Parallele Evolutionsstrategien mit der Optimierungsumgebung EvA

Dissertation

der Fakultät für Informatik
der Eberhard-Karls-Universität Tübingen
zur Erlangung des Grades eines
Doktors der Naturwissenschaften
vorgelegt von

Dipl.-Inform. Jürgen Wakunda

Tübingen
2001

Tag der mündlichen Qualifikation: 9. Mai 2001
Dekan: Prof. Dr. rer. nat. habil. Andreas Zell
1. Berichterstatter: Prof. Dr. rer. nat. habil. Andreas Zell
2. Berichterstatter: Prof. Dr. rer. nat. habil. Michael Kaufmann

Kurzfassung

Evolutionäre Algorithmen stellen verschiedene Verfahren für ein breites Gebiet von Optimierungsaufgaben bereit. Allen Verfahren liegt die von der Natur inspirierte Technik der schrittweisen Optimierung durch Prinzipien der Evolution zugrunde. Die Evolutionsstrategie als ein Vertreter der Evolutionären Algorithmen ist besonders geeignet für Anwendungen, die durch einen Vektor kontinuierlicher Parameter beschrieben werden können. Ziel dieser Arbeit ist es, die praktische Anwendung von Evolutionsstrategien bei Optimierungsaufgaben zu vereinfachen. Dazu werden drei Aspekte besonders behandelt, die bei der praktischen Anwendung relevant sind: Parallelisierung von Evolutionsstrategien, das Finden von guten Parametern für die Evolutionsstrategie selbst und die Optimierungsumgebung *EvA*, welche diese Methoden und den Einsatz von Evolutionsstrategien als Optimierungskomponente in einem System unterstützt.

Für Evolutionsstrategien existieren gut entwickelte Mechanismen zur Selbstadaption der Größe der Suchschritte während der Optimierung. Dies bringt einen wesentlichen Geschwindigkeitsgewinn. Für sehr rechenintensive Anwendungen existiert eine einfache und effiziente Standard-Methode zur Parallelisierung. Wird sie auf Evolutionsstrategien angewendet, wird dabei jedoch der wichtige Prozeß der Selbstadaption gestört und die Leistungsfähigkeit der Evolutionsstrategie vermindert. In dieser Arbeit wird die neu entwickelte *Median-Selektion* vorgestellt, welche diesen Nachteil nicht besitzt. Somit kann der Geschwindigkeitsgewinn durch Parallelisierung unter Erhaltung der Selbstadaption ausgenutzt werden.

Beim Einsatz der Evolutionsstrategie müssen durch Wahl von Parametern viele Details des Algorithmus festgelegt werden, welche die Leistungsfähigkeit deutlich beeinflussen. Besonders wenn der Optimierungsprozeß oft und mit hohen Anforderungen an eine kurze Ausführungszeit stattfinden muß, lohnt sich eine gute Wahl der Parameter. Solch eine *Meta-Optimierung* kann selbst von Experten nicht immer zufriedenstellend bewältigt werden. Deshalb wird in dieser Arbeit die Erweiterung einer Evolutionsstrategie zur Meta-Optimierung vorgestellt. Auch hier ist Selbstadaption enthalten und wegen des hohen Rechenaufwands ist der Algorithmus parallelisiert. Daher kann sich hier der Einsatz der neuen Median-Selektion ebenfalls lohnen.

Die Optimierungsumgebung EvA implementiert neben parallelen Evolutionsstrategien sowohl die Median-Selektion als auch die Meta-Optimierung. Darüberhinaus werden Aspekte des praktischen Einsatzes von Evolutionsstrategien unterstützt: einfache und flexible Implementierung von Fitneßfunktionen. Dabei sind viele Rahmenbedingungen, welche die Problemstellung vorgibt, flexibel handhabbar. Der Optimierungsschritt stellt in realen Anwendungen oft nur ein Baustein in einem komplexeren Softwaresystem dar. Es ist deshalb eine Unterstützung für eine einfache Einbettung und Interaktion mit dem Gesamtsystem vorhanden.

Es kann gezeigt werden, daß die Median-Selektion sehr gut für den Einsatz von parallelen Evolutionsstrategien geeignet ist. Die Meta-Evolutionstrategie kann als einfache Methode zur Parameterfindung für Evolutionsstrategien dienen.

Danksagung

Die vorliegende Arbeit entstand während meiner Tätigkeit als wissenschaftlicher Mitarbeiter am Lehrstuhl Rechnerarchitektur der Universität Tübingen. Ich möchte an dieser Stelle allen meinen Dank aussprechen, die durch ihre Hilfe und Anregungen direkt oder indirekt zu dieser Arbeit beigetragen haben.

An erster Stelle bin ich meinem Betreuer und Erstberichterstatter Prof. Dr. Andreas Zell zu Dank verpflichtet. Seine Anregungen trugen wesentlich zur Verbesserung der Qualität der Arbeit bei. Die freie Arbeitsatmosphäre am Lehrstuhl bot hervorragende Rahmenbedingungen für die Entstehung dieser Arbeit.

Prof. Dr. Michael Kaufmann danke ich für die Übernahme des Zweitgutachtens, welches er trotz seines engen Zeitplanes zügig anfertigen konnte.

Besonderer Dank gilt meinen Kollegen am Lehrstuhl, die zu einem hervorragenden Arbeitsklima beigetragen haben. Bei Kollege Igor Fischer möchte ich mich insbesondere dafür bedanken, daß er mich im letzten Semester dieser Arbeit wesentlich von meinen Verpflichtungen im Lehrbereich entlastet hat.

An dieser Stelle sei auch allen Studenten mein Dank ausgesprochen, die durch ihre Diplomarbeiten oder ihre Tätigkeiten als wissenschaftliche Hilfskraft einen Beitrag zu dieser Arbeit geleistet haben. Dies sind im einzelnen Volker Scheer, Hartmut Ott und Ralf Sondershaus.

Desweiteren bin ich allen Freunden und Kollegen zu Dank verpflichtet, die durch Korrekturlesen zur Verbesserung der Arbeit beigetragen haben.

Den herzlichsten Dank spreche ich hiermit meinen Eltern Johanna und Horst Wakunda aus, die es mir ermöglicht haben, meinen Weg zu gehen und auf deren Unterstützung und Vertrauen ich immer zählen konnte.

Jürgen Wakunda

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1 Einordnung

Das Gebiet der Evolutionären Algorithmen erfreut sich in den letzten Jahren zunehmender Beliebtheit in der Wissenschaft. Aber auch in der Industrie-Praxis etablieren sich diese Verfahren immer mehr. Sie gelten als leistungsfähige und robuste Optimierungsverfahren, die für Problemstellungen eingesetzt werden können, die sich mit herkömmlichen Methoden nicht oder nicht mit vertretbarem Aufwand lösen lassen.

Evolutionäre Algorithmen (EA) sind von der natürlichen Evolution inspirierte und für ein breites Einsatzgebiet verwendbare, stochastische Optimierungsverfahren. Stochastisch bedeutet dabei, daß statt deterministischer Prozesse Zufallszahlen verwendet werden, um z. B. bestimmte Entscheidungen im Ablauf zu treffen.

Die Varianten der EAs wurden zunächst unabhängig voneinander erfunden und haben sich lange Zeit auch getrennt voneinander entwickelt. Erst als die EAs populärer wurden trafen sich 1991 auf der International Conference for Genetic Algorithms (ICGA) Vertreter der unterschiedlichen Entwicklungslinien und tauschten sich erstmals untereinander aus. Hierbei wurde der Begriff *Evolutionäre Algorithmen* (*evolutionary algorithms* oder auch *evolutionary computation*) als Oberbegriff festgelegt, der die drei Teilgebiete *Genetic Algorithms* (GA), *Evolutionary Programming* (EP) und *Evolution Strategies* (ES) bezeichnete. Aus GAs hervorgegangen sind die Methoden der *Classifier Systems* (CS) und *Genetic Programming* (GP), welche sich inzwischen zu gleichwertigen Zweigen entwickelt haben. Auch die Methode des *Simulated Annealing* (SA) wird heutzutage wegen ihrer Ähnlichkeit zu den anderen Verfahren hinzugezählt (s. Abb. 1.1). Die Evolutionären Algorithmen wiederum ordnen sich in das Gebiet der *Computational Intelligence* zusammen mit *Neuronalen Netzen* und *Fuzzy Logic* ein. Dies sind alles numerische Verfahren, die auf ungenauen und nicht zuverlässigen Werten arbeiten, die aus Prozessen der realen Welt gewonnen werden. Für genau dasselbe Gebiet ist in neuerer Zeit die Bezeichnung *Soft Computing* aufgetaucht, wel-

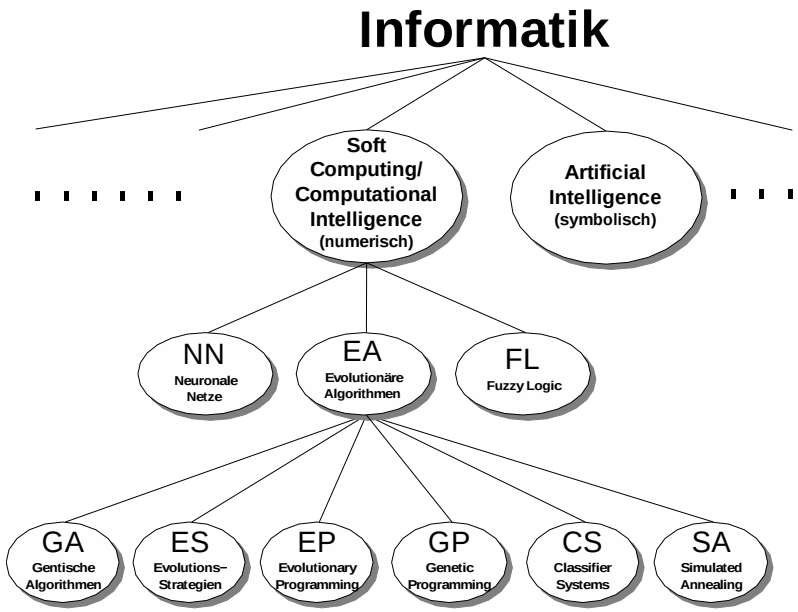


Abbildung 1.1: Eine Einordnung von Evolutionären Algorithmen in die Informatik

che die Toleranz der Verfahren gegenüber ungenauen Eingabewerten betonen soll, im Gegensatz zum konventionellen (*Hard) Computing*[1].

Die vorliegende Arbeit beschäftigt sich in erster Linie mit Evolutionsstrategien. Falls die gemachten Aussagen aber auch allgemein auf Evolutionäre Algorithmen zutreffen, so wird im folgenden dafür auch der Begriff EA verwendet.

Die den EAs zugrundeliegende Idee wurde aus der Natur kopiert und in abstrakterer Form in Optimierungsverfahren abgebildet. Die natürliche Evolution kann als ständiger, schrittweiser Optimierungsprozeß aufgefaßt werden, mit der Lebewesen im Laufe der Naturgeschichte immer besser an ihre Umwelt angepaßt werden, in der sie überleben müssen. Charles Darwin (1809-1882) wurde als bedeutendster Fürsprecher der Evolutionstheorie bekannt, die er in seinem Buch “On the Origin of Species by Means of Natural Selection, or the Preservation of Favoured Races in the Struggle for Life” beschreibt [Darwin 29]. Die beiden der Evolution zugrundeliegenden Prinzipien sind die Mutation und die Selektion. Mutationen sind zufällige Veränderungen des Erbgutes, die zu leichten Variationen der resultierenden Lebewesen führen. Selektion findet statt, wenn sich diese Unterschiede relevant auf die Fähigkeiten der Fortpflanzung und des Überlebens auswirken, so daß eine erhöhte oder verminderte Wahrscheinlichkeit besteht, daß das veränderte Erbgut an Nachkommen weitergegeben wird.

Andere Länder, wie z. B. die USA, scheinen einen Vorsprung zu haben, was den in-

[1] Der Begriff *Hard Computing* wird allerdings in der Praxis nicht verwendet.

dustriellen Einsatz solcher Verfahren angeht (s. [Evo95]). Wahrscheinlich dominieren deshalb in der einschlägigen Fachliteratur über Evolutionäre Algorithmen deutlich die Genetischen Algorithmen (GA), deren Ursprung in den USA liegt.

Die Evolutionsstrategie (ES), welche ebenfalls zu den Evolutionären Algorithmen gehört, ist besonders geeignet für eine Klasse von Anwendungen, deren Lösungen durch einen Vektor kontinuierlicher Parameter beschrieben werden können. Diese Arbeit konzentriert sich auf drei Aspekte, welche bei der praktischen Anwendung von Evolutionsstrategien relevant sind: Parallelisierung von Evolutionsstrategien, das Finden von guten Parametern für die Evolutionsstrategie selbst und die Optimierungsumgebung EvA, welche diese Methoden und den Einsatz von Evolutionsstrategien als Optimierungskomponente in einem System unterstützt.

1.2 Motivation

Praktische Optimierungsaufgaben sind oft sehr rechenintensiv, wenn z. B. komplexe Simulationen involviert sind. Der Rechenaufwand wird üblicherweise in der Anzahl der Funktionsauswertungen gemessen, die jeweils die Qualität einer Lösung berechnen. Braucht eine einzelne dieser Auswertungen relativ viel Zeit (mind. im Sekundenbereich), so kann man den Optimierungsvorgang nahezu linear beschleunigen, indem man die Auswertungen auf mehrere Prozessoren verteilt. Insbesondere im Zeitalter des Internets, wo die Vernetzung vieler Computer eine Selbstverständlichkeit ist und somit die Infrastruktur bereits gegeben ist, gewinnt Parallelisierung immer mehr an Bedeutung.

Für diese Methode der Parallelisierung sind sogenannte *Steady-State*-Algorithmen besonders geeignet. Diese berechnen im Gegensatz zu den üblichen, auf Generationen basierenden Algorithmen in einem Schritt nicht eine ganze Menge von Individuen, sondern nur jeweils ein einzelnes Individuum. Dadurch lassen sie sich besonders leicht asynchron parallelisieren, um die beteiligten Rechner besser auszulasten.

Im Gegensatz zu den weit verbreiteten Genetischen Algorithmen, die ebenfalls zur Gruppe der Evolutionären Algorithmen gehören, existieren für Evolutionsstrategien gut entwickelte Mechanismen zur dynamischen Anpassung der Suchschritte während der Optimierung. Dies wird *Selbstadaption* genannt. Dadurch wird die Optimierung wesentlich beschleunigt und die gute und effiziente Lösung mancher Aufgaben überhaupt erst möglich. Verwendet man die oben beschriebenen Steady-State-Algorithmen mit einer solchen selbstadaptiven Evolutionsstrategie, wird dabei jedoch die Selbstadaption gestört und die Leistungsfähigkeit der Optimierung vermindert. In dieser Arbeit wird das neu entwickelte Verfahren *Median-Selektion* vorgestellt, welches diesen Nachteil nicht besitzt. Somit kann der Geschwindigkeitsgewinn durch Parallelisierung unter Erhaltung der Selbstadaption ausgenutzt werden.

Das Optimierungssystem sollte eine breite Palette an algorithmischen Varianten bieten, z. B. verschiedene Selektionsverfahren, Adaptionsverfahren für die Mutationsschrittweiten, usw. so daß geprüft werden kann, ob und welche davon sich evtl. vorteilhaft bei der Bearbeitung der gegebenen Anwendung auswirken. Diese Vielfalt kann andererseits aber auch zum Problem werden, denn in der Regel ist der Anwender im Fachgebiet der speziellen Optimierungsanwendung erfahren, aber nicht unbedingt sehr gut mit dem Optimierungsverfahren an sich vertraut. Deshalb ist ein weiterer wichtiger Aspekt bei der Anwendung von Evolutionsstrategien die Wahl der Parameter des Algorithmus. Dies kann die Leistungsfähigkeit hinsichtlich der Lösungsqualität oder der Geschwindigkeit, mit der diese gefunden wird, deutlich beeinflussen. Muß der Optimierungsprozeß oft und mit hohen Anforderungen an eine kurze Ausführungszeit stattfinden, so lohnt sich eine gute Wahl der Parameter besonders. Dies stellt für sich gesehen wiederum eine Optimierungsaufgabe dar. Da diese auf einer Ebene über der eigentlichen Optimierungsanwendung abläuft, wird dies *Meta-Optimierung* genannt. Dies kann selbst von Experten nur durch Erfahrungswerte oder Testen einzelner Parameterbereiche angegangen werden. Die ganze Komplexität dieser Aufgabe kann dabei aber nicht bewältigt werden. Deshalb wird in dieser Arbeit die Erweiterung einer Evolutionsstrategie zur Meta-Optimierung vorgestellt. Auch hier ist ein Mechanismus zur Selbstadaption enthalten. Weil sich der Rechenaufwand bei der Meta-Optimierung im Verhältnis zur einfachen Optimierung vervielfacht, macht auch hier eine Parallelisierung Sinn. Deshalb bietet es sich an, ebenfalls die neue *Median-Selektion* einzusetzen.

Die Optimierungsumgebung EvA implementiert neben Evolutionsstrategien sowohl die Median-Selektion als auch die Meta-Optimierung. Darüberhinaus werden Aspekte des praktischen Einsatzes von Evolutionsstrategien unterstützt. Der Kern einer Optimierungsanwendung besteht in der Formulierung und Implementierung einer Fitneßfunktion, die einzelne (suboptimale) Lösungen der Anwendung mit einer einzigen Maßzahl bewertet. Mit EvA wird dem Anwender ein leistungsfähiges Werkzeug zur Hand gegeben, mit dem auf einfache Art und Weise eine solche Fitneßfunktion implementiert und in das System eingebunden werden kann. Dabei sind viele Rahmenbedingungen, welche die Problemstellung vorgibt, flexibel handhabbar. Dazu gehören z. B. Minimierung/Maximierung, Beschränkung der Variablen, problemspezifische Initialisierung der Menge der Anfangslösungen usw.

Der Optimierungsschritt stellt in realen Anwendungen oft nur einen Baustein in einem komplexeren Softwaresystem dar. Meist sind noch viele andere Schritte vor- oder nachgeschaltet. Daher muß sich die Optimierungssoftware möglichst einfach in das Gesamtsystem einbetten lassen und mit ihm interagieren können. Eine simple Kommunikation über Ein- und Ausgabedateien ist oft nicht ausreichend oder effizient genug.

1.3 Aufbau der Arbeit

In Kapitel 2 wird zuerst ein Überblick über das Gebiet der Evolutionären Algorithmen gegeben und als Schwerpunkt auf die in dieser Arbeit verwendeten Evolutionsstrategien eingegangen. Die Teilschritte und Kernaspekte einer Evolutionsstrategie werden im Detail dargelegt. Insbesondere wird ein Überblick über existierende Verfahren zur selbstadaptiven Mutation gegeben, da dieser Aspekt eine zentrale Rolle in der restlichen Arbeit spielt.

Kapitel 3 bringt zunächst eine Klassifikation von Parallelrechnern und stellt dann Modelle zur Parallelisierung von Evolutionären Algorithmen vor. Dies sind das Inselmodell, das Gittermodell und die parallele Fitneßevaluation. Letzteres Modell bildet die Grundlage für Kapitel 5.

In Kapitel 4 wird das Programmpaket EvA vorgestellt. Dessen Schwerpunkte liegen auf den folgenden Merkmalen, die für den praktischen Einsatz wichtig sind: einfaches Einbinden anwendungsspezifischer Fitneßfunktionen, Ein-/Ausgabe-Programmierbarkeit zur Integration als Optimierungsstufe in ein bestehendes System und die Meta-Optimierung.

Die Verfahren zur Steady-State-Parallelisierung, welche sich besonders einfach mit paralleler Fitneßevaluation parallelisieren lassen, werden in Kapitel 5 vorgestellt. Es wird aufgezeigt, daß das verbreitete Standard Steady-State-Verfahren bei Evolutionsstrategien den Selbstadaptionsmechanismus stören kann. Dadurch kann ein Teil des Geschwindigkeitsgewinns verloren gehen, den dieses Verfahren verspricht. Deshalb wird das neue Verfahren der Median-Selektion vorgestellt, das speziell entwickelt wurde, um die Vorteile des Steady-State-Algorithmus und der Selbstadaption nutzen zu können. Es werden ausführliche Testreihen präsentiert, welche die Leistungsfähigkeit des neuen Verfahrens auf Standard Benchmarkfunktionen dokumentieren.

Kapitel 6 beschäftigt sich mit dem Gebiet der Meta-Optimierung der Parameter einer Evolutionsstrategie. Hier wird ein Konzept zur Erweiterung einer Evolutionsstrategie eingeführt, um außer kontinuierlichen Parameterwerten auch ganzzahlige Parameter optimieren zu können. Zusätzlich treten noch Parameter mit einer ungeordneten, diskreten Wertemenge auf, die ebenfalls optimiert werden sollen. Für die Meta-Evolutionsstrategie wird desweiteren eine Fitneßfunktion entwickelt, welche zur Bewertung der Leistung von Evolutionsstrategien geeignet ist.

In Kapitel 7 werden die beiden neuen Verfahren auf einigen praktischen Anwendungen getestet. Eine davon ist die Positionierung von Haltemitteln bei der Holzverarbeitung. Die Positionen der Spannelemente zur Arretierung einer Holzplatte während der Bearbeitung sind durch kontinuierliche Koordinaten beschrieben. Diese werden mit der Evolutionsstrategie evolviert, so daß das Holzwerkstück möglichst gut festgehalten wird. Desweiteren wird die rechenzeitintensive Konformationsoptimierung von Molekülen durchgeführt. Hierbei wird versucht, eine dreidimensionale Konformation eines

Moleküls mit minimaler Energie zu finden. Dabei können die Winkel zwischen speziellen Molekülgruppen variieren. Diese werden durch kontinuierliche Zahlen beschrieben und sind somit für Evolutionsstrategien geeignet. Als dritte Anwendung wird die Form einer optischen Linse evolviert, so daß auftreffende Lichtstrahlen möglichst gut in einen Brennpunkt fokussiert werden.

Kapitel 2

Evolutionäre Algorithmen

Seit die Teilgebiete der EA zueinander gefunden haben, “befruchten” sie sich gegenseitig und es ist eine Konvergenz der EA-Zweige zu beobachten. Dem ist es auch zu verdanken, daß inzwischen größtenteils einheitliche Bezeichnungen und Begriffe verwendet werden, auch an einheitlichen Theorien wird gearbeitet. Deshalb ist es möglich, einen allgemeinen Evolutionären Algorithmus zu formulieren. Durch die Wahl von konkreten Operatoren und einer Repräsentation, entstehen daraus die spezielleren Varianten (z. B. GA oder ES).

Dieser allgemeine EA wird im nächsten Abschnitt erläutert. Danach folgt eine formale Darstellung der Bestandteile eines Evolutionären Algorithmus in Abschnitt 2.2. Anschließend wird auf speziellere Varianten, wie Genetische Algorithmen (Abschnitt 2.3), Evolutionsstrategien (2.4) und andere (2.5) eingegangen.

2.1 Überblick

Abbildung 2.1 zeigt den Ablauf eines Evolutionären Algorithmus in Pseudocode-Notation. Zunächst wird der Zeitzähler `t`, der in der Zeiteinheit *Generationen* zählt, auf Null gesetzt. Dann wird eine Menge zufälliger Anfangslösungen `P(0)` erzeugt und die Qualität jeder Lösung bewertet (evaluiert). Danach folgt die Hauptschleife, die solange wiederholt wird, bis die gewünschte Lösungsqualität erreicht wurde. Im ersten Schritt werden aus der Menge der momentanen Lösungen `P(t)` die Eltern bestimmt. Jeweils zwei oder mehrere Eltern werden zu einer neuen Lösung rekombiniert, welche anschließend noch durch Mutation leicht variiert wird. Die so erhaltenen neuen Lösungen werden wieder nach ihrer Qualität bewertet und schließlich die besten ausgewählt (Selektion), die als Ausgangsbasis der nächsten Iteration der Schleife dienen. Die neu erzeugten Lösungen unterscheiden sich von den bisher vorhandenen, weshalb man auch Unterschiede in der Qualität erwarten kann. Positive Veränderungen werden dann durch die Selektion bevorzugt, wodurch sich allmähliche Verbesserungen der durchschnittlichen Lösungsqualität ergeben.

```
t = 0
P(0) = initRandomPopulation()
eval(P(0))
while (not termination()) do
   P′(t) = parentSelect(P(t))
   P″(t) = recombine(P′(t))
   P‴(t) = mutate(P″(t))
   eval(P‴(t)
   P(t+1) = select(P(t), P‴(t))
   t = t+1
end
```

Abbildung 2.1: Ablauf eines Evolutionären Algorithmus.

Evolutionäre Algorithmen werden gerne angewendet, weil

- sie ein breites Anwendungsspektrum abdecken (verschiedene Varianten für sehr unterschiedliche Lösungsrepräsentationen),
- sie in der Grundform einfach einsetzbar und leicht verständlich sind,
- sie keine Zusatzforderungen, wie z. B. Differenzierbarkeit oder Stetigkeit, an die Funktion stellen, welche die Qualität der Lösungen berechnet,
- sie sehr robust sind, was Veränderungen in der Problemstellung und den Parametern der Algorithmen selbst betrifft,
- es einfach möglich ist, heuristisches Wissen über das Problem in Form von speziellen Operatoren einzubringen.

Allerdings stehen dem auch Nachteile gegenüber:

- es gibt keine Garantie, das Optimum oder überhaupt eine gute Lösung zu finden,
- der Rechenaufwand kann recht hoch werden,
- die Algorithmen besitzen selbst wieder viele Einstellungsmöglichkeiten, deren Wahl schwierig sein kann.

2.2 Grundbegriffe der Optimierung

Zunächst sollen einige grundlegende Begriffe bei der Optimierung formal definiert werden sowie weitere Bezeichnungen erläutert werden, die bei Evolutionären Algorithmen auftreten.

2.2.1 Optimierungsaufgabe

Bei einer Optimierungsaufgabe soll das globale Minimum oder Maximum einer Zielfunktion gefunden werden. Im folgenden wird ohne Beschränkung der Allgemeinheit von einer Minimierung ausgegangen. Zunächst wird der Begriff der Lösungsmenge definiert.

Lösungsmenge: $\mathbb{L}$ ist die Menge, die alle möglichen Lösungen für ein Problem in beliebiger Form kodiert enthält.

Bei Genetischen Algorithmen ist $\mathbb{L} = \mathbb{B}^n$ die Menge der binären Zeichenketten der Länge n. Bei Evolutionsstrategien ist $\mathbb{L} = \mathbb{R}^n$ der Raum der reellwertigen Vektoren der Länge n. Da Evolutionsstrategien das zentrale Thema dieser Arbeit sind, wird im folgenden von $\mathbb{L} = \mathbb{R}^n$ ausgegangen.

Menge der gültigen Lösungen: $\mathbb{M} \subseteq \mathbb{L}$ ist die Menge aller Lösungen, welche alle Randbedingungen erfüllen:

$$\mathbb{M} = \{\vec{x} \in \mathbb{L} | g_j(\vec{x}) \geq 0,\ \forall j \in \{1, \ldots, n_g\}\} \tag{2.1}$$

Die Randbedingungen g_j beschränken dabei die Lösungsmenge.

Randbedingung (Constraint): Eine Funktion g_j der folgenden Form heißt Randbedingung:

$$g_j\ :\ \mathbb{R}^n \to \mathbb{R} \qquad j = 1 \ldots n_g \tag{2.2}$$

$$\begin{array}{lll} \text{Eine Randbedingung } g_j \text{ ist } \mathbf{erfüllt}, & \text{falls} & g_j(\vec{x}) \geq 0 \\ \text{Eine Randbedingung } g_j \text{ ist } \mathbf{aktiv}, & \text{falls} & g_j(\vec{x}) = 0 \\ \text{Eine Randbedingung } g_j \text{ ist } \mathbf{inaktiv}, & \text{falls} & g_j(\vec{x}) > 0 \\ \text{Eine Randbedingung } g_j \text{ ist } \mathbf{verletzt}, & \text{falls} & g_j(\vec{x}) < 0 \end{array} \tag{2.3}$$

Bei der Bewertung von gültigen Lösungen legt eine Zielfunktion die Bewertungskriterien fest. Ergebnis ist eine reelle Zahl, die als Maß für die Tauglichkeit der Lösung verwendet wird.

Zielfunktion: Eine Zielfunktion ist eine Abbildung Φ von der Menge der gültigen Lösungen auf die Menge der reellen Zahlen:

$$\Phi : \mathbb{M} \to \mathbb{R} = f(\vec{x}), \mathbb{M} \neq \varnothing \tag{2.4}$$

Die Zielfunktion kann lokale und globale Minima enthalten.

Lokales Minimum: $\hat{f} = f(\hat{\vec{x}})$ heißt lokales Minimum, falls gilt:

$$\exists \varepsilon > 0 : \forall \vec{x} \in \mathbb{M} : \left\| \vec{x} - \hat{\vec{x}} \right\| < \varepsilon \Rightarrow \hat{f} < f(\vec{x}) \tag{2.5}$$

Wobei die Norm $\| \ \|$ meist der euklidische Abstand ist, mit dem die ε-Umgebung um das lokale Minimum definiert wird.

Das Minimum f^* ist global, wenn es nicht nur auf eine Umgebung ε, sondern in der gesamten Menge der gültigen Lösungen Minimum ist.

Globales Minimum: $f^* = f(\vec{x}^*)$ heißt globales Minimum, falls gilt:

$$\forall \vec{x} \in \mathbb{M} : f^* \leq f(\vec{x}) \tag{2.6}$$

Minimumpunkt: Der Minimumpunkt x^* ist der Punkt, für den die Zielfunktion $f(\vec{x}^*)$ das globale Minimum f^* ergibt.

$$\vec{x}^* \in \mathbb{M} \tag{2.7}$$

Nun kann der Begriff der Optimierungsaufgabe definiert werden.

Optimierungsaufgabe: Finde ein globales Minimum $f^* = f(\vec{x}^*)$ einer gegebenen Zielfunktion Φ, so daß alle Randbedingungen $G = \{g_1, \ldots, g_q\}$ erfüllt sind.

2.2.2 Begriffe bei Evolutionären Algorithmen

Über die im vorigen Abschnitt definierten Begriffe hinaus, findet man weitere Bestandteile bei Evolutionären Algorithmen, die hier definiert werden:

Individuum: $I = \{\vec{x}, \vec{s}\} \in A_x \times A_s$ ist ein *Individuum*, das aus den Objektvariablen $\vec{x} \in A_x = \mathbb{L}$ und den Strategievariablen $\vec{s} \in A_s$ besteht, die ein internes Modell enthalten, das zur Erzeugung neuer Lösungen verwendet wird. $\mathbb{I} = A_x \times A_s$ ist der Raum der Individuen.

