombination lediglich dazu Erbinformationen zu mischen und durch die Population zu, verstreuen. Die Anhänger der Genetschischen Algorithmen schreiben dagegen der Rekombination sehr große Bedeutung zu und haben ausgeklügelte Theorien über die Wirkungsweise der Rekombination entwickelt. Deshalb wird für eine ausführliche Besprechnug auf Kapitel 4 verwiesen.

Eine gebräuchliche Möglichkeit der Rekombination bei den Evolutionsstrategien ist die Mittelwertbildung der zu rekombinierenden Vektoren. Eine andere ist das Zerbrechen des Vektors in einzelne Teilstücke und dessen neue Zusammensetzung. In Abbildung 3.6 wird dies näher erläutert.

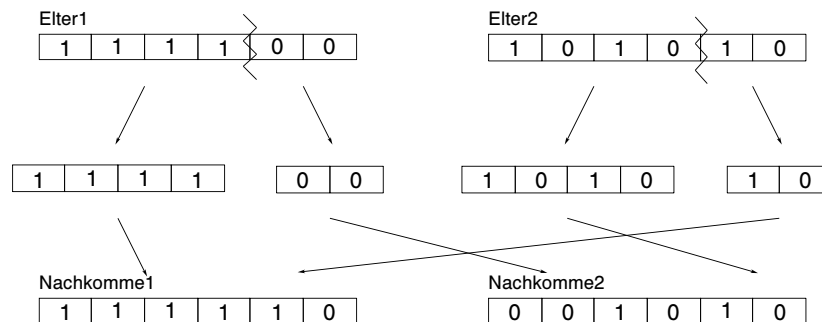


Abbildung 3.6: Einfache Rekombination reelwertiger Vektoren

## 3.4 Mathematische Theorie der Evolutionsstrategie

Um die durch Experimente und Überlegungen gewonnenen Erkenntnisse zu verifizieren, leitet Rechenberg in [Rec73] und [Rec94] ein matematisches Modell einer  $(1,1)$ -ES ab. Dabei geht es ihm vor allem darum, folgende Fragen zu klären:

- Gibt es einen Beweis für die Konvergenz einer ES?
- Gibt es eine optimale Parameterwahl und wenn ja, wie lauten die Werte?

In den folgenden Abschnitten werden die wichtigsten *Ergebnisse* zusammengefaßt. Wer sich für die Herleitung und eine weitere Studie des Rechenbergschen Modells interessiert, sei auf vorhergehend genannte Bücher verwiesen.

### 3.4.1 Grundlagen

Vorausgesetzt wird eine Optimierungsaufgabe, in der die Parameter eines  $u$ -dimensionalen Vektors  $\vec{x}$  aufgrund der Qualitätsfunktion  $Q(\vec{x})$  optimiert werden sollen.

Der Initialisierungsvektor wird als  $\vec{x}^{(0)}$  bezeichnet und seine Parameter  $(x_1, x_2, \dots, x_u)$  willkürlich gewählt.

Die aus diesem Punkt generierten mutierten Vektoren werden mit fortlaufender Hochzahl gekennzeichnet. Aus dieser Folge werden durch den Selektionsprozeß laufend Punkte eliminiert.

Um den Fortgang der Optimierung kenntlich zu machen, generieren wir eine zweite Vektorenfolge  $\vec{a}^{(m)}$ . Diese Punktfolge enthält immer nur die besten erzeugten Vektoren  $\vec{x}$ , wobei immer nur ein neuer Vektor  $\vec{x}$  in die Vektorfolge  $\vec{a}$ , als  $\vec{a}^{(m+1)}$  übernommen wird, wenn für diesen gilt:  $Q(\vec{x}^{(n)}) \geq Q(\vec{a}^{(m)})$ . Die Folge der Vektoren  $\vec{a}$  nennt man deshalb auch Aufenthaltspunkte, da sich der Prozeß gerade an dieser Stelle aufhält.

Die Mutation des Vektors  $\vec{x}$  geschieht, indem auf ihn ein Zufallsvektor  $\vec{z}$  adiert wird, der der Normalverteilung gehorcht. Somit gilt für  $\vec{z}$ :

$$w(\vec{z}) = \left( \frac{1}{\sqrt{2\pi}\sigma} \right)^2 \exp \left( -\frac{1}{2\sigma^2} \vec{z}^2 \right) \quad (3.2)$$

Mit den getroffenen Annahmen läßt sich somit der Mutations-Selektions-Algorithmus für den  $(\nu + 1)$ -ten Schritt wie folgt angeben:

Mutationskriterium:

$$\vec{x}^{(\nu+1)} = \vec{a}^{(\nu)} + \vec{z}^{(\nu)} \quad (3.3)$$

Selektionskriterium:

$$\begin{aligned} \vec{a}^{(\nu+1)} = \vec{x}^{(\nu+1)} & \quad \text{für} \quad Q(\vec{x}^{(\nu+1)}) \leq Q(\vec{a}^{(\nu)}) \\ \vec{a}^{(\nu+1)} = \vec{a}^{(\nu)} & \quad \text{für} \quad Q(\vec{x}^{(\nu+1)}) < Q(\vec{a}^{(\nu)}) \end{aligned} \quad (3.4)$$

Aufgrund des Selektionskriteriums 3.4 ergibt sich für die Qualitätswerte eine monotone nichtfallende Zahlenfolge:

$$Q(\vec{a}^{(0)}) \leq Q(\vec{a}^{(1)}) \leq \dots \leq Q(\vec{a}^{(\nu)}) \leq \dots \quad (3.5)$$

Angenommen sei, daß die Qualitätsfunktion  $Q$  ein Maximum  $\vec{a}^*$  hat, dann konvergiert die Folge 3.5 nach ihrer oberen Grenze  $\vec{a}^*$ .

Solange noch ein Gebiet  $\mathcal{G}$  mit höherer Qualität existiert, ist die Erfolgswahrscheinlichkeit  $W_e$  größer Null, daß dieser Punkt gefunden wird. Um aber eine realistische Chance zu erhalten, in endlich vielen Mutations-Selektions-Schritten eine Verbesserung zu erreichen, muß die Wahrscheinlichkeit  $W_e \geq \delta$  sein. Dafür läßt sich für alle  $\vec{a} \neq \vec{a}^*$  folgende mathematische Formulierung angeben:

$$W_e(\vec{a}) = \left( \frac{1}{\sqrt{2\pi}\sigma} \right)^2 \overbrace{\int \dots \int}_{\mathcal{G}}^u \exp \left( -\frac{1}{2\sigma^2} (\vec{x} - \vec{a})^2 \right) d\vec{x} \geq \delta \quad (3.6)$$

## Konvergenzkriterien

Nun sei ein sphärisches Umgebungsgebiet  $\mathcal{U}_\rho(\vec{a})$  mit  $(\vec{x} - \vec{a})^2 < \rho^2$  ( $\rho > 0$ ) gegeben. Dieses Gebiet  $U$  enthält einen gewissen Anteil des Erfolgsgebietes  $\mathcal{G}$ . Damit läßt sich, laut [Rec73], nun ein *notwendiges* und ein *hinreichendes* Kriterium angeben, daß von einem bestimmten Punkt des Selektions-Mutations-Verfahrens dieses mit einer bestimmten Mindestwahrscheinlichkeit weiterführt. Diese Erkenntnis ist besonders wichtig für kritische Punkte, zu denen auch Nebenmaxima der Qualitätsfunktion gehören.

**Notwendiges Kriterium:** Damit an einem kritischen Punkt  $\vec{a} \neq \vec{a}^*$  das Verfahren mit einer Wahrscheinlichkeit  $W_e \geq \delta$  weiterkonvergiert, ist notwendig, daß sich um diesen Punkt eine sphärische Umgebung  $\mathcal{U}_\rho$  mit einem darin eingeschlossenen Erfolgsgebiet  $\mathcal{G}_rho$  konstruieren läßt, so daß gilt:

$$\frac{\mathcal{G}_\rho}{\mathcal{U}_\rho} > \delta \quad (0 < \delta < 1) \quad (3.7)$$

**Hinreichendes Kriterium:** Damit an einem kritischen Punkt  $\vec{a} \neq \vec{a}^*$  das Verfahren mit einer Wahrscheinlichkeit  $W_e \geq \delta$  weiterkonvergiert, ist hinreichend, daß sich um diesen Punkt eine sphärische Umgebung  $\mathcal{U}_\rho$  mit einem darin eingeschlossenen Erfolgsgebiet  $\mathcal{G}_rho$  konstruieren läßt, so daß gilt:

$$\frac{\mathcal{G}_\rho}{\mathcal{U}_\rho} > \sqrt{\pi u} \delta \quad (0 < \delta < 1) \quad (3.8)$$

Die Kriterien 3.7 und 3.8 liegen verhältnismäßig weit auseinander. Dazwischen könnte das Verfahren mit der geforderten Mindestwahrscheinlichkeit weiterkonvergieren, muß aber nicht. Diese Unschärfe liegt daran, daß einzig und allein der anschauliche Begriff des sphärischen Umgebungsgebietes und des darin enthaltenen Erfolgsgebietes benutzt wurde. Laut Rechenberg mag es möglich sein, eine genauere Aussage herzuleiten. Dafür muß man aber mehr Informationen über die Qualitätsfunktion in die Voraussetzungen einbringen und spezialisiert damit die Aussage auf eine spezielle Topologie des Lösungsraums. Die Kriterien 3.7 und 3.8 können uns aber helfen, eine ungefähre Vorstellung einer für das Mutations-Selektions-Verfahren gut geeigneten Topologie der Qualitätsfunktion zu erhalten.

Leider helfen uns die hergeleiteten Kriterien 3.7 und 3.8 nicht sehr bei der Frage nach der Überwindbarkeit von Nebenmaxima weiter. Ein Zwischenmaxima wird sich immer störend auf die Konvergenz auswirken, da sich in einer solchen Umgebung in einem gewissen Abstand immer nur Punkte mit schlechterer Qualität befinden. Hier muß man nun die Testumgebung  $\mathcal{U}_\rho$  größer wählen, um eine genügend große Wahrscheinlichkeit zu erhalten, ein neues Erfolgsgebiet  $\mathcal{G}_\rho$  einzuschließen. Eine Möglichkeit, dies zu berücksichtigen besteht darin, eine Abhängigkeit zwischen der Verweildauer eines Punktes  $\vec{a}$  und des Mutationsvektors  $\vec{z}$  zu schaffen (siehe Abschnitt 3.5).

## Fortschrittsgeschwindigkeit

Im vorangegangenen Abschnitt wurden Kriterien hergeleitet, die sicherstellen, daß das Verfahren im Mittel nicht mehr als  $\frac{1}{5}$ -Mutationsschritte aufwenden muß um eine Qualitätsverbesserung zu erzielen.

Nun soll eine Definition für die Fortschrittsgeschwindigkeit angegeben werden:

$$\varphi = \frac{\text{Zielnäherung im Raumbereich } \mathcal{R}}{\text{Zahl der benötigten Mutationsschritte}} \quad (3.9)$$

Das Fortschrittsmaß  $\varphi$  gibt an, wieviele Mutationen im Mittel erforderlich sind, um einen geeignet abgegrenzten Raumbereich  $\mathcal{R}$  zu durchqueren.

Innerhalb des Gebietes  $\mathcal{R}$  muß man somit nur noch die Topologie der Qualitätsfunktion  $Q$  kennen. Somit läßt sich eine Qualitätsfunktion in mehrere Bereiche unterteilen, in denen sie durch evtl. einfache mathematische Funktionen angenähert werden kann.

Für dreiparametrische Qualitätsfunktionen  $Q(x_1, x_2, x_3)$  kann man sich diese Bereiche als Schalen geometrischer Körper (Quader, Kugel, Ellipsoid, Kegel, Paraboloid, ...) vorstellen, auf denen die Qualität jeweils denselben Wert besitzt.

Hat man sich einmal einen Katalog der Fortschrittsgeschwindigkeiten  $\varphi$  für solche Körper geschaffen, kann man an interessanten Stellen, durch Approximation der gegebenen Qualitätsfunktion an die gegebenen Körper, ein Maß für die Konvergenzgeschwindigkeit ablesen. Interessant wird dieser Katalog, wenn man zusätzlich noch die optimalen Mutationsstreuwerte für einzelne Topologien bestimmt. Nachfolgend werden zwei einfache Körper unter diesen Aspekten untersucht.

### 3.4.2 Korridormodell

Beim Korridormodell nimmt die Qualitätsfunktion die Form einer unendlich langen quadratischen Säule entlang einer Koordinatenachse an. Außerhalb der Säule ist die Qualität unendlich schlecht, innerhalb wächst sie in eine Richtung monoton an. Die Abbildung 3.7 soll dies verbildlichen.

Die Qualitätsfunktion läßt sich dazu formal folgendermaßen angeben, wobei  $F$  eine monoton steigende Funktion sei:

$$Q(y_1, y_2, \dots, y_n) = F(y_1) \quad \text{für} \quad \begin{array}{l} -b < y_2 < b \\ \vdots < \vdots < \vdots \\ -b < y_n < b \end{array} \\ Q(y_1, y_2, \dots, y_n) = -\infty \quad \text{sonst} \quad (3.10)$$

Durch umfangreiche Berechnungen und einige Abschätzungen gelangt man für die Fortschrittsgeschwindigkeit  $\varphi$  und die Erfolgswahrscheinlichkeit  $W_e$  zu nach-

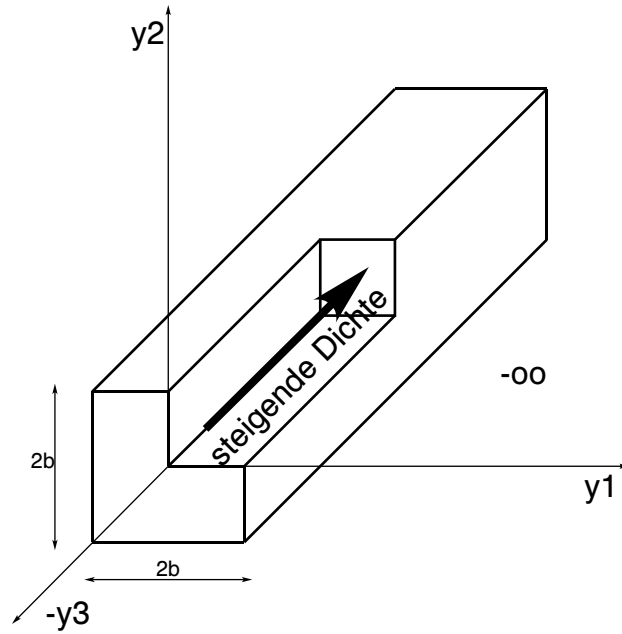


Abbildung 3.7: Qualitätsdichteverteilung beim Korridormodell

folgendem Ergebnis:

$$\varphi = \frac{\sigma}{\sqrt{2\pi}} \left(1 - \frac{1}{\sqrt{2\pi}} \frac{\sigma}{b}\right)^{u-1} \quad \left(\frac{\sigma}{b} \ll 1\right) \quad (3.11)$$

$$W_e = \frac{1}{2} \left(1 - \frac{1}{\sqrt{2\pi}} \frac{\sigma}{b}\right)^{u-1} \quad (3.12)$$

Durch Nullsetzen der ersten Ableitung der Fortschrittsgeschwindigkeit erhält man den optimalen Wert der Streuung  $\sigma$ , bei der die Fortschrittsgeschwindigkeit ihr Maximum hat:

$$\sigma_{\text{opt}} = b \frac{\sqrt{2\pi}}{u} \quad (3.13)$$

Für diesen Wert geben wir nun noch die Erfolgswahrscheinlichkeit an:

$$W_{e \text{ opt}} = \frac{1}{2e} \approx \frac{1}{5} \quad (u > 5) \quad (3.14)$$

Die Diskussion dieser Ergebnisse findet zusammen mit den des Kugelmodells in Abschnitt 3.4.4 statt.

### 3.4.3 Kugelmodell

Bei diesem Modell ist die Qualitätsdichte kugelsymmetrisch um den Ursprung des Koordinatensystems verteilt. Die Kugelschalen mit gleichem Radius weisen

dieselbe Qualität auf. Mit ansteigendem Kugelradius nimmt die Qualität ab. Die Abbildung 3.8 verbildlicht dies.

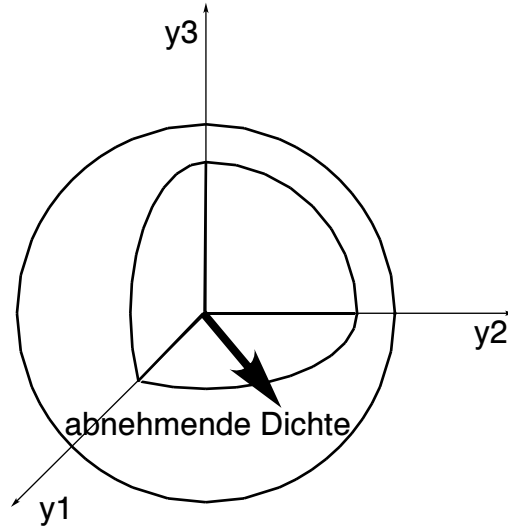


Abbildung 3.8: Qualitätsdichteverteilung beim Kugelmodell

Formal läßt sich die Qualitätsdichteverteilung folgendermaßen darstellen, wobei  $F$  eine monoton steigende Funktion sei:

$$Q(y_1, y_2, \dots, y_n) = -F(\sqrt{y_1^2 + y_2^2 + \dots + y_n^2}) \quad (3.15)$$

Hier sei nun ebenfalls wie beim Korridormodell die Fortschrittsgeschwindigkeit  $\varphi$  und die Erfolgswahrscheinlichkeit angegeben, wobei  $r$  der Aufenthaltspunkt ist:

$$\varphi = \frac{\sigma}{\sqrt{2\pi}} \left\{ \exp\left(-\left[\frac{u\sigma}{\sqrt{8}}\right]^2\right) - \sqrt{\pi} \frac{u\sigma}{\sqrt{8}r} \left[1 - \Phi\left(\frac{u\sigma}{\sqrt{8}r}\right)\right] \right\} \quad (3.16)$$

$$W_e = \frac{1}{2} \left[1 - \Phi\left(\frac{n\sigma}{\sqrt{8}r}\right)\right] \quad \left(u \gg 1, \frac{u\sigma^2}{r^2} \ll 1\right) \quad (3.17)$$

Durch Ableiten und Nullsetzen erhält man den optimalen Wert für die Standardabweichung  $\sigma$ :

$$\sigma_{\text{opt}} = r \frac{1,224}{u} \quad (3.18)$$

Dadurch ergibt sich für das Kugelmodell eine Erfolgswahrscheinlichkeit von:

$$W_{e \text{ opt}} = 0,270 \quad (3.19)$$

Im nachfolgenden Abschnitt werden diese Ergebnisse diskutiert.

### 3.4.4 Ergebnis

Es fällt bei Betrachtung der Gleichungen für  $\sigma_{\text{opt}}$  3.13 und 3.18 auf, daß beide proportional zur Dimension  $u$  der Qualitätsfunktion sind. Das bedeutet bei steigender Zahl  $u$  der Parameter, daß die wahre Größe eines Zufallsschrittes mit  $\sqrt{u}\sigma$  ansteigt, die Schrittgeschwindigkeit  $\varphi$  proportional mit  $\frac{1}{\sqrt{u}}$  abfallen muß.

Weiterhin ist zu bemerken, daß die Werte für optimale Erfolgswahrscheinlichkeit beim Korridormodell 3.14 und Kugelmodell 3.19 sehr nahe beieinander liegen. Als Approximation ließe sich der Wert  $\frac{1}{5}$  für  $W_{\varepsilon \text{ opt}}$  angeben. Jedoch gilt dieser Wert nur, wenn keine Störungen die Qualität verfälschen. Für gestörte Qualitätsfunktionen hält Rechenberg den Wert  $W_{\varepsilon} \approx \frac{1}{10}$  für angemessen. Dies bedeutet, daß bei einer  $(1 + \lambda)$ -ES  $\lambda$  für optimale Ergebnisse zwischen 5 und 10 zu wählen ist, sofern sich die Topologie des Lösungsraumes irgendwo zwischen dem des Korridormodells und dem des Kugelmodells befindet.

Das Kugelmodell bildet relativ genau die Umgebung eines Maximums nach, während die des Korridormodells bei relativ glatten Funktionen sein Äquivalent findet.