Fitneßfunktion und Fitneß: Die Fitneßfunktion F ist eine Abbildung aus dem Raum der Individuen $\mathbb{I}$ in die Menge der reellen Zahlen $\mathbb{R}$. Sie bewertet ein Individuum I mit einer Maßzahl aus $\mathbb{R}$, welche *Fitneß* genannt wird.

$$F : \mathbb{I} \rightarrow \mathbb{R} = F(\vec{x})$$

Im Unterschied zur Zielfunktion enthält die Fitneßfunktion noch vorgeschaltete Schritte wie z. B. Dekodierung der Objektvariablen von $\mathbb{L}$ nach $\mathbb{R}^n$ und nachgeschaltete Schritte wie z. B. eine Skalierung der Zielfunktionswerte zum Fitneßwert. Da bei Evolutionsstrategien $\mathbb{L} = \mathbb{R}^n$ ist, entfällt somit eine Dekodierung. Deshalb wird hier in der Regel die Zielfunktion direkt als Fitneßfunktion verwendet.

Elter: Ein Individuum E_j ($1 \leq j \leq \mu$), das als Ausgangsbasis zur Berechnung neuer Individuen dient, wird als *Elter* bezeichnet. Im folgenden werden zum Elternindividuum gehörige Komponenten mit dem Index-Zusatz Ej benannt, z. B. Objektvariable $x_{i,Ej}$.

Nachkomme: Ein Individuum N_j ($1 \leq j \leq \lambda$), das durch Anwendung von Operatoren aus einem oder mehreren Elternindividuen erzeugt wurde, wird als *Nachkomme* bezeichnet. Analog gilt auch hier die Bezeichnung von zugehörigen Komponenten mit z. B. $x_{i,Nj}$.

Population: Eine Menge von Individuen $P = \{I_1, \ldots, I_\mu\}$ ist eine *Population*. Dabei wird mit μ die Größe der Elternpopulation und mit λ die Größe der Nachkommenpopulation bezeichnet.

Generation: Ein Iterationsschritt eines Evolutionären Algorithmus, bei dem eine komplette neue Population berechnet wird, ist eine *Generation*. Zur Zählung von Generationen wird die diskrete Zeitvariable t verwendet.

Die beiden Operatoren **Rekombination** und **Mutation** dienen zur Erzeugung und Veränderung von Individuen. Die Rekombination kombiniert mehrere Individuen zu einem einzigen, neuen Individuum. Dadurch werden die Informationen aus den ursprünglichen Individuen gemischt. Die Mutation verändert ein Individuum geringfügig, so daß sich das neue Individuum vom ursprünglichen nur wenig unterscheidet.

2.3 Genetische Algorithmen

Genetische Algorithmen gehen auf Arbeiten von John Holland in den 60er Jahren zurück, in denen er sich mit adaptiven Systemen beschäftigte [Holland 62, Holland 75]. Später wurden sie dann hauptsächlich durch Arbeiten seines Schülers David Goldberg [Goldberg 89] bekannt.

```
t = 0
P(0) = initRandomPopulation()
eval(P(0))
while (not termination()) do
   P′(t) = parentSelect(P(t))
   P″(t) = crossover(P′(t))
   P‴(t) = mutate(P″(t))
   eval(P‴(t))
   P(t+1) = P‴(t)
   t = t+1
end
```

Abbildung 2.2: Ein Genetischer Algorithmus.

Die Grundidee bei Genetischen Algorithmen liegt in der direkten Imitation des biologischen Vorbildes der DNS und somit der Nachbildung auf der Ebene des Genotyps. Ein Individuum besteht im ursprünglichen GA aus einer Folge von Bits, welche die Werte 0 und 1 annehmen können (die binäre Entsprechung der 4 Basen der DNS). Die ganze Folge von Bits wird als Bitstring bezeichnet und entspricht einem Chromosom. Teilfolgen des Bitstrings entsprechen einzelnen Genen und können z. B. eine Integerzahl binär codieren. Die Codierung ist im Grunde *der* Hauptaspekt bei GAs, weil davon u. a. die Beschaffenheit des Suchraumes und somit die Fähigkeit zum Auffinden guter Lösungen entscheidend beeinflußt wird.

Abbildung 2.2 zeigt den Pseudocode eines GA. Zu Beginn wird eine Anfangspopulation mit zufällig belegten Bitstrings erzeugt und jedes Individuum daraus mit der Zielfunktion bewertet. Dann beginnt die Hauptschleife, die solange ausgeführt wird, bis ein bestimmtes Abbruchkriterium greift, z. B. das Erreichen einer vorgegebenen Zahl von Generationen oder eines bestimmten Fitneßwertes. In der Hauptschleife finden die Schritte Eltern-Selektion, Crossover[1], Mutation und Evaluation statt. Aus der Elternpopulation werden Individuen proportional zu ihrer Fitneß ausgewählt (selektiert), deren Genotyp dann als Ausgangsbasis zur Erzeugung von Nachkommen dient (z. B. mittels *Roulette-Wheel-Selektion*, siehe Abschnitt 2.4.4). Individuen mit besserer Fitneß besitzen dabei eine höhere Wahrscheinlichkeit, ausgewählt zu werden, als Individuen mit schlechterer Fitneß. Für jedes Paar von (normalerweise) zwei Individuen wird dann mit der Crossoverwahrscheinlichkeit p_c entschieden, ob sie direkt in die Nachkommenpopulation übernommen werden, oder ob Crossover stattfindet. Das

[1] In Genetischen Algorithmen wird der Begriff *Crossover* für die Rekombination benutzt, der vom biologischen Vorbild des *crossing over* von DNS-Strängen stammt.

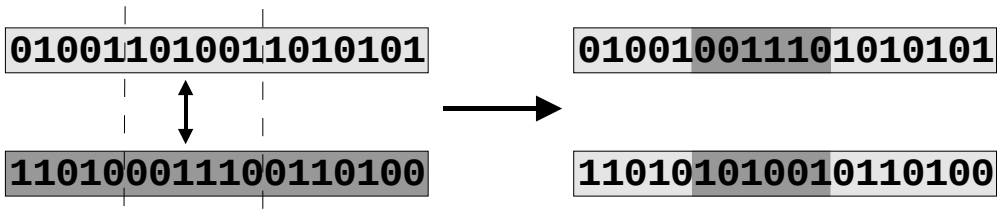


Abbildung 2.3: 2-Punkt Crossover.

Crossover mischt dabei die Bitstrings der Eltern, indem sie an zufällig ausgewählten Stellen aufgebrochen und die Teilstücke ausgetauscht werden. Dabei können verschiedene Arten des Crossover zum Einsatz kommen, wie z. B. 1-Punkt-, 2-Punkt- (siehe Abb. 2.3) oder n-Punkt-Crossover, was die Anzahl der Bruchstellen im Bitstring angibt. Beim *Uniform Crossover* dagegen wird für jedes Bit separat entschieden, von welchem Elternteil es übernommen wird. Nach dem Crossover findet noch die Mutation statt. Hierbei gibt die Mutationsrate p_m die Wahrscheinlichkeit an, mit der ein Bit mutiert wird. Die Mutation selbst wird durch Umkippen des Bitwertes von 0 nach 1 und umgekehrt realisiert. Die neuen Individuen werden dann wiederum mit der Zielfunktion bewertet. Nach Überprüfung der Abbruchbedingung wird dann entweder weiter iteriert oder der Algorithmus beendet.

Zu betonen ist hierbei, daß das Crossover auf den unstrukturierten Bitstrings arbeitet, wodurch es völlig unabhängig von der gewählten Codierung bleibt. Dies ist auch der Hauptoperator, welcher mit relativ hoher Wahrscheinlichkeit stattfindet. Die Crossoverrate p_c liegt üblicherweise im Bereich zwischen 0.5 und 0.95. Beim Crossover sollen gute Teillösungen kombiniert werden, wobei gehofft wird, daß die kombinierte Lösung eine Verbesserung darstellt[2]. Die Mutation wird dagegen als Hintergrundoperator betrachtet. Sie dient nur dazu, zu verhindern, daß sämtliche Individuen der Population an einer Stelle denselben Bitwert haben, was durch die zufällige Initialisierung durchaus vorkommen kann. Dadurch würde ein Teil des Lösungsraumes nicht durchsucht werden, da Crossover keine neuen Informationen einbringt, sondern nur vorhandene Teile neu kombiniert. Die Mutationsrate wird deshalb sehr klein gewählt, z. B. in Abhängigkeit von der Länge l des Bitstrings $p_c = \frac{1}{l}$, so daß im Durchschnitt ein Bit pro String mutiert wird.

Abweichend von der Urform des GA werden heutzutage statt der Binärcodierung auch gerne abstraktere, problemspezifischere Codierungen verwendet, z. B. Bäume oder beliebige andere Datenstrukturen. Dann müssen allerdings die Crossover- und Mutationsoperatoren so angepaßt werden, daß sie auf diesen Datenstrukturen sinnvoll arbeiten. Gerade hierdurch kann aber wichtiges problemspezifisches Wissen vorteilhaft eingebracht werden. Solcherlei modifizierte GAs werden oft für kombinatorische Pro-

[2]Beispielsweise könnten beim *Traveling Salesman Problem* zwei gute Teilstrecken, die in unterschiedlichen Individuen vorhanden sind, kombiniert werden.

bleme, wie das *Traveling Salesman Problem*, Schedulingaufgaben o. ä. erfolgreich eingesetzt.

2.4 Evolutionsstrategien

Evolutionsstrategien (ES) entstanden Mitte der 60er Jahre an der Technischen Universität Berlin. Ingo Rechenberg und Hans-Paul Schwefel setzten sie zur Optimierung ingenieurstechnischer Problemstellungen ein [Schwefel 65, Rechenberg 73]. Im Gegensatz zu GAs versuchen sie nicht, genotypische Mechanismen der biologischen Evolution zu imitieren, sondern die Evolution eher auf phänotypischer Ebene nachzubilden. Ein Individuum besteht deshalb aus einem Vektor von reellen Zahlen (Objektvariablen), auf denen auch nur Operationen für reelle Zahlen ausgeführt werden. Deren Codierung wird an keiner Stelle angerührt. Die Beobachtung, daß sich in der Natur die Nachkommen nur jeweils sehr gering von den Eltern unterscheiden, wird in eine Mutation durch Addition normalverteilter Zufallszahlen umgesetzt. Bei der Normalverteilung besteht eine Konzentration um den Mittelwert, das Auftreten von Abweichungen wird mit zunehmendem Abstand vom Mittelwert immer unwahrscheinlicher.

Bei Evolutionsstrategien gibt es zwei Hauptformen, die nach [Rechenberg 94] folgendermaßen notiert werden:

$$(\mu/\rho + \lambda)^\gamma \qquad \text{oder} \qquad (\mu/\rho, \lambda)^\gamma$$

Der Parameter μ bezeichnet die Anzahl der Individuen in der Elternpopulation, λ ist die Größe der Nachkommenpopulation. Mit ρ wird die Anzahl der Individuen angegeben, die an einer Rekombination beteiligt sind, dies können genau wie beim GA auch mehr als 2 sein (maximal jedoch μ). Mit "$+$" (Plus) oder "," (Komma) wird die Art der Selektion bezeichnet: Bei Komma werden nur aus der Nachkommengeneration die μ besten Individuen selektiert (hierbei muß $\mu \leq \lambda$ gelten), bei Plus erfolgt die Auswahl aus der Nachkommen- *plus* der Elternpopulation (veranschaulicht in Abbildung 2.7). Hierbei arbeitet die Selektion in der Art, daß eine große Menge Individuen auf eine kleinere Population reduziert wird. Hier bestehen zwei weitere Unterschiede zum GA bei der Selektion und der Handhabung von Populationen: beim GA sind Eltern- und Nachkommenpopulation meist gleich groß, die Selektion dient dazu, die Elternindividuen mit größerer Fitneß für die Erzeugung von Nachkommen auszuwählen. Bei der ES sind Eltern- und Nachkommenpopulation im Allgemeinen nicht gleich groß, alle Eltern besitzen die gleiche Wahrscheinlichkeit, Nachkommen zu erzeugen. Die Selektion findet erst nach Rekombination und Mutation statt, weil sie dazu dient, aus der Nachkommenpopulation die Eltern der nächsten Generation auszuwählen.

Der Hauptoperator bei der Evolutionsstrategie ist die Mutation, sie erzeugt neue Lösungen im Suchraum. Auf Rekombination kann ganz verzichtet werden ($\rho = 1$, diese Angabe kann dann auch weggelassen werden), doch sie bringt eigentlich immer eine zusätzliche Robustheit in die Optimierung.

```
t = 0
P_μ(0) = initRandomPopulation()
eval(P_μ(0))
while (not termination()) do
   P_λ(t) = recombine(P_μ(t))
   P'_λ(t) = mutate(P_λ(t))
   eval(P'_λ(t))
   P_μ(t+1) = select(P_μ(t), P'_λ(t))
   t = t+1
end
```

Abbildung 2.4: Algorithmus der Evolutionsstrategie.

In Abbildung 2.4 ist der Algorithmus einer Evolutionsstrategie als Pseudocode dargestellt. Er ähnelt dem GA bis auf die unterschiedliche Verwendung von Selektion. Beim GA findet eine fitneßproportionale Auswahl der Rekombinationspartner statt (Elternselektion), welche bei Evolutionsstrategien mit gleichverteilter Wahrscheinlichkeit realisiert wird. Dafür befindet sich die Selektion am Schleifenende. Hier wird ausgewählt, welche Individuen bis zur nächsten Generation überleben.

Auf die einzelnen Bestandteile und Mechanismen des Algorithmus wird in den nachfolgenden Abschnitten näher eingegangen, deshalb hier zuerst nur eine kurze Zusammenfassung: Die Elternpopulation der Größe μ wird am Anfang mit zufälligen Werten initialisiert und gleich mit der Zielfunktion bewertet. Die Hauptschleife wird durch ein Abbruchkriterium gebildet, wie z. B. das Erreichen einer vorgegebenen Anzahl von Generationen. In der Hauptschleife werden durch Rekombination von ρ Elterninidividuen und anschließender Mutation λ Nachkommen erzeugt. Von diesen (plus evtl. der Elternpopulation) werden durch die Selektion die μ besten ausgewählt, welche dann die Eltern der nächsten Generation bilden. Die Hauptschleife ist dann zu Ende, und der Algorithmus wird weiter iteriert, bis das Abbruchkriterium zutrifft.

In den folgenden Abschnitten wird nun auf die einzelnen Teilschritte und Aspekte von Evolutionsstrategien näher eingegangen. Dies ist einerseits von Interesse, da die meisten der angesprochenen Verfahren in EvA implementiert sind. Andererseits werden diese Grundlagen für das Verständnis der späteren Kapitel benötigt. Insbesondere die Mutation und Selbstadaption wird im Detail behandelt, da die im Kapitel 5 vorgestellte Selektionsmethode diese Aspekte unterstützt.

2.4.1 Initialisierung der Individuen

Zu Beginn einer Evolutionsstrategie, noch vor der ersten Generation, müssen die Objekt- und Strategievariablen mit Anfangswerten belegt werden. Dieser Prozeß wird Initialisierung genannt.

1. Objektvariablen
   Die Standardmethode, mit der die Individuen zu Beginn einer ES initialisiert werden, ist, alle Variablen mit gleichverteilten Zufallsvariablen zu belegen. Meist ist durch die Randbedingungen g_j u. a. ein Definitionsbereich für die Variablen mit Unter- und Obergrenze u_j und o_j vorgegeben, den man als Intervall zur Initialisierung wählt:

$$g_j = \begin{cases} -1 & \text{falls } x_i < u_j \\ -1 & \text{falls } x_i > o_j \\ \dots & \dots \\ 1 & \text{sonst} \end{cases} \qquad j = 1 \dots n_g \tag{2.8}$$

$$x_i = U(u_j, o_j) \text{ gleichverteilt} \qquad i = 1 \dots n$$

   Weil dadurch keine Region des Lösungsraumes vernachlässigt wird, soll die Chance vergrößert werden, ein Individuum in der Nähe des globalen Optimums zu plazieren.

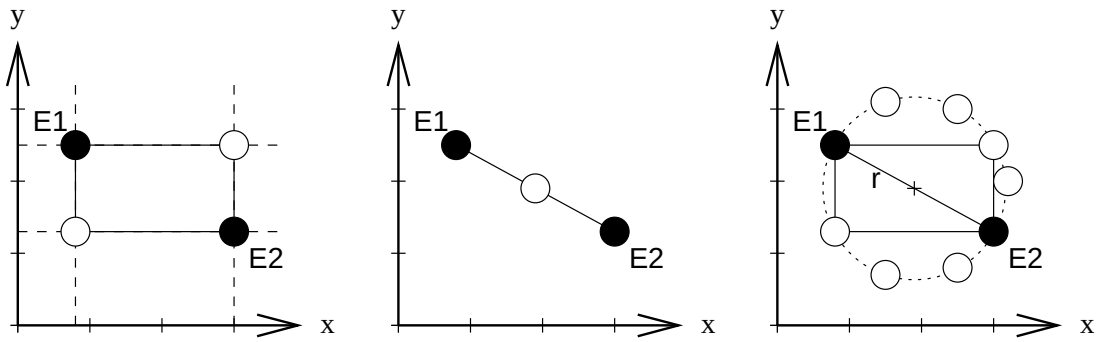
2. Strategievariablen
   Da die Strategievariable(n) je nach verwendetem Adaptionsverfahren eine unterschiedliche Bedeutung haben (Standardabweichung, Winkel, Kovarianzen) belegt man sie mit Werten, die für das jeweilige Verfahren sinnvoll sind. Handelt es sich bei den Strategievariablen um Standardabweichungen, sind sie derart mit Werten zu belegen, so daß der Suchraum zu Beginn möglichst global abgedeckt wird. Weiteres hierzu folgt im übernächsten Abschnitt bei den Verfahren zur Mutation und Selbstadaption (2.4.3).

2.4.2 Rekombination

Der Rekombinationsoperator vermischt die Informationen von mindestens zwei Individuen miteinander, um ein neues Individuum zu bilden. Bei Evolutionsstrategien ist die Rekombination optional, es kann auch nur mit Mutation gearbeitet werden. Im allgemeinen gewinnt man aber durch Einsatz von Rekombination an Robustheit, da damit stochastische Schwankungen gedämpft werden.

Rekombination: Ein Rekombinationsoperator r erzeugt aus $\rho > 1$ Individuen genau ein neues Individuum:

$$r : I^\rho \to I$$



(a) diskret (b) intermediär (c) kontinuisiert

Abbildung 2.5: Drei Rekombinationsoperatoren bei Evolutionsstrategien für den Fall Dimension $n = 2$ und Anzahl der Rekombinationspartner $\rho = 2$.

Die beiden Hauptformen der Rekombination bei Evolutionsstrategien sind die *diskrete* und die *intermediäre* Rekombination:

$$\begin{aligned} \text{diskret}: \quad x_i' &= x_{i,Ez_i} \qquad \forall i \in \{1,\ldots,n\} \\ \text{mit} \quad z_i &= U(1,\rho) \text{ gleichverteilte, ganze Zufallszahl} \end{aligned} \tag{2.9}$$

$$\text{intermediär}: \quad x_i' = \frac{1}{\rho}\sum_{j=1}^{\rho} x_{i,Ej} \qquad \forall i \in \{1,\ldots,n\} \tag{2.10}$$

Bei der diskreten Rekombination wird jede Objektvariable x_{i,Ez_i} zufällig von einem der ρ Elternindividuen Ez_i ausgewählt. Abbildung 2.5(a) zeigt die diskrete Rekombination zweier Individuen für den zweidimensionalen Fall. Die resultierenden Punkte sind dabei die Eckpunkte eines Rechtecks. Bei der intermediären Rekombination werden dagegen die Objektvariablen x_i aller ρ Elternindividuen gemittelt. Das Ergebnis ist hierbei immer der Elternschwerpunkt (siehe Abbildung 2.5(b)). Für den Spezialfall $\rho = 2$ existiert noch die verallgemeinerte intermediäre Rekombination, bei der nicht der genaue Mittelpunkt zwischen den beiden Individuen entsteht, sondern mit einem Faktor $c \in [0,1]$ gewichtet wird ($c = 0.5$ entspricht dann der intermediären Rekombination):

$$\text{verallgemeinert intermediär}: \quad x_i' = x_{i,E1} + c \cdot (x_{i,E2} - x_{i,E1}) \tag{2.11}$$

[Bäck 96] beschreibt noch weitere Rekombinationsoperatoren, die er *panmiktisch* nennt (panmiktisch diskret, panmiktisch intermediär und panmiktisch verallgemeinert intermediär). Dabei bleibt jeweils ein Elternindividuum fixiert, der Rekombinationspartner wird aber für jede Objektvariable neu aus den ρ Elternindividuen zufällig ausgewählt.

[Rechenberg 94] verallgemeinert die diskrete Rekombination am anschaulich geometrischen Beispiel zur *kontinuisierten* Rekombination für den Fall $\rho = 2$:

$$x_{1,S} = \frac{x_{1,E1} + x_{1,E2}}{2}; \quad x_{2,S} = \frac{x_{2,E1} + x_{2,E2}}{2} \tag{2.12}$$

$$r = \frac{1}{2}\sqrt{(x_{1,E1} - x_{1,E2})^2 + (x_{2,E1} - x_{2,E2})^2} \tag{2.13}$$

$$x'_1 = x_{1,S} + r\cos(2\pi z); \quad x'_2 = x_{2,S} + r\sin(2\pi z) \tag{2.14}$$

$$\text{mit } z = U[0,1) \text{ gleichverteilte Zufallszahl} \tag{2.15}$$

Der Nachkomme wird zufällig auf einem Kreis plaziert, der einen Radius r vom halben Elternabstand besitzt und dessen Mittelpunkt im Elternschwerpunkt liegt (siehe Abbildung 2.5(c)). Dies läßt sich in dieser Form nicht auf ρ Eltern bei beliebiger Dimension n verallgemeinern. Eine Möglichkeit wäre es, das rekombinierte Individuum auf einer Hyperkugel um den Elternschwerpunkt mit einem Radius vom mittleren Elternabstand von diesem Schwerpunkt zu plazieren. Diese Form der Rekombination hat sich jedoch nie durchgesetzt.

Bei Evolutionsstrategien kann auf den Einsatz von Rekombination völlig verzichtet werden ($\rho = 1$), sie hat jedoch fast immer einen positiven Effekt auf die Geschwindigkeit und Konvergenz-Zuverlässigkeit. Der größte Unterschied ergibt sich dabei, ob man Rekombination überhaupt einsetzt ($\rho = 2$) oder nicht ($\rho = 1$). Eine höhere Zahl von Rekombinationspartnern ($\rho > 2$) bringt jeweils nur noch kleinere Verbesserungen.

Auf Objekt- und die evtl. verschiedenen Typen von Strategievariablen (Standardabweichungen, Rotationswinkel) können sogar jeweils unterschiedliche Rekombinationsoperatoren angewandt werden. [Schwefel 95] empfiehlt diskrete Rekombination auf Objektvariablen, da sie die Diversität erhält und intermediäre Rekombination auf Strategievariablen, da die Mittelung derselben ein besseres Maß für die aktuell benötigte Mutationsschrittweite ergibt, als die Werte eines einzelnen Individuums. Bei der Kovarianzmatrix-Adaption (wird im nächsten Abschnitt über Mutation behandelt) existiert ein eigenes Schema zur Rekombination, da sonst die Kovarianzmatrix gewisse Eigenschaften verlieren könnte.

2.4.3 Mutation und Selbstadaption

Der wichtigste Operator bei ES ist die Mutation. Seine Leistungsfähigkeit wird ihm durch verschiedene zur Auswahl stehende Mechanismen zur Selbstadaption verliehen. Hierbei werden neben den Objektvariablen gleichzeitig noch die Strategievariablen,

oft auch als *Mutationsschrittweiten* bezeichnet, evolviert. So reguliert sich die Mutationsschrittweite selbst durch einen evolutionären Prozeß (*Schrittweitenadaption*). Dies ist ein entscheidendes Alleinstellungsmerkmal der Evolutionsstrategien. Bei den später noch genannten Verfahren wie z. B. Simulated Annealing ist dagegen das Analogon zur Mutationsschrittweite ein *exogen*, also von außen, gesteuerter Parameter.

Mutationsoperator: ein Mutationsoperator m nimmt ein Individuum I als Eingabe und verändert es:

$$m : I \to I \tag{2.16}$$

Die Mutation selbst läuft im allgemeinen dergestalt ab, daß ein Mutationsvektor $\vec{\Delta}$ auf den Vektor der Objektvariablen $\vec{x}$ aufaddiert wird, um die mutierten Objektvariablen $\vec{x}'$ zu erhalten[3]:

$$\vec{x}' = \vec{x} + \vec{\Delta} \tag{2.17}$$

oder in Komponentenschreibweise:

$$x_i' = x_i + \Delta_i \qquad i = 1 \ldots n \tag{2.18}$$

Die Erzeugung des Mutationsvektors $\vec{\Delta}$ ist bei jedem der Verfahren verschieden, sie basieren allerdings auf einer gemeinsamen Grundform, die dann jeweils noch verfeinert wird. Der Mutationsvektor besteht grundsätzlich aus einem Vektor $N(0,1)$-normalverteilter Zufallzahlen $\vec{z}$, der mit einer (in der Grundform) skalaren Mutationsschrittweite δ multipliziert wird, welche die Standardabweichung darstellt:

$$\vec{\Delta} = \delta \cdot \vec{z} \qquad \text{mit } z_i = N(0,1),\ \forall i = 1 \ldots n \tag{2.19}$$

Die Komponenten Δ_i sind somit $N(0,\delta)$-verteilt. Für große Dimensionen n beträgt der Erwartungswert $E(|\vec{\Delta}|) \approx \delta\sqrt{n}$, so daß die Nachkommen mit gleicher Fitneß auf einer Hyperkugel mit diesem Radius zu liegen kommen (siehe Abbildung 2.6).

Bevor nun auf die Adaptionsverfahren im einzelnen eingegangen wird, sollen vorab noch die Kriterien, in denen sie sich unterscheiden, vorgestellt werden:

- Die Ebene, auf der die Schrittweitenregelung stattfindet:
  - Populationsebene: eine Schrittweite für die Population.

[3] Die hier und im folgenden angegebenen Variablen sind jeweils bezogen auf ein bestimmtes Individuum zu betrachten. Auf die Benutzung von weiteren Indizes, um das Individuum in der Population zu adressieren, wird hier aus Gründen der Übersichtlichkeit verzichtet.

  - Individuenebene: eine Einzelschrittweite (auch globale Schrittweite genannt) für jedes Individuum.
  - Objektvariablenebene: separate Schrittweiten für einzelne Objektvariablen. Hierbei gibt es zwei Unterfälle:
    * unkorreliert
    * korreliert

- Die Art der Schrittweitenanpassung:
  - mutativ
  - gedämpft[4]/gedämpft + akkumulierend

Auf die genaue Bedeutung der Begriffe wird bei dem jeweiligen Verfahren eingegangen. Eine Übersicht mit Klassifikation der Verfahren nach diesen Kriterien befindet sich im Anschluß in Tabelle 2.2 und eine Klassifikation nach Speicherplatz- und Rechenzeitbedarf in Tabelle 2.3.

2.4.3.1 1/5 Erfolgsregel

Ein Individuum $I = \{\vec{x}\}$ besteht bei diesem Verfahren nur aus dem Vektor der Objektvariablen. Außerdem existiert genau eine Schrittweite δ_P für die gesamte Population. Der Mutationsvektor wird nach der Grundformel 2.19 für jedes Individuum neu berechnet als $\vec{\Delta} = \delta_P \cdot \vec{z}$.

Diese erste, einfache Methode zur Adaption der Schrittweite δ_P stammt von Rechenberg [Rechenberg 73]. Sie basiert auf den Begriffen der Fortschrittsgeschwindigkeit und Erfolgswahrscheinlichkeit:

Fortschrittsgeschwindigkeit:

$$\varphi = \frac{\text{Zielannäherung}}{\text{Zahl der benötigten Mutationsschritte}} \tag{2.20}$$

Die Zielannäherung kann dabei als Weg im Parameterraum, als Gradientenweg oder als Fitneßdistanz gemessen werden.

Erfolgswahrscheinlichkeit/Erfolgsrate:

$$W_e = \frac{\text{Zahl der erfolgreichen Mutationsschritte}}{\text{Gesamtzahl der Mutationsschritte}} \tag{2.21}$$

[4]Im Original auch als *derandomized [Ostermeier et al. 93]* oder *entstochastisiert* [Hansen et al. 95b] bezeichnet, weil die Strategieparameter keinem stochastischen Mutations-/Selektionsprozeß mehr unterliegen.

Aufgrund von Beobachtungen an Modellen der $(1+1)$-ES kam Rechenberg zu dem Schluß, daß es einen optimalen Kompromiß zwischen Erfolgswahrscheinlichkeit und Fortschrittsgeschwindigkeit gibt. Kleine Schrittweiten erzeugen zwar mit hoher Wahrscheinlichkeit gute Individuen (besser als die Eltern), bringen aber kaum Fortschritt. Große Schrittweiten dagegen bringen zwar großen Fortschritt, produzieren aber hauptsächlich schlechtere Individuen. Für die lokalen Modelle von Fitneßfunktionen, das Korridormodell und das Kugelmodell, konnte er φ und W_e mit Näherungen in Abhängigkeit von der Schrittweite δ_P berechnen. Diese Gleichungen löste er nach δ_P auf und bestimmte durch Nullsetzen der ersten Ableitung das Optimum. Aus den beiden erhaltenen Werten bildete er den groben Richtwert von $W_e \approx 1/5$.

1/5 Erfolgsregel: Protokolliere während der Optimierung den Wert W_e für die Erfolgsrate einer Mutation mit Streuung δ_P und agiere wie folgt: Sinkt die Erfolgsrate unter 1/5, so verkleinere δ_P; steigt die Erfolgsrate über 1/5, so vergrößere δ_P.

Die Vergrößerung oder Verkleinerung von δ_P wird als Multiplikation mit einem Faktor ξ realisiert, welcher bei Vergrößerung der Schrittweite den Wert α und bei Verkleinerung den Wert $\frac{1}{\alpha}$ annimmt (von [Rechenberg 94] wird als Richtwert $\alpha = 1.3$ vorgeschlagen, [Schwefel 95] benutzt einen Wert von $\frac{1}{\alpha} = 0.85 \Leftrightarrow \alpha \approx 1.18$).

$$\xi = \begin{cases} \alpha & \text{bei Vergrößerung} \\ \frac{1}{\alpha} & \text{bei Verkleinerung} \end{cases} \tag{2.22}$$

Dadurch kann sich die Schrittweite innerhalb weniger Iterationen t exponentiell (α^t), also um ganze Größenordnungen verändern, falls die Problemstellung dies erfordert. [Schwefel 95] empfiehlt, die Erfolgsrate alle q Mutationen zu überprüfen und dann die letzten $10q$ Mutationen zu betrachten.

Es lassen sich allerdings Fälle konstruieren, in denen die Erfolgsrate den Wert 1/5 nicht überschreiten kann und die Adaption fehlschlägt. Das Verfahren ist inzwischen von moderneren und leistungsfähigeren abgelöst worden. Es stellt allerdings den einzigen Vertreter der auf Populationsebene arbeitenden Verfahren dar (siehe Tabelle 2.2).