### 3.4.5 Evolutionsfenster

Bei der biologische Evolution bestimmt die das Lebewesen umgebende Umwelt die Tauglichkeitsfunktion. Setzt man voraus, daß durch das Korridor- bzw. Kugelmodell eine ausreichende Approximation dieser Tauglichkeitsfunktion gewährleistet ist, erhält man für sehr große  $u$ , was für die DNS von Lebewesen erfüllt ist, folgende Näherungen, wenn  $S = \sigma\sqrt{u}$  als Gesamtschrittweite angesehen wird:

Korridormodell:

$$\varphi^* = \frac{S^*}{\sqrt{2\pi}} \exp\left(-\frac{S^*}{\sqrt{2\pi}}\right) \quad \text{mit} \quad \varphi^* = \frac{\varphi u}{b}, S^* = \frac{S\sqrt{u}}{b} \quad (3.20)$$

Kugelmodell:

$$\varphi^* = \frac{S^*}{\sqrt{2\pi}} \left\{ \exp\left(-\left[\frac{S^*}{\sqrt{8}}\right]^2\right) - \sqrt{\pi} \frac{S^*}{\sqrt{8}} \left[1 - \Phi\left(\frac{S^*}{\sqrt{8}}\right)\right] \right\} \\ \text{mit} \quad \varphi^* = \frac{\varphi u}{r}, S^* = \frac{S\sqrt{u}}{r} \quad (3.21)$$

Trägt man diese beiden Funktionen auf einer logarithmischen Skala auf erhält man das sogenannte Evolutionsfenster (siehe Abbildung 3.9).

Betrachtet man dieses Evolutionsfenster, fällt auf, daß nur in einem sehr engen Bereich der Gesamtschrittweite  $S^*$  ein Evolutionsfortschritt  $\varphi^*$  erzielt wird. Durch eine Abschätzung mit Hilfe des Korridormodells kam Rechenberg zu dem erstaunswerten Ergebnis der Evolutionszeit von etwa 3,5 Milliarden Jahre. Dies ist eine relativ genaue Zahl, da die ältesten Fossilien auf etwa 3,2 Milliarden Jahre geschätzt werden.

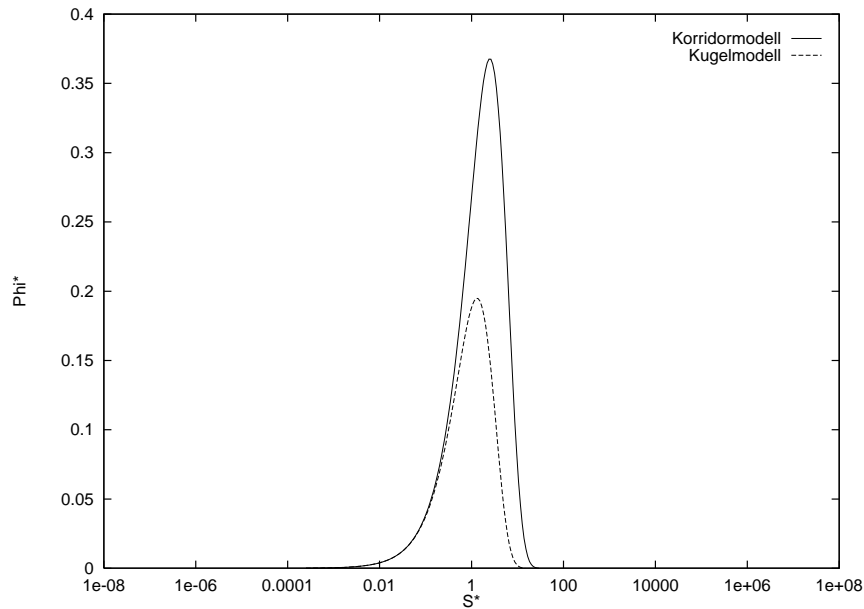


Abbildung 3.9: Fortschrittsfenster der Evolutionsstrategie

### 3.4.6 Vergleich des Mutations–Selektions–Verfahrens mit der Gradientenstrategie

Bei der Gradientenstrategie müssen im  $u$ -dimensionalen Fall  $u+1$  Qualitätsmessungen durchgeführt werden. Die erste Messung am gegenwärtigen Aufenthaltspunkt und  $u$  weitere an den Endpunkten der Prüfschritte  $\Delta x_1, \Delta x_2, \dots, \Delta x_u$ . In Richtung der steilsten Qualitätsänderung  $\Delta Q_1, \Delta Q_2, \dots, \Delta Q_u$  wird dann ein Arbeitsschritt der Länge  $s$  unternommen. Das Verfahren wiederholt sich zyklisch.

Die Gradientenstrategie wird aber nochmals ausführlich in Abschnitt 5.1.1 erklärt.

Für das Gradientenverfahren läßt sich die Fortschrittsgeschwindigkeit folgendermaßen angeben:

$$\varphi_{\text{grad}} = \frac{s}{u+1} \quad (3.22)$$

Um einen Vergleich der beiden Strategien anstellen zu können, ist es nötig, die Qualitätsfunktion für das Mutations-Selektions-Verfahren zu linearisieren, indem man beim Korridormodell  $b \rightarrow \infty$  bzw. beim Kugelmodell  $r \rightarrow \infty$  gehen läßt. Außerdem bildet man die mittlere Fortschrittsgeschwindigkeit. Dabei ergibt sich für beide Evolutionsstrategien derselbe genäherte Ausdruck:

$$\varphi = \frac{1}{\sqrt{2\pi}} \frac{s}{\sqrt{u}} \quad (3.23)$$

Aus Formel 3.22 und 3.23 erhält man, daß die Fortschrittsgeschwindigkeit beim der Gradientenstrategie mit  $\frac{1}{u}$  und bei den Evolutionsstrategien mit  $\frac{1}{\sqrt{u}}$  fällt. In Abbildung 3.10 wird dies noch einmal ersichtlich. Diese Betrachtung gilt aber nur unter der Annahme einer linearen Qualitätsfunktion. In der Abbildung ist der Bereich für kleine Anzahl Parameter mit

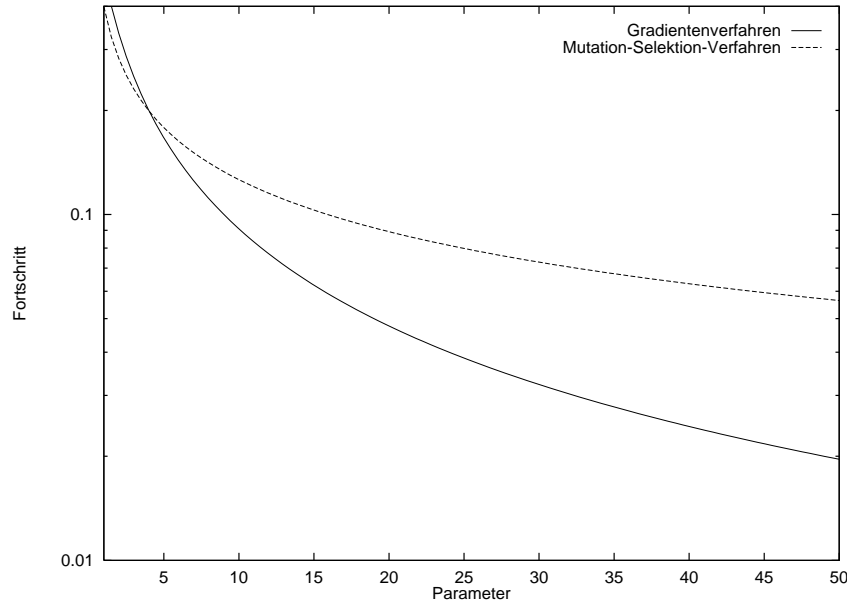


Abbildung 3.10: Fortschrittsgeschwindigkeit der Gradienten- und Evolutionsstrategie für eine lineare Qualitätsfunktion

Vorsicht zu genießen, da die Näherungen genau genommen nur für große Parameteranzahlen gelten.

Bei hohen Parameterzahlen  $u$  neigt also das Gradientenverfahren eher zu Fortschrittsverlusten als die Zufallsstrategie, während für kleine Parameterzahlen das Gradientenverfahren schnelleren Fortschritt zeigt.

### 3.5 Selbstorganisierte Evolutionsstrategien

Bei den bisher vorgestellten Evolutionsstrategien wurden nur die Lösungsparameter einer Optimierung unterzogen. Ebenso ist es in manchen Fällen von Vorteil, die Größen, die den Evolutionsalgorithmus steuern, selbst in diesen Prozeß miteinzubeziehen. Dies erscheint sinnvoll, da sich der Algorithmus somit selbst der Qualitätsfunktion anpaßt und eventuell, je nach Lösungsteilraum, der gerade durchlaufen wird, die Strategieparameter so anpaßt, daß eine möglichst gute Konvergenz erreicht wird. Dazu ist aber nötig, das Konvergenzverhalten mit in die Qualitätsfunktion einzubeziehen.

### 3.5.1 Mutation

Bei der Mutation kann man die Selbstorganisation z.B. durch Codieren der jeweiligen Varianz im Gencode erreichen. Eine weitere Möglichkeit ist die Verwendung einer Maske zur Mutation, die aussagt, welche Stellen mit welcher Wahrscheinlichkeit mutiert werden. Diese wird ebenfalls im Erbgut untergebracht. Man kann auch die Gesamtwahrscheinlichkeit, mit der eine Mutation auftritt, einbeziehen.

Durch die Selbstorganisation der Schrittweite, welche bei den Evolutionsstrategien durch die Mutation bestimmt wird, kann der Algorithmus sehr an Flexibilität und Konvergenzgeschwindigkeit gewinnen.

### 3.5.2 Rekombination

Auch hier läßt sich auf zahlreiche Arten eine Selbstorganisation erreichen. Durch die Verwendung von Masken und Sollbruchstellen oder etwa Vorlieben von bestimmten Codewörtern, mit anderen zu kombinieren, kann eine Anpassung an die jeweilige Topologie der Qualitätsfunktion erreicht werden.

### 3.5.3 Qualitätsfunktion

Um das Konvergenzverhalten bei der Bewertung der Individuen zu berücksichtigen, wie es bei den selbstorganisierenden Algorithmen notwendig ist, kann man vielfältige Strategien anwenden. Hier sollen nur wiederum einige Anregungen gemacht werden.

Eine Möglichkeit besteht darin, die Differenz der Qualität zwischen den Eltern oder die durchschnittliche Qualität der Eltern mit dem Kind zu bilden. Eine andere, falls das Maximum bekannt ist, die Differenz zwischen der Qualität des zu bewertenden Individuum und dem Maximum.

### 3.5.4 Initialisierung

Bei der Initialisierung von Evolutionsstrategien sollte beachtet werden, daß diese einen ganz entscheidenden Einfluß auf die Effektivität des Verfahrens hat.

Eine Möglichkeit die Initialisierung der Startpopulation zu optimieren ist es das Lösungsgebiet in gleichgroße Teilbereiche zu zerlegen und die Individuen gleichmäßig auf diese Gebiete zu verteilen. Auf diese Art kann der Algorithmus schon zu Beginn einen sehr guten, wenn auch noch sehr groben Überblick über die Topologie des Lösungsraumes gewinnen.

# Kapitel 4

## Genetische Algorithmen

Während Rechenberg sich für die praktische und anwendungsbezogene Seite der Evolution interessiert, stellen sich die Verfechter der Genetischen Algorithmen (GA) um Holland und Goldberg die Frage nach den Methoden der Codierung, Verarbeitung und Informationsweitergabe der Evolution.

*John Holland*, der als Vater der Genetischen Algorithmen gilt, interessierte sich dabei vor allem für den informationstheoretischen Aspekt, infolgedessen streben seine Forschungen danach ein möglichst exaktes Abbild der biologischen Evolution zu erhalten und dessen Mechanismen zu verstehen.

Im folgenden wird der Genetische Algorithmus der Einfachheit halber mit GA abgekürzt werden.

### 4.1 Grundlagen der Genetischen Algorithmen

Die Grundstruktur eines GA ähnelt denen der Evolutionsstrategien sehr. Die Feinheiten der Unterschiede liegen in der Herangehensweise an den jeweiligen Algorithmus. Während man bei den ES den Algorithmus dem Problem anpaßt, versucht man bei den GAs das Problem in das Schema des Algorithmus hineinzupressen. Dabei geht man von der Annahme aus, daß bei einer geeigneten Codierung des Problems und einer ausreichenden Nachahmungstiefe der biologischen Evolution eine optimale Lösungsstrategie zu finden sei. Dementsprechend legt man viel Wert auf die Suche einer optimalen Codierungsstrategie und untersucht die Methoden der Natur in dieser Hinsicht. Man vermutet, daß ein großer Teil des Erfolges der Evolution im Muster der Selbstorganisation des Genetischen Codes verborgen ist.

### 4.1.1 Allgemeiner Pseudocode eines Genetischen Algorithmus

Der allgemeine Code eines GA ähnelt dem der ES. Aus einer Initialisierungspopulation werden, basierend auf der Fitneß der Individuen, Eltern ausgewählt, die durch Rekombination ihres Erbgutes Nachkommen erzeugen, die in die neue Generation eingehen. Dabei fällt auf, daß hier die Mutation fast keine Rolle spielt. In Abbildung 4.1 ist in einer Pseudoprogrammiersprache der grundsätzliche Aufbau eines GA abgebildet.

```
BEGIN /* Genetischer Algorithmus */
  (Generiere eine Initialisierungspopulation);
  (Berechne die Fitne"s aller Individuen);

  WHILE NOT fertig DO
    BEGIN /* Erzeuge eine neue Generation */

      FOR (Populationsgröße) / 2 DO
        BEGIN /* Reproduktions-Zyklus */
          (Wähle zwei Individuen zur Fortpflanzung aus);
          /* Bevorzuge dabei die mit höherer Fitne"s */
          (Rekombiniere die beiden Auserwählten und
            erzeuge zwei Nachkommen);
          (Berechne die Fitne"s der beiden Nachkommen);
          (Integriere die beiden Nachkommen in die Population);
        END;

        IF (Optimierungs-Ziel erreicht) THEN
          fertig := TRUE;

      END;

    END;

  END;
```

Abbildung 4.1: Allgemeiner Pseudocode eines Genetischen Algorithmus

### 4.1.2 Codierung

Für die GA werden alle Informationen, die das Individuum bzw. das Problem beschreiben, als binärer Code im Erbgut festgehalten.

Im Gegensatz zu den Evolutionsstrategen codieren die Anhänger der Genetischen Algorithmen auch reelwertige Parameter im Binärcode. So werden z.B., falls man eine Funktion mit drei Parametern  $F(x_1, x_2, x_3)$  optimieren will, jeder der drei Parameter als 10-Bit Wert in einem Chromosom der Größe 30-Bit codiert.

Da der Gencode meist nicht direkt das Individuum repräsentiert, sondern meist erst eine Übersetzung des Erbgutes in das eigentliche Individuum stattfinden muß

unterscheidet man zwischen Genotyp und Phänotyp. Der Genotyp ist die codierte Form des Problems, während der Phänotyp die direkt von der Fitneßfunktion bewertbare Form des Problems ist.

### 4.1.3 Fitneßfunktion

Die Fitneß eines Individuums ist ein Maß für die Fortpflanzungswahrscheinlichkeit. Sie ist zwar proportional zur Qualität des Individuums, kann aber noch andere Faktoren berücksichtigen. Auch ist sie keine absolute Funktion, sondern drückt meist eine Wahrscheinlichkeit aus.

### 4.1.4 Fortpflanzung

Während des Reproduktionszyklusses werden Individuen aus der Population ausgewählt und erzeugen unter Zuhilfenahme geeigneter Strategien Nachkommen. Nachfolgend werden zwei typische Strategien vorgestellt. Die Auswahl der Eltern erfolgt aufgrund der individuellen Fitneß der Individuen, dabei sind Mehrfachauswahlen zulässig. Individuen mit hoher Fitneß erhalten im Reproduktionszyklus eine höhere Wahrscheinlichkeit zur Fortpflanzung.

#### Crossover

Eine typische Strategie zur Rekombination des Erbgutes zweier Eltern ist das *single point Crossover*. Dabei werden die Chromosomen des Erbgutes der Eltern jeweils an einer zufälligen Stelle aufgetrennt und die entstandenen vier Chromosomenstücke so auf die beiden Nachkommen verteilt, daß zwei neue, von den Eltern unterschiedliche, Individuen entstehen.

Die Wahrscheinlichkeit eines Crossovers wird typischerweise zwischen 0.6 und 1.0 gewählt, so daß jedes Individuum auch die Chance erhält, unverändert in die neue Generation einzugehen.

In Abbildung 4.2 wird dieses Vorgehen anhand eines 8-Bit großen Chromosoms demonstriert. Aus zwei Eltern werden zwei Nachkommen durch single point crossover erzeugt. In Abbildung 4.3 wird die Fitneß der Eltern und Nachkommen in einem Graphen mit der Qualitätsfunktion:

$$f(x) = \frac{\text{sgn}(x - 128)}{x - 128}$$

dargestellt. In Tabelle 4.1 werden die Werte zu den angegebenen Abbildungen im Detail angegeben.

#### Mutation

Nachdem aus den Eltern zwei Nachkommen erzeugt wurden, durchlaufen diese den Mutationsprozeß. Dabei wird ein Zufallopoperator auf die Gene angewandt. Meist handelt es sich um eine Bitinversion, die jedes Bit mit einer sehr geringen

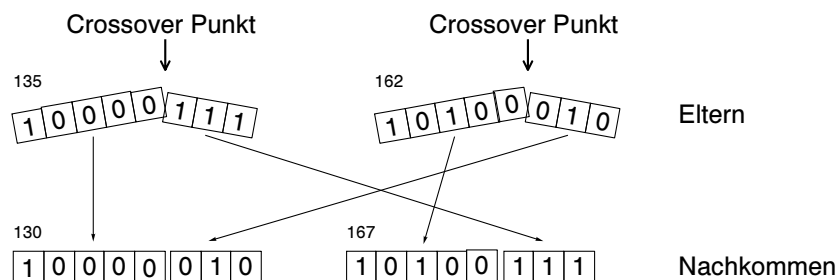


Abbildung 4.2: single point crossover

Individuum	x	Fitneß	Chromosom
Elter 1	135	0.143	10000111
Elter 2	162	0.029	10100010
Nachkomme 1	130	0.500	10000010
Nachkomme 2	167	0.026	10100111

Tabelle 4.1: Details des „single point crossover“ aus Abbildung 4.2 und Diagramm 4.3

Wahrscheinlichkeit (typisch 0,001) trifft. In Abbildung 4.4 wird dies an einem Bit verdeutlicht.

In der Theorie der GA mißt man dem Mutationsoperator nur eine untergeordnete Rolle zu. Er dient hier nur dazu, daß jeder Punkt des Lösungsraumes eine größere Auftrittswahrscheinlichkeit als Null erhält.

#### 4.1.5 Konvergenz

Nachdem ein Genetischer Algorithmus gestartet wurde, konvergiert die durchschnittliche Fitneß und die des besten Individuums gegen das globale Optimum.

Unter Konvergenz verstehen die Anhänger der Genetischen Algorithmen das Annähern der Population an eine Einheitlichkeit.

Über ein Gen, welches durch eine Bitkette bestimmter Länge repräsentiert wird, nachdem es in 95% des Erbgutes der Individuen einer Population nachweisbar ist, gesagt, es sei konvergiert.

Von einer konvergierten Population wird gesprochen, wenn alle Gene konvergiert haben. Abbildung 4.5 zeigt dieses Verhalten in einem Graphen.

## 4.2 Funktionsprinzipien

Es gibt noch keine fundamentale anerkannte Theorie, die die GAs erklären kann. Hier werden aber einige Funktionsprinzipien beschrieben, die dabei helfen, Genetische Algorithmen zu verstehen.

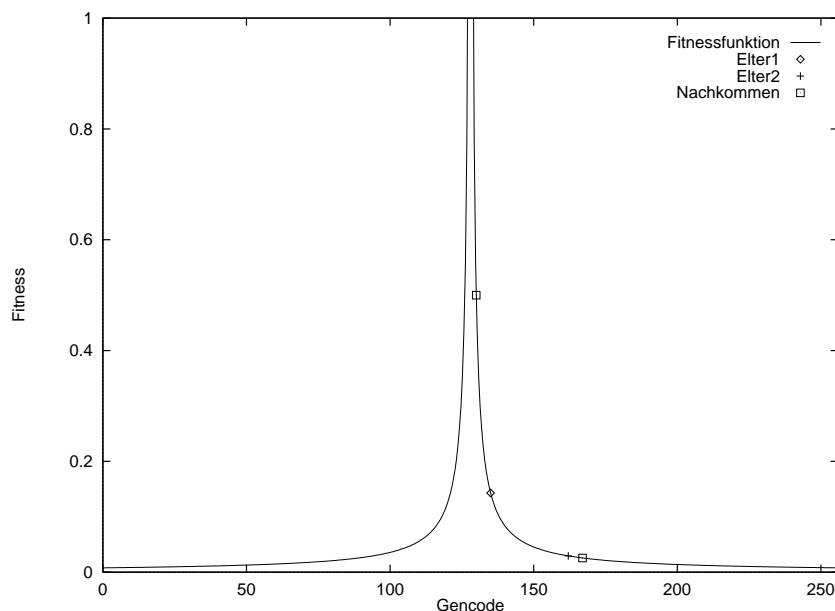


Abbildung 4.3: Fitneß der Eltern und Nachkommen nach crossover

#### 4.2.1 Das Schemata-Theorem und Implizite Parallelität

Unter einem Schema versteht man eine Folge von „0“, „1“ Bits und dem „don't care“ Zeichen „#“. Ein Chromosom enthält ein Schema, wenn es auf dieses paßt. z.B. enthält das Chromosom 1010 unter anderen die Schemata 10###, #0#0, ###0 und 101#. Die *Länge* eines Schemas ist die Anzahl der Symbole. Unter der *Ordnung*  $o(h)$  des Schemas  $h$  verstehen wir die Anzahl der nicht „don't care“-Symbole „#“ (für die Beispiele: 2, 2, 3, 1). Die *definierte Länge*  $l(h)$  ist der Abstand der äußersten „don't care“-Symbole (für die Beispiele: 2, 3, 1, 3).

Die wichtigste Interpretation der Schemata als Hyperebenen wird oft auch als Unterraum bezeichnet. So sind die oben aufgeführten Schemata Unterräume des  $M^4$ .

Ein Chromosom  $x = \langle x_1, \dots, x_n \rangle \in M^n$  ist eine *Instanz* eines Schemas  $h \in \{0, 1, \#\}$ , falls  $\{i \mid i \leq n, x_i \in x, x_i \neq \#\} = \{i \mid i \leq n, x_i \in h, x_i \neq \#\}$  ist. Für den  $M^n$  gibt es  $3^n$  Schemata. Jedes Element  $\langle x_1, \dots, x_n \rangle$  aus  $M^n$  stellt eine Instanz von  $2^n$  Unterräumen des  $M^n$ . Folglich repräsentiert eine Population von  $p$  Chromosomen der Länge  $n$  jeweils zwischen  $2^n$  und  $p2^n$  Schemata.

Der Fall  $2^n$  tritt ein, falls alle Chromosomen identisch sind, und der Fall  $p2^n$ , falls alle Chromosomen der Population verschieden sind.

Bei einer Population der Größe  $p$  werden pro Generation Schemata in der Größenordnung von  $p^3$  verarbeitet. Diesen Umstand bezeichnet man als *implizite Parallelität*. Wer an der Herleitung dieser Beziehung interessiert ist sei auf [Hol75] oder [Gol89] verwiesen.

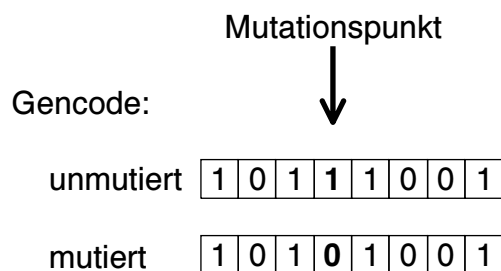


Abbildung 4.4: Mutation durch Bitinversion

Wie müssen die Schemata beschaffen sein, um eine möglichst gute Überlebenschance zu erhalten?

Es bezeichne  $m(h, t)$  die Instanzen des Schemas  $h$  zur Zeit  $t$ .  $f(h)$  sei die durchschnittliche Fitneß aller Instanzen von  $h$  zur Zeit  $t$  und  $f_{\text{mittel}}$  die durchschnittliche Fitneß der Population.  $p_c$  und  $p_m$  bezeichnen die jeweils auf die Chromosomen bezogene Crossover- bzw. Mutationswahrscheinlichkeit. Damit ergibt sich das *Schemata-Theorem* von John Holland:

$$m(h, t + 1) \geq m(h, t) \frac{f(h)}{f_{\text{mittel}}} \left( 1 - p_c \frac{l(h)}{n - 1} - p_m o(h) \right) \quad (4.1)$$

Wir wollen nun diese Gleichung interpretieren. Wenn die fixen Positionen der Schemata verantwortlich für eine überdurchschnittliche Fitneß ihrer Instanzen sind, so wird ein Schema bei einem proportional zur Fitneß gestalteten Heiratschema überdurchschnittlich oft Nachkommen erzeugen. Dadurch wird sich ein gutes Schema überdurchschnittlich oft in der Population wiederfinden. Leider stimmt dies nicht ganz, da der Prozeß durch die Mutations- und Crossoveroperationen gestört wird.

Wie man sich leicht überlegen kann, kann werden aber Schemata mit hoher definierter Länge und hoher Ordnung wesentlich anfälliger gegen diese Störungen sein, so daß Schemata niedriger Ordnung und niedriger definierter Länge eine erhöhte Überlebenschance erhalten.

Die Schlußfolgerung, die man daraus ziehen kann, bezieht sich auf die Codierung. Ein Code sollte möglichst kompakt sein und funktionell voneinander abhängige Größen möglichst in unmittelbarer Nachbarschaft codiert sein, so daß sich kleine Schemata bilden können.

## 4.2.2 Die Building-Block Hypothese

Wie aus dem Abschnitt 4.2.1 ersichtlich wird, besteht eine der Stärken der Genetischen Algorithmen darin, gute Blöcke von Schemata zu finden und diese zu immer besseren Individuen zusammenzubauen. Damit dies funktionieren kann, müssen aber zwei Hauptkriterien beim Codieren erfüllt sein:

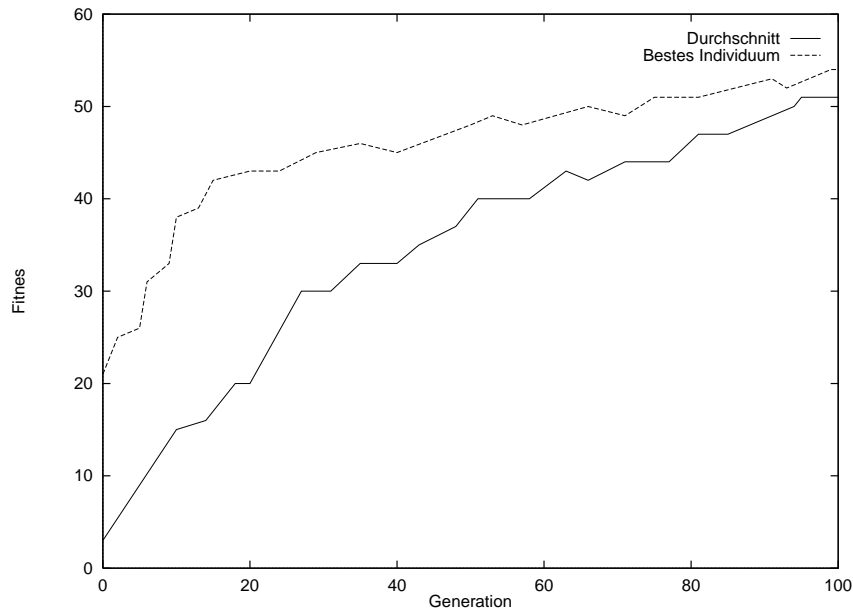


Abbildung 4.5: Konvergenzverhalten eines Genetischen Algorithmus

1. Voneinander abhängige Gene müssen nahe beieinander codiert sein.
2. Es darf keine starken Abhängigkeiten zwischen den Genen geben.

Die Forderung (1) ergibt sich unmittelbar aus den Ergebnissen des Schemata-Theorems.

Wie die Abhängigkeit der Gene untereinander zu verstehen ist erkennt man am besten an einem Beispiel. Bei einer Fledermaus z.B. müssen zur Ausprägung der Echoortung ein feines Gehör, ein Stimmorgan, das Ultraschall Schreie ausstoßen kann und eine Dekodiereinheit im Gehirn vorhanden sein. Hier besteht eine starke Abhängigkeit der Gene, die die verschiedenen genannten Merkmale ausprägen.

Die einzelnen Merkmale allein beeinflussen die Gesamtfitneß des Individuums nicht besonders. Ein Schema, welches die Kombination aller Merkmale beschreibt, wird zudem sehr lang und ist deswegen sehr anfällig. Logisch richtig scheint deshalb die Schlußfolgerung, daß die Erbinformationen, welche die Echoortung bei der Fledermaus beschreiben, dicht beieinander codiert sein müssen oder es sonst einen Mechanismus geben muß, der die Entwicklung solcher kombinierten Fähigkeiten begünstigt. Dieser Mechanismus könnte in der Fitneßfunktion liegen, die so beschaffen ist, daß unabhängig voneinander jede der benötigten Fähigkeiten eine signifikante Steigerung der Fitneß hervorruft.

Leider können diese Voraussetzungen bei den wenigsten Problemen ohne weiteres erfüllt werden.

Bei reelwertigen Parametern zum Beispiel, die als Binärzahlen codiert werden, sind beide Bedingungen nicht unbedingt erfüllt. So können hier z.B. aus zwei

guten Schemata durch Crossover ein wesentlich schlechteres entstehen. Ein Beispiel dafür ist in [SHF94] auf Seite 216 zu finden und soll hier kurz angedeutet werden:

Gesucht sei das Optimum der Funktion:

$$f(x) = x^2 \quad \text{mit} \quad x \in [0, 5] \quad (4.2)$$

Die benutzte Fitneßfunktion sei:

$$f(y) = \frac{1}{1 + |25 - y_{10}|} \quad (4.3)$$

Gegeben seien ebenfalls die vier nachfolgend aufgeführten Vektoren mit ihrer Fitneß:

$$f(c_1) = f(\langle 010001 \rangle) = \frac{1}{9} \quad (4.4)$$

$$f(c_2) = f(\langle 100000 \rangle) = \frac{1}{8} \quad (4.5)$$

$$f(c_3) = f(\langle 001001 \rangle) = \frac{1}{17} \quad (4.6)$$

$$f(c_4) = f(\langle 010000 \rangle) = \frac{1}{10} \quad (4.7)$$

Bei einem one-point-crossover zwischen den Chromosomen  $c_1$  und  $c_2$  nach der zweiten Stelle ergeben sich:

$$f(c'_1) = f(\langle 100001 \rangle) = \frac{1}{9} \quad (4.8)$$

$$f(c'_2) = f(\langle 010000 \rangle) = \frac{1}{10} \quad (4.9)$$

Aus den hinsichtlich ihrer Fitneß schlechteren Chromosomen  $c_3$  und  $c_4$  ergeben sich durch Crossover die Chromosomen:

$$f(c'_3) = f(\langle 000000 \rangle) = \frac{1}{26} \quad (4.10)$$

$$f(c'_4) = f(\langle 011001 \rangle) = 1 \quad (4.11)$$

Wobei  $c'_4$  das Optimum der Funktion  $f(x)$  darstellt. Bei diesem Beispiel ergibt sich das Optimum aus zwei schlechten Individuen. Hier wäre es also nicht sinnvoll gewesen, die schlechten Individuen auszusondern. Auch hat hier die Abhängigkeit der guten Blöcke nicht gereicht, die Gesamtfitneß so zu beeinflussen, daß diese groß genug geworden ist, um diese guten Blöcke zu erkennen.

Diese Probleme wurde von den Anhängern der Genetischen Algorithmen schon frühzeitig erkannt, so daß eine der Hauptaufgaben bei den GA die Suche nach einer **optimalen Codierung** und einer **geeigneten Fitneßfunktion** ist.

Man darf aber dennoch nicht das Potential vernachlässigen, das hinter der Building-Block Hypothese steckt, da sich durch Kenntnis dieses Verhaltens und Berücksichtigung bei der Implementation eine starke Steigerung der Effektivität der Problemlösung erreichen läßt.

### 4.2.3 Ausbreitung und Kombination

Ein guter Optimierungsalgorithmus muß zwei wichtige Techniken vereinen. Einmal die Kombination bereits untersuchter Lösungen, zum anderen die Suche nach neuen Lösungsmöglichkeiten. Wir wollen diese beiden Techniken als Kombination und Ausbreitung bezeichnen.

Idealerweise muß ein guter Algorithmus ein Optimum der Verteilung der bereitstehenden Kapazität auf die beiden Techniken finden.

Im Falle der Genetischen Algorithmen wirkt die Rekombination (Crossover) als Kombinationsoperator und die Mutation als Ausbreitungsoperator. Durch Crossover werden bereits bekannte Lösungen zu neuen Lösungen kombiniert, dabei kommen meist keine neuen Gene ins Spiel. Die Mutation sorgt dafür, daß neue Genkombinationen ins Spiel kommen.

Ist das Verhältnis zwischen den beiden Operatoren zugunsten einem unausgewogen, driftet der Algorithmus ab.

Wird der Rekombinationsoperator z.B. zu stark bevorzugt, tendiert der Algorithmus allzusehr dazu, zu konvergieren. Starke Gene dominieren sehr schnell die Population und schon bald ist der Genpool der Population so gleichförmig, daß keine neuen Kombinationen mehr hervorgebracht werden. Dies ist in der Natur, bei einer kleinen abgeschlossenen Population oft der Fall. Diese Population kann dann auf Veränderungen nicht mehr schnell genug reagieren und stirbt deshalb bei kleinen Veränderungen der Umwelt sehr schnell aus.

Wird jedoch der Mutationsoperator zu stark, so driftet der Suchprozeß sehr schnell zu einem reinem Rateprozeß ab, da die neuen Schemata gar nicht mehr ausgewertet werden können.

Laut [Hol75] besitzt ein GA unter folgenden Bedingungen das optimale Verhältnis zwischen Ausbreitung und Kombination:

1. Die Population ist unendlich groß.
2. Die Fitneßfunktion muß den Lösungsraum hinreichend gut abbilden.
3. Die einzelnen Gene dürfen keine starken Abhängigkeiten voneinander haben.

## 4.3 Praktische Aspekte

Bei der Simulation von Genetischen Algorithmen treten oft Probleme auf, die auf den ersten Blick noch nicht ersichtlich waren. In den folgenden Abschnitten werden diese Probleme angesprochen.

### 4.3.1 Fitneßfunktion

Eine der zwei empfindlichsten Teile eines Genetischen Algorithmus ist die Fitneßfunktion. Sie beeinflußt empfindlich das Konvergenzverhalten und entscheidet über die Güte der Lösung. Deshalb wollen wir zuerst einen Blick auf die

verschiedenen Probleme, die sich bei der Wahl der Fitneßfunktion ergeben können, werfen.

### **Einfache Fitneßfunktion**

Im einfachsten Fall entspricht die Fitneßfunktion der Qualitätsfunktion. Wenn diese mit einem passablen Aufwand berechnet werden kann, sollte man die Fitneßfunktion aus der Qualitätsfunktion berechnen. Lediglich eine Gewichtung oder eine Stauchung bzw. Quetschung der Qualitätsfunktion sollte bei Bedarf benutzt werden, um das Konvergenzverhalten anzupassen.

### **Kombinierte Fitneßfunktion**

Bei vielen realen Problemen besteht die Aufgabe nicht nur darin, ein einziges Kriterium zu optimieren, sondern setzt sich aus einer ganzen Menge an Anforderungen zusammen. In diesem Fall sollte man eine Gewichtung der einzelnen Kriterien vornehmen und diese dann zu einer einzigen Fitneßfunktion vereinen. Die Gewichtung sollte fix sein und sich nicht während des Optimierungsprozesses verändern, da ansonsten der Konvergenzprozeß gestört werden könnte.

### **Löchrige Fitneßfunktion**

Unter löchrigen Fitneßfunktionen versteht man Probleme, bei denen bestimmte Chromosomkombinationen unendlich schlechte Lösungen darstellen. Dies tritt besonders bei diskreten Reihenfolgeproblemen auf. Wenn man z.B. einen Stundenplan für eine Schule optimieren möchte, treten Kombinationen auf, bei denen derselbe Lehrer zur selben Zeit in verschiedenen Klassen sein soll oder ein Klassenzimmer doppelt besetzt wird.

Eine mögliche Lösung, um solche Probleme dennoch mit einem üblichen Genetischen Algorithmus behandeln zu können stellt, der Entwurf einer Fitneßfunktion dar, die solche Lösungen anhand des Aufwandes, der nötig ist, um einen solche ungültige in eine gültige Lösung überzuführen, bewertet. Man glättet in gewissem Sinne die Löcher aus, um so einen löcherlosen zusammenhängenden Lösungsraum zu erhalten.

### **Approximierte Fitneßfunktion**

Für manche zu optimierenden Probleme läßt sich keine exakte berechenbare Qualitätsfunktion angeben oder der Aufwand zur exakten Berechnung ist so hoch, daß sich eine Behandlung mit Genetischen Algorithmen nicht lohnt, da bei diesen mehrere tausend mal eine Berechnung notwendig ist.

In solchen Fällen lassen sich oft schon sehr gute Ergebnisse erzielen, indem man eine approximierete Qualitätsfunktion benutzt. Der Genetischen Algorithmus ist meistens tolerant genug, um die Fehler, die sich durch diese Approximation ergeben auszugleichen.

### **Unbekannte Fitneßfunktion**

In ganz schwierigen Fällen weiß man nicht, wie die Qualitätsfunktion eines Individuums auszusehen hat. In solchen Fällen wurden schon mit der Approximation einer Fitneß aus der der Eltern relativ gute Erfolge erzielt. Leider sind die Ergebnisse, die bei solchen Approximationen entstehen, meist sehr unzuverlässig. Durch gelegentliches Manuales Abstimmen der Ergebnisse lassen sich jedoch erhebliche Verbesserungen erzielen. Diese Methoden können aber nicht mehr als ein grobes Abschätzungsverfahren sein.

### **Kombinierte Bewertungskriterien**

Bei den meisten wirklichen Problemen ist es sinnvoll, mehrere der oben aufgeführten Möglichkeiten in geeigneter Weise zu kombinieren.

Ein Genetischer Algorithmus kann immer nur so gut sein wie seine Bewertungsfunktion.

Dies sollte man beim Entwurf von GAs immer im Gedächtnis behalten.

### **4.3.2 Streuung der Fitneß und Konvergenzverhalten**

Vor dem Start eines Genetischen Algorithmus muß eine Initialisierungs-Population erzeugt werden. Diese wird meist durch einen Zufallsprozeß erzeugt und sollte möglichst gleichmäßig den Lösungsraum ausfüllen. In den meisten realen Fällen wird der Lösungsraum aber nicht gleichmäßig ausgefüllt, da man nicht mit unendlich großen Populationen arbeiten kann. Dabei ergeben sich zwangsläufig Häufungen und Lücken. Dadurch entstehen Probleme, die zu einer frühzeitigen oder einer zu langsamen Konvergenz führen können.

#### **Frühzeitige Konvergenz**

Wenn in der Initialisierungs-Population ein Gebiet mit relativ guter Fitneß überdurchschnittlich gut besetzt wurde, passiert es sehr schnell, daß diese Lösung die Population beherrscht und zu einer frühzeitigen Konvergenz eines lokales Maximums führt. Um dies zu verhindern kann man die Qualitätsfunktion etwas stauchen. Damit werden lokale Maximas überproportional schlechter bewertet, als dies normalerweise der Fall wäre.

Eine andere Möglichkeit ist es nicht, von vornherein die besten Individuen zur Nachkommensbildung heranzuziehen, sondern auch schlechten Individuen eine reele Chance einzuräumen, Nachkommen zu bilden (siehe dazu Abschnitt 4.3.3).

#### **Langsame Konvergenz**

Eine Verzögerung der Konvergenz kann manchmal dazu führen, daß es zu keiner Konvergenz mehr kommt. In solchen Fällen muß die Qualitätsfunktion wieder

gestreckt werden. Man kann den Streck- oder Dehnungskoeffizienten auch durch einen geeigneten Automatismus steuern bzw. ihn selbst zum Gegenstand der Optimierung machen.

### 4.3.3 Auswahl der Eltern

Eine weitere Methode, auf die Konvergenz der Genetischen Algorithmen Einfluß zu nehmen, ist die Wahl des Eltern-Selektionsprozesses. Nach der Bewertung der Individuen durch die Fitneßfunktion tritt dieser Prozeß in kraft und bestimmt die Partner zur Zeugung von Nachkommen. Dieser Prozeß hat die Funktion eines Heiratsinstitutes. Anzumerken ist, daß bei den Genetischen Algorithmen von einer Auswahl mit Zurücklegen ausgegangen wird.