2.4.3.2 Mutative Schrittweitenregelung (MSR) mit globaler Schrittweite

Anstatt wie bei der 1/5-Erfolgsregel die Erfolgswahrscheinlichkeit fest in das Verfahren zu kodieren, wird hier die Selektion dazu benutzt, zu entscheiden, welche Schrittweiten sich als erfolgreich bewährt haben: Es werden Nachkommen mit größerer und kleinerer Schrittweite erzeugt, denen die veränderte Schrittweite als Strategieparameter vererbt wird. Die Schrittweite, die der lokalen Form des Fitneßgebirges besser

angepaßt ist und somit bessere Individuen erzeugt, wird nach der anschließenden Selektion häufiger in Individuen anzutreffen sein. Die neue Schrittweite wird dann als Ausgangsbasis für weitere Mutationen verwendet.

Ein Individuum setzt sich in der einfachsten Form aus dem Objektvariablenvektor $\vec{x}$ und einer Mutationsschrittweite δ zusammen: $I = \{\vec{x}, \delta\}$. Die Schrittweite δ heißt *global,* da sie für alle Objektvariablen des Individuums gemeinsam verwendet wird. Das Verfahren modifiziert dann zuerst die Mutationsschrittweite und führt mit dem neuen Wert δ' die eigentliche Mutation der Objektvariablen x_i durch:

$$\begin{aligned} \delta' &= \xi \cdot \delta \\ \vec{\Delta} &= \delta' \cdot \vec{z} \\ \vec{x}' &= \vec{x} + \vec{\Delta} \end{aligned} \tag{2.23}$$

Hierbei sind die Komponenten z_i jeweils unabhängige Instanzen $N(0,1)$-normalverteilter Zufallsvariablen.

Die Vergrößerung oder Verkleinerung von δ wird als Multiplikation mit einem Faktor ξ realisiert, welcher mit gleicher Wahrscheinlichkeit den Wert α oder $\frac{1}{\alpha}$ annimmt (α wie in 2.4.3.1). [Rechenberg 94] schlägt auch eine Variante mit der dritten Möglichkeit $\xi = 1$ vor.

$$\xi = \begin{cases} \alpha & \text{mit W'keit } p_\alpha = 1/2 \\ \frac{1}{\alpha} & \text{mit W'keit } p_{\frac{1}{\alpha}} = 1/2 \end{cases} \quad \text{oder} \quad \xi = \begin{cases} \alpha & \text{mit W'keit } p_\alpha = 1/3 \\ 1 & \text{mit W'keit } p_1 = 1/3 \\ \frac{1}{\alpha} & \text{mit W'keit } p_{\frac{1}{\alpha}} = 1/3 \end{cases} \tag{2.24}$$

Allerdings existiert eine modernere Variante, die statt der diskreten eine kontinuierliche, logarithmische Verteilung benutzt (Gleichung 2.25). ξ wird hierbei gebildet, indem als Exponent der Eulerschen Konstante e eine mit $\tau_1 \approx 1/\sqrt{n}$ skalierte $N(0,1)$-normalverteilte Zufallszahl z verwendet wird.

$$\xi = e^{\tau_1 \cdot z} \tag{2.25}$$

Die so modifizierte Schrittweite δ' wird dem neuen Individuum vererbt. Die Idee dahinter ist, daß diese neue Schrittweite fortan verwendet wird, falls der dadurch entstandene Nachkomme eine höhere Fitneß als andere Individuen erreicht hat, und sich somit bei der Selektion durchsetzt.

2.4.3.3 Mutative Schrittweitenregelung (MSR) mit separaten Schrittweiten

Statt eine globale Schrittweite δ pro Individuum zu benutzen, versprechen separate Schrittweiten für jede Objektvariable, also ein Vektor $\vec{\delta}$, mehr Flexibilität bei der Adaption an komplexere Fitneßlandschaften. Die Mutationsverteilung stellt hier keine

Hyperkugel mehr dar, sondern eine Hyperellipse (siehe Abbildung 2.6), da in jeder der n Dimensionen der Radius unterschiedlich sein kann. Dieses Verfahren wird z. B. in [Saravanan et al. 96] erläutert.

$$\begin{array}{rcl} & \xi & = & e^{\tau_1 \cdot z} \\ \vec{\xi} = (\xi_1, \ldots, \xi_n) \quad \text{mit} & \xi_i & = & e^{\tau_2 \cdot z_{\xi,i}} \qquad 1 \leq i \leq n \end{array} \tag{2.26}$$

$$\begin{array}{rcl} \vec{\delta}' & = & \xi \cdot \vec{\xi} \cdot \vec{\delta} \\ \vec{\Delta} & = & \vec{\delta}' \cdot \vec{z} \\ \vec{x}' & = & \vec{x} + \vec{\Delta} \end{array} \tag{2.27}$$

Auch hier sind z, $z_{\xi,i}$ und die Komponenten z_i von $\vec{z}$ unabhängige Instanzen $N(0,1)$-normalverteilter Zufallsvariablen. Die logarithmisch normalverteilten Modifikationsfaktoren ξ (global) und ξ_i (Komponenten von $\vec{\xi}$) werden dabei mit den unterschiedlichen Skalierungen τ_1 (Gleichung 2.28) auf globaler Ebene und τ_2 (Gleichung 2.29) auf Objektvariablenebene angepaßt.

$$\tau_1 \approx 1/\sqrt{2n} \tag{2.28}$$

$$\tau_2 \approx 1/\sqrt{2\sqrt{n}} \tag{2.29}$$

Damit dieses Verfahren funktioniert, muß eine genügend große Population verwendet werden [Schwefel 87]. [Ostermeier et al. 94] schätzt ab, daß Populationsgrößen von mindestens $10n$ Individuen benötigt werden, damit keine Degeneration der Schrittweiten $\vec{\delta}$ stattfindet.

2.4.3.4 Gedämpfte Schrittweitenregelung mit separaten Schrittweiten (GSR)

In [Ostermeier et al. 93] wird ein weiteres Verfahren mit separaten Schrittweiten für jede Objektvariable vorgestellt, das außerdem noch eine weitere Verbesserung mit sich bringt: anstatt eine mit ξ modifizierte (mutierte) Schrittweite direkt an die Nachkommen zu vererben, wird nur eine durch den Dämpfungsexponenten β mit $0 < \beta < 1$ verminderte Schrittweitenänderung ξ^β vererbt:

$$\vec{\Delta} = \xi \cdot \vec{\delta} \cdot \vec{z} \tag{2.30}$$

$$\vec{\delta}' = \xi^\beta \cdot \vec{\xi}_{\vec{z}}^{\beta_{\text{scal}}} \cdot \vec{\delta} \tag{2.31}$$

$$\vec{x}' = \vec{x} + \vec{\Delta} \tag{2.32}$$

ξ ist dabei der aus Gleichung 2.25 schon bekannte skalare Schrittweitenmodifikationsfaktor.
$\vec{\xi}_{\vec{z}}$ wird aus dem Vektor unabhängiger, $N(0,1)$-verteilter Zufallszahlen $\vec{z}$ gebildet, der schon zur Bildung des Mutationsvektors benutzt wurde:

$$\vec{\xi}_{\vec{z}} = (\xi_1, \ldots, \xi_n) \quad \text{mit} \quad \xi_i = e^{\left(|z_i| - \sqrt{2/\pi}\right)} \tag{2.33}$$

$\sqrt{2/\pi}$ ist dabei der Erwartungswert von $|z_i|$, so daß der Exponent einen Erwartungswert von 0 hat und die ξ_i einen Erwartungswert von 1 haben und eine ähnliche Verteilung wie mit Gleichung 2.25 erreicht wird. Dies bewirkt, daß eine in einer Komponente i realisierte Mutation z_i, die betragsmäßig größer ausgefallen ist als der Erwartungswert, auch zu einer direkten Erhöhung der Schrittweitenkomponente δ_i führt. Dieser direkte kausale Zusammenhang ist bei der mutativen Schrittweitenregelung nicht gegeben: dort kann auch eine durch ξ_i vergrößerte Schrittweite durch eine zufällig sehr klein realisierte Komponente z_i in einem kleinen Mutationsschritt resultieren, wobei ein Rückschluß über den Fitneßwert dann keine sichere Aussage über die Tauglichkeit der Schrittweite mehr zuläßt.

Die Dämpfungsfaktoren werden für den globalen Modifikationsfaktor mit $\beta = \sqrt{1/n}$ und für die separaten (Skalierungs-) Modifikationsfaktoren $\beta_{\text{scal}} = 1/n$ gewählt, die separaten Schrittweiten werden also stärker gedämpft, und die globale Schrittweite kann sich schneller verändern. Die eigentliche Mutation wird mit ungedämpftem Modifikationsfaktor ξ durchgeführt, so daß die große Diversität bei den Nachkommen und somit die Relevanz einer Selektion erhalten bleibt. Die separaten Schrittweiten selbst werden aber gedämpft vererbt, so daß störende zufällige Einflüsse bei der Schrittweitenänderung reduziert werden. Daher heißt diese Methode auch “derandomized”.

2.4.3.5 Kumulierende Schrittweitenregelung (KSR)

Die im vorigen Abschnitt vorgestellte gedämpfte Schrittweitenanpassung benutzt zur Adaption der neuen Schrittweite nur Informationen des Vektors $\vec{z}$ aus dem letzten Mutationsschritt (Gleichung 2.33). Die Kumulierende Schrittweitenregelung KSR [Ostermeier et al. 94] dagegen akkumuliert die Informationen aus *mehreren* erfolgreichen Mutationsschritten auf, was eine Verbesserung der Adaptionsgeschwindigkeit zur Folge hat. Ein Individuum besteht dabei aus den Komponenten $I = \{\vec{x}, \delta, \vec{\delta}, \vec{\vec{Z}}\}$. Es enthält also neben den Objektvariablen $\vec{x}$ eine globale Schrittweite δ und einen Vektor mit separaten Schrittweiten $\vec{\delta}$. Im Vektor $\vec{\vec{Z}}$ werden die erfolgreichen (selektierten) Mutationsschritte $\vec{z}$ gewichtet aufsummiert. Die Trennung von globaler und separaten Schrittweiten wurde gemacht, damit sich die globale Schrittweite schneller an den Verlauf der Fitneßfunktion anpassen kann. Die separaten Schrittweiten dienen dann zur Festlegung einer Skalierung zwischen den Variablen.

Zuerst wird eine Mutation mit den gespeicherten Schrittweiten δ, $\vec{\delta}$ und einem komponentenweise $N(0,1)$-verteilten Vektor $\vec{z}$ durchgeführt:

$$\vec{\Delta} = \delta \cdot \vec{\delta} \cdot \vec{z} \tag{2.34}$$

$$\vec{x}' = \vec{x} + \vec{\Delta} \tag{2.35}$$

Der nächste Schritt findet erst nach der Selektion statt, also nachdem sich ein $\vec{z}$ (jetzt als $\vec{z}_{\text{sel}}$ bezeichnet, da es keine neue Instanz eines Zufallsvektors darstellt) als erfolgreicher Mutationsvektor herausgestellt hat[5]. Er wird mit dem Faktor c gewichtet auf den Vektor $\overrightarrow{\overrightarrow{Z}}$ aufsummiert. Es wird ein Wert von $c = \sqrt{1/n}$ vorgeschlagen. $\overrightarrow{\overrightarrow{Z}}$ wird mit dem Null-Vektor $\vec{0}$ initialisiert.

$$\overrightarrow{\overrightarrow{Z}}' = (1-c)\overrightarrow{\overrightarrow{Z}} + c \cdot \vec{z}_{\text{sel}} \tag{2.36}$$

Dann wird die globale Schrittweite mit der schon bekannten logarithmischen Verteilung, gedämpft durch den Exponenten β, vergrößert oder verkleinert, je nachdem ob $|\overrightarrow{\overrightarrow{Z}}|$ größer oder kleiner als der Erwartungswert ausgefallen ist.

$$\delta' = \delta \cdot \left(e^{\left(\frac{|\overrightarrow{\overrightarrow{Z}}|}{\sqrt{n} \cdot \sqrt{\frac{c}{2-c}}} - 1 + \frac{1}{5n} \right)} \right)^{\beta} \tag{2.37}$$

Der Betrag $|\overrightarrow{\overrightarrow{Z}}|$ wird noch durch Korrekturfaktoren normiert und dann 1 subtrahiert, damit der Exponent einen Erwartungswert von 0 hat. Der Term $\sqrt{\frac{c}{2-c}}$ korrigiert dabei den Einfluß der Gewichtungsfaktoren c und $1-c$ aus Gleichung 2.36 und $\frac{1}{5n}$ ist ein Korrekturterm für kleine Dimensionen n.

Die separaten Schrittweiten werden ganz ähnlich adaptiert, wobei ebenfalls auf den Vektor $\overrightarrow{\overrightarrow{Z}}$ zurückgegriffen wird. Hierbei wird allerdings ein anderer Dämpfungsfaktor β_{scal} verwendet. Es gelten wie schon bei dem Verfahren mit einfacher Dämpfung die Richtwerte $\beta = \sqrt{1/n}$ und $\beta_{\text{scal}} = 1/n$. Die Addition von 0.35 dient zur Normierung des geometrischen Mittels von $(|z| + 0.35)$ auf 1, um eine systematische Drift der Schrittweiten $\vec{\delta}$ zu verhindern.

$$\vec{\delta}' = \vec{\delta} \cdot \left(\frac{|\overrightarrow{\overrightarrow{Z}}|}{\sqrt{\frac{c}{2-c}}} + 0.35 \right)^{\beta_{\text{scal}}} \tag{2.38}$$

[5] Im Algorithmus können die nachfolgenden Berechnungen natürlich auch gleich zusammen mit obiger Mutation ausgeführt werden – die Ergebnisse werden dann allerdings für nicht selektierte Individuen nicht weiterverwendet, sondern mitsamt den Individuen verworfen.

2.4.3.6 Korrelierte Schrittweiten mit Rotationswinkeln

Die erste Methode, die eine Korrelation zwischen den einzelnen Schrittweiten herstellen konnte, war die Methode nach [Schwefel 81] mit Rotationswinkeln. Korrelation zwischen Schrittweiten bedeutet, daß ein vorteilhafter Quotient zwischen zwei Schrittweiten erkannt wird und bei der Erzeugung der Mutationsverteilung berücksichtigt wird. Anschaulich können damit Mutationsverteilungen in Form von Hyperellipsen erzeugt werden, deren Achsen nicht mehr parallel zu den Koordinatenachsen des Lösungsraumes sein müssen, sondern frei im Raum gedreht sein können (Abbildung 2.6). Ziel der Korrelation ist es, eine verallgemeinerte, n-dimensionale Normalverteilung $N(\vec{0}, \boldsymbol{C})$ mit Zentralwert $\vec{0}$ und der Kovarianzmatrix $\mathbf{C}^{-1}$ zu erzeugen und diese den lokalen Gegebenheiten der Fitneßfunktion anzupassen. Bei diesem Verfahren werden allerdings nicht direkt die Kovarianzen der Matrix mutiert, da sonst darauf geachtet werden müßte, daß die Basisvektoren des Koordinatensystems, das sie bilden, orthogonal zueinander sind (bzw. daß die Matrix positiv definit bleibt). Stattdessen wird das Koordinatensystem des Lösungsraumes durch eine Kette von Rotationen in das Ziel-Koordinatensystem überführt.

Bei der Korrelation mit Rotationswinkeln enthält ein Individuum $I = \{\vec{x}, \vec{\delta}, \vec{\omega}\}$ die Strategieparameter $\vec{\delta}$ (1 bis n_δ separate Schrittweiten) und $\vec{\omega}$ (0 bis n_ω Rotationswinkel).

$$\delta_i' = \delta_i \cdot e^{(\tau_1 z + \tau_2 z_i)} \qquad \forall i \in \{1, \ldots, n_\delta\} \tag{2.39}$$

$$\omega_j' = \omega_j + \beta \cdot z \qquad \forall j \in \{1, \ldots, n_\omega\} \tag{2.40}$$

$$\vec{\Delta} = \vec{\delta}' \cdot \prod_{i=1}^{n-1} \prod_{j=i+i}^{n} R(\omega_{ij}) \tag{2.41}$$

$$\vec{x}' = \vec{x} + \vec{\Delta} \tag{2.42}$$

$R(\omega_{ij}) = (r_{ij})$ sind dabei Rotationsmatrizen mit Dimension $n \times n$, die aus der Einheitsmatrix $\boldsymbol{I}$ bestehen plus den 4 zusätzlichen Einträgen $r_{ii} = r_{jj} = \cos(\omega_{ij})$ und $r_{ij} = -r_{ji} = -\sin(\omega_{ij})$. Es werden die Werte $\tau_1 = \frac{1}{\sqrt{2n}}$, $\tau_2 = \frac{1}{\sqrt{2\sqrt{n}}}$ und $\beta \approx 0.0873$ vorgeschlagen. Die konkatenierten Rotationen aus Gleichung 2.41 lassen sich programmtechnisch relativ kompakt implementieren, da die Matrizen regelmäßig aufgebaut sind.

Der Parameter n_δ gibt die Anzahl der separaten Schrittweiten des Vektors $\vec{\delta}$ an, mit $1 < n_\delta \leq n$. Ist $n_\delta < n$, so gelten die Schrittweiten δ_1 bis $\delta_{n_\delta - 1}$ für die Objektvariablen x_1 bis $x_{n_\delta - 1}$ und die Schrittweite δ_{n_δ} für die restlichen Objektvariablen x_{n_δ} bis

n_δ	n_ω	Verfahren
1	0	MSR mit globaler Schrittweite
n	0	MSR mit separaten Schrittweiten (unkorreliert)
n	$\frac{n\cdot(n-1)}{2}$	MSR mit separaten Schrittweiten, korreliert

Tabelle 2.1: Spezialfälle, die im Verfahren nach Schwefel mit Rotationswinkeln enthalten sind. MSR mit korrelierten, separaten Schrittweiten ist dabei neu.

x_n. Die Anzahl der Rotationswinkel ist durch $n_\omega = (2n - n_\delta)(n_\delta - 1)/2$ gegeben. Alternativ kann auch $n_\omega = 0$ gewählt werden, wodurch die Korrelation nicht verwendet wird. Durch Wahl spezieller Werte für n_δ und n_ω ergeben sich einfachere Adaptionsverfahren als Spezialfall des Rotationsverfahrens, diese sind in Tabelle 2.1 aufgelistet.

Der Zusammenhang zwischen den Werten von δ_i, ω_{ij} und den Elementen c_{ij} der Kovarianzmatrix ist gegeben durch:

$$c_{ii} = \delta_i \tag{2.43}$$

$$\tan(2\omega_{ij}) = \frac{2c_{ij}}{\delta_i^2 - \delta_j^2} \tag{2.44}$$

Dieses Adaptionsschema ist theoretisch in der Lage, beliebige, koordinatensystemunabhängige n-dimensionale Normalverteilungen zu generieren. In [Hansen et al. 95a] wird jedoch beobachtet, daß es in manchen Fällen versagt, die Verteilung einem zufällig rotierten Hyperellipsoiden anzupassen. Auch Permutationen der Koordinatenachsen der Zielfunktion können sich negativ auf die Performance auswirken. Dies wurde in neueren Arbeiten auf die geringere Selektionsrelevanz der Rotationswinkel gegenüber den Schrittweiten $\vec{\delta}$ zurückgeführt. Dadurch unterliegen die Winkel großen stochastischen Fluktuationen, so daß sie beinahe zufällig variiert werden. Genauso wie die MSR mit separaten Schrittweiten, die ja auch als Spezialfall enthalten ist, benötigt dieses Verfahren ausreichend große Populationen, um funktionieren zu können. Schwefel verwendet deswegen als Standard eine $(15, 100)$-ES.

Diese hier genannten Nachteile werden mit dem nachfolgend vorgestellten Verfahren CMA ausgeglichen.

2.4.3.7 Kovarianz-Matrix-Adaption (CMA)

Die *Kovarianz-Matrix-Adaption* (engl. covariance matrix adaption - CMA) [Hansen et al. 96] ist die leistungsfähigste und komplexeste der hier vorgestellten Schrittweitenadaptionsmethoden. Sie ist eine Weiterentwicklung der *Erzeugendensystem-Adaption*

(engl. generating set adaptation - GSA) [Hansen et al. 95b] [Hansen et al. 95a], auf die hier nicht näher eingegangen werden soll[6].

Die Grundidee bei CMA ist, eine beliebige $N(\vec{0}, \boldsymbol{C})$ Verteilung zu erzeugen und zu adaptieren, indem direkt die Kovarianzmatrix $\boldsymbol{C}$ adaptiert wird. Ein Individuum besteht dabei aus den Elementen $I = \{\vec{x}, \delta, \vec{s}, \vec{s}_\delta, \boldsymbol{C}\}$. δ ist die bereits bekannte globale Schrittweite. $\vec{s}$ und $\vec{s}_\delta$ sind die gewichtet aufsummierten, erfolgreichen Mutationsschritte, ähnlich zu $\vec{\vec{Z}}$ in Gleichung 2.36, wobei $\vec{s}_\delta$ eine normalisierte Mutationsschritte aufsummiert. $\boldsymbol{C}$ ist die Kovarianzmatrix, welche zu Beginn mit $\boldsymbol{C}_0 = \boldsymbol{I}$ initialisiert wird. Anfangswerte für die Summationsvektoren sind $\vec{s} = \vec{s}_\delta = \vec{0}$.

$$\vec{\Delta} = \delta \cdot \boldsymbol{B} \cdot \vec{z} \tag{2.45}$$

$$\vec{x}' = \vec{x} + \vec{\Delta} \tag{2.46}$$

Der erste Schritt ist bei CMA - ebenso wie bei den vorhergehenden Verfahren - die Mutation des Objektvariablenvektors $\vec{x}$. Hierbei wird der komponentenweise $N(0,1)$-verteilte Zufallsvektor $\vec{z}$ mit Hilfe einer Matrix $\boldsymbol{B}$ transformiert, um einen Vektor $\vec{\Delta}$ mit $N(\vec{0}, \boldsymbol{C})$-Verteilung zu erhalten. Der Zusammenhang zwischen $\boldsymbol{B}$ und $\boldsymbol{C}$ ist dabei gegeben durch:

$$\boldsymbol{C} = \boldsymbol{B}\boldsymbol{B}^T. \tag{2.47}$$

Mit dieser Gleichung ist die Bestimmung von $\boldsymbol{B}$ nicht eindeutig. Die Autoren wählen deshalb als Spaltenvektoren von $\boldsymbol{B}$ die normalisierten Eigenvektoren $\frac{1}{|\vec{c}_i|}\vec{c}_i$ $(i = 1 \ldots n)$ von $\boldsymbol{C}$ jeweils multipliziert mit der Wurzel des dazugehörigen Eigenwertes e_i:

$$\boldsymbol{B} = \left(\begin{array}{ccc} \frac{\sqrt{e_1}}{|\vec{c}_1|}\vec{c}_1 & \ldots & \frac{\sqrt{e_n}}{|\vec{c}_n|}\vec{c}_n \end{array} \right). \tag{2.48}$$

Es ist also eine Eigenvektor- und Eigenwertbestimmung von $\boldsymbol{C}$ nötig, welche mit steigender Dimension n sehr rechenaufwendig werden kann.

Zur Adaption der Kovarianzmatrix wird zuerst die gewichtete Summe der erfolgreichen Mutationsschritte gebildet. Dieser Schritt des Verfahrens findet nach der Selektion statt, der in Gleichung 2.45 erzeugte Zufallsvektor $\vec{z}$ wird hier unter dem Namen $\vec{z}_{\text{sel}}$ wiederverwendet (analog zu KSR, Seite 25):

$$\vec{s}' = (1 - c) \cdot \vec{s} + c \cdot c_u \boldsymbol{B} \vec{z}_{\text{sel}} \tag{2.49}$$

[6] Auf einen Unterschied sei noch hingewiesen: GSA verwendet zum Wechsel des Koordinatensystems ein Erzeugendensystem, also eine Menge von Vektoren. Aufgrund der Anzahl der Vektoren braucht das Verfahren Speicherplatz in der Größenordnung von $O(n^3)$. CMA dagegen speichert die Koordinatensystemtransformation in einer Matrix, welche nur einen Speicherplatzbedarf der Größenordnung $O(n^2)$ hat.

Der Gewichtungsfaktor c regelt die Geschwindigkeit der Adaption und kann im Intervall $(0, 1]$ gewählt werden. $c_u = \sqrt{(2-c)/c}$ dient zur Normalisierung der Varianz von $\vec{s}$.

Die Kovarianzmatrix selbst wird dann folgendermaßen adaptiert:

$$\boldsymbol{C}' = (1 - c_{\text{cov}}) \cdot \boldsymbol{C} + c_{\text{cov}} \cdot \vec{s}\,(\vec{s})^T\,. \tag{2.50}$$

Hier tritt ein weiterer Gewichtungsfaktor c_{cov} auf, der (wie c) im Intervall $(0, 1]$ gewählt werden kann und die Adaptionsgeschwindigkeit der Kovarianzmatrix regelt. Von den Autoren wird ein Wert von $c \lessapprox \frac{2}{n^2}$ vorgeschlagen.

Um die globale Schrittweite δ zu adaptieren, wird eine weitere kumulierte Summe der Mutationsschritte ähnlich zu Formel 2.49 benötigt:

$$\vec{s}_\delta' = (1 - c) \cdot \vec{s}_\delta + c \cdot c_u \boldsymbol{B}_\delta \vec{z}_{\text{sel}} \tag{2.51}$$

Der einzige Unterschied besteht darin, daß statt $\boldsymbol{B}$ hier $\boldsymbol{B}_\delta$ verwendet wird, welches gleich $\boldsymbol{B}$ mit normalisierten Spalten ist. Dadurch hat $\boldsymbol{B}_\delta \vec{z}_{\text{sel}}$ eine $N(\vec{0}, \boldsymbol{I})$-Verteilung. Zusammen mit der Normalisierung der gewichteten Summe durch $c_u = \sqrt{(2-c)/c}$ unterliegt auch $\vec{s}_\delta$ einer $N(\vec{0}, \boldsymbol{I})$-Verteilung. Abweichungen vom Erwartungswert können also leicht zur Adaptions-Entscheidung für δ benutzt werden:

$$\delta' = \delta \cdot e^{(\beta \cdot c \cdot (\|\vec{s}_\delta\| - \hat{\chi}_n))}. \tag{2.52}$$

Hierbei ist $\hat{\chi}_n = \sqrt{n}(1 - \frac{1}{4n} + \frac{1}{21n^2})$, welches den Erwartungswert der χ_n-Verteilung annähert, was wiederum die Verteilung der Länge eines $N(\vec{0}, \boldsymbol{I})$-verteilten Zufallsvektors in $\mathbb{R}^n$ ist. Der Faktor β ($0 < \beta \leq 1$) dämpft die Schrittweitenanpassung und sollte als $\beta \leq c$ gewählt werden, um Oszillationen (Sequenzen von abwechselnd vergrößertem und verkleinertem δ) zu vermeiden.

CMA ist in dieser Form nicht für einfache Rekombinationsverfahren geeignet. Der Adaptionsprozeß würde gestört werden. In [Hansen et al. 97] wird ein weiterentwickeltes Schema der CMA mit Rekombination beschrieben, auf das hier nicht näher eingegangen werden soll.

Der Autor weist bei diesem Verfahren noch darauf hin, daß eine gelegentlich vorkommende Degeneration der Verteilung verhindert werden kann, indem man das Verhältnis zwischen größtem und kleinstem Eigenwert beschränkt. Desweiteren kann das Verfahren auch so implementiert werden, daß die Strategievariablen δ, $\vec{s}$, $\vec{s}_\delta$ und $\boldsymbol{C}$ nicht pro Individuum existieren, sondern global für die ganze Population. Zur Bildung der aufsummierten Vektoren $\vec{s}$ und $\vec{s}_\delta$ wird dann der Mutationsvektor $\vec{z}_{\text{sel}}$ des besten selektierten Individuums herangezogen. Es genügt sogar, wenn die Kovarianzmatrix nicht in jeder Generation adaptiert wird. Eine Implementierung dieses Verfahrens in Matlab-Code ist in [Hansen et al. 00] veröffentlicht.

2.4.3.8 Differential Evolution

Differential Evolution (DE) [Price et al. 95, Price et al. 97a, Price et al. 97b] paßt nicht ganz in das Schema einer Mutationsschrittweitenregelung. Einerseits wird es als kompletter Algorithmus vorgestellt, der deswegen auch Selektion usw. beinhaltet, andererseits gibt es darin nicht einmal eine explizit abgespeicherte Variable für die Mutationsschrittweite. Dennoch läßt sich die Mutationsidee daraus isolieren und in eine Evolutionsstrategie übertragen.

Die Quelle für Mutationsvektoren $\vec{\Delta}$ bildet die Verteilung der Population im Lösungsraum. Zur Mutation von Individuen werden einfach Vektordifferenzen zwischen zwei zufällig ausgewählten Individuen benutzt:

$$\vec{\Delta} = c \cdot (\vec{x}_j - \vec{x}_k) \qquad j = U(1,\mu),\ k = U(1,\mu),\ c \approx 0.7 \tag{2.53}$$

$$\vec{x}' = \vec{x} + \vec{\Delta} \tag{2.54}$$

Dabei kann völlig auf die Verwendung von normalverteilten Zufallszahlen verzichtet werden. Das Verfahren ist extrem einfach zu implementieren, aber trotzdem leistungsfähig. Es ist damit möglich, Adaptionen an die Fitneßfunktion zu erreichen, wie sie sonst nur mit der vergleichsweise komplexen Kovarianzmatrix-Adaption zu erreichen sind. Allerdings werden deutlich mehr Funktionsauswertungen benötigt.

Anschaulich gesehen ist die Wahrscheinlichkeit hoch, aus einer Population, die sich z. B. entlang eines schmalen Grates der Fitneßfunktion verteilt hat, wieder Mutationsvektoren zu gewinnen, die in Richtung des schmalen Grates verlaufen. Der Rest des originalen Algorithmus ist bewußt einfach gehalten und kann innerhalb weniger Zeilen Code implementiert werden. Es wird auf komplexe Selektionsschemata zugunsten einfacher, lokaler Verfahren verzichtet.

Eine Zusammenfassung der beiden Formeln 2.53 und 2.54 ergibt:

$$\vec{x}' = \vec{x} + c \cdot (\vec{x}_j - \vec{x}_k) \tag{2.55}$$

Diese Formel ähnelt sehr der Formel 2.11 für die verallgemeinerte intermediäre Rekombination. Der entscheidende Unterschied ist, daß bei Differential Evolution ein drittes Individuum beteiligt ist, auf welches die gewichtete Vektordifferenz aufaddiert wird.

Ein großer Nachteil von Differential Evolution ist aber die geringe Adaptionsgeschwindigkeit. Erst wenn sich die Verteilung der gesamten Elternpopulation wesentlich verändert hat, ändert sich die durchschnittliche Länge des Mutationsvektors $\vec{\Delta}$. Damit die Verteilung der Population aber eine ausreichend gute Quelle für Mutationen bildet, muß auch eine entsprechend große Population verwendet werden. Desweiteren ist fraglich, ob die Verwendung des Faktors $c \approx 0.7$ nicht eine ständig kleiner werdende durchschnittliche Länge des Mutationsvektors hervorruft. Dies ist zwar im Allgemeinen gewünscht, aber vielleicht versagt das Verfahren, wenn die Schrittweite vergrößert werden muß. Darüber existieren allerdings noch keine Untersuchungen.

Art der Regelung → Adaptionsebene ↓	mutativ	gedämpft	gedämpft + kumulierend
Population	1/5-Erf.-Regel		
Individuum	MSR (Rot. $n_\delta = 1;\ n_\omega = 0$)		
Objektvariable			
- unkorreliert	MSR separate Schrittw. (Rot. $n_\delta = n;\ n_\omega = 0$)	GSR	KSR
- korreliert	Rot. $n_\delta = n;\ n_\omega = \frac{n\cdot(n-1)}{2}$		CMA

Tabelle 2.2: Einordnung der Adaptionsverfahren nach Adaptionsebene und Art der Schrittweitenregelung. Mit *Rot.* wird das Verfahren mit korrelierten Schrittweiten und Rotationswinkeln bezeichnet.

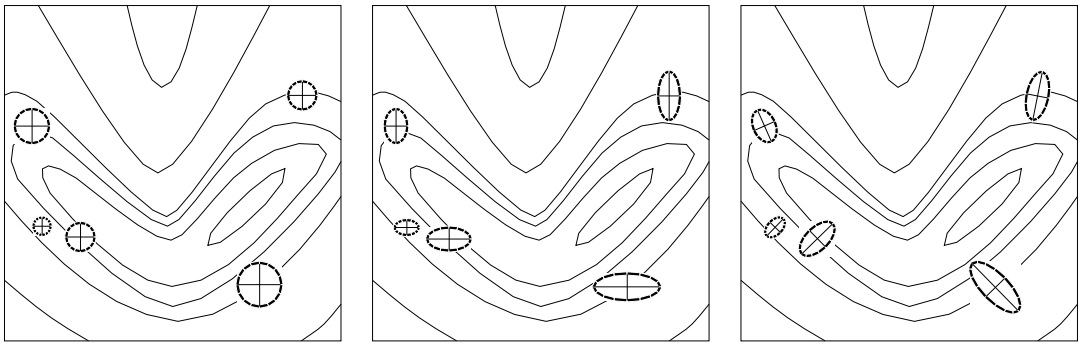
Verfahren	Speicherplatzbedarf	Rechenzeitbedarf
1/5 Erfolgsregel	$O(1)$ (pro Population)	$O(1)$
MSR	$O(1)$ (pro Individuum)	$O(n)$
MSR separat	$O(n)$	$O(n)$
GSR	$O(n)$	$O(n)$
KSR	$O(n)$	$O(n)$
Rotationswinkel	$O(n^2)$	$O(n^2)$
CMA	$O(n^2)$	$O(n^2)$
DE	0	$O(n)$

Tabelle 2.3: Klassifizierung der Adaptionsverfahren nach Speicherplatzverbrauch in Abhängigkeit von der Dimension n des Problems.

2.4.3.9 Klassifizierung der Verfahren

Die hier vorgestellten Schrittweitenadaptionsverfahren lassen sich durch die auf Seite 19 vorgestellten Kriterien der Art der Schrittweite und der Art der Schrittweitenanpassung klassifizieren (siehe Tabelle 2.2). Differential Evolution läßt sich allerdings in dieses Schema nicht problemlos einordnen. Ein Vergleich des Speicherplatzbedarfs der Verfahren ist in Tabelle 2.3 aufgezeigt.

Wie sich die unterschiedlichen Fähigkeiten auf die Erzeugung von Mutationsverteilungen auswirken, ist für den zweidimensionalen Fall in Abbildung 2.6 veranschaulicht. Dargestellt ist ein Höhenlinienbild eines “Fitneßgebirges”. Auf den Höhenlinien ist die Fitneß jeweils gleich, die Kreise und Ellipsen stellen die Linien dar, auf denen mit gleicher Wahrscheinlichkeit Nachkommen plaziert werden. Die verwendete Schrittweitenregelung wirkt sich wie folgt aus:



(a) globale Schrittweite

(b) unkorrelierte, separate Schrittweiten

(c) korrelierte Schrittweiten

Abbildung 2.6: Veranschaulichung der drei verschiedenen Typen von Mutationsverteilungen (nach [Bäck 96]).

- Mit einer globalen Schrittweite lassen sich nur kreisförmige Mutationsverteilungen erzeugen (siehe Abbildung 2.6(a)).
- Mit separaten Schrittweiten (unkorreliert) lassen sich ellipsenförmige Mutationsverteilungen erzeugen, deren Achsen allerdings immer parallel zu den Koordinatenachsen des Lösungsraumes sind, siehe Abbildung 2.6(b).
- Mit korrelierten, separaten Schrittweiten können frei im Raum gedrehte, ellipsenförmige Mutationsverteilungen erzeugt werden, siehe Abbildung 2.6(c).

[Hansen 00] stellt die Forderung nach Invarianz des Adaptionsverfahrens gegenüber bestimmten Transformationen der Zielfunktion, wie z. B. Translation, Rotation, ordnungserhaltende Transformation, skalare Multiplikation oder sogar beliebigen linearen Transformationen. Das bedeutet, daß das Optimierungsverfahren exakt dieselbe Performance erzielen soll, auch wenn die Zielfunktion und der Anfangspunkt der Optimierung transformiert werden. Der Autor kommt zu dem Schluß, daß nur das CMA-Verfahren invariant gegen beliebige lineare Transformationen ist.

2.4.4 Selektion

Die Selektion ist in allen Arten von Evolutionären Algorithmen ein unverzichtbarer Operator. Bei der Evolutionsstrategie interagieren Mutation und Selektion eng miteinander[7]. Die Mutation ist ein Operator, der Vielfalt auf der Ebene der Objektvariablen

[7] Bei Genetischen Algorithmen übernimmt die Rekombination die Aufgabe, welche die Mutation bei Evolutionsstrategien hat.

erzeugt. Die Selektion dagegen reduziert diese Vielfalt wieder, indem sie eine Auswahl der Individuen anhand der Fitneß der Objektvariablen trifft. Sind diese beiden Operatoren nicht gut genug aufeinander abgestimmt, so kann es passieren, daß die Population schnell zu lauter sehr ähnlichen Individuen konvergiert. Im anderen Fall kann sie auch zur Zufallssuche degenerieren.

Selektion: Ein Selektionsoperator s wählt aus einer Menge von p Individuen q aus.

$$s : I^p \rightarrow I^q \tag{2.56}$$

Bei Evolutionären Algorithmen wird an zwei Punkten im Algorithmus Selektion eingesetzt:

Elternselektion: (*vor* Mutation und Rekombination) beim Genetischen Algorithmus wird an dieser Stelle üblicherweise eines der stochastischen Verfahren Roulette-Wheel- oder Ranking-Selektion eingesetzt, da hier ein direkter Zusammenhang zwischen der Fitneß und der Fortpflanzungswahrscheinlichkeit besteht. Bei einer Evolutionsstrategie werden die Eltern dagegen gleichverteilt ausgewählt, die eigentliche Selektion findet später statt (siehe nächster Punkt)

Nachkommenselektion: (*nach* Mutation, Rekombination und Fitneßevaluation) bei einer Evolutionsstrategie werden erst, nachdem eine große Nachkommenpopulation erzeugt worden ist, diejenigen Individuen ausgewählt, die nun die Eltern der nächsten Generation bilden. Beim Genetischen Algorithmus werden dagegen alle Individuen der Nachkommenpopulation in die nächste Generation übernommen.

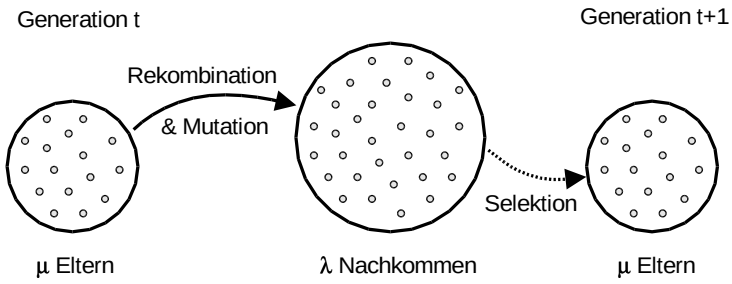
Bei Evolutionsstrategien gibt es zwei Arten von Selektion (siehe Abbildung 2.7):

Komma-Selektion: diese wird notiert als (μ, λ). Bei der Komma-Selektion werden aus μ Elternindividuen λ Nachkommen erzeugt. Dann werden alleine aus den λ Nachkommen μ Individuen als Eltern der nächsten Generation ausgewählt. Hier muß natürlich $\lambda \geq \mu$ gelten.

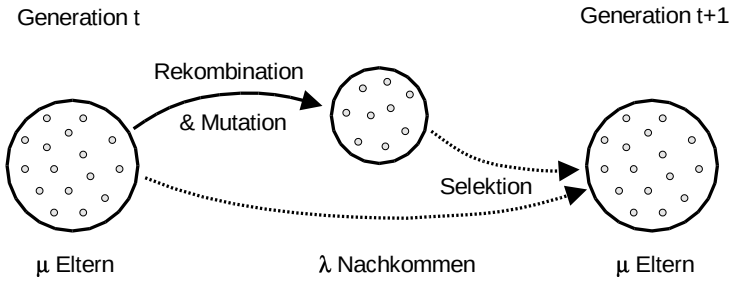
Plus-Selektion: diese wird notiert als $(\mu + \lambda)$. Bei der Plus-Selektion bilden die λ Nachkommen *plus* die ursprünglichen μ Eltern die Menge, aus der selektiert wird. Hier kann λ unabhängig von μ gewählt werden.

Das Verhältnis $\frac{\mu}{\lambda}$ (Komma-Selektion) bzw. $\frac{\mu}{\mu+\lambda}$ (Plus-Selektion) bildet hierbei ein Maß für den sogenannten *Selektionsdruck.*

Grundsätzlich kann ein Selektionsmechanismus eine der beiden folgenden Eigenschaften *extinctive* oder *preservative* haben [Bäck 91]:



(a) Selektion bei (μ, λ)



(b) Selektion bei $(\mu + \lambda)$

Abbildung 2.7: Selektion bei der (μ, λ) und $(\mu + \lambda)$ Evolutionsstrategie.

extinctive: auslöschende Selektion, d. h. es gibt mindestens ein Individuum, das beim gewählten Selektionsverfahren eine Selektionswahrscheinlichkeit von 0 hat. Sowohl die $(\mu + \lambda)$ als auch die (μ, λ)-Selektion sind auslöschend.

preservative: erhaltende Selektion, d. h. jedes Individuum der Population hat eine Selektionswahrscheinlichkeit größer 0. Beispiel: Roulette-Wheel-Selektion.

Desweiteren unterscheidet man noch die Eigenschaften *elitist* oder *pure* bei der Selektion:

elitist: Eltern werden mit in den Selektionsprozeß einbezogen und stehen in Konkurrenz zu den Nachkommen. Beispiel: $(\mu + \lambda)$-Selektion. Insbesondere bei Genetischen Algorithmen bedeutet dies auch, daß das beste Individuum auf jeden Fall in die nächste Generation übernommen wird.

pure: Eltern werden nicht in den Selektionsprozeß einbezogen. Ein Individuum hat grundsätzlich die Lebensdauer von nur einer Generation. Beispiel: (μ, λ)-Selektion.

2.4.5 Terminierung

Evolutionäre Algorithmen sind keine “abgeschlossenen” Algorithmen, d. h. sie sind nicht irgendwann am Ende der Berechnung angelangt, mit dem Ergebnis, die optimale Lösung gefunden zu haben. Es kann sogar ohne zusätzliches Wissen über die Zielfunktion überhaupt nicht festgestellt werden, ob es sich bei einer gefundenen Lösung um das globale Optimum handelt oder nicht. Vielmehr muß eine Haltebedingung angegeben werden, um festzulegen, wann der Algorithmus nicht weiter iteriert werden soll. Über die Güte der Lösung, die gefunden wird, kann keine Zusicherung gemacht werden. Aus diesem Grunde kann man auch nicht von der Komplexität des Algorithmus auf einer bestimmten Problemstellung sprechen. Ein NP-vollständiges Problem wird durch den Einsatz eines Evolutionären Algorithmus nicht in der Komplexität reduziert, i. A. können aber sehr gute Näherungslösungen in akzeptabler Zeit erzielt werden.

Als Terminierungsbedingung kommen verschiedene Möglichkeiten in Betracht:

Erreichen einer maximalen Berechnungszeit: dies kann sowohl eine absolute Zeit in Sekunden sein, als auch - basierend auf der Zeiteinheit des Algorithmus - eine bestimmte Anzahl an Generationen.

Erreichen einer vorgegebenen Fitneß: wenn das Fitneßmaß eine Zahl mit einer bestimmten Bedeutung ist (z. B. Weglänge beim TSP, usw.) und nicht nur als maßstabsloses Ordnungskriterium verwendet wird, macht die Angabe einer Fitneß Sinn, die das Problem für die Ansprüche des Benutzers hinreichend gut löst. Sobald ein Individuum diesen Fitneßwert oder einen besseren erreicht, wird abgebrochen.

Stagnation: macht die Optimierung ein bestimmtes Zeitintervall lang keinen signifikanten Fortschritt mehr, so ist es auch sinnvoll, abzubrechen. Bei Genetischen Algorithmen existieren Maße wie Bitkonvergenz oder -diversität, anhand derer festgestellt werden kann, ob die Population praktisch nur noch aus gleichartigen Individuen besteht. In diesem Falle würde ein Weiterrechnen wenig Sinn machen, da der Hauptoperator Crossover keine neuen Informationen einbringen kann. Bei Evolutionsstrategien sind diese Maße nicht anwendbar. Es kann aber das Unterschreiten einer minimalen Änderung im Fitneß- oder Individuenraum als Stopkriterium verwendet werden:

- **Fitneßraum:** Terminierungsbedingung ist das Unterschreiten einer minimalen Fitneß-Differenz zwischen bestem und schlechtestem Individuum.
- **Individuenraum:** Terminierungsbedingung ist das Unterschreiten einer minimalen Mutationsschrittweite. Hierzu wird am besten die größte in der Population vorkommende Schrittweite herangezogen. Mutationen würden dann nur noch relativ kleine Änderungen bewirken. Dies würde z. B. Sinn machen, wenn die Objektvariablen Koordinaten sind und eine Genauigkeit von weniger als 1 mm bei der Anwendung ausreicht.

2.4.6 Randbedingungen

In vielen Anwendungen existieren Randbedingungen, die von den Lösungen eingehalten werden müssen, damit sie als gültig anerkannt werden (Definition siehe 2.2.1). Der englische Originalbegriff lautet *Constraint*.

Zunächst kann man die Art der Problemstellung feiner klassifizieren, abhängig davon, wie schwierig die Randbedingungen zu erfüllen sind, oder ob überhaupt Randbedingungen vorhanden sind:

Free Optimization Problem (FOP): besteht aus einem Suchraum und einer Fitneßfunktion. Die Lösung des FOP ist dasjenige Element des Suchraumes, welches eine optimale (minimale/maximale) Fitneß besitzt. Der Begriff *Free* kommt daher, daß der Suchraum frei von Randbedingungen ist.

Constraint Satisfaction Problem (CSP): besteht aus einem Suchraum und einer Menge von Randbedingungen. Lösung des CSP ist dasjenige Element des Suchraumes, welches alle Randbedingungen erfüllt (Definition *erfüllt*, s. Abschnitt 2.2.1, Gleichung 2.3). Je nach Problemstellung sind eine, mehrere oder alle Lösungen des CSP gesucht.

Constraint Optimization Problem (COP): besteht aus einem Suchraum, einer Menge von Randbedingungen und einer Fitneßfunktion. Lösung des COP ist dasjenige Element des Suchraumes, welches alle Randbedingungen erfüllt und eine optimale Fitneß besitzt.

Im CSP geht es also darum, überhaupt eine Lösung zu finden, welche die Randbedingungen erfüllt. Weitere Ansprüche an die Lösung sind nicht gestellt. Beim COP dagegen soll eine Lösung gefunden werden, welche die Randbedingungen erfüllt und dabei noch optimal bezüglich der Fitneßfunktion ist.

Hier sind einige grundsätzliche Möglichkeiten gezeigt, wie Randbedingungen in Evolutionsstrategien oder allgemein in Evolutionären Algorithmen integriert werden können:

Strafterme: Lösungen, die Randbedingungen verletzen, werden zugelassen, allerdings wird ein Strafterm auf die Fitneß aufaddiert, so daß diese Lösungen schlechter bewertet werden. Hierbei kann z. B. der Wert, den die Randbedingung g_i ausgibt, direkt im Strafterm verwendet werden und muß nur noch entsprechend gewichtet werden. Die Gewichtung kann den Suchprozeß allerdings stark beeinflussen, d. h. es stellt sich die Frage, wie die Gewichte gewählt werden sollen. Dieser Problematik nimmt sich z. B. das Verfahren der Koevolution an (siehe unten).
Die Straftermaddition hat den Vorteil, daß die Kontinuität des Lösungsraumes

erhalten bleibt und bietet sich auch an, wenn Verletzungen von Randbedingungen öfter vorkommen. Die Hoffnung dabei ist, daß die Stärke der Verletzung den Suchprozeß in Regionen mit nur wenig verletzten bzw. sogar erfüllten Randbedingungen leitet.

Verwerfen und Neugenerieren: Kommt die Verletzung von Randbedingungen im Lösungsraum relativ selten vor, so können ungültige Individuen einfach verworfen werden. Es muß dann der Prozeß der Generierung (Elternselektion, Rekombination, Mutation) solange wiederholt werden, bis ein gültiges Individuum erzeugt wurde.

Reparieren: Lösungen, die Randbedingungen verletzen, werden repariert. Dies setzt voraus, daß eine Möglichkeit bekannt ist, gültige Lösungen zu erzeugen. Beim Reparieren stehen noch weitere Möglichkeiten zur Wahl: das Individuum wird nicht wirklich verändert, sondern die reparierte Lösung nur zur Berechnung der Fitneß verwendet; oder das Individuum wird tatsächlich verändert und so weiterverwendet.

Decoder: Der Evolutionäre Algorithmus arbeitet hierbei auf einem modifizierten Suchraum, dessen Elemente mittels einer Transformation (Decoder) in den originalen Suchraum überführt werden können. Der Decoder erzeugt dabei nur Lösungen, die die Randbedingungen erfüllen. Die Schwierigkeit liegt nun darin, die Decoder-Funktion zu konstruieren. Möglicherweise kann der Decoder auch nicht alle Lösungen des originalen Suchraums erzeugen, so daß nur ein Teilraum abgesucht wird.

Randbedingungen-erhaltende Operatoren: Eine weitere Möglichkeit ist es, Mutations- und Crossover-Operatoren zu verwenden, die aus gültigen Individuen wiederum nur gültige Individuen erzeugen. Als Voraussetzung muß hierbei natürlich eine Anfangspopulation generiert werden, die nur aus gültigen Individuen besteht.

Koevolution: Hierbei wird die Menge der Randbedingungen als eine zweite Population neben der Menge der Individuen angesehen[8]. Die "Fitneß" K_j einer Randbedingung g_j ermittelt sich proportional zu der Anzahl der Individuen, welche die Randbedingung verletzen. Diese Fitneßwerte werden dann als Gewichtung der Randbedingungen benutzt, die in den Straftermen S in die Fitneßfunktion einfließen. Dadurch erhält man eine dynamische Gewichtung, bei der jeweils stark verletzte (oder "schwierig" zu erfüllende) Randbedingungen stärker gewichtet werden.

Am Anfang müssen alle Gewichtungen der Randbedingungen auf einen Anfangswert K_{init} gesetzt werden. Dieser sollte einerseits groß gewählt werden,

[8] Die Population der Constraints ist streng genommen keine solche, sondern nur eine normale Menge, da sie nicht mithilfe von Mutation, Rekombination und Selektion evolviert wird.

damit die Randbedingungen auch angemessen gewichtet werden, andererseits aber klein, damit nicht von vorneherein Wege zu guten Lösungen ausgeschlossen werden.

$$K_j = K_{\text{init}} \qquad \forall j : 1 \leq j \leq n_g \tag{2.57}$$

Bei der Fitneßberechnung der Nachkommenpopulation werden nun die Verletzungen der Randbedingungen mit diesen Werten in einem Strafterm S gewichtet:

$$S(\vec{x}_i) = \sum_{j=1}^{n_g} K_j \cdot g_j(\vec{x}_i) \qquad \forall i : 1 \leq i \leq \lambda \tag{2.58}$$

Die Strafterme werden dann auf die normale Fitneß aufaddiert, um die Gesamtfitneß zu bekommen:

$$F(\vec{x}_i) = f(\vec{x}_i) + S(\vec{x}_i) \qquad \forall i : 1 \leq i \leq \lambda \tag{2.59}$$

Nach der Selektion, wenn also nur noch μ Individuen übrig sind, wird anhand des Prozentsatzes der Individuen, die eine Randbedingung verletzen, die Fitneß (das Gewicht) der Randbedingung erhöht. Dazu wird zuerst die binäre Funktion $g'_j(\vec{x})$ benötigt, die entscheidet, ob eine Randbedingung verletzt wird:

$$g'_j(\vec{x}_i) = \begin{cases} 1 & \text{falls} \quad g_j(\vec{x}_i) > 0 \\ 0 & \text{falls} \quad g_j(\vec{x}_i) = 0 \end{cases} \tag{2.60}$$

Dann kann der Prozentsatz p_j ermittelt werden, welcher Anteil der Individuen der Elternpopulation die Randbedingung g_j verletzen:

$$p_j = \frac{1}{\mu} \sum_{i=1}^{\mu} g'_j(\vec{x}_i) \tag{2.61}$$

Mithilfe eines Schwellwertes $0 \leq q \leq 1$ werden dann die neuen Gewichte $K_j(t+1)$ der Randbedingungen berechnet:

$$K_j(t+1) = \begin{cases} \alpha_{\text{coev}} \cdot K_j(t) & \text{falls} \quad p_j > q \\ \frac{1}{\alpha_{\text{coev}}} \cdot K_j(t) & \text{falls} \quad p_j \leq q \end{cases} \tag{2.62}$$

Der Schwellwert entscheidet, ob die Gewichtung einer Randbedingung verstärkt oder abgeschwächt wird. Dieser kann während der Evolution verändert werden und sollte am Ende 0 betragen, damit nur erfüllte Randbedingungen abgeschwächt werden. Der Faktor α_{coev} ist der Modifikationsfaktor für die Gewichte K_j.

[Paredis 92] macht einen etwas anderen Ansatz bei der Koevolution. Hier finden vor dem Crossover und der Mutation zuerst eine Anzahl (20) “Begegnungen”

(orig. “encounter”) zwischen jeweils einem Individuum und einer Randbedingung statt. Diese werden mit einer Ranking-Selektion ausgewählt. Ein Individuum bekommt eine Belohnung von +1, wenn es eine Randbedingung erfüllt und eine Strafe von -1, wenn es eine Randbedingung verletzt. Die Randbedingung dagegen bekommt genau den negierten Wert gutgeschrieben. Jedes Individuum und jede Randbedingung halten nun eine Historie der Ergebnisse ihrer letzten 25 Begegnungen. Die Fitneß ergibt sich aus der Summe der Punkte in der Historie.

Für eine Vertiefung zu Randbedingungen in Evolutionären Algorithmen sei auf [Michalewicz 91] und [Bäck et al. 97], Abschnitt C5 und die dort genannte, weiterführende Literatur verwiesen.

2.5 Andere Evolutionäre Algorithmen

2.5.1 Genetic Programming

Die Besonderheit bei Genetic Programming (GP) ist, daß die zu evolvierenden Objekte Computerprogramme sind, die in einer abstrakten Repräsentation als Syntaxbaum vorliegen. Diese bestehen aus Funktionen, Variablen und Konstanten. Die bedeutendsten Arbeiten auf diesem Gebiet stammen von John Koza [Koza 92, Koza 94]. Computerprogramme stellen universelle Problemlöser dar, somit umfaßt das GP ein weites Anwendungsspektrum von symbolischer Funktionsapproximation, Klassifikation, Steuerung/Regelung, usw. Die ursprünglich verwendete Sprache war LISP, da sich die Baumstruktur einfach in LISP umsetzen läßt und es sich um eine interpretierte Sprache handelt. Aktuelle Forschungsarbeiten beschäftigen sich aber auch mit der Verwendung von Maschinencode. Genetic Programming kann heute als eigenständiger EA-Zweig angesehen werden.

2.5.2 Evolutionary Programming

Evolutionäre Programmierung (EP) basiert auf frühen Arbeiten von L. J. Fogel, A. J. Owens und M. J. Walsh [Fogel et al. 66] und wird heute u. a. von D. B. Fogel, dem Sohn von L. J. Fogel, fortgesetzt [Fogel 92, Fogel 95]. Ursprünglich wurden damit endliche Automaten optimiert, grundsätzlich gilt aber das Paradigma, daß eine dem Problem angepaßte Repräsentation verwendet werden soll. Dementsprechend wird bei modernem EP für reellwertige Parameteroptimerungsaufgaben auch ein Vektor reellwertiger Zahlen zur Lösungsrepräsentation verwendet. Hierbei existieren auch selbstadaptive Mutationsoperatoren (allerdings wird auf Rekombination in der Regel komplett verzichtet). Dadurch ist Evolutionäre Programmierung näher mit Evolutionsstrategien verwandt, als mit den genotyplastigen Genetischen Algorithmen.

2.5.3 Classifier Systeme

Classifier Systeme (CS) gehen auf Arbeiten von Holland und Reitman zurück [Holland et al. 78, Holland et al. 86]. Sie sind selbst eigentlich keine Evolutionären Algorithmen. Allerdings sind sie eine besondere Form von binären, regelbasierten Klassifizierungssystemen, die als Lernstufe einen Genetischen Algorithmus benutzen. Sie sind in der Umgebung von Genetischen Algorithmen entstanden und dadurch sehr eng mit dem GA als Lernsystem verbunden. Der dritte Bestandteil von CS ist neben der Regelbasis und dem GA eine Zielfunktion zur Bewertung von Regeln, die *Bucket-Brigade-Algorithmus* heißt. Sie bewertet diejenigen Regeln gut, die zu einer Klassifizierung einen hohen Beitrag leisten [Goldberg 89].

2.5.4 Simulated Annealing

Simulated Annealing (SA) [Kirkpatrick et al. 83, Cerny 85] lehnt sich an den physikalischen Prozeß der Abkühlung einer Schmelze zu einem Festkörper an. Grundlage ist die Beobachtung, daß sich mit kontrollierter, langsamer Abkühlung eines Stoffes sehr regelmäßige und geordnetere Kristallstrukturen der Moleküle ausbilden. Diese stellen einen energieminimierten Zustand dar. Mit sinkender Temperatur wird die Bewegungsfreiheit der Moleküle immer weiter eingeschränkt. Dementsprechend existiert bei Simulated Annealing ein Steuerungsparameter, der die Abkühlungsgeschwindigkeit bestimmt. Mit sinkender "Temperatur" (Energiezustand/Fitneßwert) sinkt auch die Schwelle, mit der auch schlechtere Lösungen akzeptiert werden (bessere werden immer akzeptiert). Die Abkühlung erfolgt somit nach einem vorgegebenen Abkühlungsplan, sie ist also nicht selbstadaptiv, sondern exogen gesteuert. Bei Simulated Annealing wird in der ursprünglichen Form auch nur ein Lösungskandidat optimiert.

Simulated Annealing zählte ursprünglich nicht zum Gebiet der Evolutionären Algorithmen, wurde aber später wegen seiner Ähnlichkeit zu den anderen Verfahren, was die Klasse der Anwendungen und den Algorithmus selbst betrifft, dazugezählt. Außerdem ist es ebenso ein an einen natürlichen Vorgang angelehntes Verfahren.

Die Verfahren des *Threshold-Accepting* (TA) [Dueck et al. 90] und der *Sintflut-Algorithmus* (engl. *Great Deluge Algorithm*, GD) [Dueck 93, Dueck et al. 93] unterscheiden sich von Simulated Annealing im Hinblick auf die Akzeptanzbedingung. Bei Threshold-Accepting ist die Akzeptanzbedingung, daß die neue Lösung nur maximal um einen Schwellwert (threshold) schlechter sein darf, als die alte Lösung. Der Schwellwert wird im Laufe der Optimierung bis auf 0 abgesenkt. Beim Sintflut-Algorithmus ist die Akzeptanzbedingung noch weiter vereinfacht: die neue Lösung muß einfach besser sein, als ein globales Limit ("Wasserstand"), welches im Laufe der Optimierung immer weiter angehoben wird.

Nachdem in diesem Kapitel neben der detaillierten Darstellung von Evolutionsstrategien ein Überblick über das Gebiet der Evolutionären Algorithmen gegeben wurde,

wird im nächsten Kapitel auf die Parallelisierung dieser Verfahren eingegangen. Diese ist unabhängig von der gewählten Repräsentation der Individuen und den verwendeten Operatoren. Die Parallelisierung kann also auf alle Arten von Evolutionären Algorithmen gleichermaßen angewendet werden.

Kapitel 3

Parallelisierung Evolutionärer Algorithmen

Der Rechenzeitbedarf von Evolutionären Algorithmen kann je nach Problemstellung recht hoch sein. Dies liegt zum einen daran, daß eine Population von Lösungen verwendet wird und nicht nur eine einzige Lösung. Andererseits sind es iterative Verfahren, die gegebenenfalls eine große Anzahl von Schritten ausführen. Bei sogenannten *real-world* Anwendungen, also praktischen Anwendungen aus der Industrie, wird der Hauptteil der Rechenzeit von der Fitneßauswertung der Individuen beansprucht. Deshalb wird die Anzahl der Fitneßfunktionsauswertungen auch oft als ein vom verwendeten Rechner unabhängiges Maß zum Leistungsvergleich verschiedener Evolutionärer Algorithmen verwendet. Da einzelne Fitneßauswertungen unabhängig voneinander sind, liegt es nahe, durch parallele Berechnung die Gesamtrechenzeit zu verkürzen. Hierbei werden entweder tatsächlich nur die Fitneßberechnungen parallelisiert (wie in Abschnitt 3.2.3 erläutert) oder zusätzlich andere Teile des Algorithmus, wie Rekombination, Mutation und Selektion (wie beim Gittermodell, Abschnitt 3.2.2). Dies kann soweit gehen, daß schließlich mehrere (kleinere) Instanzen eines EA parallel laufen (Inselmodell, Abschnitt 3.2.1), die nur noch sehr lose zusammenhängen.

In diesem Kapitel sollen zunächst verschiedene Arten von Parallelrechnern anhand ihrer Klassifikationen vorgestellt werden. Danach folgen dann in Abschnitt 3.2 Beschreibungen der drei Hauptmodelle, Evolutionäre Algorithmen zu parallelisieren. Für diese Arbeit ist das Modell der parallelen Fitneßevaluation von besonderem Interesse. Wenn es zum Einsatz kommt, wird es fast immer mit einem Steady-State-Algorithmus kombiniert. Diese werden im Kapitel 5 behandelt. Besonders bei der Meta-Optimierung (Kapitel 6) ist es sinnvoll, dieses Parallelisierungsmodell einzusetzen, da hier ein hoher Rechenaufwand auftritt.

3.1 Klassifikation von Parallelrechnern

Für Rechnerarchitekturen existieren verschiedene Klassifikationen. Eine sehr bekannte, die Rechner in vier Klassen einteilt, ist die Klassifikation nach Flynn [Flynn 66]. Diese ist im Hinblick auf heutige Rechnerarchitekturen nicht mehr sehr passend. Sie soll an dieser Stelle aber dennoch aufgeführt werden, da die Parallelmodelle der Evolutionären Algorithmen aus diesen Rechnermodellen hervorgegangen sind.