#### Direkte Methode

Die einfachste Methode besteht darin, jedem Individuum eine Fortpflanzungswahrscheinlichkeit proportional zur erreichten Fitneß zuzuteilen. Dabei kann es aber sehr schnell zu einer frühzeitigen Konvergenz kommen. Ein oder mehrere überdurchschnittlich gute Individuen dominieren sehr schnell den Genpool und führen zur vorzeitigen Konvergenz der Gene.

#### Fitneß-Rang

Die Probleme einer frühzeitigen Konvergenz können gemildert werden, indem den einzelnen Individuen entsprechend ihrer Fitneß ein Rang zugeteilt wird. Dabei erhält das beste Individuum den ersten Rang, das zweitbeste den zweiten und so fort. Die Fortpflanzungswahrscheinlichkeit wird nun entsprechend des Ranges vergeben. Dadurch erreicht man eine Linearisierung. Ein überdurchschnittliches Individuum kann maximal den ersten Rang erreichen und erhält somit auch nur eine festgelegte Höchstwahrscheinlichkeit zur Fortpflanzung.

#### Wettkampf

Eine weitere Möglichkeit zur Milderung des Dominierungseffektes ist es eine Art Turnier zu veranstalten. Es werden immer zwei oder mehr Individuen per Zufall aus der Population ausgewählt und treten zu einem Wettkampf an. Das bessere gewinnt und wird zur Fortpflanzung zugelassen.

### 4.3.4 Ersetzungsschema

Nachdem die Zeugung der Nachkommen durch Auswahl der Eltern und Rekombination ihrer Gene erfolgt ist, muß aus den Eltern und den Nachkommen die nächste Generation erzeugt werden. Auch hier kann man wieder nach den verschiedensten Strategien vorgehen.

## Komplettersetzung

Die einfachste Methode ist es, die komplette Elterngeneration durch ihre Nachkommen zu ersetzen. Dabei kann es aber vorkommen, daß gute Ergebnisse verlorengehen, da durch die Rekombination genau die Genkombinationen zerstört wurden, die eine gute Lösung dargestellt haben.

## Teilersetzungen

Um den Verlust von Erbgut in Grenzen zu halten kann, man sich Ersetzungsschemata vorstellen, die nur einen Teil der Elterngeneration ersetzen. Dies muß natürlich schon bei der Nachkommensbildung berücksichtigt werden.

Eine Möglichkeit besteht darin, weniger Nachkommen zu erzeugen und nach einem Auswahlverfahren, das dem der *Auswahl der Eltern* in Abschnitt 4.3.3 entspricht, nur invertiert wird, Todeskandidaten aus der Elterngeneration zu wählen und diese durch Kinder zu ersetzen.

Eine weitere Möglichkeit besteht darin, Eltern und Nachkommen in einen Topf zu werfen und einen Vernichtungsalgorithmus anzuwenden.

Durch die Schaffung eines Kindergartens, in dem die Nachkommen zuerst unter ihresgleichen einen Wettkampf auf Leben und Tod ausführen müssen, und gleichzeitig dasselbe in der Elterngeneration stattfindet, schafft man eine weitere Möglichkeit.

## 4.3.5 Populationsgröße

Bisher sind wir stillschweigend davon ausgegangen, daß sich die Populationsgröße während des Algorithmus nicht ändert. Nun wollen wir diese genauer betrachten und abwägen, welche Faktoren von ihr abhängen.

### Fixe Populationsgröße

In der Natur findet man selten die Umstände, die zu einer völlig fixen Populationsgröße führen. Meist wird durch Futterknappheit oder Überfluß, Klimawechsel oder ähnliches die Größe der Population schwanken. Indem wir die Population konstant halten, erhalten wir einen *konstanten Selektionsdruck*.

### Simulation von Populationswellen

Simuliert man die Umstände, durch die es zu Schwankungen in der Populationsgröße kommt, erhält man eine Änderung des Selektionsdrucks.

Erhöht sich die Größe der Population von einer Generation auf die nächste, führt dies zu einem *niedrigerem Selektionsdruck*. Individuen, die normalerweise gestorben wären, überleben, was zu einer Verbreiterung des Suchgebietes führt.

Dies kann vor allem dann sinnvoll sein, wenn der Algorithmus im Begriff ist, sich auf ein lokales Maximum festzufahren.

Erniedrigt man die Populationsgröße, *erhöht* man den *Selektionsdruck*. Man zwingt die Population, sich auf seine stärksten Individuen zu konzentrieren und treibt so die Konvergenz voran. Dies ist sinnvoll wenn die Population nicht mehr weiterkonvergiert.

Man kann anhand der oben aufgeführten Faustregeln eine automatische Anpassung der Populationsgröße an den gewünschten Selektionsdruck verwirklichen.

In beiden Fällen hört der Algorithmus auf zu konvergieren, d.h. es wird über mehrere Generationen hinweg keine Verbesserungen mehr geben. Man berechnet nun die Streuung der Population und erhöht im Falle einer kleinen Streuung den Selektionsdruck durch Verkleinern der Populationsgröße oder verringert den Selektionsdruck im Falle einer großen Streuung durch Vergrössern der Population.

# Kapitel 5

## Andere Optimierungsprozesse

In diesem Kapitel werden einige verbreitete Optimierungsprozesse vorgestellt und mit Evolutionsalgorithmen verglichen.

### 5.1 Strategische Verfahren

Im Gegensatz zu den stochastischen Optimierungsstrategien spielt bei den strategischen Methoden der Zufall nur eine untergeordnete Rolle.

#### 5.1.1 Gradientenstrategie

Dieses Verfahren setzt einen Parameterraum voraus, der mit reelwertigen Vektoren beschrieben werden kann. Jedem Parameter wird eine Komponente eines  $n$ -dimensionalen Lösungsvektors zugewiesen. Jeder Parameter stellt nun eine Achse eines  $n$ -dimensionalen Lösungsraumes dar.

Der Wert, auf die der Vektor zeigt, ist die Qualität der Lösung an dieser Stelle. Man erhält eine Art Dichte im  $n$ -dimensionalen Raum.

Beim Start des Suchprozesses wird ein zufälliger Startvektor gewählt. An diesem Startvektor wird nun der Gradient mit der Differenzenmethode bestimmt, indem in jede Achsenrichtung ein  $\delta$ -Schritt vollzogen wird und zwischen diesem neuen Punkt und dem Startvektor die Steigung berechnet wird. Nun wird die Richtung des steilsten Anstieges bestimmt und in diese Richtung ein  $\delta$  Schritt vollzogen. Das Verfahren wiederholt sich solange bis die Abbruchbedingung erreicht ist.

In Abbildung 5.1 wird der Algorithmus anhand einer Pseudo-Programmiersprache verdeutlicht.

Die Abbruchbedingung kann das Erreichen einer bestimmten Qualität, die auf den Suchprozeß verwendete Zeit, die Anzahl der Schleifendurchläufe oder auch

```

BEGIN /* Gradientenstrategie */
  Startvektor := beliebig;
  WHILE (Abbruch)
    FOR (alle n-Achsen)
      (Berechne alle Steigungen
       zwischen Startvektor und
       Delta-Schritt in Achsenrichtung);
    END FOR;
    Startvektor := Startvektor +
      Delta-Schritt in Richtung des
      hoechsten Anstieges;
    IF (Abbruchbedingung erreicht) Abbruch := TRUE;
  END WHILE
END;

```

Abbildung 5.1: Allgemeiner Pseudocode einer Gradientenstrategie

der Stillstand des Suchprozesses sein. Unter dem Stillstand des Suchprozesses versteht man, daß innerhalb einer bestimmten Anzahl an Schleifendurchläufen keine nennenswerten Verbesserungen mehr eingetreten sind.

Man erhält einen monoton steigenden Qualitätsverlauf.

Solche Methoden werden als „Hillclimbing“-Methoden bezeichnet. Sie sind vor allem geeignet in Lösungsräumen mit einem Maximum, dieses schnell zu finden. Bei Funktionen mit mehreren Maxima wird das erste gefundene Maximum erreicht und der Suchprozeß läuft sich dort fest. Meist ist dies nur ein relatives Maximum.

Für einen 1-dimensionalen Lösungsraum ist ein Beispiel in Abbildung 5.2 zu sehen. Die Suche startet an dem zufällig gewähltem Punkt  $X$  und bewegt sich bergauf zu  $B$ . Dort angekommen steckt die Suche fest. Die besseren Maxima in  $A$  und  $C$  werden nicht gefunden. Im Gegensatz zu den Evolutionsalgorithmen

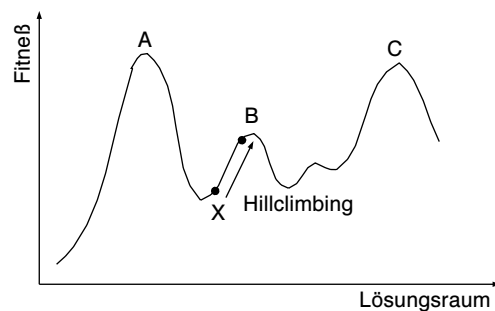


Abbildung 5.2: Das „Hillclimbing“-Problem

kann sich die Gradientenmethode kein vollständiges Bild über den Lösungsraum machen. Es entscheidet den Fortgang der Suche allein anhand der unmittelbaren

Umgebung des augenblicklichen Standpunktes, wohingegen sich Evolutionsalgorithmen zuerst ein grobes Bild der Lösungstopologie verschaffen und dann die Suche auf vermeintlich erfolgreiche Gebiete konzentrieren.

## 5.2 Stochastische Optimierungsprozesse

Bei den stochastischen Optimierungsprozessen ist der Zufall einer der wichtigsten Triebfedern im Fortgang der Suche nach dem Optimum.

### 5.2.1 Zufallssuche

Die „Zufallssuche“ ist einer der simpelsten Optimierungsprozesse. Es wird eine Zufallslösung auf ihre Tauglichkeit geprüft, dies vermerkt und durch Zufall eine neue Lösung bestimmt, die wiederum bewertet wird. Dies wird solange wiederholt bis eine hinreichend gute Lösung gefunden wurde.

Das dieses Vorgehen meist selten von Erfolg gekrönt wird muß wohl nicht näher erläutert werden.

### 5.2.2 Monte–Carlo Verfahren

Bei diesem Verfahren wird der Lösungsraum wiederum durch einen  $n$ -dimensionalen Vektor beschrieben, welcher alle zu optimierenden Größen beinhaltet. Von einem zufällig gewählten Startpunkt  $X$  aus wird ein zweiter Zufallspunkt  $Y$  gewählt, der innerhalb des Gebietes  $\mathcal{G}$  liegt. Das Gebiet  $\mathcal{G}$  beschreibt einen Raum, dessen Rand  $\Gamma$  maximal einen festen Abstand  $\rho$  vom Startpunkt  $X$  hat. Beide Punkte  $X$  und  $Y$  werden nun bewertet und der bessere als Startpunkt übernommen. Das Verfahren wird abgebrochen, nachdem einen ausreichend gute Lösung gefunden wurde oder eine maximale Anzahl an Iterationsschritten erreicht wird.

Das Verfahren trägt den Namen *Monte–Carlo*, weil bei der Erfindung des Algorithmus zur Bestimmung des jeweils nächsten Punktes *Würfel* benutzt wurden.

Der *Monte–Carlo Algorithmus* kann unter gewissen Voraussetzungen direkt mit einer  $(1 + 1)$ -Evolutionstrategie gleichgesetzt werden. Diese Voraussetzung betrifft den bei der  $(1 + 1)$ -ES benutzten Mutationsoperator. Kann man nämlich beweisen, daß der Mutationsoperator lediglich eine mutiertes Gen erzeugt, welches maximal den Abstand  $\rho$  vom unmutierten Ausgangs-gen im Lösungsraum besitzt, ersetzt dieser das Wählen des nächsten Punktes.

Somit lassen sich alle Aussagen, die für die  $(1 + 1)$ -ES gemacht wurden auf das *Monte Carlo Verfahren* übertragen.

Einige Besonderheiten des *Monte–Carlo Verfahrens* sollen jedoch hier aufgeführt werden.

Man erhält bei diesem Algorithmus einen monoton steigenden Fitneßverlauf.

Nebenmaxima werden nicht erkannt und führen, falls es innerhalb des Gebietes  $\mathcal{G}$  kein besseres Maximum gibt, zum Stillstand des Suchprozesses.

### 5.2.3 Simulated Annealing

Dieses Verfahren verläuft analog zum *Monte-Carlo Verfahren* (siehe Abschnitt 5.2.2), nur anstatt immer den jeweils besten Punkt als nächsten Startpunkt zu wählen, führt man einen Akzeptanzrahmen ein, d.h. die Fitneß des Zufallspunktes  $Y$  darf innerhalb des vorgegebenen Rahmens auch schlechter als die des Startpunktes  $X$  sein um dennoch akzeptiert zu werden. Dieser Akzeptanzrahmen wird exponentiell mit der Zeit kleiner gemacht, so daß irgendwann nur noch Verbesserungen möglich sind und das Verfahren konvergiert.

Aufgrund des exponentiellen Zeitverlaufs des Akzeptanzrahmens nennt man dieses Verfahren *Simulated Annealing* (Simulierte Abkühlung). Zuerst sind relativ grosse negative Sprünge im Fitneßverlauf erlaubt, die mit der Zeit immer kleiner werden. Dies soll die Abkühlung eines Festkörpers zu simulieren.

Diese Methode besitzt ebenfalls keinen Überblick über den Lösungsraum und kein Gedächtnis, mit dem vielversprechende Wege gemerkt werden können, aber im Gegensatz zur reinen Gradientenmethode oder des Monte-Carlo Algorithmus können Nebenmaxima im Anfangsstadium des Verfahrens überwunden werden. Läuft das Verfahren am Anfang auf ein Nebenmaxima, kann es dieses überwinden, solange der Akzeptanzrahmen noch hinreichend groß ist.

Die Methode des *Simulated Annealing* läßt sich auch in Evolutionsstrategien einsetzen. So kann man eine Evolutionsstrategie dahingehend ändern, daß man dort ebenfalls einen zeitlich konvergierenden Akzeptanzrahmen für die Varianz der Wahl der Eltern einführt.

## 5.3 Gemischte Verfahren

Bei gemischten Verfahren haben Zufall und systematische Suche in etwa den gleichen Stellenwert beim Lösungsfindungsprozeß.

### 5.3.1 Iterative Verfahren

Das *Iterative Verfahren* ist eine Kombination aus der Zufallssuche und der Gradientenstrategie. Man startet die Gradientenstrategie mit einem Zufallspunkt. Wenn dieser konvergiert hat, wählt man einen neuen Startpunkt per Zufallsprozeß und verfährt ebenfalls nach der Gradientenstrategie. Dies wird beliebig oft wiederholt. Am Ende ist der beste Endpunkt die Lösung der Suche.

Dieses Verfahren mag sehr primitiv wirken, verspricht aber bei Lösungsräumen, in denen das Maximum einen sehr kleinen Bereich einnimmt und von großen Bereichen geringer Fitneß eingeschlossen ist, einen sehr guten Erfolg.

# Kapitel 6

## Vergleich verschiedener Evolutionsalgorithmen

### 6.1 2–Dimensionale Testfunktionen

Anhand dreier charakteristischer 2–dimensionaler Testfunktionen soll das Lösungsfindungsverhalten verschiedener Evolutionsalgorithmen untersucht werden.

Folgende Punkte sollten bei allen Betrachtungen berücksichtigt werden:

1. Bei allen Untersuchungen wird das absolute **Maximum** gesucht.
2. Die **Ränder** des Suchbereiches **sind** als **absolut** zu betrachten, d.h. ist ein Punkt auf dem Rand der höchste im Suchbereich ist er automatisch das absolute Maximum und somit die Lösung der Suche.

### 6.1.1 Quadratische Funktion

Die Gleichung 6.1 steht als Stellvertreter von Funktionen mit einem Maximum und gutartigem Verhalten. Bei dieser Funktion sollte jeder vernünftige Lösungsalgorithmus innerhalb akzeptabler Zeit zum Ziel kommen.

$$f_0(x, y) = -(x^2 + y^2) \quad \text{und} \quad x, y \in [-10, 10] \quad (6.1)$$

In Abbildung 6.1 wird diese Funktion in einer 3-dimensionalen Ansicht mit Höhenlinien zur besseren Orientierung dargestellt. Die Grenzen wurden für  $x$  und  $y$  zwischen  $-10$  und  $10$  gewählt. Das einzige Maximum befindet sich bei den Koordinaten  $(0, 0)$  auf der Höhe  $0$ .

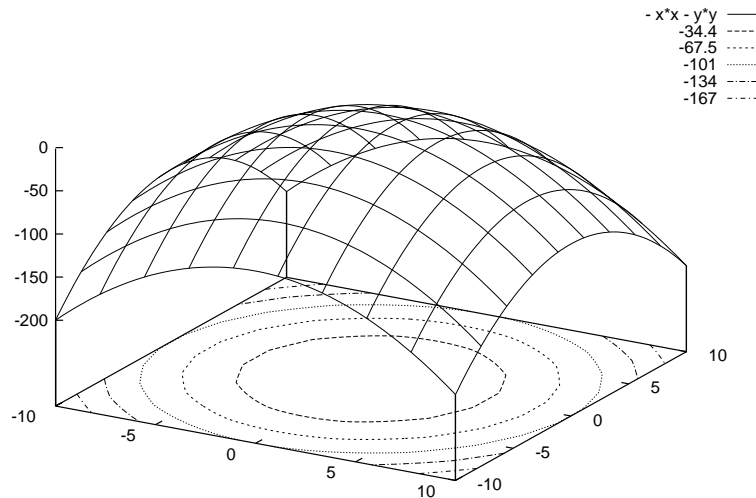


Abbildung 6.1: Quadratische Funktion

### 6.1.2 Treppenfunktion

Die Treppenfunktion 6.2 steht für Funktionen, die sich sprungartig ändern. Bei solchen Funktionen haben einige Algorithmen schon erhebliche Schwierigkeiten. So ist z.B. vorstellbar, daß sich eine Gradientenstrategie mit zu kleiner Schrittweite auf einer Treppenstufe verfangen kann und den Schritt zur nächsten Stufe nicht mehr schafft, da diese ausserhalb der Schrittweite liegt.

$$f_1(x, y) = \text{int}(x) + \text{int}(y) \quad \text{und} \quad x, y \in [-5, 5] \quad (6.2)$$

In Abbildung 6.2 ist die Treppenfunktion ebenfalls mit Höhenlinien zur besseren Übersicht dargestellt. Die Bereichsgrenzen liegen für  $x$  und  $y$  bei  $-5$  und  $5$ . Das absolute Maximum ist auf dem Rand bei den Koordinaten  $(5, 5)$  mit dem Wert  $10$  zu finden.

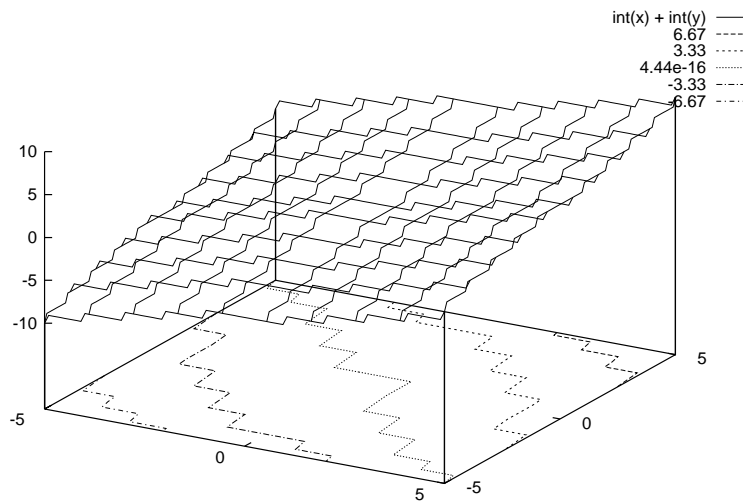


Abbildung 6.2: Treppenfunktion

### 6.1.3 Lineare cos Überlagerung mit Nebenmaxima

Die meisten Optimierungsalgorithmen haben Schwierigkeiten Nebenmaxima zu überwinden. Die Gleichung 6.3 soll als Vertreter solch gearteter Funktionen dienen.

$$f_2(x, y) = x \cos(x) + y \cos(y) \quad \text{und} \quad x, y \in [-\pi, \pi] \quad (6.3)$$

Wiederum wird die Funktion in einer Abbildung 6.3 mit Höhenlinien zur verdeutlichung dargestellt. Die Grenzen der  $x$  und  $y$  Koordinaten wurden zu  $-\pi$  und  $\pi$  gewählt. In Tabelle 6.1 werden die absoluten und lokalen Maxima aufgeführt.