Flynn unterscheidet Rechner nach den folgenden beiden Kriterien:

- ein (*single*) oder mehrere (*multiple*) Befehlsströme (*instruction stream*)
- ein (*single*) oder mehrere (*multiple*) Datenströme (*data stream*)

Daraus ergeben sich die folgenden Abkürzungen für die Rechnerklassen:

SISD: Single Instruction stream, Single Data stream.
(klassisches System mit einer zentralen CPU, “von-Neumann-Rechner”)

SIMD: Single Instruction stream, Multiple Data streams.
(Feld- oder Vektorrechner, systolische Arrays)

MISD: Multiple Instruction streams, Single Data stream.
(Pipelinerechner)

MIMD: Multiple Instruction streams, Multiple Data streams.
(Multiprozessorsysteme/Multicomputer mit mehreren unabhängigen Prozessoren)

Die Klasse SISD wird von klassischen Einprozessor-Rechnern dargestellt. Sie können ein Programm ausführen (single instruction stream), welches einen Datenstrom verarbeitet (single data stream). MIMD-Rechner sind durch ein Kommunikationsnetzwerk verbundene Prozessoren, die vom selben Typ wie diejenigen in SISD-Rechnern sind. Jeder Prozessor hat einen eigenen, lokalen Speicher zur Verfügung, so daß es sich um einen Verbund unabhängiger Rechner handelt. Jeder Rechner kann ein anderes Programm ausführen (multiple instruction stream) und durch den lokalen Speicher auf seinen eigenen Daten arbeiten (multiple data streams). MISD ist eine eher theoretische Kombination, wobei hier das interne Verarbeitungsmodell von Prozessoren darunterfällt, die eine Befehlspipeline mit mehreren Verarbeitungsstufen besitzen. SIMD-Rechner bestehen aus einer großen Menge von relativ einfach gebauten Prozessoren, wegen ihrer Einschränkungen auch Prozessor-Elemente (PEs) genannt. Ein zentrales Steuerwerk verarbeitet einen Befehlsstrom (single instruction stream) und alle PEs führen denselben Befehl auf ihren lokalen Daten aus (multiple data stream).

PEs können aber auch inaktiv gesetzt werden, um eine bedingte Ausführung von Anweisungen zu erreichen, wie z. B. bei if-then-else Konstrukten. Einschränkungen der PEs sind meist im Bereich der Bitbreite der Register, des lokalen zugreifbaren Speichers oder der arithmetischen Einheiten (z. B. keine Fließkomma-Einheit) vorhanden. Dies liegt in der meist recht großen Anzahl der PEs begründet (ca. 2^8 bis 2^{14}).

Der Begriff "massiv parallel" wird für Systeme verwendet, die einfach "relativ viele" Prozessoren (z. B. >128) besitzen. Früher waren dies ausschließlich SIMD-Systeme, heute sind dies große MIMD-Systeme. Deren Größe ist inzwischen bei einigen tausend Prozessoren angelangt. Massiv parallele Systeme vom SIMD-Typ sind inzwischen fast gänzlich vom Markt verschwunden.

Die Klassifikation nach Flynn ist relativ grob, und durch moderne Entwicklungen sind die Grenzen zwischen den Systemen inzwischen nicht mehr scharf zu ziehen. PCs und Workstations sind eigentlich vom Typ SISD, allerdings besitzen heutzutage alle Prozessoren eine interne Befehlspipeline (MISD) und es sind starke Entwicklungstendenzen sowohl in Richtung SIMD, als auch in Richtung MIMD zu erkennen: viele moderne Prozessoren besitzen Recheneinheiten, die mehrere gleichartige Befehle nach SIMD-Art parallel verarbeiten können, wie z. B. die MMX-Erweiterung (Multimedia-Extensions) von Intel Pentium Prozessoren (hauptsächlich zur Ganzzahlverarbeitung), ISSE (Intel Streaming SIMD Extension) von Intel Pentium III (hier auch viele Floatingpoint-Befehle), AltiVec im Motorola PowerPC Prozessor, VIS (Visual Instruction Set) in Sun's UltraSPARC Prozessoren[1], usw. Andererseits gibt es Systeme mit 2 oder 4 Prozessoren auf einer Platine, was bereits eine MIMD-Architektur ist. Diese Mehrprozessorrechner oder auch Einprozessorrechner lassen sich durch die inzwischen schon zum Standard gewordene Vernetzung mit immer höher werdender Bandbreite leicht als Prozessorcluster nach MIMD-Art nutzen. Allerdings dürfen hier die Ansprüche an die Kommunikationsbandbreiten nicht so hoch gesetzt werden wie in wirklichen MIMD-Systemen, da dort meist Vernetzungstopologien eingesetzt werden, die die gleichzeitige, breitbandige Nutzung von vielen Prozessoren zulassen. Dies ist in LANs (Local Area Networks) meist nicht gegeben.

Beispiele für SIMD-Systeme waren die MasPar MP-1 und MP-2 mit 16384 Prozessor-Elementen, die Connection Machine CM-2 und die Adaptive Solutions CNAPS, mit bis zu 512 PEs (speziell auf die Simulation Neuronaler Netze ausgerichtet). Diese Spezialrechner konnten sich nicht auf Dauer durchsetzen, da sich ihre hohen Entwicklungskosten deutlich im Preis niederschlugen, aber der Markt aufgrund des engen Einsatzgebietes nicht sehr groß war. Außerdem stoßen sie relativ schnell an ihre Grenzen, was die lokale Speichergröße [Evo95] und die Effizienz bei Parallelisierung algorithmisch komplizierter Berechnungen angeht.

Bei SISD-Prozessoren dagegen hat der große Markt eine relativ schnelle Weiterent-

[1] Die Namensgebung ist vom Marketing geprägt und zielt verstärkt auf den Bereich Multimedia ab, also 2D-, 3D-Grafik, Soundverarbeitung usw., da dies die Hauptanwendungsgebiete der Erweiterungen in Desktopsystemen sind.

wicklung sichergestellt, so daß diese nach kurzer Zeit ein deutlich besseres Preis/Leistungsverhältnis aufweisen konnten. Heute sind praktisch alle sogenannten "Supercomputer"[2] MIMD-Rechner, die entweder mit Kommunikationsnetzwerken oder mit gemeinsamem Speicher (shared memory) ausgestattet sind. Außerdem werden dabei gerne Standard-Prozessoren genutzt, da hier die rasche Weiterentwicklung relativ sicher ist und die Preise niedrig sind.

Beispiele für MIMD-Rechner sind die Cray T3E-900 (512 Prozessoren, Verbindungstopologie: dreidimensionaler Torus), NEC SX-4/32 (32 Prozessoren, Shared Memory mit Crossbar) und HP V2200 (16 Prozessoren, Shared Memory).

Aktuellere Klassifikationsschema erlauben es, MIMD-Rechner feiner zu unterscheiden. Dabei werden auf Kriterien wie z. B. Verbindungstopologie, Netzwerktyp und Speicherorganisation (gemeinsamer/verteilter Speicher) eingegangen.

3.2 Parallelmodelle Evolutionärer Algorithmen

Im folgenden werden die drei am häufigsten verwendeten Parallelmodelle Evolutionärer Algorithmen vorgestellt. Diese stellen meist keine funktional äquivalente parallele Implementierung des entsprechenden sequentiellen Algorithmus dar, sondern bringen jeweils Modifikationen mit, die es erlauben, die Parallelisierung effizienter durchzuführen. Somit sind die parallelen Algorithmen nicht direkt mit der sequentiellen Form vergleichbar. Dies wird oft bei Vergleichen vernachlässigt, und es ist teilweise auch der Grund dafür, daß sogenannter "superlinearer" Speedup beobachtet wird. Portiert man den modifizierten parallelen Algorithmus zurück auf einen Einprozessor-Rechner, so werden sich dort die Modifikationen ebenfalls auf das Laufzeitverhalten des Algorithmus auswirken.

Die Modifikationen sind aber nicht nur wegen der einfacheren Parallelisierbarkeit vorhanden. Man verspricht sich auch einen zusätzlichen Gewinn davon, wie z. B. bessere globale Sucheigenschaften oder Erhaltung von Diversität. Im Nachfolgenden werden bei jedem vorgestellten Modell die Modifikationen in Bezug zum sequentiellen EA deutlich hervorgehoben.

Grundsätzlich wird Evolutionären Algorithmen oft eine "inhärente" Parallelität zugesprochen, da sie naturanaloge Verfahren sind und deren Vorbild in der Natur auch ein hochgradig paralleles System darstellt. Die Modelle versuchen die Parallelität auf verschiedenen Ebenen auszunutzen.

[2] Eine Liste der Top 500 Supercomputer ist unter `http://www.netlib.org/benchmark/top500.html` zu finden.

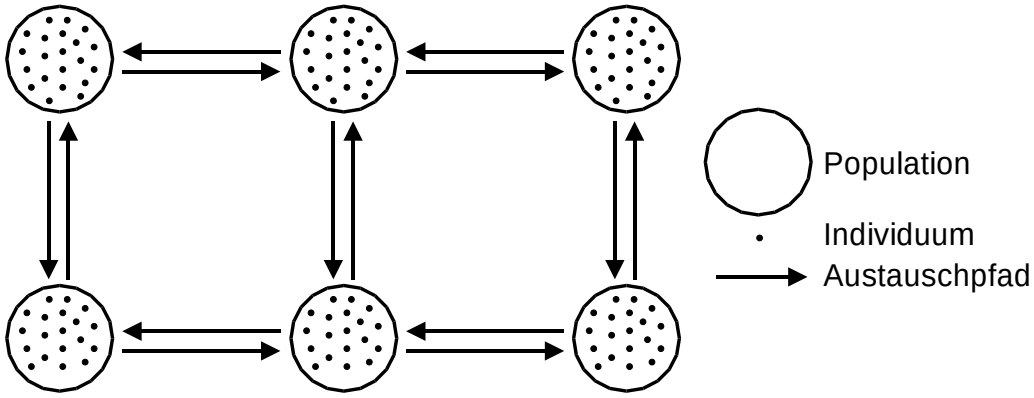


Abbildung 3.1: Das Inselmodell mit Individuenaustausch, hier: Gittertopologie.

3.2.1 Inselmodell

Für die Implementierung auf MIMD-Rechnern ist das Inselmodell sehr verbreitet. Dabei wird einfach auf jedem Prozessor eine unabhängige Instanz eines EA ausgeführt und zusätzlich noch ein Austausch von Individuen zwischen den Prozessoren eingeführt. Dieses Modell wird oft auch durch den Begriff *grobkörnige Parallelität* charakterisiert.

Das Inselmodell ist in Abbildung 3.1 dargestellt. Dies sind die algorithmischen Modifikationen zum sequentiellen Standard-EA:

- statt einer, jetzt mehrere, unabhängige Instanzen des EA,
- in regelmäßigen Abständen Austausch von Individuen zwischen den Inseln. Im Vergleich zum sequentiellen EA verändert sich dadurch die Population auf einer Insel.

Betrachtet man das Modell im Detail, muß man sich über die folgenden Punkte Gedanken machen:

Austauschtopologie: für jede Population muß festgelegt werden: zu welcher anderen Population werden Individuen geschickt?
Dies wird über eine Nachbarschaftstopologie festgelegt, wobei eine Population nur mit ihren unmittelbaren Nachbarn kommuniziert. Die Topologie ist fast immer ungerichtet bzw. bidirektional und statisch. Es bietet sich z. B. an, die schon vorhandene Kommunikationstopologie des Rechners auch für die Nachbarschaft der Populationen zu nutzen. Andererseits hat die Austauschtopologie Einfluß auf das Verhalten des Algorithmus und deshalb sollte eine vorteilhafte Austauschtopologie unter Vernachlässigung der Rechner-Kommunikationstopologie

benutzt werden. Denn die Kommunikationskosten sind oft vernachlässigbar klein: die dominanten Kosten sind meist die Funktionsauswertungen. Die übertragene Datenmenge für Individuen ist außerdem in der Regel sehr gering. Sie besteht aus einer Menge von Objekt- und Strategievariablen, Bitstrings oder abstrakten (meist kompakten) Darstellungen von Lösungen.
Verbreitete Topologien sind z. B. Gitter (Grid), Torus, Ring, Hypercube oder verschiedene Baum-Topologien [Bräunl 93]. Die Topologien beeinflussen hauptsächlich die Geschwindigkeit, mit der sich die ausgetauschten Individuen über die gesamte Population verbreiten können.

Austauschintervall: nach wie vielen Generationen werden Individuen ausgetauscht? Dies legt die Dauer fest, wie lange eine Population sich isoliert von den anderen entwickeln kann und wird daher auch Isolationsintervall oder Epoche genannt. Zu häufiger Austausch zerstört den Effekt von Isolation. Eine zu lange Isolation dagegen (im Extremfall gar kein Austausch) kommt der Ausführung von mehreren unabhängigen Läufen gleich (Multistart). Es sollte also ein Kompromiß gefunden werden.

Austauschrate: die Austauschrate legt die Anzahl der Individuen fest, die ausgetauscht werden (absolut oder relativ zur Populationsgröße).
Ist sie zu klein, so findet kaum Austausch statt. Der positive Effekt, den man sich davon verspricht, kann also nicht allzu groß sein. Ist die Austauschrate zu groß, so kann eine Population zu stark von externen Individuen überschwemmt werden, so daß die lokalen Individuen verschwinden können. Auch hier sollte also ein Kompromiß gefunden werden.

Selektion der auszutauschenden Individuen: bestimmt die zu versendenden Individuen.
Üblicherweise werden die besten versendet.

Integration: wie werden die von anderen Populationen ankommenden Individuen in die eigene Population integriert?
Ein übliches Verfahren ist, die schlechtesten Individuen durch die neuen zu ersetzen. Es können aber auch normale Selektionsmethoden oder die bei Steady-State-Algorithmen üblichen Integrationsmethoden verwendet werden, siehe Abschnitt 5.3 im Kapitel über parallele Steady-State-ES. Insbesondere die Median-Selektion läßt sich, wenn auf einem Prozessor schon ein Steady-State-Algorithmus verwendet wird, sowohl für die Selektion von lokal erzeugten als auch von extern ankommenden Individuen verwenden. Vor allem werden nicht einfach alle externen Individuen akzeptiert, wie dies bei manchen Verfahren der Fall ist.

Kopie oder Migration: soll nur eine Kopie eines Individuums versendet werden oder verschwindet das Individuum aus der lokalen Population, wenn es versendet wird (Migration)? Standard ist das Kopieren.

Eine typische Implementierung des Inselmodells bei einem GA und einer ES stellen die Programme VEGA [Baumann 95] und VEES [Wakunda 95] des EvA-Paketes dar. Austauschtopologie, -intervall und -rate sind dabei frei wählbar. Versendet werden nur die besten Individuen, welche dann die schlechtesten in der Zielpopulation ersetzen. Es werden hierbei Kopien versendet, es findet keine Migration statt.

Der Hauptvorteil, den man sich von dieser Parallelisierung verspricht, ist daß die Populationen durch die Isolation unterschiedliche Regionen des Lösungsraumes eine zeitlang ungestört durchsuchen können, ohne daß sie zu schnell von momentan besseren Lösungen dominiert werden, welche vielleicht nur zu einem lokalen Optimum führen. Der Individuenaustausch soll dann aber doch wieder dazu führen, daß gute Lösungen auch in Populationen eingebracht werden, die bisher nicht sehr erfolgreich waren. Dies muß allerdings nicht immer positive Auswirkungen haben. Es kann auch vorkommen, daß die Rekombination von zwei lokal sehr guten Individuen zu einem sehr schlechten Individuum führt. Man spricht dann vom sog. *Mule-Effekt*.

Desweiteren kann man entweder mehr Individuen in derselben Zeit berechnen und somit den Lösungsraum breiter durchsuchen, oder die Populationen auf den Inseln werden kleiner gemacht als eine einzelne Population, womit die Rechenzeit insgesamt verkürzt wird. Allerdings ist eine Population nicht in beliebig viele kleine Populationen aufteilbar. Außerdem kommt noch der Effekt hinzu, daß kleine Populationen mit größerer Wahrscheinlichkeit in lokalen Optima stagnieren.

In vielen wissenschaftlichen Arbeiten über parallele GAs wird von “near linear” oder sogar “superlinear” Speedup geschrieben [Tanese 97,Tanese 89,Belding 95]. In [CP97] wird der Verdacht geäußert, daß dabei meistens die Frage der Effizienz und die Frage der erreichten Qualität getrennt behandelt werden, d. h. bei den Läufen mit linearem Speedup wird oft nicht die gleiche Qualität erreicht wie beim entsprechenden sequentiellen GA.

3.2.2 Gittermodell

Für die Implementierung auf SIMD-Rechnern ist das Gittermodell gedacht. Auf einem Gitter (oder Torus, zyklisch durchverbunden) befindet sich in jedem Knotenpunkt (auf ein PE abgebildet) ein Individuum. Zwischen den Knoten besteht eine Nachbarschaftstopologie in Form eines Gitters mit horizontalen und vertikalen Verbindungen, manchmal kommen auch noch diagonale Verbindungen hinzu (X-Gitter). Das Suchen von Rekombinationspartnern und die Selektion findet in der lokalen Nachbarschaft statt, auf die in SIMD-Rechnern mit niedrigen Kommunikationskosten für alle PEs parallel zugegriffen werden kann.

Dies sind die algorithmischen Modifikationen zum sequentiellen Standard EA:

- Der oder die Rekombinationspartner werden nur in der lokalen Nachbarschaft, nicht global ausgewählt.

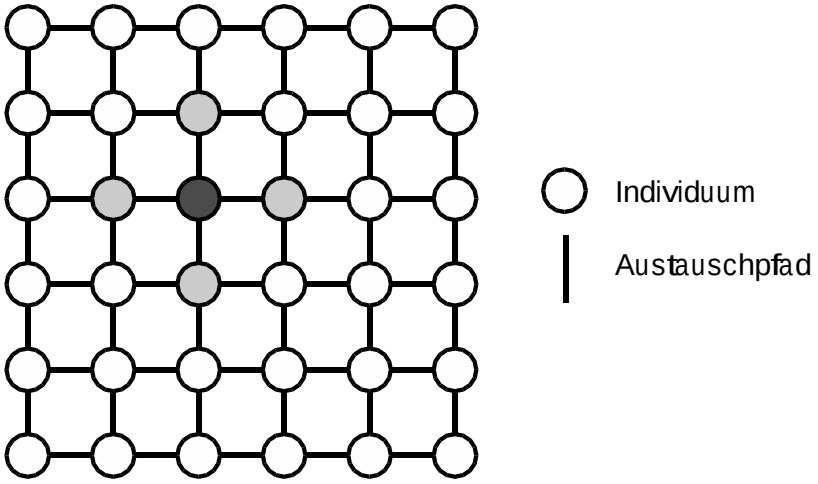


Abbildung 3.2: Das Gittermodell: auf jedem Gitterpunkt befindet sich ein Individuum. Die hellgrauen Individuen sind die direkten Gitter-Nachbarn des dunklen Individuums (hier keine Diagonalverbindungen/X-Gitter).

- Selektion findet auch nur lokal statt, meist eine Art Wettkampfselektion bzw. Ersetzungsstrategie für das lokale Individuum.

Dieses Parallelmodell wird auch Diffusionsmodell genannt und durch den Begriff *feinkörnige Parallelität* charakterisiert. Diffusion deshalb, weil gute Individuen oder zumindest Teile ihrer Objektvariablen sich mit der Zeit (da durch den Selektionsprozeß bevorzugt) in ihrer Nachbarschaft ähnlich der Diffusion eines Gases ausbreiten. In einigen Arbeiten wird auch der Begriff *Zellulärer EA* verwendet.

Betrachtet man das Modell im Detail, muß man sich über die folgenden Punkte Gedanken machen:

Nachbarschaftstopologie: welche Individuen sind direkt miteinander verbunden?
Es können beispielsweise bei zweidimensionaler Anordnung nur horizontale und vertikale Verbindungen zugelassen werden (Gitter) oder zusätzlich noch diagonale Verbindungen. Die Topologie kann zyklisch durchverbunden sein (Torus) oder nicht.

Nachbarschaftsradius: welche Individuen zählen zur Nachbarschaft?
Der Nachbarschaftsradius gibt hierbei die maximale Anzahl der Schritte an, die zum Erreichen eines Knotens über die Nachbarschaftstopologie benötigt werden darf, damit es zur Nachbarschaft gezählt wird.

Partner-Selektion: welche Individuen werden rekombiniert?

Es kann zunächst zwischen zwei Methoden unterschieden werden: wird das Ausgangsindividuum (im Zentrum der Nachbarschaft) grundsätzlich rekombiniert, oder werden alle Rekombinationspartner aus der Nachbarschaft ausgewählt?

Die Auswahl der Selektionspartner kann dann z. B. fitneßproportional (roulette wheel), rangbasiert oder auch aus einer lokalen Wettkampfselektion erfolgen. Gerne wird auch ein sogenannter *random walk* benutzt. Hierbei werden die besten Individuen auf einem zufälligen Pfad mit einer maximalen Länge verwendet. Der Pfad kommt zustande, indem auf jedem Knoten erneut zufällig eine Richtung ausgewählt wird.

Ersetzungsbedingung: wird das Individuum im Zentrum der Nachbarschaft durch das neu erzeugte Individuum ersetzt oder nicht?

Hierzu können auch wieder die im Abschnitt über Steady-State erläuterten Ersetzungsbedingungen zum Einsatz kommen: Ersetzung immer (hier auch oft generationelle Ersetzung genannt), Ersetzung nur durch bessere Individuen (elitär) oder andere Methoden.

Es gibt einige Arbeiten, die sich mit dem Zusammenhang zwischen Nachbarschaftsradius bzw. der Nachbarschaftstopologie und der Selektion beschäftigen, z. B. [Sarma et al. 96] und [Alba et al. 00]. Als Ergebnis können Aussagen über den Selektionsdruck gemacht werden, die es erlauben, bezüglich des Selektionsdrucks gleichwertige Topologien und Radien zu wählen, die aber weniger Kommunikationsaufwand erfordern.

Im Unterschied zum Insel-Modell gibt es kein Austauschintervall, es wird in jedem Schritt kommuniziert und auf Nachbarn zugegriffen. Betrachtet man das Austauschintervall als zeitliche Isolation, kann man beim Gittermodell von einer räumlichen Isolation sprechen. Diese führt ebenso dazu, daß schlechtere Lösungen, die aber später evtl. noch zum globalen Optimum führen können, nicht sofort durch zeitweilig bessere Lösungen dominiert werden.

Eine typische Implementierung des Gittermodells eines GA und einer ES stellen die Programme MPGA (Massiv Parallele Genetische Algorithmen) [Hummler 95] und MPES (Massiv Parallele Evolutionsstrategien) [Görzig 95] des EvA-Paketes dar (siehe Kapitel 4).

3.2.3 Parallele Fitneßevaluation

Für Problemstellungen, bei denen die Bewertung einer einzigen Lösung sehr lange dauert und die Ausführungszeit des eigentlichen Algorithmus (Mutation, Rekombination, Selektion, Verwaltung von Populationen usw.) dagegen vernachlässigbar klein ist, eignet sich das Modell der parallelen Fitneßevaluation, auch Farming-Modell genannt. Dabei wird der Algorithmus des EA auf einem einzigen zentralen Rechner

ausgeführt, und nur die Berechnung der Fitneßfunktion wird für mehrere Individuen parallel auf die restlichen Rechner verteilt. Sobald ein Rechner ein Ergebnis zurückliefert, wird ihm ein neues Individuum geschickt, dessen Fitneß er berechnen soll.

Dieser Ansatz läßt sich in zwei Varianten implementieren:

- synchron
- asynchron mit überlappenden Generationen

Die synchrone Implementierung bedeutet, daß der Original-Algorithmus verwendet wird mit der einzigen Modifikation, daß die Individuenevaluation auf andere Prozessoren verteilt wird. Dies ändert aber die Semantik des Algorithmus nicht. Dadurch bleibt der Algorithmus direkt mit der sequentiellen Version vergleichbar. Allerdings ist dann nicht immer eine hundertprozentige Auslastung aller Rechner erreichbar. Gegen Ende einer Generation, wenn die letzten Individuen evaluiert werden, kann es vorkommen, daß viele Prozessoren nicht genutzt werden, denn neue Individuen können erst wieder generiert und evaluiert werden, wenn die vorige Generation komplett ist und die Selektion stattgefunden hat (Synchronisationspunkt). Die Effizienz hierbei bzw. der Effizienzverlust hängt von verschiedenen Faktoren ab:

- dem Verhältnis von λ zur Anzahl p der zur Verfügung stehenden Prozessoren:
  je größer λ gegenüber p ist, desto weniger fallen Leerlaufzeiten vor der Synchronisation ins Gewicht.
- der Rest der bei Division von λ durch die Anzahl p der zur Verfügung stehenden Prozessoren entsteht:
  bei konstanten Evaluationszeiten bestimmt diese Zahl, wieviele Prozessoren vor der Synchronisation leer laufen.
- der Varianz in den Evaluationszeiten:
  variieren die Evaluationszeiten stark, werden vor der Synchronisation immer Leerlaufzeiten vorhanden sein, auch wenn λ ein Vielfaches von p ist.

Eventuelle Leerlaufzeiten werden beseitigt, wenn man überlappende Generationen zuläßt und somit auf die Synchronisation am Ende jeder Generation verzichtet. Dann werden zwar einige Individuen der nächsten Generation noch auf Basis der Elternindividuen der aktuellen Generation erzeugt, aber die Hoffnung dabei ist, daß die schnellere Beendigung einer Generation die evtl. verminderte Fortschrittsgeschwindigkeit ausgleicht. Die Auswirkungen dieser Modifikation hängen ebenfalls von den oben genannten Faktoren ab.

Besonders beliebt für die asynchrone Implementierung sind die sogenannten Steady-State-Algorithmen (s. Abschnitt 5). Hierbei gibt es keine Generationen mehr, in jedem Schritt wird nur ein einzelnes Individuum erzeugt und nach der Evaluation sofort

wieder in die Elternpopulation integriert. Somit ist nur noch die asynchrone Variante überhaupt sinnvoll mit dieser Methode parallelisierbar, da ansonsten immer nur ein Prozessor beschäftigt wäre.

Ebenso wie das Inselmodell ist das Modell der parallelen Fitneßevaluation speziell für die Implementierung auf MIMD-Rechnern gedacht. Es eignet sich sogar noch besser für ein heterogenes Rechnernetz als das Inselmodell, da eine Synchronisation, wie z. B. beim Individuenaustausch, nicht unbedingt nötig ist und somit auch unterschiedlich leistungsfähige Rechner eingesetzt werden können. Parallele Fitneßevaluation eignet sich allerdings kaum zur Implementierung auf SIMD-Systemen, zumindest nicht, wenn eine speicherintensive (PE Speicherplatzbeschränkung) oder algorithmisch komplexe Auswertungsfunktion verwendet werden soll (parallele Ausführung unterschiedlicher Programmzweige nicht möglich - sequentielle Ausführung ineffizient).

3.2.4 Entkopplung von der Rechnerarchitektur

Die in den letzten Abschnitten erläuterten Parallelmodelle für Evolutionäre Algorithmen sind oder waren zumindest ursprünglich für die Implementierung auf einem bestimmten Parallelrechnermodell gedacht. [Schwehm 97] schlägt dagegen vor, das Parallelmodell des Algorithmus vom Parallelmodell des Rechners, auf dem er implementiert ist, zu trennen.

Die Parallelmodelle der Algorithmen heißen dann:

- globales Modell (wie beim sequentiellen EA)
- regionales Modell (entspricht Inselmodell)
- lokales Modell (entspricht Gittermodell)

Durch die Trennung gewinnt man neuartige Kombinationen und kann so die potentiellen Vorteile des Parallelmodells eines Algorithmus nutzen, auch wenn das Parallelmodell des Rechners ein anderes ist, z. B. wenn ein Rechner dieses Typs gar nicht verfügbar ist. Auch die Vermischung und somit die Nutzung verschiedener Parallelmodelle der Algorithmen gleichzeitig ist denkbar. So könnte man z. B. ein regionales Modell wie oben erläutert auf einem MIMD-Rechner implementieren (Inselmodell) aber zusätzlich jede der Populationen gemäß dem lokalen Modell realisieren, um so von dessen Eigenschaften zu profitieren.

Von den hier im Überblick vorgestellten Parallelisierungsmethoden ist für diese Arbeit besonders das Modell der parallelen Fitneßevaluation interessant, da diese bei rechenzeitintensiven praktischen Anwendungen fast unumgänglich ist. Dabei wird dann fast immer ein Steady-State-Algorithmus eingesetzt. Diese Art Algorithmus wird in Kapitel 5 behandelt. Zunächst soll aber das EvA-System vorgestellt werden, welches u. a. das Inselmodell und die parallele Fitneßevaluation implementiert.

Kapitel 4

Das EvA-System

Das EvA-System ist ein Softwarepaket, das verschiedene Implementierungen paralleler und sequentieller Evolutionärer Algorithmen (daher auch der Name) und eine graphische Benutzeroberfläche enthält [Wakunda et al. 97].

Die Entwurfsziele des EvA-Systems waren:

- effiziente parallele Implementierungen von Evolutionsstrategien und Genetischen Algorithmen auf SIMD und MIMD Parallelrechnern zum Leistungsvergleich,
- eine einheitliche graphische Benutzeroberfläche für alle Programme,
- stark parametrisierte Algorithmen, d. h. freie Wahl des Selektionsverfahrens, der Populationsgröße, Selbstadaptionsverfahren, usw.,
- Bereitstellung einer großen Menge von Standardbenchmarkfunktionen,
- einfache Einbindung neuer Fitneßfunktionen.

In Abbildung 4.1 sind orthogonal zu den Rechnerarchitekturen MIMD (verteilt) und SIMD (massiv parallel) die Implementierungen von GA und ES aufgeführt. Zusammengesetzt ergeben sich dann die Namen der vier ersten Module: MPGA, MPES, VEGA, VEES. Als weitere Implementierung von GAs auf einer massiv parallelen Rechnerarchitektur ist CNGA auf der Adaptive Solutions CNAPS entstanden. An einem neuen Verfahren, den *(Verteilten) Selbstorganisierenden Evolutionsstrategien* (VESE) [Huhse et al. 00], einer Variante einer Evolutionsstrategie, die Ideen aus den Selbstorganisierenden Karten[1] übernimmt, wird noch geforscht.

Die ersten Teile der EvA-Software sind ab 1994 am Institut für Parallele und Verteilte Höchstleistungsrechner (IPVR) der Universität Stuttgart entstanden. EvA wird seit

[1] SOM - self organizing maps - eine spezielle Form Neuronaler Netze.

		massiv parallel MP	verteilt VE
Genetische Algorithmen	GA	MPGA CNGA	VEGA
Evolutions-Strategien	ES	MPES	VEES
Selbstorganisierende Evolutionsstrategien	SE	-	VESE
User Interface for Evolutionary Algorithms (UIEA)			

Abbildung 4.1: Übersicht über die Bestandteile von EvA.

Ende 1995 am Wilhelm-Schickard-Institut für Informatik (WSI) der Universität Tübingen weiterentwickelt. Die MP-Module für die massiv parallelen Rechner MasPar MP-1 und Adaptive Solutions CNAPS, also MPGA, MPES, und CNGA, werden mangels verfügbarer Hardware nicht mehr weitergepflegt. Diese Rechner werden aus den im Abschnitt 3.1 genannten Gründen nicht mehr hergestellt.

Der Entwicklungsschwerpunkt liegt deshalb seither auf den verteilten Varianten. Die Parallelisierung wird durch Message-Passing nach dem MPI-Standard (Message Passing Interface) erreicht. Es existieren verschiedene (auch freie) Implementierungen des MPI-Standards in Form von Bibliotheken für Unix-Workstations und Windows-PCs (z. B. MPICH [MPIa] oder LAM [MPIb]). Auf allen modernen MIMD-Systemen - auch auf shared memory Systemen - wird heutzutage eine herstellereigene Implementierung von MPI oder eine Portierung einer freien Implementierung mitgeliefert.

Im folgenden Abschnitt 4.1 wird zuerst eine Bestandsaufnahme von im Internet erhältlicher freier EA-Software präsentiert. Auf dieser Vergleichsbasis wird dann in Abschnitt 4.2 auf die Eigenschaften die aus der Software-Architektur von EvA resultieren, eingegangen. Aspekte davon sind die Anbindung der graphischen Benutzeroberfläche (Abschnitt 4.2.2) und die Datenkommunikation zur online Ergebnisverarbeitung und Visualisierung (Abschnitt 4.2.3). Danach wird das Modul VEES für die verteilten Evolutionsstrategien beschrieben (Abschnitt 4.3), welches die in den nachfolgenden Kapiteln behandelten neuen Verfahren implementiert. Betont werden dabei die Aspekte, die bei einer Anwendung von Evolutionsstrategien im Vordergrund stehen: die verschiedenen ES-Varianten, die Möglichkeiten der Einbindung neuer Fitneßfunktionen und Behandlung von Randbedingungen.

4.1 EA-Software

Im Internet findet man viele verschiedene Software für Evolutionäre Algorithmen mit unterschiedlichen Zielrichtungen. Unter `ftp://www.aic.nrl.navy.mil/pub/`

galist/src/index.html und http://www.it.uom.gr/pdp/DigitalLib/EC/ec_soft.htm findet man sehr umfangreiche Auflistungen von EA-Software aller Art. Beim größten Teil davon handelt es sich um Programmier-Bibliotheken oder Klassen-Frameworks, die sich flexibel für alle Arten von Codierungen und Varianten von Algorithmen anpassen lassen. Um solch ein Software-System auf spezielle Problemstellungen anzupassen, ist es immer nötig, mehr oder weniger viel Programmierarbeit zu leisten. Dies liegt in der Natur der Algorithmen begründet: eine Optimierungsaufgabe definiert sich durch die Fitneßfunktion und diese läßt sich am einfachsten und effizientesten als Code in der Programmiersprache des Optimierungssystems formulieren und einbinden. Auch andere Teile des Algorithmus, wie z. B. die Lösungsrepräsentation (Codierung) und spezielle Operatoren für Mutation und Rekombination auf dieser Codierung, erfordern das Schreiben von Code. Dies ist bei anderen Gebieten des Soft Computing, wie z. B. den Neuronalen Netzen, anders geartet: hier kann eine Software bereits implementierte Modelle Neuronaler Netze ohne Veränderungen flexibel auf neue Problemstellungen anpassen, da sich hierbei nur die Struktur des Netzes (Anzahl Ein-/Ausgabeneuronen, Verbindungstopologie, Gewichte) ändert, was ohne Änderung des Codes passieren kann. Die Architektur der Grundbestandteile des NNs, die Neuronen und die Arbeitsweise des Netzes und des Lernverfahrens bleibt dabei ebenfalls unverändert. Auch eine Änderung der Vernetzung ist leicht handhabbar.

Aus diesen Gründen bietet sich eine baukastenartige Bibliothek zur Programmierung von EA-Software an. Die Flexibilität von objektorientierten Programmiersprachen (Klassen, Vererbung, Templates, Polymorphismus) ist hierfür geradezu prädestiniert und wird von der Mehrzahl der Frameworks und Programmbibliotheken verwendet. Allerdings erfordert das Einarbeiten in ein Klassenmodell und seine Interfaces doch einiges an Lernaufwand. Da solche Frameworks sehr auf die Programmierung abzielen, wird dabei oft der Aspekt der Benutzerschnittstelle offen gelassen. Auch der Aspekt der Einbindung einer Optimierungskomponente in ein bestehendes Softwaresystem wird nur selten berücksichtigt.

Die Programmiersprache Java qualifiziert sich also für EA-Software, da sie sowohl objektorientiert ist, sich aber auch zur Erstellung von graphischen Benutzerschnittstellen eignet. Allerdings haftet ihr immer noch der Ruf an, nicht effizient genug für rechenintensive Aufgaben zu sein. Gerade aber bei der Optimierung ist dies ein wesentlicher Aspekt.

Der Ansatz von EvA und speziell VEES versucht nun, einige Punkte zu adressieren, die speziell für die Anwendung in der Industrie von Wert sind. Bei Evolutionsstrategien ist die Codierung in reellwertigen Variablen vorgegeben und der Mutationsoperator (mit Selbstadaption) ist speziell auf diese Codierung zugeschnitten. Daher beschränkt sich hier die Anpassung auf eine neue Optimierungsaufgabe i. A. auf die Formulierung der Fitneßfunktion und Randbedingungen, was in VEES besonders einfach und flexibel zu gestalten versucht wurde. Für manche Problemstellungen ist es außerdem noch nötig, die Anfangspopulation mit sinnvollen Werten zu belegen. Welche Möglichkeiten hierfür in VEES gegeben sind, ist in den Abschnitten 4.3.2 und 4.3.3 erläutert.

Kriterium	Ergebnis
Anzahl Pakete	33 (19 + 14 Kleinprogramme)
verwendete Sprache	*ohne Kleinprogramme*: Java: 4; C++: 4; C: 11; andere: 5 *alle*: Java: 5; C++: 5; C: 18; andere: 5
Algorithmus	*alle*: EA: 7; GA: 24; ES 1; GP: 1
GUI	9 mit GUI, 10 ohne
Lizenz	GPL: 4, GPL mit Einschr.: 9, Shareware: 1, unbekannt: 5
Alter	aktuell: 6; mittel: 6; alt: 7

Tabelle 4.1: EA-Software im Internet. Falls nicht angegeben, beziehen sich die Zahlen nur auf die 19 Pakete ohne die Kleinprogramme.

Um eine Übersicht über die im Internet erhältliche Software zu EAs zu bekommen, wurde eine Recherche betrieben. Dabei wurden insgesamt 33 Softwarepakete gefunden. 19 davon wurden in die nähere Auswahl genommen und genauer untersucht, die restlichen 14 wurden als zu klein (vom Funktionsumfang), zu speziell oder wesentlich zu alt eingestuft oder waren in einer wenig gebräuchlichen Sprache implementiert bzw. waren die einzigen Vertreter einer Programmiersprache (Eiffel, Fortran, LISP, Matlab, Scilab). Rein kommerzielle Software wurde ebenfalls nicht untersucht. Es handelt sich größtenteils um “freie” Software, die als Lizenz der GNU GPL unterliegt [GPL]. Eine Übersicht der Ergebnisse im Detail ist in Anhang B zu finden.

Die meiste Software ist in der Sprache C implementiert, einige jedoch auch in einer moderneren, objektorientierten Sprache, wie Java oder C++. Nimmt man noch die 14 Kleinprogramme hinzu, verschiebt sich der Schwerpunkt noch weiter zur Sprache C. Berücksichtigt man jedoch das Alter und den Wartungszustand der Software, so kann man eine starke Korrelation zwischen jüngeren Softwarepaketen und den moderneren, objektorientierten Sprachen feststellen: alle neueren Pakete sind durchweg in Java oder C++ geschrieben. Dies liegt auch nahe, da ein Evolutionärer Algorithmus aus sehr vielen Objekten (Populationen, Individuen, Chromosomen, usw.) und Methoden (Rekombination, Mutation, Selektion, usw.) besteht. Mit abgeleiteten Klassen lassen sich leicht Modifikationen bei der Codierung und den Operatoren einbringen.

Da sich das Forschungsgebiet Evolutionäre Algorithmen gerade wachsender Beliebtheit erfreut und viele neue Erkenntnisse gewonnen werden, wurde auch das Alter und der Wartungszustand der Software als wichtiges Kriterium erachtet. Die Bewertungsstufen waren hier: *aktuell* - die letzte Version ist nicht älter als zwei Jahre und/oder es gibt eine aktive Mailingliste; *mittel* - die Software ist zwischen zwei und vier Jahren alt; *alt* - fünf Jahre und älter, es findet keine erkennbare Weiterentwicklung mehr statt.

Bei der Art der implementierten Algorithmen ist der GA deutlich am stärksten vertreten (24 von 33 Paketen). Dies liegt wohl daran, daß der GA in Amerika erfunden wurde, wo auch die größte Forschergemeinde residiert und sich deshalb großer Be-

liebtheit erfreut. Außerdem benutzen viele EAs mit problemangepaßten Modifikation, wie nicht-binärer Codierung und entsprechenden Operatoren, das algorithmische Grundgerüst des GA. 7 Pakete enthalten allgemeine Software zu EAs und beinhalten außer GAs oft auch Evolutionsstrategien (5). EP und GP ist eher selten vertreten.

9 der 19 Pakete haben ein graphisches Benutzerinterface (GUI - Graphical User Interface).

Nur vier Programme unterliegen uneingeschränkt der GNU General Public Licence (GPL). Die meisten anderen Pakete (9) sind zwar für private, Lehr- und Forschungszwecke frei verwendbar, modifizierbar und verbreitbar, verbieten aber z. B. den kommerziellen Einsatz ohne Zustimmung des Autors (6 von 9). Eine Einschränkung fordert lediglich ein Erwähnen der Software und des Autors in davon abgeleiteten Arbeiten und Dokumentationen, ein Paket verbietet militärische Nutzung und eines läßt eine Verbreitung von Änderungen nur als Patch und unter anderem Namen zu. Ein Java-Paket wird in Form von Bytecode als Shareware vertrieben, bei 5 Paketen war keine Information zur Lizenz vorhanden, es ist anzunehmen, daß sie zumindest für nicht-kommerziellen Einsatz frei sind.

Die Merkmale von EvA im Vergleich mit der gesichteten Software sind:

Sprache: EvA ist in der Sprache C (Teile auch in C++) geschrieben. Es ist geplant, den Kern nach C++ zu portieren.

Algorithmus: Genetische Algorithmen, Evolutionsstrategien und Selbstorganisierende Evolutionsstrategien. Dabei sind zahlreiche algorithmische Varianten implementiert. Als einziges der gesichteten Pakete enthält das Modul für Evolutionsstrategien die moderne Kovarianzmatrixadaption.
Als eines der wenigen Pakete sind in EvA alle Algorithmen auch für parallele Systeme implementiert.

GUI: EvA besitzt eine von den Algorithmen unabhängige, graphische Benutzeroberfläche, die mit OSF/Motif realisiert wurde. Dabei sind HTML-Hilfeseiten und Erklärungen zu den Parametern der Algorithmen vorhanden.

Lizenz: es ist beabsichtigt, EvA unter der GNU GPL zu vertreiben, wodurch auch der kommerzielle Einsatz erlaubt wäre.

Alter: EvA wird seit 1995 ständig weiterentwickelt. Neueste Forschungsergebnisse fließen in die Software ein.

Sonstiges: EvA besitzt weitere nützliche Eigenschaften. Diese werden in den folgenden Abschnitten erläutert:

- es ist Unterstützung für die Einbindung von EvA als Optimierungskomponente in ein komplexeres System vorhanden,

- es ist eine umfangreiche Bibliothek an Testfunktionen vorhanden,
- die Einbindung von neuen Fitneßfunktionen ist ohne Neucompilierung des Gesamtprogrammes möglich,
- es sind Programme für die online-Visualisierung des Traveling Salesman Problem und die Optimierung einer optischen Linse enthalten,
- umfangreiche Untersuchungen werden durch die Eingabesprache Tcl erleichtert, es ist ein Tcl-Erweiterungspaket zur Erstellung von Meßreihen enthalten,
- es ist ein Auswertungstool für die bei Optimierungsläufen entstehenden Protokolldateien vorhanden.

Es existiert keine andere freie Software, welche diesen Funktionsumfang in sich vereint.

4.2 Die Software-Architektur von EvA

Die Architektur von EvA ist in Abbildung 4.2 dargestellt. Ein Rahmen mit abgerundeten Ecken (durchgehende Linie) stellt eine funktionale Einheit innerhalb eines Programmes dar, mehrere funktionale Einheiten sind durch einen gestrichelten Rahmen zu einem Programm zusammengefaßt, dessen Namen oder Funktion angegeben ist. Ein einfacher Pfeil zwischen den Programmen zeigt einen Datenaustausch mittels unterschiedlichster Mechanismen an. Doppelpfeile zwischen funktionalen Einheiten stellen Datenaustausch über Funktionsaufrufe dar.

Die EvA-Software besteht im wesentlichen aus einer Menge von *Modulen* und einer einheitlichen graphischen Benutzeroberfläche, die nach einem Client-Server-Modell miteinander kommunizieren (siehe Abb. 4.2). Die Module implementieren jeweils spezielle Arten von Algorithmen, z. B. Evolutionsstrategien oder Genetische Algorithmen, die in Abbildung 4.1 dargestellt sind. Desweiteren können Programme zu Zwecken der Visualisierung oder ähnlichem, mit einem Modul kommunizieren.

Ein EvA-Modul ist ein Optimierungsalgorithmus, der mindestens die Benutzerschnittstelle und das Ausgabedateiformat von EvA unterstützt. Optional ist die Unterstützung von Kommunikationsmechanismen, die auf dem Data-Manager aufsetzen. Diese dienen dazu, während des Optimierungslaufs anfallende Daten, wie z. B. gefundene Lösungen, an externe Programme weiterzuleiten. Dies wird in Abschnitt 4.2.3 näher erläutert.

Im Rahmen dieser Arbeit sind die folgenden Teile von EvA entstanden:

- der Data-Manager mit den Schnittstellen Tcl-Data und RPC-Data,

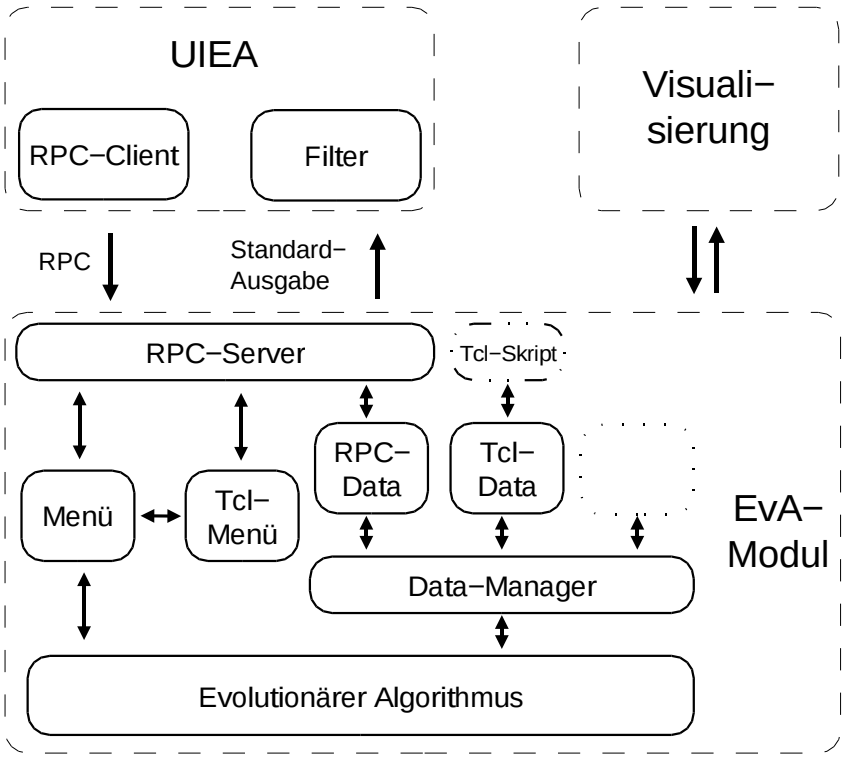


Abbildung 4.2: Architektur von EvA-Modulen und das Zusammenspiel mit der Benutzeroberfläche und Visualisierungskomponenten.

- darauf Aufbauende Visualisierungskomponenten,
- wesentliche Erweiterungen des Moduls für Evolutionsstrategien:
  - moderne Schrittweitenadaptionsverfahren: KSR, CMA, Differential Evolution,
  - parallele Fitneßevaluation,
  - Steady-State-Evolutionsstrategien,
  - die *applib* Applikationsbibliothek,
  - Methoden zur Constraint-Behandlung.

4.2.1 Die Modul-Schnittstelle

Die Bedienung eines Moduls ist auf eine Schnittstelle abstrahiert, die als *Menü* bezeichnet wird. Das Menü besteht aus einer Menge von Menüpunkten, die nach ihrer Funktion gruppiert sind. Ein Menüpunkt kann ein Datum von einem bestimmten Typ

aufnehmen (Integer, Liste, Dateiname, usw.) und wird durch einen Namen eindeutig identifiziert. Hinzu kommt noch ein abgekürzter Name, Informationen über den Wertebereich, ein Defaultwert und eine kurze Beschreibung. Zwischen Menüpunkten können Abhängigkeiten programmiert werden, so daß z. B. die Wahl der Rekombinationsart nur dann möglich ist, wenn die Anzahl der zu rekombinierenden Individuen mindestens 2 beträgt.

Die Erstellung des Menüs wird durch eine Bibliothek unterstützt. Sie enthält auch die RPC[2]-Server-Schnittstelle und die Umsetzung des Menüs in eine textuelle Benutzerschnittstelle. Letztere verarbeitet als Eingabe die interpretierte Skriptsprache Tcl [Ousterhout 95]. Für jeden Menüpunkt wird hier ein Tcl-Kommando generiert, mit welchem der Wert des Parameters verändert werden kann.

Wird das Modul direkt auf der Kommandozeile gestartet, stellt es das Menü textuell dar und präsentiert das Tcl-Kommandoprompt des integrierten Tcl-Interpreters. Alternativ kann durch Angabe eines Dateinamens auch eine Skriptdatei eingelesen und somit eine Batchverarbeitung erreicht werden. Durch die Eingabesprache Tcl ist es möglich, auch kompliziertere Messungen durch Programmierung der EvA-Module direkt durchzuführen. Im Gegensatz dazu muß bei vielen Programmen, die nur Kommandozeilenparameter verarbeiten, auf eine externe Programmiersprache (z. B. eine Kommandoshell) zurückgegriffen werden.

Durch die Abstraktion des Benutzerinterfaces eines Evolutionären Algorithmus auf ein Menü ist es möglich, dieses alternativ auch graphisch darzustellen. UIEA (User Interface for Evolutionary Algorithms) ist die Modul-unabhängige, einheitliche graphische Benutzeroberfläche für alle EvA-Module. Sie wird in Abschnitt 4.2.2 näher beschrieben.

Das einheitliche Ausgabedateiformat aller EvA-Module ist ein einfaches Format, das gut von Analysetools eingelesen und weiterverarbeitet werden kann. Es besteht aus zwei Blöcken. Im ersten Block sind Daten, die global für einen Lauf gelten und somit nur einmal in der Datei vorkommen. Der zweite Block enthält zeitabhängige Daten, die z. B. in jeder Generation neu anfallen. Dieses Format wird für alle drei von EvA-Modulen generierten Dateien verwendet:

Protokolldatei: pro Optimierungslauf wird eine Protokolldatei erstellt. Sie enthält alle Eingabeparameter, die nötig sind, um diesen Lauf zu reproduzieren. Desweiteren enthält sie eine Reihe von statistischen Daten, die während der Optimierung angefallen sind, wie z. B. die beste, durchschnittliche und schlechteste Fitneß und Zeitangaben. Diese Daten werden je nach angegebenem Statistik-Intervall (in Generationen) protokolliert.

Individuendatei: pro Optimierungslauf wird außerdem eine Individuendatei erstellt. Sie enthält zumindest das beste Individuum am Ende des Laufes.

[2] Remote Procedure Call - ein Protokoll zur entfernten Ausführung von Prozeduren auf einem Server.

Summary-Datei: in dieser Datei werden die Parameter mehrerer Optimierungsläufe protokolliert. Solange derselbe Dateiname verwendet wird, werden auch nach Beendigung eines Moduls bei einem späteren Neustart Daten zu weiteren Läufen angehängt.

4.2.2 Die graphische Benutzeroberfläche

Die graphische Benutzeroberfläche UIEA (User Interface for Evolutionary Algorithms) ist ein eigenständiges Programm, welches als separater Prozeß unabhängig von einem EvA-Modul läuft. Ein Modul läßt sich alleine im Batch- oder interaktiven Modus mit Textbedienung betreiben. Wird es aber von UIEA mit bestimmten Optionen gestartet, fungiert es als RPC-Server und UIEA greift als Client auf das Modul zu. Der Aufbau des Menüs wird ausgelesen, neue Werte nach den Benutzerangaben gesetzt und auf Klick der Algorithmus gestartet.

In UIEA werden die Menüpunkte auf Standardelemente der graphischen Benutzerschnittstelle, wie z. B. Eingabefelder, Auswahlboxen, usw. abgebildet. Jede Gruppe von Parametern erscheint in einem separaten Dialogfenster. Die graphische Oberfläche ist mit dem Toolkit OSF/Motif Version 1.2 programmiert[3].

Durch das Client-Server-Modell ist es auch möglich, daß die Oberfläche auf einer lokalen Workstation läuft, während z. B. ein Evolutionärer Algorithmus auf einem entfernten Parallelcomputer ausgeführt wird.

In Abbildung 4.3 ist die Oberfläche UIEA zu sehen. Oben links befindet sich das Hauptfenster mit dem Menü. In dessen Textbereich ist die Ausgabe des geladenen Moduls (hier VEES) zu sehen. Die vier Fenster mit den Eingabefeldern und Auswahlboxen in der Mitte oben und rechts unten, zeigen die Parameter der Evolutionsstrategie an. Beispielsweise sind im Fenster in der Mitte oben die Grundparameter der Evolutionsstrategie zu sehen: Elternindividuen μ, Anzahl Rekombinanten ρ, Nachkommenindividuen λ, Generationen γ, usw. Die Grafik oben rechts zeigt den Verlauf der besten, durchschnittlichen und schlechtesten Fitneß über die Zeit an. Unten links ist ein Visualisierungsprogramm für das Problem des Handlungsreisenden zu sehen, welches die aktuell kürzeste Rundreise anzeigt.

Desweiteren bietet UIEA einige Funktionen, welche zur Benutzerfreundlichkeit beitragen, wie das Laden und Speichern von Parametereinstellungen, das Wiederherstellen der offenen Fenster und ihrer Positionen aus der letzten Sitzung und einen Skript-Editor. Mit dem Editor können die aktuellen Einstellungen der Parameter-Fenster in ein Tcl-Skript übernommen werden. Skripte können dann zur Ausführung an das Modul gesendet werden.

[3]Version 2.1.30 wurde am 15. Mai 2000 von der *Open Group [OG]* unter dem Namen *Open Motif* unter einer Open Source Lizenz freigegeben. Damit kann es auf Open Source Betriebssystemen wie Linux oder FreeBSD unentgeltlich genutzt werden.

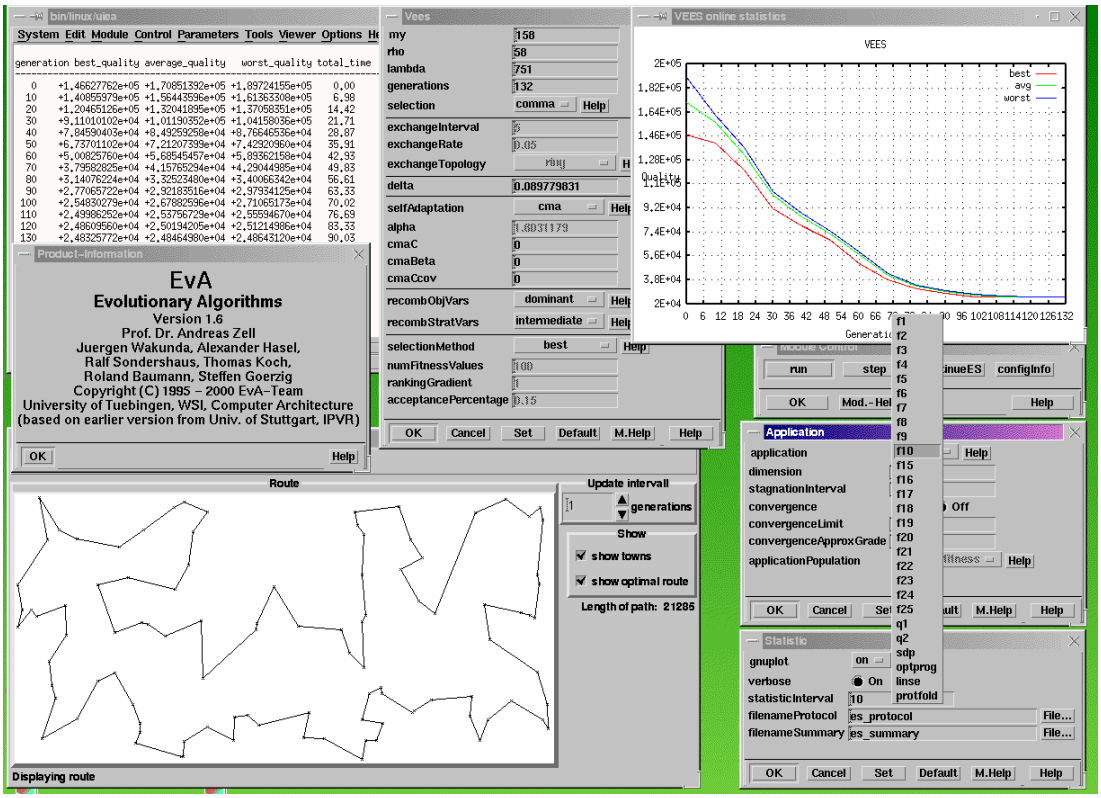


Abbildung 4.3: Die graphische Benutzeroberfläche UIEA.

4.2.3 Online Ergebnisverarbeitung und -visualisierung

In praktischen Anwendungen wird die Optimierung oft nur als eine Komponente in einem umfangreicheren System benutzt. Die naheliegende Möglichkeit ist es, das EvA-Modul im Batchmodus durch ein Skript mit den nötigen Eingaben zu versorgen. Nachdem die Optimierung fertig ist können die Ergebnisse aus den Protokoll- und Individuendateien ausgelesen werden. Diese Lösung ist oft nicht zufriedenstellend, da während des Batch-Laufes kein Feedback erfolgt. Bei Optimierungen in interaktiven Anwendungen ist es wünschenswert, z. B. die aktuell beste Lösung laufend zu visualisieren. Hierzu existiert in EvA eine Realisierung, die es erlaubt, Visualisierungsprogramme während der Optimierung mit den entsprechenden Daten zu versorgen.

Es wurde ein XML[4]-Format für die übertragenen Daten gewählt. Dadurch werden nur textuelle Daten übertragen, was den Vorteil hat, daß sich die Programmierschnittstellen nicht ändern, wenn neue Datentypen hinzugefügt werden. Ein Beispiel für die übertragenen Daten ist in Abbildung 4.4 gezeigt. Nicht jedes Visualisierungsprogramm empfängt allerdings das komplette XML-Dokument in einem Datenstrom. Stattdessen werden nur diejenigen Tags versendet, die das jeweilige Visualisierungsprogramm

[4]eXtensible Markup Language

```
<Optimization>
  <Generation number=''1''>
    <Individual type=''currentBest''>
      <Processor> 0 </Processor>
      <Fitness> 108.7 </Fitness>
      <Variables number=''3''>
        <Var> 1.0  </Var>
        <Var> 2.57 </Var>
        <Var> 3.8  </Var>
      </Variables>
    </Individual>
  </Generation>
  <Generation number=''2''>
  . . .
  </Generation>

  . . .

</Optimization>
```

Abbildung 4.4: Beispiel für die XML-Daten eines EvA-Moduls.

vor Start der Optimierung angefordert hat. Dies geschieht auch nur in einem ebenfalls vorher festgelegten Generationen-Intervall.

Bisher können z. B. vom Modul VEES die folgenden Informationen erzeugt werden:

- Beginn und Ende:
  - der Optimierung,
  - einer Generation.
- Individuen:
  - alle Eltern, alle Individuen nach der Rekombination, nach der Mutation und nach der Selektion (ist identisch mit Eltern der nächsten Generation),
  - das beste der aktuellen Generation,
  - das beste aller bisherigen Generationen.
- Fitneßwerte: beste, durchschnittliche und schlechteste Fitneß der aktuellen Generation.

- Für die Median-Selektion (s. Kapitel 5):
  - gemessene Rate der akzeptierten Individuen zu nicht akzeptierten Individuen,
  - gemessene Akzeptanzrate wie im vorigen Punkt, aber nur über die Anzahl an Generationen seit der letzten Abfrage dieses Wertes,
  - Inhalt des Fitneß-FIFO-Puffers.

Der Data-Manager (siehe Abbildung 4.2) fungiert als Schnittstelle zwischen dem Evolutionären Algorithmus und Kommunikationsmodulen, welche das Kommunikationsprotokoll bestimmen. Bisher sind Kommunikationsmodule für RPC und für Tcl implementiert. Weitere Module, z. B. für eine Anbindung über CORBA, sind denkbar (durch den gepunkteten Rahmen im EvA-Modul in Abb. 4.2 angedeutet). Der Data-Manager ist in der Lage, die anfallenden Daten an mehrere Visualisierungsprogramme weiterzuleiten.

Das Kommunikationsmodul Tcl-Data ermöglicht es, mit einem Skript zu kommunizieren, welches direkt im Tcl-Interpreter des EvA-Moduls abläuft (gestrichelter Rahmen im EvA-Modul in Abb. 4.2). Für jedes XML-TAG kann eine Callback-Funktionen eingerichtet werden, die dann während der Optimierung aufgerufen wird. Wird die Erweiterung Tk zur Programmierung von graphischen Oberflächen in den Tcl-Interpreter geladen, so kann dadurch direkt eine anwendungsspezifische Benutzeroberfläche und Visualisierung programmiert werden. Ein Beispiel hierfür ist die Linsenoptimierung mit VEES (Beschreibung der Anwendung siehe Kapitel 7).

4.3 VEES: Verteilte Evolutionsstrategien

Die folgenden herausragenden Eigenschaften von VEES machen es einfach und vielseitig einsetzbar, ohne daß sich ein Anwender Varianten des Kernalgorithmus oder aufwendige Datenflußmechanismen selbst programmieren müßte.