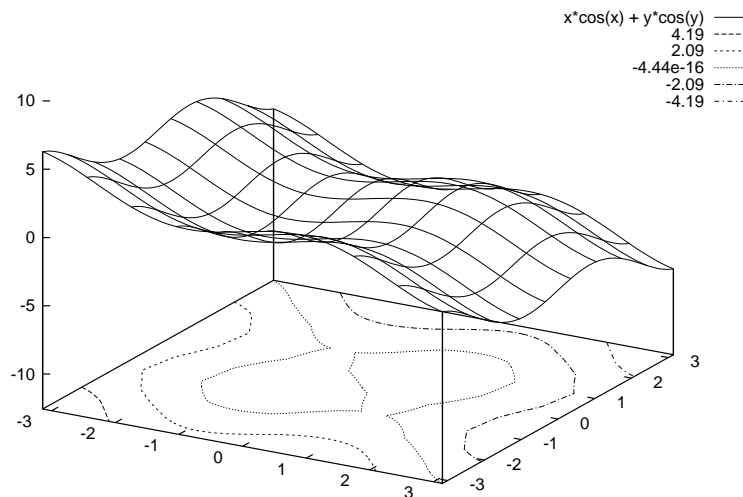


Abbildung 6.3: Lineare cos Überlagerung mit Nebenmaxima

Kennung	Maxima	$x$	$y$	$z$
a	absolut	$-\pi$	$-\pi$	$2\pi$
n	relativ	0.86	0.86	1.122
o	relativ	0.86	$-\pi$	3.703
p	relativ	$-\pi$	0.86	3.703

Tabelle 6.1: Absolutes und lokale Maxima der Funktion 6.3

## 6.2 Beschreibung der getesteten Algorithmen

In den in Abschnitt 6.3 durchgeführten Simulationen wurden Algorithmen mit in Tabelle 6.2 aufgeführten Parametern getestet. Für das Verständnis der einzelnen Algorithmen lese man in den entsprechenden Kapiteln nach. Die Evolutionsstrategien findet man in Kapitel 3 und die Genetischen Algorithmen in Kapitel 4. Die Bezeichnungen für die Selektionsstrategie, Mutations- und Rekombinationsoperatoren wird in Kapitel 7 näher erläutert.

Algorithmus	Name	selektion	mutation	rekombination	mu	lambda
(1 + 1)-ES	es	—	mut0	sex0	1	1
(1 + 5)-ES	es	—	mut0	sex0	1	5
(10 + 30)-ES	es	—	mut0	sex0	10	30
(5, 16)-ES	mukommalambda	—	—	—	5	16
GA	ga	wettkampf	mut0	sex1	6/8/12	12

Tabelle 6.2: Parametereinstellungen der getesteten Algorithmen

## 6.3 Simulationsergebnisse

In den folgenden Tabellen wurden jeweils die Simulationsergebnisse von 10 Testläufen bei Evolutionsstrategien und 5 Testläufe bei Genetischen Algorithmen festgehalten.

In den folgenden Abschnitten steht jeweils am Anfang eine Tabelle in der die für das Verfahren individuell eingestellten Parameter dargestellt sind. Es sind dies *maxloop*, *break*, *maxwidth* und *maxstep*. Die Beschreibung der Funktion der Parameter findet man in Kapitel 7. Bei den Genetischen Algorithmen wurde bei allen Simulationen der Parameter **immortal** zu 0 gewählt.

Es wurde jeweils die Qualität des besten Individuums der Startpopulation (**Start**) und die Anzahl der durchlaufenen Generationen bis zur Konvergenz (**Konv.**) gemessen. Anschließend wurde die Durchschnittliche Schrittweite (**S.W.**) berechnet. Dabei wurde die Schrittweite linear approximiert und nach folgender Formel berechnet:

$$S.W. = \frac{|\text{absolutes Maximum} - \text{Start}|}{\text{Konv.}} \quad (6.4)$$

Diese Schrittweite darf nur mit Schrittweiten derselben Qualitätsfunktion verglichen werden.

Zusätzlich wurde die durchschnittliche Qualität des besten Startindividuum berechnet.

Das Konvergenzverhalten wurde mittels *Gnuplot* graphisch analysiert und ein typischer Verlauf abgedruckt.

Ebenfalls graphisch mittels *Gnuplot* wurde die Spur, d.h. das Abtastverhalten des Algorithmus analysiert und ein charakteristischer Verlauf abgedruckt.

### 6.3.1 Quadratische Funktion

Parameter	$(\mu + \lambda)$ -ES	$(\mu, \lambda)$ -ES	GA
maxloop	1000	1000	10000
break	-0.01	-0.01	-0.01
maxwidth	10	10	10
maxstep	0.05	0.05	0.001

(1 + 1)-ES				(1 + 5)-ES		
#	Start	Konv.	S.W.	Start	Konv.	S.W.
0	-85.62	438	0.20	-98.02	163	0.60
1	-19.00	233	0.08	-53.90	125	0.43
2	-98.52	472	0.21	-53.33	124	0.43
3	-43.12	351	0.12	-89.66	156	0.58
4	-45.01	329	0.13	-105.97	176	0.60
5	-6.47	133	0.05	-30.03	92	0.33
6	-21.79	273	0.08	-10.45	57	0.18
7	-89.45	452	0.20	-57.78	140	0.41
8	-38.93	279	0.14	-24.22	92	0.26
9	-93.78	560	0.17	-154.23	213	0.72
$\phi$	54.17	352	0.15	67.77	133.8	0.45

(10 + 30)-ES				(5, 16)-ES		
#	Start	Konv.	S.W.	Start	Konv.	S.W.
0	-7.80	45	0.17	-18.46	76	0.24
1	-9.67	46	0.21	-6.27	43	0.17
2	-5.61	35	0.16	-5.80	42	0.14
3	-3.28	26	0.13	-4.56	37	0.12
4	-10.16	49	0.21	-31.75	102	0.31
5	-3.06	28	0.11	-42.76	117	0.37
6	-22.53	75	0.30	-24.15	83	0.29
7	-26.81	82	0.33	-11.73	56	0.21
8	-12.69	51	0.25	-1.39	21	0.07
9	-8.00	44	0.18	-17.52	69	0.25
$\phi$	10.96	48	0.20	16.44	65	0.21

Bei der Betrachtung obiger Simulationsergebnisse fällt auf, daß die durchschnittliche Qualität der Individuen der ersten Generation mit steigender Anzahl der Eltern zunimmt. Dies ist leicht erklärbar, da bei einer höheren Anzahl der Eltern, beim Start die Wahrscheinlichkeit steigt durch Zufall einen guten Startpunkt zu finden.

Mit steigendem  $\lambda$  nimmt auch die durchschnittliche Schrittweite zu. Dies ist wiederum leicht ersichtlich, da hier die Wahrscheinlichkeit steigt einen möglichst guten Nachfolger zu finden. Dieses Verhalten nutzt bei sequentieller Bearbeitung relativ wenig, da gleichzeitig mit dem steigen der Konvergenzgeschwindigkeit auch die Anzahl der nötigen Stichproben steigt. Allerdings läßt sich dieses bei paralleler Bearbeitung ausnutzen um den Faktor Konvergenzzeit zu minimieren.

Bei der graphischen Auswertung fiel auf, daß die (5, 16)–ES nicht wie erwartet zu Schwingungen neigt. Wie auch in der Abbildung 6.8 zu erkennen ist führt die Eliminierung aller Eltern zwar manchmal zu Rückschritten, diese werden aber sehr bald wieder aufgeholt.

In Abbildung 6.4 wird ein typisches Konvergenzverhalten einer (1 + 1)–ES angewandt auf die hier behandelte Funktion 6.1 dargestellt. Es ist, wie bei dieser Funktion nicht anders zu erwarten, sehr deutlich das gleichmäßige Konvergenzverhalten zu erkennen.

Genetischer Algorithmus						
mu=6, lambda=12			mu=8, lambda=12		mu=12, lambda=12	
#	Start	Konv.	Start	Konv.	Start	Konv.
0	-5.472542	3338	-1.18	7	-0.80	8
1	-30.03	7877	-2.21	1854	-0.19	389
2	-5.18	2603	-19.96	4629	-1.41	16
3	-8.25	4094	-8.33	100	-6.07	11
4	-3.57	6	-2.79	1361	-5.19	560

Der Genetische Algorithmus konvergiert bei diesem Problem sehr langsam. Bei der graphischen Auswertung fiel auf, daß meist am Anfang des Suchprozesses ein starker Qualitätsanstieg zu verzeichnen ist. Danach wird der Fortgang nur noch durch den Mutationsoperator bestimmt, da sich alle Individuen sehr dicht im Lösungsraum aneinander drängen. Wie man sich leicht überlegen kann muß die Mittelwertbildung beim Rekombinationsprozeß unmittelbar zu diesem Phänomen führen. Ein Genetischer Algorithmus in dieser Form kann deshalb nicht für solche Probleme empfohlen werden.

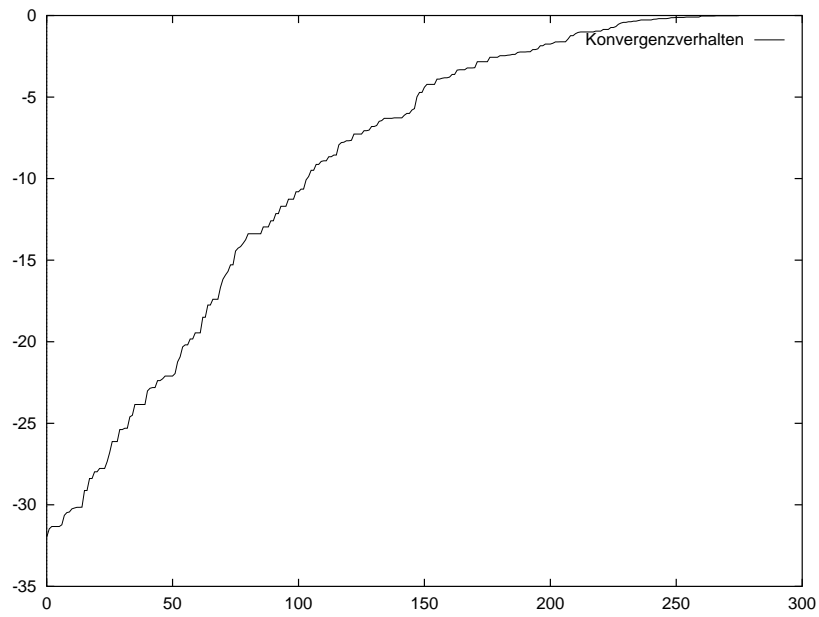


Abbildung 6.4: Konvergenzverhalten einer  $(1 + 1)$ -ES

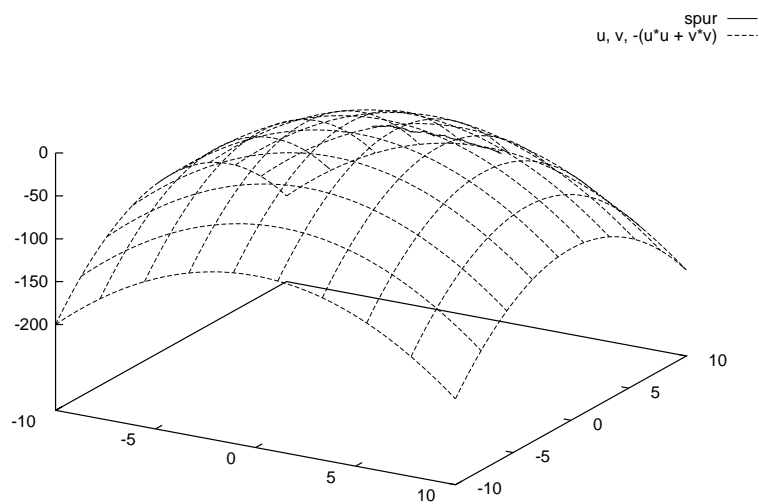


Abbildung 6.5: Spur für das Konvergenzverhalten einer  $(1 + 1)$ -ES

### 6.3.2 Treppenfunktion

Parameter	$(\mu + \lambda)$ -ES	$(\mu, \lambda)$ -ES	GA
maxloop	1000	1000	10000
break	10	10	10
maxwidth	5	5	5
maxstep	0.6	0.6	0.6

(1 + 1)-ES				(1 + 5)-ES		
#	Start	Konv.	S.W.	Start	Konv.	S.W.
0	3	687	0.010	-3	50	0.26
1	-6	170	0.082	1	82	0.11
2	7	19	0.159	-1	341	0.03
3	-5	161	0.093	2	295	0.03
4	-8	999	0.018	1	65	0.14
5	1	619	0.015	2	48	0.16
6	2	95	0.082	-1	546	0.02
7	-3	264	0.049	-2	31	0.39
8	-1	388	0.028	1	26	0.35
9	-5	262	0.057	1	29	0.31
$\phi$	-1.5	366	0.059	0.1	151	0.18

(10 + 30)-ES				(5, 16)-ES		
#	Start	Konv.	S.W.	Start	Konv.	S.W.
0	5	8	0.62	1	17	0.64
1	6	5	0.80	4	13	0.46
2	3	11	0.64	6	4	1.00
3	5	7	0.71	8	2	1.00
4	2	9	0.89	4	8	0.75
5	7	4	0.75	7	3	1.00
6	6	6	0.67	4	11	0.54
7	3	11	0.64	4	9	0.67
8	5	6	0.83	0	20	0.50
9	6	7	0.57	6	5	0.80
$\phi$	4.8	7	0.71	4.4	9	0.74

Grundsätzlich gilt hier das in Abschnitt 6.3.1 über Evolutionsstrategien gesagte.

Der Parameter *maxstep* musste hier auf einen Wert gewählt werden, der ein Überspringen der Stufen mit einer gewissen Wahrscheinlichkeit ermöglicht, da ansonsten das Verfahren auf einer Stufe festläuft.

Bei der Betrachtung des Konvergenzverhaltens in Abbildung 6.6 fällt auf, daß trotz der hohen Mutation sich das Verfahren oftmals relativ lange auf einer Stufe aufhält bevor es den Sprung zur nächsten Stufe schafft. Man hätte in diesem Fall die Mutation noch größer wählen dürfen. Allerdings muß man aufpassen, daß er nicht zu groß wird und das Verfahren zur Zufallssuche wird.

Im Spurbild 6.7 erkennt man wiederum ein sehr zielstrebiges Konvergenzverhalten. Die Befürchtung liegt nahe, daß sich dies bei Funktionen mit Nebenmaxi-

ma nachteilig auswirken kann.

Genetischer Algorithmus						
mu=6, lambda=12			mu=8, lambda=12		mu=12, lambda=12	
#	Start	Konv.	Start	Konv.	Start	Konv.
0	5	16	-2	31	5	35
1	6	15	-4	25	5	19
2	7	11	-4	23	4	30
3	2	42	-5	20	5	19
4	3	17	-4	12	6	15

Da im vorherigen Versuch erkannt wurde, daß die Rekombinationsstrategie *Mittelwertbildung* in diesem Fall nicht sehr sinnvoll ist, wurde diesmal die Mutation mehr betont. Jetzt liegen die Ergebnisse im Bereich einer vergleichbaren  $(\mu + \lambda)$ -ES bzw. einer  $(\mu, \lambda)$ -ES. Auch die graphische Untersuchung zeigt, daß das Suchverhalten mit dem einer Evolutionsstrategie vergleichbar ist. Im folgenden wird deshalb der Genetische Algorithmus nicht weiter untersucht.

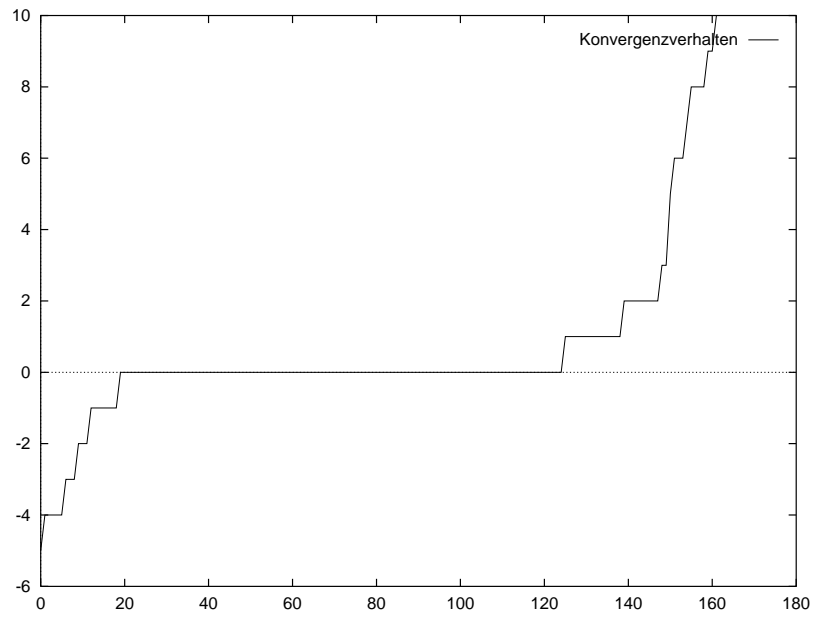


Abbildung 6.6: Konvergenzverhalten einer  $(1 + 1)$ -ES

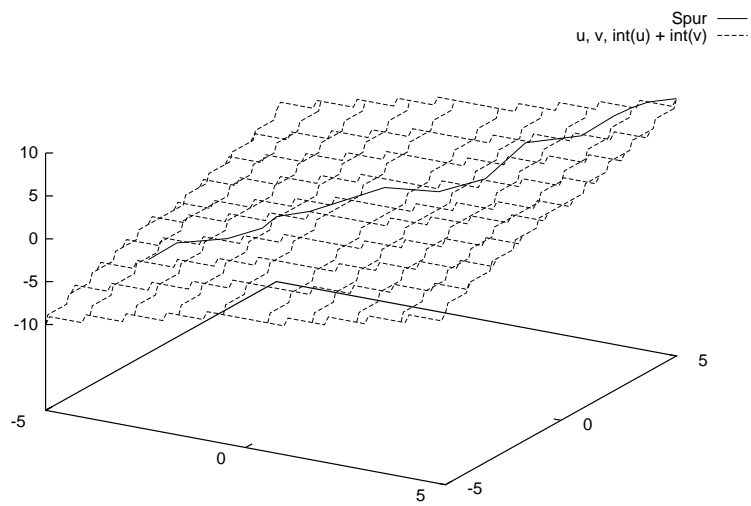


Abbildung 6.7: Spur für das Konvergenzverhalten einer  $(1, 5)$ -ES

### 6.3.3 Lineare $\cos$ Überlagerung mit Nebenminima

Parameter	$(\mu + \lambda)$ -ES	$(\mu, \lambda)$ -ES
maxloop	1000	250
break	6.2	6.2
maxwidth	3.1415	3.1415
maxstep	0.02	0.02

(1 + 1)-ES				(1 + 5)-ES		
#	Start	Konv.	Min.	Start	Konv.	Min.
0	-4.20	320	n	0.74	80	p
1	0.50	100	n	-2.09	80	n
2	-1.40	360	o	0.10	60	p
3	0.13	150	n	-1.32	50	n
4	0.16	250	p	1.58	90	p
5	2.51	250	p	-3.42	100	n
6	2.72	230	a	0.12	80	p
7	-0.32	200	n	1.97	95	a
8	1.24	150	o	2.79	20	p
9	-0.32	330	p	4.81	25	a

(10 + 30)-ES				(5, 16)-ES		
#	Start	Konv.	S.W.	Start	Konv.	S.W.
0	1.19	65	p	2.39	65	p
1	2.47	80	a	0.05	90	p
2	2.72	50	o	3.10	25	o
3	5.72	11	a	5.27	20	a
4	2.73	30	o	3.39	20	p
5	3.55	10	o	3.56	30	p
6	4.61	29	a	2.71	60	p
7	1.72	65	p	2.65	75	o
8	3.51	52	a	1.59	60	o
9	5.09	22	a	3.33	20	o

In lokalen Bereichen um relative Maxima ist das Suchverhalten wie erwartet nicht anders als bei der *quadratischen* Qualitätsfunktion.