- viele verschiedene Algorithmen-Varianten (Abschnitt 4.3.1),
- einfaches und doch flexibles Einbinden neuer Fitneßfunktionen, auch ohne das ganze Programm neu zu übersetzen (Abschnitt 4.3.2),
- verschiedene Möglichkeiten, Randbedingungen zu implementieren (Abschnitt 4.3.3),
- leistungsfähige Skript-Programmiersprache Tcl,
- Event-gesteuerte Schnittstelle zur Online Datenvisualisierung und -weiterverarbeitung (Abschnitt 4.2.3),

- Meta-Optimierung, um gute, problemspezifische Parametereinstellungen zu bekommen.

4.3.1 ES-Varianten in VEES

In diesem Abschnitt werden alle Varianten von Evolutionsstrategien beschrieben, die das Programm VEES bisher bietet. VEES implementiert ein hochgradig parametrisiertes Modell einer Evolutionsstrategie, das durch Wahl der Parameter entsprechend angepaßt werden kann.

- Standard-Parameter:
  - Größe der Elternpopulation μ
  - Größe der Nachkommenpopulation λ
  - Anzahl Generationen γ
  - Anzahl der Individuen für die Rekombination ρ
  - Rekombinationsmethoden jeweils getrennt für Objekt- und Strategievariablen: dominant, intermediär, kontinuisiert (nur für Objektvariablen sinnvoll)
- Strategietyp:
  - Generationenbasierte Standardalgorithmen:
    - (μ, λ) “Komma”-Strategie
    - $(\mu + \lambda)$ “Plus”-Strategie
  - Steady-State-Algorithmen:
    - Standard:
      - Ersetzungsstrategie: Ersetzung des schlechtesten oder Ersetzung des ältesten
      - Ersetzungsbedingung: immer ersetzen oder nur durch ein besseres Individuum
    - Median-Selektion:
      - Länge des Fitneßpuffers n_p
      - Akzeptanzrate r_p
    - local Tournament Selection
  - Verteilte ES nach dem Inselmodell mit Individuenaustausch:
    - Austauschrate relativ zur Anzahl der Elternindividuen μ

        - Austauschintervall in Generationen
        - Austauschtopologie: Ring, Grid, X-Netz, Hypercube, voll verbunden.
    - Verteilte geschachtelte Evolutionsstrategie:
        - Anzahl Elternpopulationen μ' (die Anzahl der Nachkommenpopulationen λ' ist auf die Anzahl der verfügbaren Prozessoren festgelegt)
        - Anzahl der Populationen bei der Rekombination ρ'
    - parallele Fitneßevaluation

- Schrittweiten-Parameter:
    - relative Anfangsschrittweite δ (bezogen auf den Wertebereich jeder einzelnen Objektvariablen)
    - Schrittweitenadaptionsverfahren:
        - mutative Schrittweitenregelung (MSR) mit globaler Schrittweite
        - mutative Schrittweitenregelung mit unkorrelierten separaten Schrittweiten
        - Kumulierende Schrittweitenregelung (KSR)
        - Kovarianz-Matrix-Adaption (CMA)
        - Differential Evolution Mutation (DE)
        - ohne Adaption
    - Adaptionsfaktor α für die MSR-Verfahren

- Selektionsmethode für Eltern der nächsten Generation:
  Bestenselektion, Rouletterad-Selektion, Ranking-Selektion

- Behandlung von Rauschen nach Ostermeier (in [Rechenberg 94], S. 195) mit den Faktoren κ und k

4.3.2 Fitneßfunktionen in VEES

Wie bereits erwähnt, ist ein zentraler Punkt bei Evolutionären Algorithmen die Formulierung einer Fitneßfunktion. Die Kodierung ist bei Standard-Evolutionsstrategien bereits als Vektor reeller Zahlen vorgegeben, so daß hier keine Änderungen bei neuen Optimierungaufgaben nötig sind. Um die Fitneßfunktion sinnvoll einbinden zu können, sind aber noch einige andere Parameter wie z. B. Definitionsbereich der Variablen, Initialisierungsfunktionen, usw. notwendig. Eine Fitneßfunktion inklusive aller zusätzlichen Parameter und Hilfsfunktionen heißt in EvA *Applikation*. Welche Möglichkeiten VEES hier bietet, wird in den folgenden Abschnitten erläutert. Die Behandlung von Randbedingungen folgt im Abschnitt 4.3.3.

Alle Applikationen sind in EvA in einer Bibliothek namens *applib* zusammengefaßt. Dies hat den Vorteil, daß die *applib* auch in anderen Optimierungsmodulen verwendet werden kann. So wird sie z. B. auch von VESE genutzt. Die *applib* verwaltet die Applikationen auch insofern, daß sie Initialisierung und Finalisierung (siehe Abschnitt "Die Fitneßfunktion") selbständig durchführt, die Randbedingungen verwaltet und eine Menge von "Standard"-Randbedingungen beinhaltet, welche sie auch auf die Individuen anwenden kann.

Die Kapselung der Applikationen in einer Bibliothek hat auch den Vorteil, daß die *applib* nicht fest zum Optimierungsmodul hinzugebunden werden muß, sondern als sogenannte *shared library* dynamisch beim Programmstart hinzugebunden werden kann. Auf diese Weise kann ein Benutzer seine Anwendung auf einfache Art abändern und entwickeln, ohne daß das gesamte Optimierungsmodul mitübersetzt und gebunden werden muß.

Die *applib* bietet auch optionale Unterstützung der Menü-Bibliothek *ea_menu*, welche es Applikationen ermöglicht, mit wenigen Zeilen Code Menüpunkte zu erzeugen, um variable Parameter einzulesen. Beispielsweise kann es nötig sein, daß eine Applikation den Namen einer Datei benötigt, aus der Daten für die Optimierung Daten eingelesen werden. Diese Menüpunkte werden dann als Tcl-Befehle in der Textversion realisiert oder in der graphischen Oberfläche angezeigt.

Optional ist auch die Unterstützung der sogenannten Viewer-Schnittstelle (siehe Abschnitt 4.2.3), welche es ermöglicht, auf einfache Art und Weise Daten mit externen Programmen auszutauschen. Dies ist sehr nützlich, um das Optimierungsmodul in andere Programmsysteme einzubinden. Dadurch ist eine einfache Kommunikation der Applikation mit externen Programmen ohne Umweg über Dateien möglich.

4.3.2.1 Die Fitneßfunktion

Eine Applikation besteht zunächst aus drei Grund-Funktionen:

- Fitneßfunktion
  Sie bekommt als Eingabedaten den Vektor der reellwertigen Objektvariablen und die Dimension des Vektors. Desweiteren bekommt sie noch einen Zeiger auf den sog. *Container*, welcher in Individuen dazu dient, zusätzliche Informationen zu speichern, die über die Objektvariablen hinausgehen. Diese können somit auch in der Fitneßfunktion genutzt werden.

- Initialisierungsfunktion
  Diese Funktion wird nur einmal bei Aktivierung der Applikation aufgerufen und dient dazu, vorbereitende Aktionen auszuführen, die nicht bei jedem Aufruf der Fitneßfunktion passieren müssen, wie z. B. Speicherplatz anzufordern oder Dateien zu öffnen. Danach erst wird die Fitneßfunktion (mehrfach) für Evaluie-

rungen aufgerufen. Außerdem werden hier die Randbedingungen festgelegt und zur weiteren Behandlung an die *applib* weitergereicht.

- Finalisierungsfunktion
  Dies ist das Gegenstück zur Initialisierungsfunktion. Sie wird nur einmal aufgerufen, nachdem alle benötigten Evaluierungen der Fitneßfunktion gemacht wurden. Hier kann Speicherplatz wieder freigegeben werden oder Dateien geschlossen werden.

4.3.2.2 Wertebereich und Initialisierung

Die Initialisierungsfunktion muß einen Wertebereich für jede Variable festlegen. Dieser kann für die Initialisierung der Individuen und/oder für Randbedingungen verwendet werden. Im ersteren Fall werden alle Individuen mit gleichverteilten Zufallszahlen im Wertebereich initialisiert. Für komplexere Fälle kann auch eine eigene Funktion zur Initialisierung verwendet werden.

Für die Randbedingungen kann der Wertebereich verwendet werden, um Individuen zu verwerfen, die den Wertebereich verlassen oder um die Individuen mit verschiedenen Methoden wieder in den Wertebereich zurückzutransformieren, also sie zu reparieren. Dies wird im Abschnitt 4.3.3 über Randbedingungen genauer behandelt.

4.3.2.3 Zusätzliche Parameter

- Name der Applikation
  Jede Applikation hat einen eindeutigen, kurzen Namen, mit dem Sie aus der Liste der Applikationen ausgewählt werden kann.

- Minimierung/Maximierung, vorzeichenbehaftet oder absolut
  Jede Applikation legt fest, wie Fitneßwerte zu interpretieren sind. Wenn eine Maximierung stattfinden soll, stellen größere Fitneßwerte eine Verbesserung dar, soll dagegen eine Minimierung stattfinden, sind kleinere Werte besser. Desweiteren wird festgelegt, ob Fitneßwerte vorzeichenbehaftet oder vorzeichenlos betrachtet werden. Dies hat Auswirkungen auf den Wert der besten und schlechtesten möglichen Fitneß.

- Dimensionsbereich und Defaultwert der Dimension
  Eine Anwendung kann angeben, mit wievielen Variablen sie umgehen kann. Dazu wird ein Intervall festgelegt, in dem die Dimension des Problems liegen muß und einen Standardwert. Beispielsweise kann die Applikation f_{10}, eine Instanz des TSP, mit genau 100 Variablen umgehen, da diese Probleminstanz 100 Städte hat. Die Funktion f_2 (Generalized Rosenbrock's Function) muß dagegen mindestens 2 oder mehr Variablen haben, usw.

- Anfangswert für relative Mutationsschrittweite
  Bei allen Schrittweitenadaptionsverfahren benötigt man einen sinnvollen Anfangswert für die Mutationsschrittweite[5]. Es macht z. B. wenig Sinn, wenn alle Variablen im Wertebereich $[0 \ldots 100]$ optimiert werden sollen, mit einer Mutationsschrittweite von 10^{-7} zu beginnen. Diese kann sich zwar anpassen, wird aber selbst bei einer nur wenig komplexen Funktion schnell in ein lokales Optimum konvergieren. Die ES sollte zuerst global suchen und später dann die Suche mit kleinerer Schrittweite verfeinern.
  Die Mutationsschrittweite ist in VEES ein Wert relativ zum Wertebereich jeder Objektvariablen. Dies hat sich als sinnvoll herausgestellt, wenn z. B. eine Variable einen Winkel im Bereich $[0 \ldots 2\pi]$ darstellt und eine andere eine Länge im Bereich $[0 \ldots 100cm]$. Hier wäre eine absolute Schrittweite entweder zu groß für die Winkel-Variable oder zu klein für die Längen-Variable. Als geeignet hat sich der vorgegebene Wert von 0.1 des Definitionsbereiches herausgestellt.

- Optimum
  Zur Skalierung des Fitneßgraphen und als Standardwert für die Konvergenzgrenze kann ein Wert für das mögliche/bekannte Optimum angegeben werden. Dieser ist teilweise sogar bei praktischen Anwendungen bekannt, wenn z. B. ein Fehler auf 0 minimiert werden soll. Unbekannt ist dagegen die Lösungsinstanz, die diesen Wert erreicht.

4.3.2.4 Menüfunktionalität

Jede Applikation kann bei Bedarf beliebig viele eigene Menüpunkte definieren, die zusätzlich zu denen des Algorithmus erscheinen. Dies können jeweils ein Datum vom folgenden Typ aufnehmen: ganze Zahl (Integer), reelle Zahl (Double), binärer Schalter (on/off), Dateiname, Zeichenkette oder Optionsauswahl (mehrere benannte Werte).

4.3.3 Randbedingungen

In Abschnitt 2.4.6 wurden bereits einige Möglichkeiten der Behandlung von Randbedingungen aufgezeigt. Die Verwendung von *Straftermen*, oder eines *Decoders* kann in der Fitneßfunktion realisiert werden, deswegen ist hierzu keine spezielle Unterstützung durch das Programmsystem nötig. Die verbleibenden Möglichkeiten sind *Verwerfen und Neugenerieren*, *Reparieren* und *Koevolution*. Natürlich ist es auch möglich, keine Randbedingungen zu verwenden.

Die Randbedingungen kommen immer dann zur Anwendung, wenn neue Individuen generiert wurden. Dies ist zum einen der Fall bei der Initialisierung der Anfangspo-

[5] Mit Mutationsschrittweite ist hier die globale Mutationsschrittweite δ gemeint oder bei den komplexeren Verfahren die globale Komponente δ der Mutationsschrittweite.

pulation und zum anderen innerhalb einer Generation nach der Rekombination und Mutation zur Bildung von Nachkommenindividuen.

4.3.3.1 Verwerfen und Neugenerieren

Diese Methoden sind in VEES als *zurückweisend* (reject) bezeichnet. Als vordefinierte, einfache Methode kann hier die Kombination aus der Verwendung der Wertebereiche für die Randbedingungen und dem Attribut *zurückweisend* benutzt werden. Für komplexere Randbedingungen kann eine eigene, zurückweisende Funktion definiert werden.

4.3.3.2 Reparierende Randbedingungen

Für den allgemeinen Fall ist auch hier die Möglichkeit gegeben, eine benutzerdefinierte Funktion einzusetzen. Es existieren allerdings auch einige vordefinierte Methoden, die die Wertebereiche verwenden. Es sind dies im einzelnen:

- abschneiden
  Variablen, die die Obergrenze des Wertebereich überschreiten, werden auf die Obergrenze zurückgesetzt. Variablen, die die Untergrenze unterschreiten, werden auf die Untergrenze zurückgesetzt.
- zyklischer Wertebereich
  Variablen, die den Wertebereich auf einer Seite verlassen, werden um den entsprechenden Betrag auf der anderen Seite des Wertebereichs wieder eingesetzt. Das Intervall ist damit zyklisch durchverbunden.
- spiegeln
  Ähnlich wie beim zyklischen Wertebereich werden Variablen, die den Wertebereich auf einer Seite verlassen, an den Intervallgrenzen gespiegelt.

Die besondere Variante des Reparierens, daß die Fitneß für ein repariertes Individuum ermittelt wird, ansonsten aber das die Randbedingung verletzende Individuum weiterverwendet wird, läßt sich alleine in der Fitneßfunktion realisieren.

4.3.3.3 Koevolution der Randbedingungen

VEES unterstützt auch koevolvierende Randbedingungen, wie in Abschnitt 2.4.6 beschrieben. Statt einer Funktion pro Randbedingung $g_j(\vec{x})$, muß hierbei eine Funktion implementiert werden, welche die Werte für alle Randbedingungen, also einen Vektor $\vec{g}(\vec{x})$, zurückgibt.

Kapitel 5

Parallele Steady-State-Evolutionsstrategien

In diesem Kapitel wird auf eine spezielle Variante von Evolutionären Algorithmen eingegangen - die Steady-State-EAs - die sich besonders für die Parallelisierung mit paralleler Fitneßevaluation eignet. Diese ist für Anwendungen sinnvoll, die lange Evaluationszeiten haben, z. B. aufgrund von aufwendigen Berechnungen oder Simulationen. Verwendet man eine Steady-State-Variante bei Evolutionsstrategien, ergeben sich allerdings Probleme bei der wichtigen Eigenschaft der Selbstadaption. Diese werden zu beheben versucht, durch Konstruktion eines neuen, speziell auf Evolutionsstrategien zugeschnittenen Selektionsverfahrens, der *Median-Selektion*.

In vielen Arbeiten wird ein Steady-State-EA verwendet, wenn die benutzte Anwendung lange Evaluationszeiten hat. Es gibt auch einige Arbeiten, die sich explizit mit Steady-State-EAs beschäftigen, dabei wird aber immer ein Genetischer Algorithmus verwendet. In seiner frühen Arbeit verwendet Rechenberg [Rechenberg 73] zwar eine Steady-State-Evolutionsstrategie, allerdings noch mit fester Mutationsschrittweite. Laut [Bäck et al. 91] ist die Steady-State-Evolutionsstrategie nicht in der Lage, die Mutationsschrittweiten korrekt zu adaptieren. Grund dafür ist, daß durch die elitäre Selektion verkleinerte Schrittweiten bevorzugt werden. Seither wurde die Steady-State-Evolutionsstrategie nicht weiter berücksichtigt. Doch gerade im Zusammenhang mit der parallelen Fitneßevaluation bietet sie ein beachtliches Leistungspotential. Dem Autor ist keine Arbeit bekannt, bei der eine parallele Steady-State-Evolutionsstrategie benutzt wird. Dies ist die erste Arbeit, die sich explizit mit diesem Thema und den dabei evtl. auftretenden Problemen beschäftigt.

In den folgenden Abschnitten wird zunächst auf die Standard Selektionsverfahren bei Evolutionsstrategien und allgemein bei Steady-State-EAs eingegangen. Danach wird die Median-Selektion vorgestellt, die beeinflussenden Parameter genauer untersucht und Vergleichstests anhand von Standard Benchmarkfunktionen gezeigt.

5.1 Steady-State Grundlagen

Eine wesentliche Eigenschaft und Neuerung von Evolutionären Algorithmen gegenüber anderen (auch stochastischen) Optimierungsverfahren ist das Konzept der Population. Es wird nicht mehr nur eine einzelne Lösung verändert und verbessert, sondern parallel mehrere Lösungen gleichzeitig. Dies ermöglicht es erst, neben der Mutation auch Rekombination zu verwenden. Außerdem kann durch vergleichen von neuen Lösungsalternativen untereinander eine Auswahl getroffen werden, anstatt nur mit einer älteren Lösung zu vergleichen. In einem Schritt, auch eine *Generation* genannt, wird eine ganze Nachkommenpopulation neu erzeugt, aus welcher dann die Eltern der nächsten Generation selektiert werden (siehe Abbildung 2.7 in Kapitel 2).

Dieses Generationen-Konzept hat aber Nachteile, wenn man einen solchen Algorithmus nach dem Modell der parallelen Fitneßevaluation (s. Abschnitt 3.2.3) parallelisieren will. Es kann vorkommen, daß Prozessoren nicht vollständig ausgelastet werden, wenn am Ende einer Generation vor der Selektion noch Individuen evaluiert werden, andere Prozessoren aber bereits fertig sind, denn neue Individuen werden erst nach der Selektion erzeugt. Wenn man diese Restriktion fallen läßt und sich aufeinanderfolgende Generationen "überlappen" läßt, so können diese Leerlaufzeiten vermieden werden. Die Überlappung bedeutet dabei, daß Nachkommenindividuen der nächsten Generation teilweise noch aus Eltern der aktuellen Generation erzeugt werden. Dies hat sicherlich kleinere Einbußen bei der Fortschrittsgeschwindigkeit zur Folge, aber die Hoffnung dabei ist, daß die dadurch früher beendete Berechnung der nächsten Generation dies mehr als ausgleicht.

Da eine einzelne Evaluation schon relativ lange dauert, braucht es noch mehr Zeit, bis eine ganze Generation durchgerechnet ist. Aber erst dann werden die neu berechneten Individuen (die hoffentlich eine Verbesserung bringen) in die Population übernommen. Eine Überlegung ist nun, daß die Lösungen, die dabei eine Verbesserung gebracht haben, vielleicht schon relativ früh innerhalb der Generation berechnet wurden und die ganze restliche Zeit bis zur Selektion im Speicher lagen, ohne weiter verwendet zu werden. Um dies zu vermeiden, ist eine Methode nötig, welche die evaluierten Individuen, die beim Master-Prozessor eintreffen, sofort der Population als neues Elternindividuum zuführt. Diesen Ansatz verfolgen die sogenannten *Steady-State*-Algorithmen. Hierbei wird das Generationen-Konzept verworfen.

Steady-State bedeutet *stetiger Zustand* und bezieht sich auf die Population, die ja den Zustand des EAs darstellt. Die Individuen der Population ändern sich nun nicht mehr in einem Schritt alle auf einmal. Die Änderungen, also der Austausch von Individuen durch neue, finden nun graduell in kleineren Schritten statt.

Der Begriff *Generation* ist bei einem Steady-State-Algorithmus nicht mehr sinnvoll anwendbar, stattdessen wird im folgenden einfach *Schritt* verwendet. Ein Schritt besteht dann aus der Erzeugung, Evaluation und Integration von einem Individuum. Der Vergleich zwischen einem Schritt eines Steady-State-EA und einer Generation eines

generationenbasierten Algorithmus ist durch ein *Generationenäquivalent* gegeben:

Generationenäquivalent: Vergleicht man einen generationenbasierten EA der Form $(\mu,^{+}\lambda)$ mit einem $(\mu + 1)$ Steady-State-EA, so entspricht ein Generationenäquivalent beim Steady-State-EA der Größe der Nachkommenpopulation λ des generationenbasierten EA.

Ein Steady-State-EA hat nach λ Schritten genausoviele Individuen behandelt wie ein generationenbasierter $(\mu,^{+}\lambda)$ Algorithmus. [Watson et al. 97] definiert diesen Begriff bei Genetic Programming, wo genauso wie bei GAs die Nachkommenpopulation die gleiche Größe wie die Elternpopulation hat $(\mu,^{+}\mu)$. Ein Generationenäquivalent kann also nur im Vergleich mit einem bestimmen generationenbasierten Algorithmus definiert werden, da dieser erst über λ die Dauer einer Generation festlegt.

Manchmal wird für Steady-State-Algorithmen auch das Attribut *overlapping* verwendet, womit die Überlappung mehrerer, aufeinanderfolgender Generationen gemeint ist. Bei GAs gibt es auch Varianten, bei denen der Grad der Überlappung eingestellt werden kann. Dabei wird nur der angegebene Anteil der Population neu erzeugt, der restliche Anteil wird aus der vorigen Generation unmodifiziert übernommen (dieses Feature ist z. B. in GAlib vorhanden, siehe Anhang B). [Grefenstette 86] wiederum bezeichnet diesen Anteil der Individuen, der durch neue ersetzt wird, als *generation-gap*. Ist die generation-gap minimal, so resultiert dies in einem Steady-State-Algorithmus, ist sie maximal, so hat man einen generationenbasierten Algorithmus.

Was sind die Vorteile von Steady-State-EAs?

- Die Tournaround-Zeit bei der Verarbeitung der Individuen ist kürzer: neu erzeugte Individuen können früher weiterverwendet werden. Gerade bei paralleler Fitneßevaluation werden fast immer Steady-State-Algorithmen verwendet, da hier besonders deutlich ins Auge fällt, daß bei einem Generationen-basierten Algorithmus bereits fertig evaluierte Individuen auf dem Master-Prozessor längere Zeit bis zur Selektion warten würden. Dieser Vorteil wirkt sich auch bei nicht-paralleler Ausführung aus, der Fortschritt pro Generationenäquivalent ist größer.
- Der Algorithmus läßt sich sehr einfach nach der Methode der parallelen Fitneßevaluation (siehe Abschnitt 3.2.3) parallelisieren. Auf einem Netz von Workstations kann es sich schon ab ca. 0.5 Sekunden Auswertungszeit lohnen.

Nachteile:

- Asynchrone Parallelisierung bringt sicher keinen exakt linearen Speedup gegenüber einem sequentiellen Steady-State-EA, da sich durch die Asynchronität Evaluierungen zeitlich überlappen. Für neu erzeugte Individuen können die

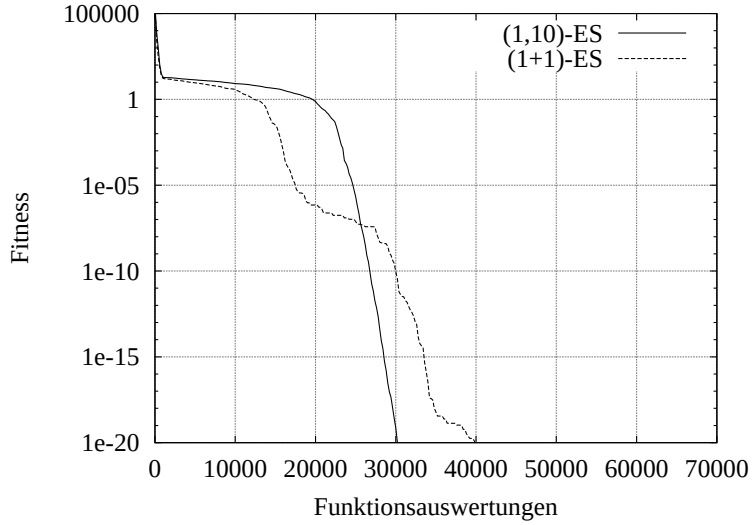


Abbildung 5.1: Optimierung von f_2 mit einer $(1, 10)$-ES und einer $(1+1)$-ES. Auf der x-Achse ist die Zahl der Funktionsauswertungen aufgetragen (welche als rechnerunabhängiges Zeitmaß dient) und auf der y-Achse die Fitneß.

momentan in Evaluierung befindlichen Individuen noch nicht als Eltern verwendet werden. Dies wird sich in einer gegenüber dem äquivalenten sequentiellen Steady-State-EA in einer leicht erhöhten Anzahl von Funktionsauswertungen niederschlagen. Dies kann man anhand der Messungen in Kapitel 7 sehen.

- Meist ist der Selektionsdruck gar nicht oder nicht eindeutig einstellbar.

5.2 Optimierungsbeispiel

Durch dieses Kapitel führt als roter Faden ein Optimierungsbeispiel, das an geeigneter Stelle bei den jeweiligen Algorithmen auftaucht. Dadurch soll schrittweise die Motivation veranschaulicht werden, die zur Konstruktion des in diesem Kapitel erarbeiteten Steady-State-Algorithmus mit Median-Selektion geführt hat. Im Beispiel wird die Funktion $f_2(\vec{x})$ minimiert (Generalized Rosenbrock's Function) (siehe auch Gleichung C.1 in Anhang C):

$$f_2(\vec{x}) = \sum_{i=1}^{n} (100 \cdot (x_{i+1} - x_i^2)^2 + (x_i - 1)^2) \tag{5.1}$$

Die weiteren Rahmenbedingungen des Beispiels sind: Dimension $n = 20$, Abbruchkriterium ist das Erreichen eines Fitneßwertes von weniger als 10^{-20} mit max. 100000

Funktionsauswertungen. Hierbei wird die Kovarianzmatrix-Adaption (CMA) verwendet. Bis auf das Abbruchkriterium sind dies die gleichen Bedingungen wie in [Hansen et al. 96], hier wurde bereits bei 10^{-10} abgebrochen. Als Ausgangssituation soll die dort verwendete $(1, 10)$-ES dienen (Abbildung 5.1). Dessen Ergebnisse decken sich auch mit den hier erzielten. Pro Strategie wurden 50 Optimierungsläufe berechnet. Das Schaubild zeigt dann immer denjenigen Lauf, der mit der Anzahl der Funktionsauswertungen bei Erreichen des Abbruchkriteriums am nächsten am Mittelwert der 50 Läufe liegt.

Eine normale $(\mu + 1)$-Evolutionsstrategie kann als Steady-State-ES bezeichnet werden. Doch wie man am Optimierungsbeispiel in Abbildung 5.1 sehen kann, bringt die $(1 + 1)$-ES gegenüber der $(1, 10)$-ES keine Verbesserung, sondern benötigt sogar ein Drittel mehr Funktionsauswertungen (ca. 40000 statt ca. 30000), um den Konvergenzwert zu erreichen. Der Optimierungsfortschritt läuft hier nicht so glatt wie bei der Komma-Strategie, sondern es sind immer wieder Phasen mit flacherem Verlauf dabei, die Kurve ist insgesamt zackiger. Dies liegt daran, daß bei der Plus-Strategie nur Verbesserungen zugelassen werden. Finden lange keine Verbesserungen statt, stagniert die Optimierung für diese Zeit. Die Selbstadaption funktioniert hier also nicht mehr richtig und es wird mit schlechter Mutationsschrittweite mutiert, wodurch die Wahrscheinlichkeit einer erfolgreichen Mutation (Verbesserung der Fitneß) geringer ist, also auch die Fortschrittsgeschwindigkeit der Optimierung. Überlebt ein Individuum recht lange, so wird in dieser Zeit auch nur die Mutationsschrittweite, die in diesem Individuum gespeichert ist, zur Erzeugung von neuen Individuen verwendet. Dies ist besonders deutlich am Optimierungsbeispiel zu sehen, wo $\mu = 1$ verwendet wurde und somit in der Population keine alternativen Individuen mit besseren Mutationsschrittweiten enthalten sind.

Man sieht an diesem Beispiel, daß sich die elitäre Plus-Selektion nachteilig auf die Selbstadaption bei der ES auswirken kann. Auch [Schwefel 92] kommt zu dem Schluß, daß eine (μ, λ)-ES einer $(\mu + \lambda)$-ES im Hinblick auf die Geschwindigkeit bei der Selbstadaption überlegen ist. Bei paralleler und asynchroner Fitneßevaluation bietet es sich aber an, eine Steady-State-Selektion einzusetzen, da dann die Parallelisierung sehr einfach ist. Bei Evolutionsstrategien haben aber die gebräuchlichen Integrationsverfahren zuviel Ähnlichkeit mit der elitären Plus-Selektion. Deshalb lohnt es sich, einen genaueren Blick auf die Selektionsverfahren zu werfen. Dies wird im nächsten Kapitel getan, danach wird die Beispieloptimierung mit dem Standard Steady-State-Algorithmus betrachtet. Mit der lokalen Tournament-Selektion wird eine weitere Variante von Steady-State-Algorithmen vorgestellt und danach die neuartige Median-Selektion, die den oben genannten Nachteil ausgleicht. Zum Schluß des Kapitels werden dann noch einmal alle vorgestellten Verfahren anhand der Beispieloptimierung verglichen.

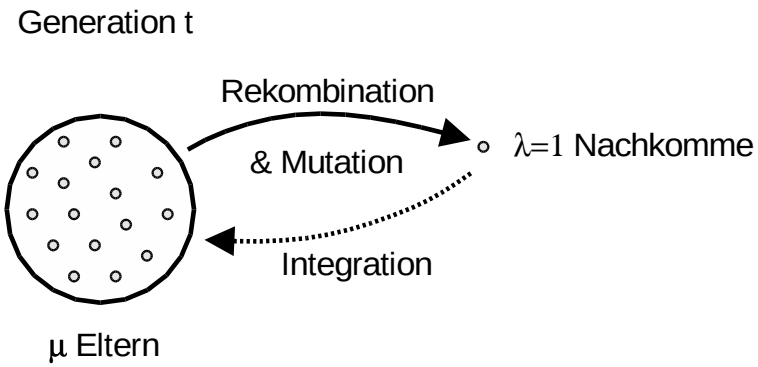


Abbildung 5.2: Ablauf beim Steady-State-EA.

5.3 Selektion bei Steady-State

An die Stelle der Selektion tritt bei Steady-State-Algorithmen die *Integration* (siehe Abb. 5.2). Sie besteht aus zwei Teilschritten:

1. Auswahl eines Elternindividuums, welches gegebenenfalls ersetzt werden soll.
2. Entscheidung, ob das Elternindividuum tatsächlich durch den Nachkommen ersetzt wird.

Bei Schritt 1 wird eine *Ersetzungsstrategie* gebraucht, die das zu ersetztende Individuum auswählt. Hierfür gibt es unter anderem die folgenden Möglichkeiten:

- Ersetzung des schlechtesten Individuums,
- Ersetzung des ältesten Individuums,
- Ersetzung eines zufällig gewählten Individuums.