Leider wirkt sich hier die Zielstrebigkeit des Konvergenzvorschrittes negativ aus. Alle Untersuchten Algorithmen fahren sich unmittelbar im, vom Startpunkt aus, nächsten lokalen Maxima fest. Diesem Verhalten ließe sich nur durch Wahl eines stärkeren Mutationsoperator entgegenwirken. Damit würde sich die Konvergenzgeschwindigkeit verkleinern aber gleichzeitig würde sich auch die Varianz des Suchverfahrens vergrößern. Allerdings ist dabei zu Beachten, daß der Mutationsoperator so groß sein müsste, daß er von einem lokalen Maximum mindestens an eine Stelle höherer Qualität mutieren kann. Dabei gleitet das Verfahren jedoch wieder zu einer Zufallsuche ab.

In Abbildung 6.9 sind diesmal mehrere Spuren eingezeichnet, die zu unterschiedlichen Maxima führen. Bei Verstärkung des Mutationsoperators würde die Spur breiter werden; Die Varianz der Spur würde sich verbreitern.

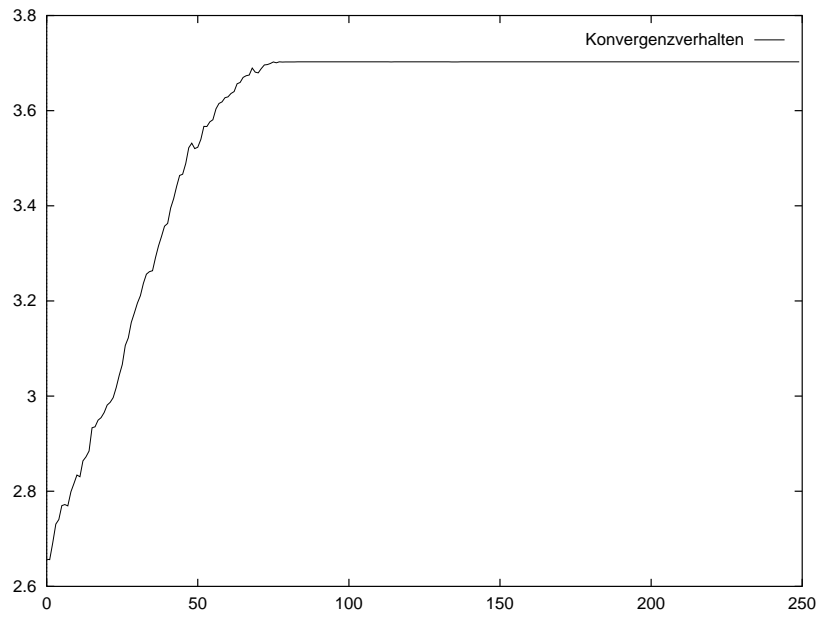


Abbildung 6.8: Konvergenzverhalten einer (5, 16)-ES

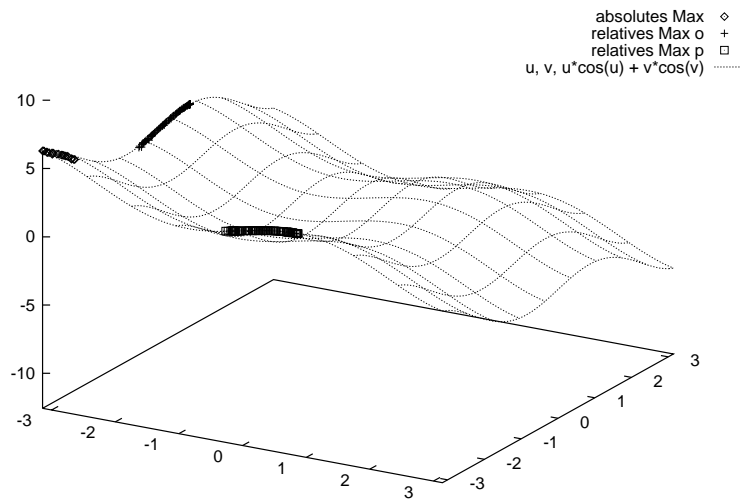


Abbildung 6.9: Spur für das Konvergenzverhalten einer (1 + 1)-ES

## 6.4 Zusammenfassung der Ergebnisse

Bei einfachen glatten Funktionen konvergieren die hier untersuchten Evolutionsstrategien sehr gut. Durch Erhöhen der Werte  $\mu$  und  $\lambda$  läßt sich die Konvergenzgeschwindigkeit erhöhen. Diese Eigenschaft läßt sich sehr gut ausnutzen um parallele Strategien zu entwickeln.

Bei Funktionen mit mehreren Maxima tendieren die hier untersuchten Evolutionsstrategien dazu sich in Nebenmaxima festzulaufen. Um solche Probleme zu lösen sollte man erweiterte Evolutionsstrategien entwickeln. Eine Möglichkeit, die in [Abl87] beschrieben wird, scheint auch hier angebracht zu sein. Fällt die Konvergenzschrittweite über mehrere Generation hinweg unter einen bestimmten Wert wird der Einfluß des Mutationsoperators erhöht. Somit werden größere Sprünge möglich, die eventuell dazu führen das lokale Maximum zu überwinden. Eine andere Möglichkeit ist es bei der Selektionsstrategie ähnlich wie beim *Simulated Annealing* (siehe Abschnitt 5.2.3 einen Akzeptanzparameter einzuführen, der bei Konvergenzstagnation für eine gewisse Anzahl an Generationen vergrößert wird.

Bei den Untersuchungen hat sich außerdem gezeigt das sich ein einfacher Evolutionsalgorithmus ähnlich dem eines verbesserten *Monte-Carlo* Verfahrens verhält. Dieses Verhalten lag nahe, da sich das Monte-Carlo Verfahren mit der  $(1+1)$ -ES vergleichen läßt und die weiteren Evolutionsstrategien nur Verallgemeinerungen dieses Grundalgorithmus sind.

Der Genetische Algorithmus hat sich in diesem Test leider nicht bewehrt. Die Mittelwertbildung als Rekombinationsoperator führte zu einer zu schnellen Vereinheitlichung des Erbgutes, so daß keine weiteren Entwicklungen durch diesen Operator mehr möglich waren. Eine mögliche Verbesserung mag darin bestehen nicht alle Komponenten des Vektors zu mitteln. Bei Funktionen höherer Dimension dürfte dies genügen um eine frühzeitige Konvergenz des Genpools zu verhindern.

Insgesamt gesehen können Evolutionsalgorithmen erst ihre Stärken bei einer hohen Anzahl an Parametern ausspielen. Dies wurde auch schon in Abschnitt 3.4.4 hervorgehoben.

## 6.5 Das Travelling-Salesman-Problem

Das Travelling-Salesman-Problem (TSP) ist ein klassischer Vertreter von Reihenfolgeproblemen und wird sehr oft als Referenzproblem für dererlei Problemstellungen herangezogen.

Ein Handelsvertreter hat die Aufgabe eine Reihe von Städten zu besuchen und anschließend wieder an seinem Ausgangsort zurückzukommen. Es gibt Straßen von jeder Stadt zu jeder anderen.

Dieses Problem ist deshalb so schwer zu lösen, da die Zahl der Möglichkeiten faktoriell mit der Zahl der Orte zunimmt. Die Zahl der möglichen Wege berechnet

sich wie folgt:

$$\text{Anzahl möglicher Wegkombinationen} = (\text{Anzahl Städte})! \quad (6.5)$$

Dabei gilt auch ein Weg in anderer Laufrichtung als neue Kombination.

Bei dem hier untersuchten Beispiel wurde die Aufgabe gestellt 12 Städte miteinander zu verbinden. Dabei gibt es schon 479001600 mögliche Wegkombinationen.

Der optimale Weg hat eine Länge von 280.578046 Einheiten und ist in Abbildung 6.11 dargestellt.

Mit einem rekursiven Algorithmus wurde zuerst die Dichte der Wahrscheinlichkeitsverteilung für die Weglängen ermittelt. Diese Verteilung ist in Abbildung 6.10 dargestellt. Dabei fällt auf, daß die Verteilung als normalverteilt angenommen werden kann. Der Mittelwert wird dabei mittels der Graphik zu 690 und die Varianz zu 100 geschätzt.

Bei mehreren Testläufen wurde mit einem  $(1+4)$ -ES im Durchschnitt nach 58 Generationen annähernd das Minimum der Weglänge erreicht. Der Algorithmus wurde nach Unterschreiten der Länge 285 abgebrochen und als Konvergent betrachtet.

Bei der gewählten  $(1+4)$ -Evolutionsstrategie entspricht das ca. 4 Stichproben · 58 Generationen = 232 Stichproben.

Die Wahrscheinlichkeit bei 232 rein zufälligen Stichproben einen Weg zu finden, der besser oder gleich einer Weglänge von 285 Einheiten ist liegt weit unter einem Hunderttausendstel. Hierbei wird die Effektivität der Evolutionsstrategie gegenüber einer reinen Zufallssuche ersichtlich.

Bei der Untersuchung einer  $(\mu, \lambda)$ -ES wurde deutlich, daß der Mutationsoperator sehr große Sprünge im Lösungsraum verursacht. Dies wird auch durch die großen Sprünge im Qualitätsverlauf der Abbildung 6.12 deutlich. Die genannte Strategie führt zu keiner Konvergenz. Die Qualität der Nachkommen liegt weit ab von der der Eltern, so daß das Verfahren zum Schwingen neigt und nicht konvergiert.

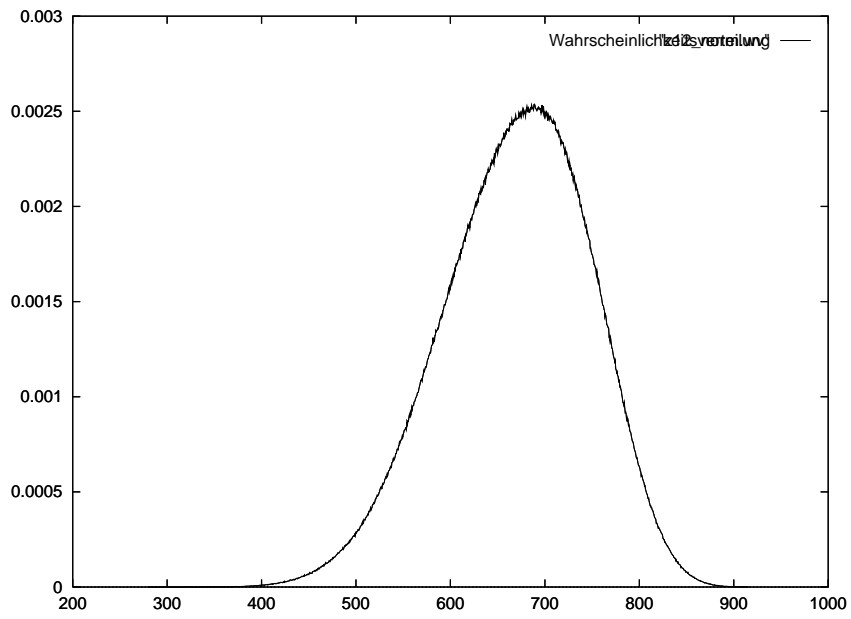


Abbildung 6.10: Wahrscheinlichkeitsverteilung beim TSP

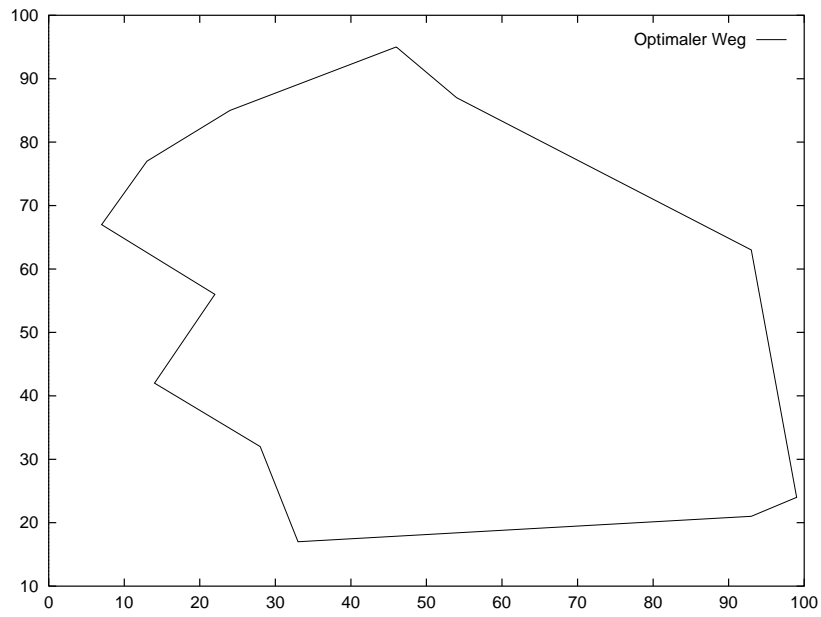


Abbildung 6.11: Optimaler Weg des gewählten Beispiels

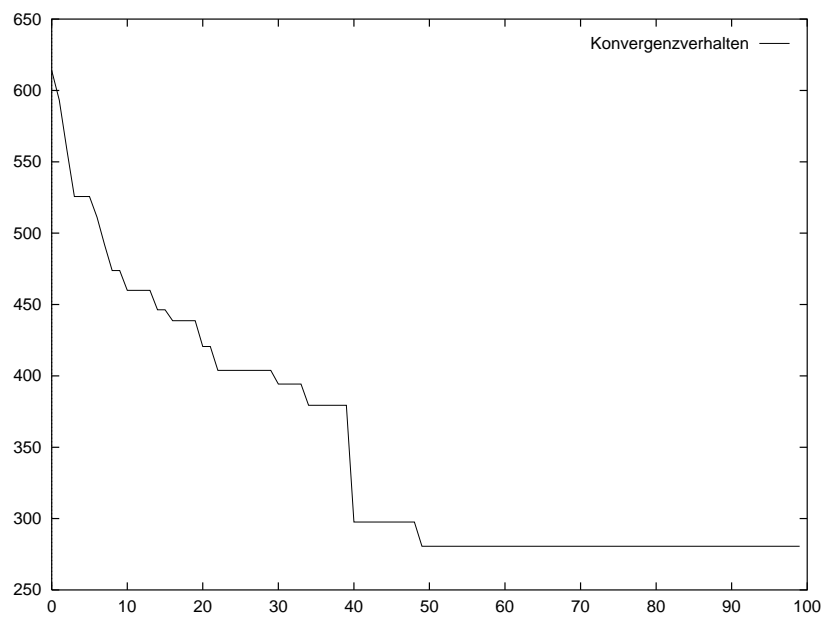


Abbildung 6.12: Typisches Konvergenzverhalten beim TSP mit einer (1+4)-ES

# Kapitel 7

## EVDice

Um Experimente mit verschiedenen Problemstellungen durchführen zu können, wurde das Programmpaket EVDice (Evolution Dice) entworfen. Es ist hauptsächlich für Reihenfolgeprobleme und parametrisierbare Probleme konzipiert, läßt sich aber auch auf andere Problemstellungen erweitern.

Der Entwurf zielt darauf, das Travelling-Salesman-Problem (TSP) und  $n$ -dimensionale parametrisierte Funktionen zu lösen.

### 7.1 Überblick

Das Programm beinhaltet verschiedene Algorithmen, die sich über ein Konfigurationsfile ansprechen und einstellen lassen.

Desweiteren bietet das Programm verschiedene Mutations-, Rekombinationsoperatoren und Selektionsstrategien an.

Um ein anderes Problem anzusprechen ist leider ein erneuter Compilervorgang notwendig, da sich bestimmte Routinen der verschiedenen Problemstellungen nicht vereinheitlichen lassen. Dies sind insbesondere die Fitneß- oder Qualitätsfunktion, Ausgabe des Gencodes und das Konfigurieren und Einlesen problemspezifischer Parameter.

#### 7.1.1 Algorithmen

Nachfolgend aufgelistete und beschriebene Algorithmen wurden bisher verwirklicht:

**MonteCarlo** Hierbei handelt es sich um ein klassischen Monte-Carlo Verfahren. (siehe Abschnitt 5.2.2)

**muPlusLambda** Bei diesem Algorithmus handelt es sich um eine klassischen  $(\mu + \lambda)$ -Evolutionsstrategie nach Rechenberg (siehe Abschnitt 3.2.1)

**muKommaLambda** Diese Implementation ist einer klassischen  $(\mu, \lambda)$ -Evolutinssstrategie nach Rechenberg nachempfunden. (siehe Abschnitt 3.2.2)

**ES** Dieser Algorithmus stellt einen erweiterten Evolutionsalgorithmus dar. (siehe Abschnitt 7.2.6)

**GA** hierbei handelt es sich um eine Umsetzung eines Genetischen Algorithmus. (siehe Abschnitt 7.2.6)

### 7.1.2 Informationsausgabe

Das Programm kann verschiedenste Informationen ausgeben. Diese Informationen können sowohl auf den Bildschirm als auch in ein File geschrieben werden. Das Format wurde so gehalten, daß es mit Standard–Unix–Befehlen wie *awk* weiterverarbeitet oder mit *gnuplot* visualisiert werden kann.

Es können Informationen über den Fortschrittsverlauf des besten Individuums oder des Durchschnitts erstellt werden. Desweiteren läßt sich eine Spur des besten Individuums oder auch aller Individuen erzeugen. Dies sind nur einige von mir getesteten Möglichkeiten. Sicherlich sind noch weitere möglich.

### 7.1.3 Problemspezifisches

Einige Einstellungen sind bei verschiedenen Problemen zu beachten.

Beim Travelling–Salesman–Problem (TSP) muß ein Ortefile existieren, das die miteinander zu verbindenden Orte beinhaltet.

Bei parametrisierten Problemen muß eine Qualitätsfunktion im Code implementiert sein. Es muß die Dimension des zu optimierende Problems bekannt sein und es muß ein rechteckiger Arbeitsbereich definiert werden. Desweiteren muß noch die maximale Schrittweite für den Mutationsoperator angegeben werden.

Wie diese Parameter einzustellen sind und wie das Format des Ortefiles zu wählen ist wird in Abschnitt 7.4 erläutert.

## 7.2 Bedienung und Konfiguration

Das Programm verfügt teils über eine Befehlszeilen orientierte, teils über eine Konfigurationsdatei orientierte Steuerung.

### 7.2.1 Bedienung

Eine Übersicht der in der Befehlszeile möglichen Kommandos läßt sich mit der Option **-h** erreichen.

Der Aufruf des Programmes entspricht folgender Schablone:  
`evd [ -h || -a || -t (algorithmus) || (name.rc) ]`

Dabei bedeuten die Optionen folgendes:

- h Es wird eine kurze Hilfe zur befehlszeilenorientierten Bedienung auf den Bildschirm ausgegeben.
  - a Alle im Programm implementierten Algorithmen werden aufgelistet. Die angegebenen Namen entsprechen denen für das Tag **algo**, das im Konfigurationsfile den Algorithmus wählt.
  - t (*Algorithmus*) Listet alle für den angegebenen *Algorithmus* möglichen Tags auf. Leitet man diese Ausgabe mit dem Filedescriptor „*l* (*name.rc*)“ in ein File um, hat man eine gute Grundlage für die Konfiguration eines bestimmten Algorithmus.
- (*name.rc*) Startet einen Programmlauf mit den im Konfigurationsfile (*name.rc*) angegebenen Einstellungen.

## 7.2.2 Konfigurationsdatei und Tagkonzept

Das Konfigurationsfile ist zeilenorientiert aufgebaut. Eine Zeile kann durch ein abschließendes “\” auf die nachfolgende Zeile ausgedehnt werden. Ansonsten endet die Zeile mit dem Returncode.

Eine Kommentarzeile wird durch ein „#“ als erstes Zeichen der Zeile eingeleitet.

Das erste Wort einer Zeile ohne voranstehende Leerzeichen wird als Tag interpretiert. Das Tag wird durch einen abschließenden Doppelpunkt oder durch ein Space beendet. Alle Zeichen dahinter bis zum Zeilenende werden als Wert für das Tag interpretiert. Eine Tagzeile hat also folgendes Format:

`<tag>: <Wert>`

Der Wert eines Tags wird der durch das Tag spezifizierten Konfigurationsroutine als String übergeben.

## 7.2.3 Ausgabe

Für die Steuerung der Ausgabe sind folgende Tags vorgesehen:

**info** Steuert die erweiterte Informationsausgabe. Wird durch die Werte **on** und **off** ein- bzw. ausgeschaltet. Ist die erweiterte Informationsausgabe eingeschaltet, werden alle Einstellungen des Konfigurationsfiles zusätzlich auf den Bildschirm ausgegeben. Ausserdem werden nach jeder Generation alle Individuen mit Qualität und Alter ausgegeben. Auf die Ausgaben des Outputfiles hat dieses Tag keinen Einfluß.

**echo** Gibt den hinter dem Tag stehenden Text auf den Bildschirm aus.

**line** Gibt eine horizontale Linie auf den Bildschirm aus.

**set\_comm** Kann dazu benutzt werden, Text als Kommentar in das Ausgabefile zu schreiben. Dem Text wird ein „#“ vorangestellt um es als Kommentar zu kennzeichnen.

**mask** Mit diesem Tag wird die Auswertung gesteuert. Es sind folgende Werte in beliebiger Kombination und Reihenfolge möglich: **counter**, **top.worth**, **top.prop**, **top.age**, **bottom.worth**, **bottom.prop**, **bottom.age**, **mean.worth**, **mean.prop**, **mean.age**, **best**, **living**. Zur näheren Diskussion sei auf Abschnitt 7.2.4 hingewiesen.

## 7.2.4 Auswertung und Outputfile

Mit dem Tag **out** wird das Outputfile gewählt. Wird als Ausgabefile der Name **STDOUT** angegeben, wird die gesamte Ausgabe, die ansonsten in das angegebene File geschrieben wird, auf Standardout (meist der Bildschirm) ausgegeben.

Zuerst werden alle konfigurationsrelevanten Tags mit ihren Werten als Kommentar in das Outputfile geschrieben. Danach erfolgt die Ausgabe der durch das Tag **mask** gewählten Daten. Diese Daten werden in einer Zeile durch Space getrennt ausgegeben. Als mögliche Daten stehen nachfolgend aufgelistete Optionen offen, die in beliebiger Anzahl, Reihenfolge und Kombination angegeben werden können.

**counter** Die Generationen werden fortlaufend durchnummeriert. Hiermit kann dieser Zähler ausgegeben werden.

**top.worth** Die Qualität des besten Individuums wird ausgegeben.

**top.age** Das Alter in Generationen des besten Individuums wird hiermit registriert. Dieser Wert wird bisher nur vom Algorithmus *ga* und *es* unterstützt.

**bottom.worth** Wie *top.worth*, nur wird hier das schlechteste Individuum betrachtet.

**bottom.age** Wie *top.age*, für das schlechteste Individuum.

**mean.worth** Hiermit wird die mittlere Qualität der gesamten Population ermittelt und ausgegeben.

**mean.age** Berechnet das mittlere Alter in Generationen der gesamten Population. Diese Option wird bisher nur vom Algorithmus *ga* und *es* unterstützt.

**best** Gibt den Gencode des besten Individuums in einer einzeiligen Darstellung aus. Die Darstellung variiert mit der Implementation der Problemstellung. Diese Option ist nicht beim Problem *TSP* möglich.

**living** Gibt den Gencode aller lebenden Mitglieder der augenblicklichen Population aus. Nach jedem Gencode eines Individuums wird ein Return angefügt. Dies muß in Kombination mit anderen Flags beachtet werden. Für die Darstellung gilt das bei *best* gesagte. Diese Möglichkeit wird bisher nicht vom Problem *TSP* unterstützt.

### 7.2.5 Zufallszahlengenerator

Zufallszahlen werden durch die im Modul *random* implementierten Routinen erstellt. Diese Algorithmen wurden dem Buch [PTVF92] entnommen.

Zum Festlegen des Startwertes des Zufallszahlengenerators dient das Tag **sran**. Wird hier ein negativer Wert (vorzugsweise  $-1$ ) angegeben, so wird der augenblickliche Stand des im Rechner eingebaute Timers als Startwert benutzt. Bei einem positiven Werte wird dieser als Startwert verwendet.

Um eine Reproduktion von Ergebnissen zu gewährleisten, wird der Startwert des Zufallszahlengenerators in das Outputfile geschrieben.

### 7.2.6 Algorithmen

Die Wahl des Algorithmus erfolgt durch das Tag **algo**. Hinter diesem Tag muß der Name des gewünschten Algorithmus angegeben werden.

Bisher wurden folgende Algorithmen für das Programm implementiert: **montecarlo** (Monte-Carlo Verfahren), **mupluslambda** (klassische  $(\mu + \lambda)$ -Evolutionstrategie nach Rechenberg), **mukommalambda** (klassische  $(\mu, \lambda)$ -Evolutinsstrategie nach Rechenberg), **es** (Erweiterte Evolutionstrategie) und **ga** (Genetischer Algorithmus).

Nachfolgend werden die Algorithmen in ihrer Arbeitsweise beschrieben und die für sie möglichen Einstellungen erklärt.

#### Monte-Carlo Verfahren

Der Monte-Carlo Algorithmus wurde als erster implementiert. Er läßt sich einzig durch das Tag **maxloop** steuern. Es gibt die Anzahl der zu durchlaufenden Schleifenzyklen bis zum Abbruch an.

Eine Beschreibung des Monte-Carlo Verfahrens findet sich in Abschnitt 5.2.2.

Zur Erzeugung des zweiten Testpunktes wird bei dieser Implementierung eine Copy des Ergutes angelegt und dies mittels des Mutationsoperators **mut0** zufällig verändert.

#### $(\mu + \lambda)$ -Evolutionstrategie

Eine Beschreibung der  $(\mu + \lambda)$ -Evolutinsstrategie findet sich in Abschnitt 3.2.1.

Nachfolgend werden alle Tags für diesen Algorithmus aufgelistet und deren Wirkungsweise beschrieben:

**maxloop** Nach der angegebenen Anzahl an Schleifendurchläufen wird der Vorgang abgebrochen.

**mu** Gibt die Anzahl der Eltern an und sollte sich im Bereich  $[1, \infty]$  bewegen. Hierbei muß es sich um eine ganze Zahl  $\mathcal{Z}$  handeln.

**lambda** Gibt die Anzahl der aus den Eltern zu erzeugenden Nachkommen an. Es handelt sich wiederum um eine ganze Zahl  $\mathcal{Z} \in [1, \infty]$ .

Aus den  $\mu$  besten der Liste werden durch Zufall  $\lambda$  Nachkommen durch den Rekombinationsoperator **sex0**, welcher eine reine Verdoppelung darstellt, erzeugt und durch den Mutationsoperator **mut0** mutiert. Danach findet eine Sortierung der Liste statt, wodurch die Selektion gewährleistet wird.

### $(\mu, \lambda)$ -Evolutinssstrategie

In Abschnitt 3.2.2 wird dieser Evolutionsalgorithmus näher beschrieben.

Die für diesen Algorithmus benötigten Einstellungen sind dieselben wie für die der  $(\mu + \lambda)$ -Evolutionsstrategie. Für eine Beschreibung der Tags lese man deshalb in Abschnitt 7.2.6 nach.

Aus den  $\mu$  ersten der Liste werden durch Zufall  $\lambda$  Nachkommen erzeugt. Diese werden durch den Verdoppelungsoperator **sex0** erzeugt und durch den Mutationsoperator **mut0** mutiert. Bei der anschließenden Sortierung werden nur die  $\lambda$  Nachkommen berücksichtigt.

### Erweiterte Evolutionsstrategie

Bei der erweiterten Evolutionsstrategie handelt es sich im Grunde genommen um eine  $(\mu + \lambda)$ -Evolutinssstrategie mit freier Wahlmöglichkeit des Mutations- und des Rekombinationsoperators. Zusätzlich läßt sich noch eine weitere Abbruchbedingung angeben, bei deren Unterschreiten der Algorithmus abgebrochen wird.

Nachfolgend eine Auflistung der Einstellungsmöglichkeiten und ihre Beschreibung:

**mutation** Es kann ein geeigneter implementierter Mutationsoperator angegeben werden. Um eine Liste der implementierten Operatoren zu erhalten rufe man das Programm mit der Option **-t es** auf.

**rekombination** Aus den implementierten Rekombinationsoperatoren kann man einen auswählen. Für eine Liste der implementierten Operatoren rufe man das Programm mit der Option **-t es** auf.

**maxloop** Nach der angegebenen Anzahl an Schleifendurchläufen wird der Vorgang abgebrochen.

**break** Nach Unterschreiten der angegebenen reellen Zahl  $\mathcal{R}$  für die Qualität des besten Individuums wird der Algorithmus abgebrochen.

**mu** Gibt die Anzahl der Eltern an und sollte sich im Bereich  $[1, \infty]$  bewegen. Hierbei muß es sich um eine ganze Zahl  $\mathcal{Z}$  handeln.

**lambda** Gibt die Anzahl der aus den Eltern zu erzeugenden Nachkommen an. Es handelt sich wiederum um eine ganze Zahl  $\mathcal{Z} \in [1, \infty]$ .

## Genetischer Algorithmus

Für eine Beschreibung der Wirkungsweise eines Genetischen Algorithmus lese man in Kapitel 4 nach. Zur Steuerung des Genetischen Algorithmus finden folgende Tags eine Anwendung:

**selektion** Wählt die Selektionsstrategie. Für eine Übersicht der implementierten Strategien rufe man das Programm mit der Option `-t ga` auf.

**rekombination** Wählt den Rekombinationsoperator. Eine Liste der möglichen Operatoren erhält man mit den Optionen `-t ga`.

**mutation** Wählt den Mutationsoperator. Eine Auflistung der implementierten Operatoren erhält man mit den Optionen `-t ga`.

**mu** Mindestgröße der Population. Dieser Parameter muß eine ganze Zahl  $Z \in [1, \infty]$  sein.

**lambda** Gibt die Zahl der zu erzeugenden Nachkommen an. Bei der Anzahl der Nachkommen muß es sich um eine ganze Zahl  $Z \in [1, \infty]$  handeln.

**maxloop** Nach der angegebenen Anzahl an Schleifendurchläufen wird der Vorgang abgebrochen.

**break** Nach Unterschreiten der angegebenen reellen Zahl  $\mathcal{R}$  für die Qualität des besten Individuums wird der Algorithmus abgebrochen.

**immortal** Macht die *immortal* besten Individuen unsterblich.

Aus einer Startgeneration von  $\mu$  Individuen werden durch den Selektionsprozeß Eltern selektiert, die durch den Rekombinationsprozeß  $\lambda$  Nachkommen erzeugen und durch den Mutationsprozeß mutiert werden.

Anschließend werden aus der sortierten Liste mittels eines inversen Gaußverteilung solange zufällig Individuen ausgewählt und eliminiert, bis die Mindestzahl  $\mu$  an Individuen erreicht wird.

Jetzt kann ein neuer Zyklus beginnen.

## 7.3 Operatoren

Nachfolgend werden schon implementierte Operatoren und Verfahren näher erläutert.

### 7.3.1 Mutationsoperatoren

**Integer Vertauschung** Zwei Integerwerte des Gencodes, dessen Positionen durch Zufall bestimmt werden, tauschen die Plätze.

**Double Linear einfach** Auf eine Komponente des als Vektor aus double Zahlen interpretierten Gencodes wird ein zufälliger Wert zwischen  $-maxstep$  und  $maxstep$  addiert. Dabei wird eine eventuelle Überschreitung des durch  $maxwidth$  festgelegten Wertebereiches beachtet.

**Double Linear mehrfach** Auf alle Komponenten des als Vektor aus double Zahlen interpretierten Gencodes wird ein zufälliger Wert zwischen  $-maxstep$  und  $maxstep$  addiert. Ansonsten gilt das für *Double Linear einfach* gesagte.

**Double Gauß einfach** Es gilt das für *Double Linear einfach* gesagte, nur daß hier der Zufallswert durch eine Gaußverteilung bestimmt wird.

**Double Gauß mehrfach** Es gilt das für *Double Linear mehrfach* gesagte, nur daß hier der Zufallswert durch eine Gaußverteilung bestimmt wird.

### 7.3.2 Rekombinationsoperatoren

**Verdoppelung** Führt eine einfache Verdoppelung des Gencodes durch.

**Mittelwertsbildung** Das Erbgut zweier Eltern wird als Vektor interpretiert und eine komponentenweise Mittelwertsrechnung durchgeführt. Der Nachkomme stellt den komponentenweisen Durchschnitt beider Eltern dar.

### 7.3.3 Selektionsstrategien

**Standard** Aus der sortierten Liste wird anhand einer Gaußverteilung zufällig ein Individuum ausgewählt.

**Wettkampf** Zwei zufällig ausgewählte Individuen treten zu einem Zweikampf an, daß mit der höheren Fitneß gewinnt.

**random** Ein zufälliges Individuum wird selektiert.

Die Selektionsstrategie wird bei den meisten Strategien vom Algorithmus selbst bestimmt. Nur beim Algorithmus **ga** läßt sich die Selektionsstrategie frei wählen.

## 7.4 Problemorientierte Module

Durch Setzen der Variablen *PROBLEM* im Makefile und erneutes compilieren läßt sich eine Anpassung an spezielle Probleme verwirklichen. Nachfolgend werden zwei vorhandene problemspezifische Module beschrieben.

### 7.4.1 Parametrisierbare Probleme

Das Modul **fkt** beinhaltet Funktionen um das Minimum von parametrisierbaren Funktionen zu finden.

Das Tag **maxwidth** gibt die Grenzen des Suchverfahrens an. Es wird nur innerhalb des Bereichs  $[-\text{maxwidth}, \text{maxwidth}]$  gesucht, wobei die Grenzen als absolut anzusehen sind, d.h. ist ein Punkt auf dem Rand der niederste innerhalb des Bereiches so stellt er ein absolutes Minima dar.

Mit dem Tag **maxstep** wird die Maximale Schrittweite angegeben, d.h. die bei einem Schritt erreichbare Abweichung einer Koordinate beträgt zwischen  $-\text{maxstep}$  und  $\text{maxstep}$ .

Mit dem Tag **funktion** läßt sich eine der folgenden Qualitätsfunktionen auswählen. Die Dimension der Qualitätsfunktion muß über das Tag **dimension** angegeben werden.

**fkt0**  $f(x, y) = x^2 + y^2$

**fkt1**  $f(x, y) = \text{int}(x) + \text{int}(y)$

**fkt2**  $f(x, y) = x \cos(x) + y \cos(y)$

Die Mutationsoperatoren sind folgendermaßen zugeordnet:

**mut0** Double Gauß mehrfach

**mut1** Double Gauß einfach

**mut2** Double Linear einfach

**mut3** Double Linear mehrfach

Die Rekombinationsoperatoren wurden folgendermaßen benannt:

**sex0** Verdoppelung

**sex1** Mittelwertsbildung

Die Selektionsoperatoren wurden folgendermaßen gewählt:

**standard** Standard

**wettkampf** Wettkampf

**random** Random

## 7.4.2 Das Traveling-Salesman Problem

Ein Handlungsreisender muß eine bestimmte Anzahl an Orten hintereinander besuchen und soll wieder an seinem Ausgangspunkt zurückkommen. Es existieren Wege von jedem Ort zu allen anderen. Gesucht wird nun der kürzeste Weg dieser Reise. In Abschnitt 6.5 wird dieses Problem näher erläutert.

Mit dem Tag **ortefile** wird der Filname des einzulesenden Ortefiles angegeben. In diesem File sind alle Koordinaten der zu besuchenden Orte verzeichnet. Pro Zeile steht ein  $(x, y)$ -Koordinatenpaar, wobei die einzelnen Komponenten durch Space getrennt werden. Das Format stellt sich also folgendermaßen dar:

<x> <y>

Kommentarzeilen werden durch ein führendes „#“ als erstes Zeichen der Zeile gekennzeichnet.

Als einziger Mutationsoperator ist der Operator **mut0** als „Integer Vertauschung“ implementiert.

Für den Rekombinationsoperator und die Selektionsstrategie gilt die Zuordnung aus Abschnitt 7.4.1.

Die Algorithmen *ga* und *es* sind für dieses Problem nicht integriert.

## 7.5 Konzept und Source

Das Programm teilt sich in folgende Module auf:

**ev** Hauptprogramm

**evsup** Algorithmen und zugehörige Unterprogramme

**tager** Konfiguration

**random** Zufallszahlen

**problem** Übergeordnete problemspezifische Deklarationen

**fkt** Routinen für parametrisierbare Probleme

**traveller** Routinen für das Travelling-Salesman-Problem

Die einzelnen Module sind soweit es geht in sich gekapselt. Leider ließen sich im Fortgang des Projektes Abweichungen von dieser Kapselung nicht unbedingt vermeiden. Somit wird keine Garantie dafür übernommen, daß sich wirklich jedes Modul für sich verwenden läßt. Vor allem zwischen dem Modul *ev* und dem Modul *evsup* bestehen zahlreiche Querreferenzen. Das Modul *tager* hingegen läßt sich ohne weiteres auch für andere Aufgaben einsetzen. In ihm sollte es keine Querreferenzen geben. Die Module *fkt* und *traveller* benötigen jeweils das Includefile *problem* und sind nicht als eigenständige Module zu betrachten, sondern stellen lediglich eine Auslagerung problemspezifischer Teilroutinen aus dem Modul *evsup* dar.

Die Auslagerung problemrelevanter Programmteile garantiert eine universelle Verwendbarkeit und erleichtert die Implementation neuer Problemstellungen. Im Makefile muß lediglich der Variable *PROBLEM* das zu berücksichtigende Modul angegeben werden. Nun kann durch einen erneuten Compilervorgang ein an das Problem angepaßtes Programm erzeugt werden.

Für eine nähere Betrachtung des Sourcecodes sei auf Anhang 4 hingewiesen. Dort ist der gesamte Sourcecode des Programmes abgedruckt.

## 7.6 Erweiterungsmöglichkeiten

Wie schon im vorhergehenden Abschnitt erläutert, kann das Programm durch Austauschen eines der Module *fmt* oder *traveller* an ein spezielles Problem angepaßt werden. Wie das problemspezifische Modul auszusehen hat, läßt sich leicht durch Betrachtung der beiden genannten Module ersehen.

Das Programm kann durch die Implementierung weiterer Algorithmen erweitert werden. Oft genügt es aber einfach weitere Mutationsoperatoren, Rekombinationsoperatoren oder Selektionsstrategien zu implementieren und einen Algorithmus zu wählen, bei dem diese frei wählbar sind.

## 7.7 Bekannte Fehler

Das Programm enthält einen Fehler in der Speicherverwaltung. Es gibt nach Programmende nicht allen belegten Speicher frei. Dies stellt unter UNIX kein Problem dar, da dort nach Programmende alle vom Prozeß belegten Ressourcen automatisch freigegeben werden. Auf DOS ähnlichen Umgebungen sollte dieser Umstand jedoch berücksichtigt werden.

Der Algorithmus *ga* führt wegen diesem Problem manchmal zu segmentation faults.

# Kapitel 8

## Simulationsmodell

In diesem Kapitel soll versucht werden, ein universelles Simulationsmodell für Evolutionsalgorithmen zu entwickeln. Dazu muß aber bemerkt werden, daß es keinen universellen Problemlöser gibt, auch nicht mit Hilfe von Evolutionsalgorithmen.

In diesem Kapitel wird vielmehr versucht ein Konzept zu entwerfen, welches, soweit möglich, alle Ideen der Evolutionstheorie in einem gesamtheitlichen Mechanismus unterbringt.

### 8.1 Die Idee

Auf einer abgelegenen grünen Insel, die nur über eine Fähre zu betreten oder zu verlassen ist, lebt ein vegetarischer Schäfer mit einer Herde Schafe. Auf der Insel lebt außerdem noch ein Wolf, der besonders gerne die Schafe des Schäfers reißt. Zu allem Unglück verstrahlt ein leckgeschlagenes Atomkraftwerk die Landschaft.

Die Insel bietet ausreichend Nahrung für eine maximale Anzahl an Schafen. Der Schäfer wird in die Nahrungsbilanz nicht miteinbezogen. Solange mindestens ein Schaf lebt kann sich der Schäfer von dessen Produkten ernähren. Der Wolf ernährt sich von den gerissenen Schafen und wird bei dieser Bilanz auch nicht berücksichtigt.

Die Insel wird durch nichts beeinflußt, was nicht den Weg über die Fähre genommen hat, d.h. sie ist isoliert und stellt einen Mikrokosmos dar.

### 8.2 Der Lebenskreislauf

Die einzelnen Komponenten des Simulationsmodells werden durch Symbole dargestellt, die ähnliche Funktionen in einem Modul zusammenfassen.

Der Algorithmus wird durch eine Initialisierungsroutine, die sich im Modul *Insel* befindet, gestartet, indem eine Gründungsgeneration an Schafen erzeugt wird.

Die Gründerpopulation wird nun durch das Modul *Auge* bewertet. Nun kann der Evolutionszyklus beginnen.

Die Schafe suchen sich Partner und zeugen Nachkommen oder erzeugen durch klonen identischen Nachkommen. Dies wird durch das Modul *Herz* symbolisiert. Nun tritt das Modul *Atomkraftwerk* in Aktion und mutiert die Schafe. Das *Auge* bewertet wiederum die gesamte Schafepopulation. Danach führt der *Wolf* die Selektion durch, indem er Schafe eliminiert. Nun legt die *Fähre* an die Insel an und tauscht Informationen von den Nachbarinseln aus. Zu guter Letzt führt der *Schäfer* eine Bilanz durch und entscheidet, ob der Zyklus fortgesetzt wird oder abgebrochen wird.

In Abbildung 8.1 ist der ganze Zyklus durch eine Grafik dargestellt.

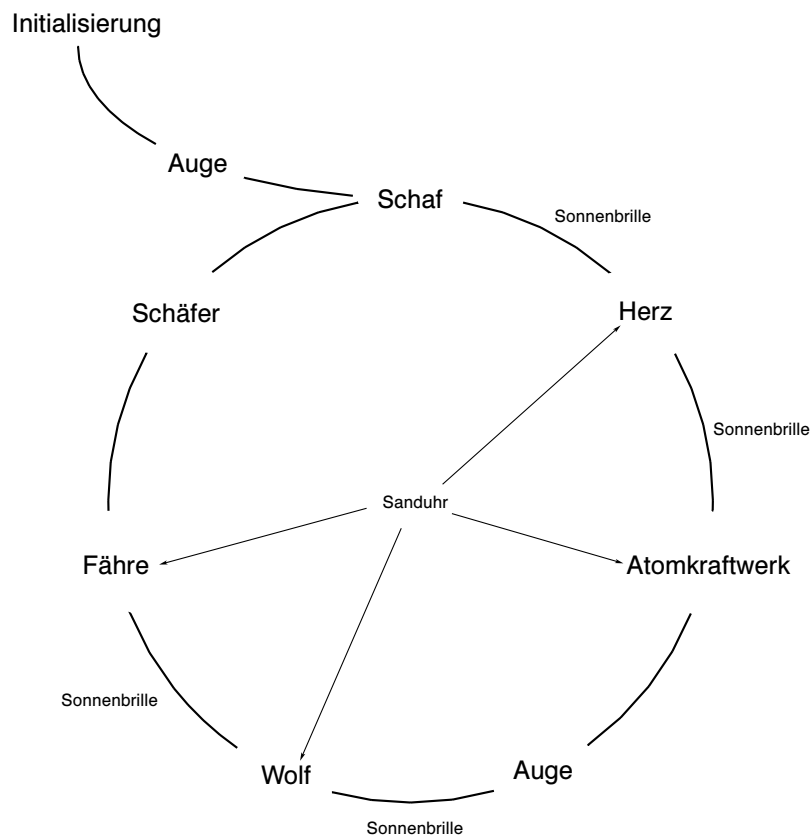


Abbildung 8.1: Der Lebenskreislauf auf der Schafinsel

### 8.3 Objekte und Operatoren

Die einzelnen Komponenten des Zyklusses lassen sich in Objekte und Operatoren einteilen. Die Schafe sind dabei das Objekt, auf das der Reihe nach die

Modul	Aufgabe	Art
Schaf	Träger des Gencodes Träger Individuum abhängiger Informationen	Objekt
Insel	Konfiguration Initialisieren der Gründungspopulation Reservieren der Ressourcen (Speicher u.s.w.)	Modul
Herz	Partnerwahl Rekombination	Operator
Auge	Qualitätsfunktion Fitneßfunktion	Operator
Wolf	Selektionsoperator	Operator
Schäfer	Abbruchbedingung Statistikberechnung	Operator
Fähre	Austausch von Informationen mit anderen Populationen	Operator
Sonnenbrille	Filter	Filter
Sanduhr	Steuerfunktionen	Funktionen

Tabelle 8.1: Übersicht der Module

Operatoren: Herz, Atomkraftwerk, Auge, Wolf, Fähre und Schäfer angewendet werden. Zusätzlich wird noch ein Filter angeboten, der vor die einzelnen Operatoren geschaltet werden kann, um eine Vorauswahl aus den Objekten (Schafe) zu erhalten. Als weitere Option läßt sich ein Modul einrichten, daß Funktionen zur Verfügung stellt, die von der Anzahl der durchlaufenen Generationen oder der seit dem Start des Zyklusses verstrichenen Zeit abhängen. Wir nennen dieses Modul Wolke.

## 8.4 Übersicht der Module

In Tabelle 8.1 werden alle Module mit einer Kurzbeschreibung ihrer Funktionen aufgelistet.

## 8.5 Beschreibung der einzelnen Module

Im folgenden werden die einzelnen Module nur kurz in ihrer wesentlichen Aufgabe beschrieben.

**Insel** Das Modul Insel beinhaltet mehrere Funktionen und dient gleichzeitig als Austragungsort für den Evolutionsprozeß.

Bei der Konfiguration werden die anderen Operatoren eingestellt und mit den benötigten Einstellungen für den gewünschten Algorithmus versorgt.

Der für die Objekte benötigte Speicher wird alokiert und alle benötigten Ressourcen reserviert.

Mittels einer Initialisierungsroutine wird eine Gründerpopulation an Schafen geschaffen und mit einem zufälligen Gencode versorgt. Die Mutations-

und die Rekombinationsmaske werden gesetzt. Danach findet eine erste Bewertung durch den Operator Auge statt.

**Schaf** Die Schafe stellen die Stichproben, d.h. den Phänotyp, im Lösungsraum dar und müssen sich in ihrem Lebensraum behaupten, um dem Wolf zu entkommen.

Das Objekt Schaf ist im Grunde genommen nur eine Datenstruktur, welche folgende Daten beinhaltet:

- Gencode
- Mutationsmaske
- Rekombinationsmaske
- Fitneßbewertung
- Alter in Generationen

Alle Operatoren arbeiten auf dieser Datenstruktur.

**Wolf** Der Wolf übernimmt den Part der Selektion. Durch den Filter Sonnenbrille selektierte Schafe werden durch das Modul Wolf eliminiert. Der Selektionsprozeß kann durch Funktionen des Moduls Sanduhr beeinflusst werden.

**Auge** Stellt die für das Problem spezifische Qualitätsfunktion zur Verfügung. Aus der Qualität wird durch eine Fitneßfunktion anhand weiterer Einflußgrößen, wie Populationsgröße oder Funktionen der Sanduhr eine für jedes Schaf spezifische Fitneß berechnet.

**Herz** Der Operator Herz wählt durch den Filter Sonnenbrille eine Untermenge der Objekte Schafe aus. Aus dieser Untermenge werden wiederum Eltern-Schafe ausgewählt und diese erzeugen die gewünschte Anzahl Nachkommen durch Rekombination oder klonen. Der ganze Prozeß kann durch das Modul Sanduhr beeinflusst werden.

**Atomkraftwerk** Das Atomkraftwerk mutiert mit seiner Strahlung die Schafe und sorgt so für neue Spielmasse des Selektion-Rekombination-Zyklus.

**Schäfer** Der Schäfer prüft die Abbruchbedingung und leitet, falls diese erfüllt ist, das Abbruchverfahren ein. Es werden die Ergebnisse und die ebenfalls in diesem Modul erstellte Statistik auf das gewünschte Medium ausgegeben.

**Fähre** Über die Fähre können mehrere parallel laufende Evolutionsprozesse miteinander kommunizieren. Welche Individuen in diesem Austauschprozeß berücksichtigt werden kann durch den Filter Sonnenbrille und durch das Modul Sanduhr beeinflusst werden.

**Sonnenbrille** Die Sonnenbrille stellt Routinen zur Bildung einer Untermenge der aktuellen Schafpopulation zur Verfügung. Sie stellt einen Filter mit Kriterien wie z.B. Alter oder Fitneß für die Operatoren dar.

**Sanduhr** Die Sanduhr beinhaltet Funktionen, die von der Anzahl der seit dem Start der Simulation durchlaufenen Generationen oder der verstrichenen Zeit abhängen. Mit Hilfe von diesen Funktionen lassen sich Algorithmen mit Steuergrößen verwirklichen, die von der Zeit abhängen. Desweiteren werden Funktionen angeboten, die von dem Fortschrittsverhalten der Population abhängen.

# Kapitel 9

## Zusammenfassung

Im folgenden werden die wichtigsten Erkenntnisse zusammengefasst und ein Ausblick auf zukünftige Möglichkeiten gegeben.

### 9.1 Ergebnis

Naturanaloge Optimierungsverfahren werden durch die steigende Rechnerkapazität immer interessanter. Die meisten der Natur abgeschauten Verfahren, wozu insbesondere die Evolutionsalgorithmen gehören, zeichnen sich durch eine hochgradige Parallelität aus.

Meistens ist eine möglichst schnelle Lösung gesucht und nicht immer die Optimale. Gerade bei solchen Anforderungen lassen sich Evolutionsalgorithmen einsetzen. Kommt zur Problemstellung noch eine gewisse Unschärfe dazu und hat man über die eigentlichen Internen Zusammenhänge kein Bild, ist dieses geradezu prädestiniert für die Lösung durch Evolutionsalgorithmen.

Zusammengefasst lässt sich sagen, daß Probleme für die ein alternatives in seiner Leistungsfähigkeit genügendes Lösungskonzept existiert nicht geeignet sind durch EAs gelöst zu werden. Erst wenn ungenügendes Wissen und die Forderung nach einer guten, aber nicht unbedingt optimalen Lösung, gepaart mit hohem bis unmöglichen Aufwand der alternativen Lösungsfindung zusammentreffen sollte man auf EAs zurückgreifen.

Sollte auf eine schnelle Lösungsfindung bei individuellen Problemen Wert gelegt werden, sind ebenfalls EAs zu empfehlen, da ohne das Wissen um interne Zusammenhänge meist schon hinreichend gute Lösungen gefunden werden können. Auch die Anpassung von EA an das Problem hält sich meist in Grenzen, sofern man einen hohen Ressourcenverbrauch nicht scheut. Die beiden Hauptäste der Evolutionstheorie, die Evolutionstrategien und die Genetischen Algorithmen, werden sich meiner Meinung nach irgendwann zu einer einheitlichen Theorie verschmelzen, da sie nur zwei Blickwinkel auf dasselbe Objekt darstellen.

## 9.2 Ausblick und Schlußbemerkung

Bisher müssen die EAs noch durch Experimente an die zu optimierenden Probleme angepasst werden, damit sie optimal konvergieren und Nebenmaxima überwinden können. Durch die Suche nach neuen Strategien der Anpassung könnte sich die Lösung von Optimierungsproblemen mittels Evolutionsalgorithmen jedoch in vielen Bereichen etablieren.

Auch die Verbreitung von Parallelrechnern wird dazu beitragen Evolutionsalgorithmen in immer mehr Bereichen einzusetzen.

Jedoch muß der Euphorie Grenzen gesetzt werden, da auch die Evolutionsalgorithmen nicht als ultimative Problemlöser einsetzbar sind, da sie immer gegenüber speziell angepassten Lösungsstrategien im Nachteil sein werden.

In Fällen in denen die Ermittlung der Qualität der Individuen einen zu großen Anteil an der Gesamtlast des Aufwandes ausmacht sind immer alternative Lösungswege zu bevorzugen.

Um Evolutionsalgorithmen effektiv einzusetzen sind immer große Rechnerkapazitäten notwendig.

# Anhang A

## Weitere Informationsquellen

### A.1 Vorlesungen

An der Universität Stuttgart wird seit Wintersemester 95/96 am Institut für Informatik eine Vorlesung *Naturallogie Verfahren* von Prof. Dr. rer. nat. Volker Claus gelesen. Diese beschäftigt sich unter anderem mit Evolutionsalgorithmen.

Hier eine grobe Gliederung der Vorlesung als Übersicht der behandelten Themen:

1. Einführung
2. Basis-Algorithmus
3. Simulated Annealing
4. Threshold Algorithmen
5. Genetische Algorithmen
6. Evolutionsstrategien
7. Genetische Programmierung
8. Problemanalyse
9. Unterschiede der Verfahren
10. Marktübersicht, Werkzeuge, ...
11. Ausblick

## **A.2 Internet**

### **A.2.1 Encore – the Evolutionary Computation Repository Network**

Unter diesem Titel verbirgt sich ein weltweiter Zusammenschluß von ftp- und www-Servern. Dort befinden sich eine Menge Artikel zu Evolutionsalgorithmen und ähnlichen Themen.

Man kann diesen Server über das Deutsche EUnet unter der Adresse:

*ftp://ftp.Germany.EU.net/pub/rearch/softcomp/EC*

erreichen oder unter der WWW-Adresse:

*http://www.gemany.eu.net.*

### **A.2.2 Newsgroups**

Zum Thema Genetische Algorithmen existiert auch eine Newsgruppe in der aktuelle Themen des Gebietes diskutiert werden. Ihr Name ist:

*comp.ai.genetic*

## Anhang B

# Maschinell erzeugte Zufallszahlen

### B.1 Linearverteilung

Für die Erzeugung linearverteilter Zufallszahlen stehen im Modul *random* mehrere Algorithmen zur Verfügung. Diese Algorithmen wurden dem Buch [PTVF92] entnommen.

Es folgt eine kurze Diskussion der Vor- und Nachteile der einzelnen Routinen, die im Modul *random* integriert sind (siehe Anhang 4).

**ran0** Standard Algorithmus, der nur für anspruchslose Aufgaben geeignet ist.

**ran1** Dieser Algorithmus wird für Aufgaben empfohlen, bei denen nicht mehr als 100.000.000 Zufallszahlen in einer Folge benötigt werden. Die Ausführungszeit beträgt ca. das 1.3 fache des *ran0* Algorithmus.

**ran2** Falls man eine Sequenz mit mehr als 100.000.000 Zufallszahlen benötigt wird dieser Algorithmus empfohlen. Seine relative Ausführungszeit beträgt das 2.0 fache des *ran0*.

**ran3** Benötigt zur Ausführung nur das 0.6 fache wie der Referenzalgorithmus *ran0*, ist aber noch nicht sehr gut erforscht und stellt keinen Standard dar.

### B.2 Gaußverteilung

Die Abbildung B.1 zeigt eine normierte Darstellung der Verteilung des Gauß-Zufalls-Zahlen-Algorithmus aus [PTVF92]. Dieser Algorithmus wurde in das Modul *random* als Funktion *double dNorm (double mean, double max)* integriert. Zur Erstellung dieses Graphen wurden 1.000.000 Stichproben mit einer

„Varianz“ von 1 und einem Mittelwert von 0, genommen und diese normiert aufgetragen.

Wie man unschwer erkennen kann handelt es sich bei dem verwendeten Algorithmus nur um eine Aproximation an die Gaußverteilung.

Hier wurde die Gaußkurve durch eine Halbkreisfunktion angenähert, was im Bereich um den Mittelwert gute Ergebnisse hervorbringt, aber in den äußeren Bereichen zu großen Abweichungen führt. Diese Abweichungen fallen aber nicht zu sehr ins Gewicht, da naturgemäß die äußeren Bereiche der Gaußverteilung weniger selten angesprochen werden.

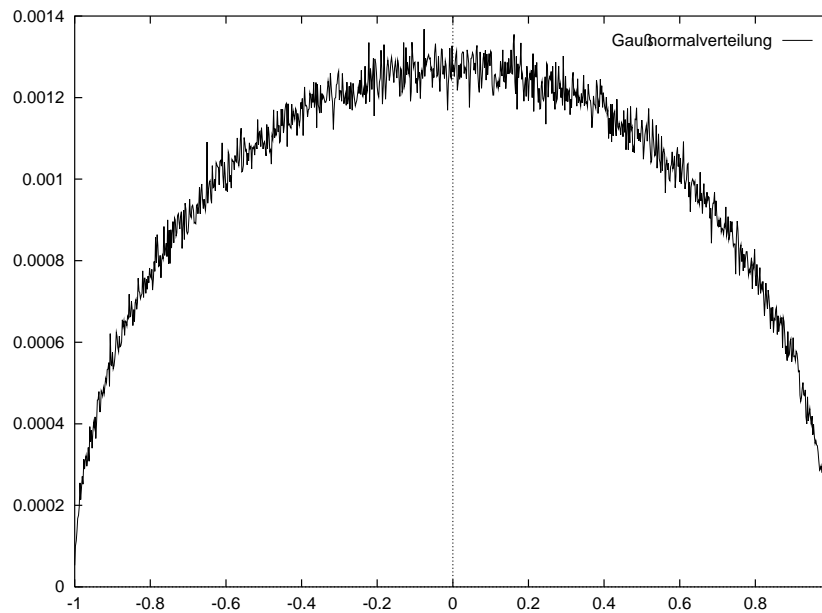


Abbildung B.1: Gaußnormalverteilung nach einem Algorithmus aus [PTVF92]

# Anhang C

## Begriffe und Abkürzungen

In diesem Teil des Anhangs werden die meistverwendeten Begriffe und Abkürzungen erklärt, die in dieser Arbeit verwendet werden und nicht als allgemein bekannt vorausgesetzt werden können oder einer näheren Definition ihres Gebrauches bedürfen.

**Allele** Merkmale eines Individuums.

**Chromosom** Fadenförmige doppelsträngige Spiralstruktur, die die Erbinformation einer Zelle in Form von Genen trägt.

**DNS** Desoxyribonukleinsäure, die wichtigste in den Chromosomen vorkommende Nukleinsäure stellt den Träger der Erbinformation auf molekularer Ebene dar.

**EA** Evolutionsalgorithmen (Oberbegriffe für alle Algorithmen, die auf der Nachahmung der Evolution beruhen)

**Erbgut** Gesamtheit aller in den Genen codierten, an die Nachkommen weitergebbaren Allele.

**ES** Evolutionsstrategien (von Rechenberg entwickelte Variante der Evolutionstheorie)

**GA** Genetische Algorithmen (von Holland begründete Variante der Evolutionstheorie)

**Gen** Einheit im DNA-Doppelstrang des Chromosoms, die den Bauplan für ein Proteinmolekül enthält.

**Gendrift** Die auf Zufall beruhende signifikante Veränderung der Merkmalsausbildungen einer Population.

**Genotyp** Die Gesamtheit des Erbgutes eines Individuums.

**Hamming-Abstand** Anzahl der unterschiedlichen Bits zweier Codeworte.

**Individuum** Ein einzelner Vertreter einer Population repräsentiert durch sein Erbgut und seine Allele.

**Phänotyp** Die Summe aller beobachtbaren Merkmale eines Individuums.

**TSP** Travelling-Salesman-Problem (Eine Anzahl Orte soll auf dem kürzesten Weg miteinander verbunden werden).

# Kapitel 4

## Source

Der als extra Einlage beigefügte Sourcecode wurde durch die UNIX Befehlsfolge:  
`a2ps <sourcefile> | lpr -P<drucker>`  
erzeugt.

# Literaturverzeichnis

- [Abl87] Paul Ablay. Optimieren mit Evolutionsstrategien. *Spektrum der Wissenschaft*, Juli 1987.
- [BBM93a] David Beasley, David R. Bull, and Ralph R. Martin. An overview of genetic algorithms, part 1, fundamentals. Technical report, University Computing, 1993.
- [BBM93b] David Beasley, David R. Bull, and Ralph R. Martin. An overview of genetic algorithms, part 2, research topics. Technical report, University Computing, 1993.
- [Gol89] D.E. Goldberg. *Genetic Algorithms in search, optimization and machine learning*. Addison-Wesley, 1989.
- [Hol75] John H. Holland. *Adaptation in Natural and Artificial Systems*. MIT Press, 1975.
- [Hol92] John H. Holland. Genetische Algorithmen. *Spektrum der Wissenschaft*, September 1992.
- [PTVF92] William H. Press, Saul A. Teuholsky, William T. Vetterling, and Brian P. Flanery. *Numerical Recipes in C, The Art of Scientific Computing*. Cambridge University Press, 1992.
- [Rec73] Ingo Rechenberg. *Evolutionsstrategie, Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Friedrich Frommann Verlag, 1973.
- [Rec94] Ingo Rechenberg. *Evolutionsstrategie '94*. ??, 1994.
- [RS] Günter Rudolph and Hans-Paul Schwefel. Evolutionäre Algorithmen: Ein robustes Optimierkonzept. Technical report, Universität Dortmund Fachbereich Informatik, ??
- [SHF94] Eberhard Schöneburg, Frank Heintzmann, and Sven Feddersen. *Genetische Algorithmen und Evolutionsstrategien*. Addison-Wesley, 1994.
- [Wec93] Guido Weckwerth. Zeugende Zahlen. *mc*, Oktober 1993.
- [Wec94] Guido Weckwerth. Der Weg zur Drei. *mc*, Januar 1994.

[WHR<sup>+</sup>87] Watson, Hopkins, Roberts, Steifz, and Weiner. *Molecular Biology of the Gene Volume I, Fourth Edition*. The Benjamin / Cummings Publishing Company, Inc, 1987.