Bei Schritt 2 braucht man eine *Ersetzungsbedingung*, z. B.:

- ein Individuum wird nur ersetzt, wenn das neue Individuum eine bessere Fitneß besitzt,
- immer ersetzen.

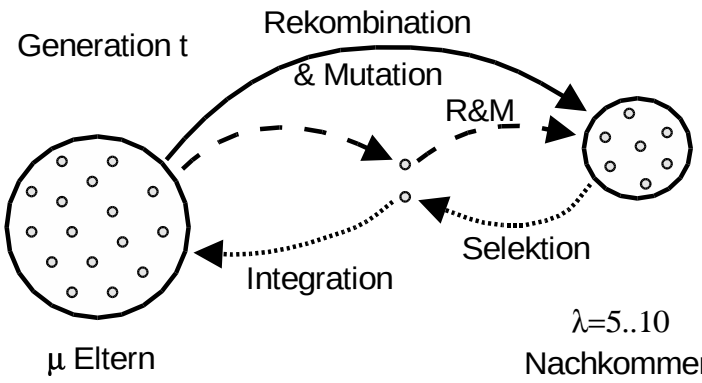


Abbildung 5.3: Ablauf eines Steady-State-ES mit lokaler Tournament-Selektion und eines Steady-State-GA nach Smith und Fogarty.

Die bei weitem am häufigsten verwendete Kombination ist “Ersetzung des schlechtesten Individuums, nur wenn das neue Individuum besser ist”. Dies ist auch die auf den ersten Blick plausibelste Kombination und wird deshalb im folgenden als “Standard Steady-State-Algorithmus” bezeichnet. Diese Kombination entspricht genau einer $(\mu+1)$-ES. Hier ist das Prinzip bei der Selektion leicht anders, führt aber genau zu demselben Ergebnis. Es werden aus den $\mu + 1$ Individuen die μ besten selektiert. Ist das neue Individuum besser als das schlechteste, so wird es selektiert und genau das schlechteste fällt heraus.

Eine (μ, λ)-ES entspricht am ehesten der Ersetzung des ältesten Individuums, da hierbei nach einer gewissen Anzahl von Schritten die ursprüngliche Elternpopulation auf jeden Fall durch neue Individuen ersetzt wurde. Die Ersetzungsbedingung ist allerdings nicht klar. Diese Lücke versucht die später vorgestellte Median-Selektion zu füllen (siehe Abschnitt 5.4).

5.3.1 Lokale Tournament-Selektion

Smith und Fogarty [Smith et al. 96] haben die Idee der Selbstadaption von Evolutionsstrategien aufgegriffen und auf einen Steady-State-GA übertragen, da ihnen die globale, konstante Mutationsrate des Basisalgorithmus suboptimal und unflexibel erschien. Sie speichern dazu in einem Individuum zusätzlich eine Mutationsrate ab. Die Reproduktion läuft nun dergestalt ab, daß einige Individuen mit Varianten dieser Mutationsrate erzeugt werden, um zu testen, ob eine vergrößerte oder verkleinerte Mutationsrate in der Lage ist, fittere Individuen zu erzeugen (siehe Abb. 5.3). Von dieser kleinen Menge erzeugten Individuen ($\lambda \leq 10$) wird dann genau eines ausgewählt und mit den üblichen Mechanismen in die Elternpopulation integriert. Diese Form der Selektion wurde gewählt, da laut Smith und Fogarty zur Unterstützung der

Selbstadaption ein relativ hoher Selektionsdruck nötig ist.

Der Algorithmus ist ursprünglich nicht zum Zweck der Parallelisierung entworfen worden und mußte deswegen auch minimal abgeändert werden, um bei der Parallelisierung nicht allzu komplex zu werden. In Abbildung 5.3 ist der ursprüngliche Ablauf der Reproduktion mit zwei gestrichelten Pfeilen eingezeichnet. Wie bereits erwähnt, wird zuerst ein Elternindividuum zufällig ausgewählt, aus welchem dann alle λ Nachkommen des Tournaments mit variierter Mutationsrate erzeugt werden. In einer parallelisierten Version mit asynchroner Fitneßevaluation müßte man dann mit einigem Aufwand darauf achten, daß bei der Selektion genau die Individuen im Tournament gesammelt werden, die aus demselben Elternindividuum erzeugt worden sind. Ist die Anzahl der Prozessoren deutlich größer als die Größe des Tournaments λ, so müssen evtl. mehrere solcher Tournaments verwaltet werden, bis sie aufgefüllt sind und die Selektion stattfinden kann.

Um den parallelen Algorithmus zu vereinfachen und näher am Vorbild der Evolutionsstrategie zu halten, wurde er dahingehend modifiziert, daß für jeden der λ Nachkommen die Eltern neu ausgewählt werden (durchgehender Pfeil “Rekombination & Mutation” in der Abbildung), genau wie es bei einer Generationen-basierten (μ^+,λ)-ES geschieht. Dann kann das Tournament mit den ankommenden Individuen gefüllt werden und immer wenn es voll ist, findet die Selektion statt.

Als Notation für die Steady-State-Evolutionsstrategie mit lokaler Tournament-Selektion wurde die folgende Form gewählt:

$$(\mu + (1, \lambda)) \tag{5.2}$$

Die Klammer $(1, \lambda)$ anstelle von nur λ bei der Generationen-basierten ES soll das Tournament repräsentieren.

Für den Fall $\mu = 1$, spielt die Ersetzungsstrategie (ältestes/schlechtestes/zufälliges Individuum) keine Rolle. Die Ersetzungsbedingung “immer ersetzen” führt dann exakt zu einer $(1, \lambda)$-ES. Für diesen Fall unterscheidet sich das Schaubild des Optimierungsbeispiels auch nicht von dem der $(1, \lambda)$-Strategie und wird deshalb hier nicht gezeigt. Die Ersetzungsbedingung “nur durch fitteres Individuum ersetzen” entspricht dagegen genau einer $(1 + \lambda)$-Strategie. Für $\mu > 1$ unterscheidet sich die Steady-State-ES mit lokaler Tournament-Selektion mit entsprechenden Ersetzungsstrategien und -bedingungen aber von den (μ^+,λ)-Strategien. Sie wird dann im Kapitel 7 auch als Alternativmöglichkeit getestet.

5.4 Median-Selektion

Die bisher vorgestellten Möglichkeiten, eine Steady-State-Evolutionsstrategie zu implementieren, welche die Selbstadaption der Mutationsschrittweiten ähnlich gut bewältigt wie die (μ, λ)-ES, hatten - zumindest bei der Beispieloptimierung - nicht den

gewünschten Erfolg. Deshalb wird in diesem Abschnitt ein speziell zu diesem Zweck konstruierter Algorithmus vorgestellt [Wakunda et al. 00a, Wakunda et al. 00b].

Die Wahl der Ersetzungsstrategie fällt dabei auf die "Ersetzung des ältesten Individuums". Dies kommt dem Vorbild der (μ, λ)-ES am nächsten, da hierbei kein Individuum eine unbegrenzt lange Zeitspanne überleben kann, es sei denn, die Ersetzungsbedingung läßt gar keine Ersetzung mehr zu, was deshalb in dem neuen Verfahren gezielt vermieden werden soll.

Die Standardersetzungsbedingungen führen in Kombination mit der Ersetzungsstrategie "Ersetzung des ältesten Individuums" nicht zu den erwünschten Eigenschaften:

immer ersetzen: Dies ist nicht sinnvoll in Kombination mit Ersetzung des ältesten Individuums, da dann ein sog. *random walk* stattfindet, bei dem keine Selektion mehr vorhanden ist. In jedem Schritt würde einfach ein Individuum der Elternpopulation durch das nächste, zufällig erzeugte, wahllos überschrieben werden.

nur durch ein fitteres Individuum ersetzen: Dies führt wieder zu einer elitären Ersetzung, bei der kein Rückschritt in der Fitneß zugunsten einer Variation der Schrittweite stattfinden kann. Hat das zu ersetzende Individuum zufällig eine relativ hohe Fitneß, so kann es genau wie beim Standard Steady-State zu langen Stagnationen kommen.

Es ist also eine andere Ersetzungsbedingung nötig. Da die (μ, λ)-ES nachgebildet werden soll, soll die Selektion hierbei noch einmal genauer betrachtet werden (Abb. 5.4). Die Fitneß der λ Nachkommen besitzt eine bestimmte Verteilung, die abhängig von den μ Elternindividuen, dem Rekombinations- und Mutationsoperator und der Fitneßfunktion ist[1]. Sie könnte z. B. wie in der Abbildung 5.4 ähnlich einer Normalverteilung aussehen (weil ja die Mutation auf einer Normalverteilung basiert), kann aber auch anders geartet sein. Darüber wird keine feste Annahme gemacht. Es werden bei der Komma-Selektion nun immer die μ besten Individuen selektiert und die restlichen $\lambda - \mu$ Individuen verworfen. Im folgenden wird ohne Beschränkung der Allgemeinheit von einer Maximierung der Fitneß ausgegangen, für eine Minimierung gilt analog das gleiche.

Um die Selektion durchführen zu können, wird üblicherweise die Population nach der Fitneß sortiert (aufsteigend), so daß gilt:

$$f(I_\lambda) \leq f(I_{\lambda-1}) \leq \ldots \leq f(I_{\mu+1}) \leq f(I_\mu) \leq \ldots \leq f(I_2) \leq f(I_1) \tag{5.3}$$

[1] Dies ist die zentrale Annahme, auf der das Verfahren beruht. Z. B. bei der Verwendung einer dynamischen Fitneßfunktion ist diese Annahme nicht mehr unbedingt gegeben, falls sich die Fitneß zu schnell ändert. Die Verteilung wäre dabei auch noch von der Zeit abhängig. Es ließe sich dann ein extremer Fall konstruieren, bei dem mit stetig fallender Fitneß kein Individuum mehr akzeptiert werden würde.

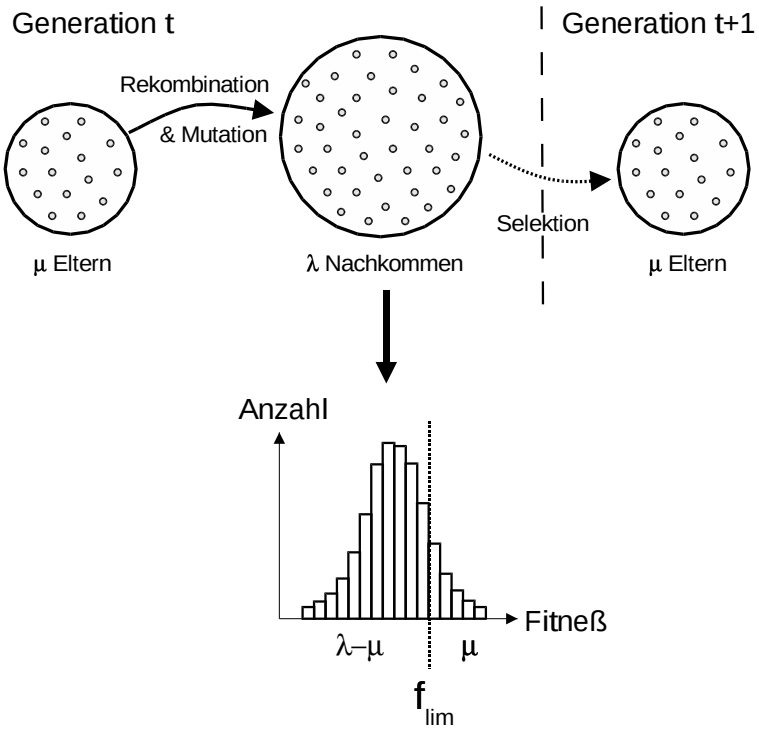


Abbildung 5.4: Selektion bei der (μ, λ) Evolutionsstrategie.

Die Selektionsentscheidung (selektiert = 1/nicht selektiert = 0) sieht für die Komma-Selektion nun folgendermaßen aus: Die Individuen mit Index $1 \leq i \leq \mu$ werden selektiert, die Individuen mit Index $\mu < i \leq \lambda$ werden nicht selektiert:

$$\text{select}(I_i) = \begin{cases} 1 & \text{gdw} \quad 1 \leq i \leq \mu \\ 0 & \text{gdw} \quad \mu < i \leq \lambda \end{cases} \tag{5.4}$$

Die Selektionsentscheidung wird also anhand des Index des Individuums in der nach Fitneß sortierten Population getroffen. Soll nun eine ähnliche Selektionsentscheidung in einem Steady-State-Algorithmus getroffen werden, so ist dies in dieser Art und Weise nicht mehr möglich, da ja keine Nachkommenpopulation der Größe λ existiert. Es kann aber eine andere Betrachtungsweise der Komma-Selektion angewendet werden, die die Fitneßwerte der Individuen direkt betrachtet und nicht indirekt über den Index der Individuen:

$$f_{\text{lim}} = f(I_\mu) \tag{5.5}$$

f_{lim} ist der Fitneßgrenzwert, so daß alle selektierten Individuen eine Fitneß größer oder gleich f_{lim} besitzen und alle nicht selektierten Individuen eine Fitneß kleiner oder gleich f_{lim}. Es wird f_{lim} als der Fitneßwert des Individuums mit Index μ definiert

(Gleichung 5.5), so daß mit Gleichung 5.3 daraus folgt:

$$\begin{cases} f(I_i) \geq f_{\text{lim}} & \text{falls} \quad 1 \leq i \leq \mu \\ f(I_i) \leq f_{\text{lim}} & \text{falls} \quad \mu < i \leq \lambda \end{cases} \tag{5.6}$$

$$\Leftrightarrow \begin{cases} 1 \leq i \leq \mu & \Rightarrow \quad f(I_i) \geq f_{\text{lim}} \\ \mu < i \leq \lambda & \Rightarrow \quad f(I_i) \leq f_{\text{lim}} \end{cases} \tag{5.7}$$

Dies ergibt zusammen mit Gleichung 5.4:

$$\begin{cases} \text{select}(I_i) = 1 & \Rightarrow \quad f(I_i) \geq f_{\text{lim}} \\ \text{select}(I_i) = 0 & \Rightarrow \quad f(I_i) \leq f_{\text{lim}} \end{cases} \tag{5.8}$$

Der Umkehrschluß ($\Leftarrow$) gilt hier nicht, da der Fall $f(I_i) = f_{\text{lim}}$ in beiden Zeilen enthalten ist. Vernachlässigt man den Fall in einer Zeile (z. B. der unteren), so wird die Komma-Selektion in Spezialfällen nicht exakt nachgebildet. Die Spezialfälle sind dabei das Auftreten von einigen gleichen Fitneßwerten um den Indexbereich $i = \mu$ und größer. Das anvisierte Ziel ist aber ein Steady-State-Algorithmus, der ähnliche Eigenschaften wie die (μ, λ)-ES hat. Um dies realisieren zu können muß (und soll) u. a. auch das Konzept der Nachkommenpopulation fallen gelassen werden, was eine größere Modifikation darstellt, als auf die genannten Spezialfälle zu verzichten.

Der modifizierte Umkehrschluß kann also als neue Selektionsentscheidung dienen und sieht folgendermaßen aus

$$\begin{cases} f(I_i) \geq f_{\text{lim}} & \Rightarrow \quad \text{select}(I_i) = 1 \\ f(I_i) < f_{\text{lim}} & \Rightarrow \quad \text{select}(I_i) = 0 \end{cases} \tag{5.9}$$

$$\Leftrightarrow \quad \text{select}(I_i) = \begin{cases} 1 & \text{falls} \quad f(I_i) \geq f_{\text{lim}} \\ 0 & \text{falls} \quad f(I_i) < f_{\text{lim}} \end{cases} \tag{5.10}$$

Im Gegensatz zu Gleichung 5.4, bei der noch die ganze Population betrachtet wird ($1 \leq i \leq \lambda$), tritt hier nur noch ein Individuum I_i auf, für das die Selektionsentscheidung gefällt wird. Kann man nun noch erreichen, daß f_{lim} anhand z. B. eines Modells geschätzt werden kann, anstatt es aus den λ Nachkommen zu bestimmen, so kann man die Selektionsentscheidung direkt für jedes erzeugte Individuum fällen. Dieses kann dann sofort in die Elternpopulation integriert werden, was einen Steady-State-Algorithmus darstellt. Die Erzeugung aller λ Nachkommen vor der Selektion würde damit vermieden werden. Im folgenden Absatz soll nun die Methode beschrieben werden, wie die Median-Selektion den Fitneßgrenzwert f_{lim} schätzt.

Die Fitneßverteilung liegt bei der Komma-Selektion in Form der Menge der λ Nachkommenindividuen (und ihrer Fitneßwerte) vor. Nach der Selektion hat sich die Elternpopulation verändert und somit auch die Fitneßverteilung der Individuen, die daraus

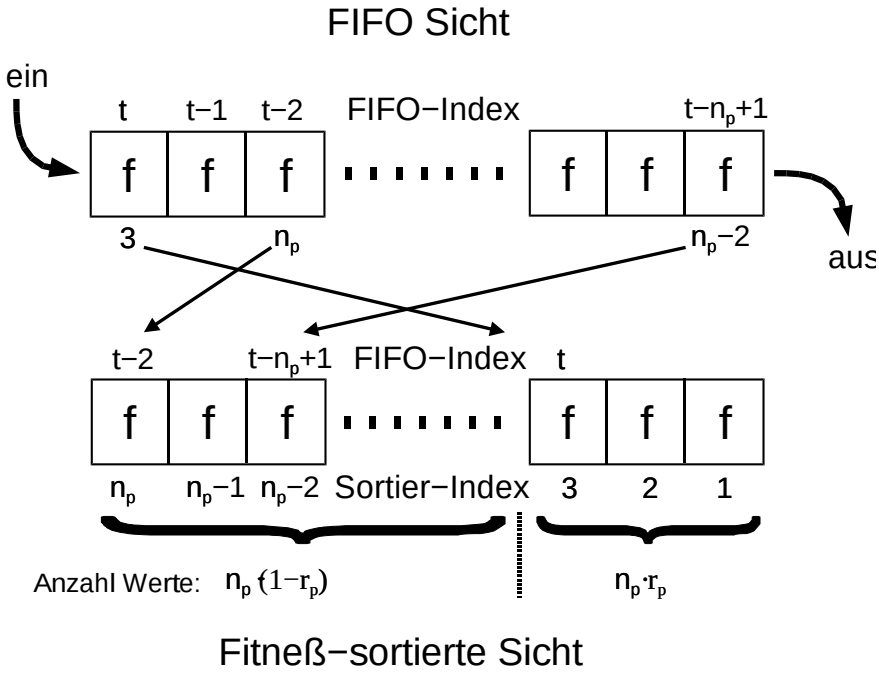


Abbildung 5.5: Organisation des Fitneß-Puffers. Die Zeitangabe (*t, t-1, . . .*) bezeichnet den Zeitpunkt des Einfügens des Wertes in den Puffer. Der älteste Wert vom Zeitpunkt $t - n_p$ fällt als nächster heraus.

produziert werden. Da es bei einem Steady-State-Algorithmus keine Nachkommenpopulation in dieser Größe gibt, muß man die Fitneßwerte der erzeugten Individuen separat in einer Liste oder einem Puffer der Länge λ abspeichern. Der Fitneßwert im Puffer am Index μ wird dann als Fitneßakzeptanzgrenze $f_{\lim}$ verwendet. Diese Verteilung in Form einer Stichprobenmenge muß mit neuen Werten aktualisiert werden, um das immer wieder daraus bestimmte $f_{\lim}$ an die sich verändernde Elternpopulation anzupassen. Deswegen ist der Fitneß-Puffer nach dem *First In First Out* (FIFO) Prinzip organisiert, d. h. alte Fitneßwerte werden nach λ Schritten automatisch von neuen Werten aus dem Puffer verdrängt und verworfen. Dies ist in Abbildung 5.5 in der obere Hälfte dargestellt. Um das Modell allgemeiner zu halten, wird die Pufferlänge mit n_p bezeichnet. Sie muß nicht unbedingt den Wert λ einer zu simulierenden Komma-Selektion haben, sondern kann, wie später gezeigt wird, auch anders, insbesondere kleiner, gewählt werden. Um auf $f_{\lim}$ zugreifen zu können, müssen die Fitneßwerte auch in nach Fitneß sortierter Form vorliegen (Abb. 5.5 untere Hälfte). Dies kann mit einer zweiten Verkettung der Liste erreicht werden. Details zur Datenstruktur werden in einem späteren Abschnitt erläutert.

Will man mit diesem Modell eine (μ, λ)-Strategie simulieren, so sollte dies einer Pufferlänge $n_p = \lambda$ entsprechen. Das Verhältnis $\frac{\mu}{\lambda}$ dient zur Bestimmung von $f_{\lim}$, wel-

ches die Fitneß des v-besten Individuums ist:

$$v = n_p \cdot \frac{\mu}{\lambda} = \lambda \cdot \frac{\mu}{\lambda} = \mu \tag{5.11}$$

$v = \mu$ deckt sich wieder mit dem Modell der (μ, λ)-Strategie.

Das Verhältnis $\frac{\mu}{\lambda}$ spiegelt den Selektionsdruck wieder und soll im weiteren mit r_p (Akzeptanzrate) bezeichnet werden. r_p dient dazu, im Modell den Index zum Zugriff auf f_{lim} relativ zur Pufferlänge n_p zu bestimmen und kann im Intervall $(0, 1]$ frei gewählt werden. Mit $r_p = 0.5$ stellt f_{lim} den sogenannten *Median* dar[2]. Andernfalls wird f_{lim} zum sog. *k-kleinsten Element* mit

$$k = \lfloor n_p \cdot (1 - r_p) + 1 \rfloor \qquad k \in \{1, \ldots, n_p\} \tag{5.12}$$

(bei Maximierung und aufsteigender Sortierung der Fitneßwerte), was als gewichteter Median angesehen werden kann. Aus diesem Grunde wird das Verfahren *Median-Selektion* genannt. Bei Minimierung und aufsteigender Sortierung wird k (mit $n_p \in [0, 1)$) berechnet durch:

$$k = \lfloor n_p \cdot r_p + 1 \rfloor \qquad k \in \{1, \ldots, n_p\} \tag{5.13}$$

Der Algorithmus für die Steady-State-Evolutionsstrategie mit Median-Selektion ist in Abbildung 5.6 gegeben. Neben der üblichen Initialisierung wird in Zeile 3 der FIFO-Fitneßpuffer als leerer Puffer angelegt und danach die μ Fitneßwerte der Population `P(0)` hineingegeben. In den Zeilen 6 bis 9 wird im Unterschied zum Generationen-basierten Algorithmus nur ein einziges neues Individuum mit Crossover und Mutation erzeugt. In Zeile 10 wird mit der gewählten Ersetzungsstrategie das potentiell zu ersetzende Individuum I_r ausgewählt. Die Zeilen 11 bis 13 realisieren die Ersetzungsbedingung mittels Median-Selektion, d. h. die Fitneß des neuen Individuums `I'` wird mit der Fitneßakzeptanzgrenze verglichen, die aus dem Puffer mithilfe der Akzeptanzrate r_p gewonnen wird. In Zeile 14 wird schließlich der Fitneßwert des neuen Individuums in den Fitneßpuffer gegeben. Dies geschieht unabhängig davon, ob das Individuum akzeptiert wurde oder nicht, da der Puffer ja ein Abbild der Verteilung der gesamten Nachkommenpopulation darstellen soll, worin sowohl die selektierten als auch die nicht selektierten Individuen enthalten sind.

Einige implementierungstechnische Details seien hier betont:

- Der Fitneß-Puffer kann als Array realisiert werden. Zur Benutzung als FIFO benötigt man lediglich einen Zähler für die Anzahl der im Puffer befindlichen Elemente und eine Variable, die den Index des nächsten einzufügenden Elements angibt. Dieser Index wird modulo n_p weitergezählt.

[2]Genaugenommen nur, falls n_p ungerade ist. Für gerade n_p ist der Median definiert als der Mittelwert der beiden Elemente mit Index $0.5n_p$ und $0.5n_p + 1$.

```
1   P(0)=initRandomPopulation()
2   eval(P(0))
3   fifo={}; fifoInsert(P(0),fifo)
4   t=0
5   while (not termination()) do
6      Parents = ParentSelect(P(t))
7      I = crossover(Parents)
8      I'= mutate(I)
9      eval(I')
10     I_r = individualToReplace(P(t))
11     if (getFitness(I') >= acceptanceLimit(fifo,r_p)) then
12        replace(I_r, I', P(t))
13     endif
14     fifoInsert(getFitness(I'), fifo)
15     inc(t)
16  end
```

Abbildung 5.6: Algorithmus der Steady-State-Evolutionsstrategie mit Median-Selektion.

Zum Zugriff in sortierter Reihenfolge wird jedes Element zusätzlich doppelt verkettet, es enthält also einen Verweis (in Form eines Array-Index) auf den Vorgänger und Nachfolger. Als Vorgänger/Nachfolger ist hier der nächst größere/kleinere Wert gemeint. Wird ein Element in den Puffer eingefügt, so fällt zunächst das älteste Element heraus und dessen Link muß entfernt werden (es werden stattdessen einfach der Vorgänger mit dem Nachfolger verlinkt). Danach muß die Sortierposition des neuen Elements gesucht und die Verlinkung an dieser Stelle aktualisiert werden.

- Die Funktion `acceptanceLimit(fifo,` r_p`)` dient dazu, die Fitneß-Akzeptanzgrenze f_{lim} aus dem FIFO-Puffer zu erhalten. Dazu liefert sie aus den sortierten Fitneßwerten den Wert mit dem Index k nach Gleichung 5.12 zurück: `fifoGetSorted(`k`)`.

- Zu Beginn des Algorithmus ist der Puffer leer und muß gefüllt werden:

  - zuerst werden die Fitneßwerte der μ Elternindividuen in den Puffer gegeben
  - falls $n_p > \mu$ ist, also der Puffer immer noch nicht voll ist, aber mit der

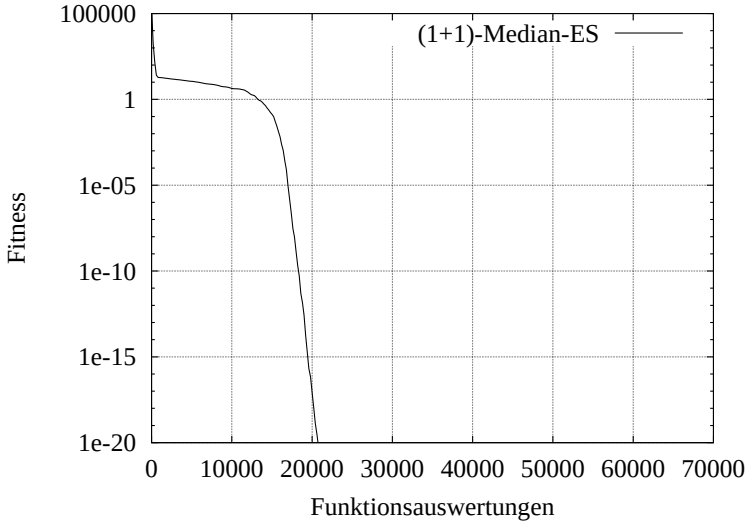


Abbildung 5.7: Das Optimierungsbeispiel mit der $(1+1)$ Median-Strategie.

Funktion `fifoGetSorted()` schon auf f_{lim} zugegriffen werden soll, wird statt der Pufferlänge n_p die aktuelle Anzahl der Werte im Puffer n'_p verwendet: $f_{\text{lim}} = \texttt{fifoGetSorted}(\lceil n'_p \cdot r_p \rceil)$. Für eine kurze Zeit am Anfang kann es dann sein, daß die Stichprobe an Fitneßwerten im Puffer noch nicht repräsentativ genug ist, um eine relevante Selektionsentscheidung zu treffen. Da EAs aber sehr robust gegenüber stochastischen Schwankungen sind, wird dies sehr schnell wieder ausgeglichen.

- Das Update des Fitneßpuffers mit der Fitneß des neuen Individuums (Zeile 14) findet erst nach der Selektion dieses Individuums statt (Zeilen 11-13). Dies verhindert insbesondere bei relativ kleinen Pufferlängen, daß beim Zugriff auf den Puffer genau der gerade eingefügte Fitneßwert als Fitneßakzeptanzgrenze extrahiert wird. In diesem Fall würde der Fitneßvergleich (Zeile 11) mit $\geq$ immer wahr liefern und das Individuum würde akzeptiert werden.

In Abbildung 5.7 ist das Optimierungsbeispiel für die $(1+1)$ ES mit Median-Selektion gezeigt. Es wurden die Parameter $n_p = 10$ und $r_p = 0.15$ verwendet. Wie man sieht, wird der erwünschte Effekt erreicht. Die Kurve verläuft glatt, d. h. die Selbstadaption wird ausreichend unterstützt. Dadurch kann der Effekt der kürzeren Tournaround-Zeit ausgenutzt werden und es werden fast ein Drittel weniger Funktionsauswertungen im Vergleich zur Generationen-basierten $(1, 10)$-ES benötigt. Ein Vergleich aller Verfahren ist in Abbildung 5.8 dargestellt.

Bäck [Bäck 94b] nennt drei Kriterien, die bei Selektionsverfahren besonders wünschenswert sind, da man dadurch auf einfache Art Einfluß auf den Suchprozeß nehmen kann:

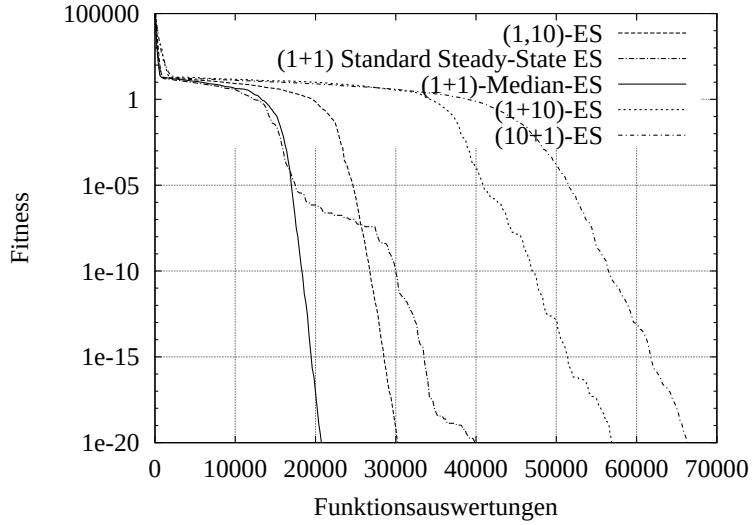


Abbildung 5.8: Vergleich aller Strategien beim Optimierungsbeispiel.

1. Der Einfluß des/der Steuer-Parameter(s) auf den Selektionsdruck sollte einfach und vorhersagbar sein.

2. Ein einzelner Steuerparameter ist vorzuziehen.

3. Die Bandbreite an Selektionsdruck, die durch die Steuerparameter erreicht werden kann, sollte möglichst groß sein.

Bei der Median-Selektion verhält es sich folgendermaßen:

1. Die beiden Parameter Pufferlänge n_p und die Akzeptanzrate r_p haben einen relativ klaren Einfluß (dieser wird in den nächsten beiden Abschnitten noch genauer untersucht):

   - Pufferlänge n_p:
     - Eine zu kurze Pufferlänge bewirkt, daß die Fitneßverteilung durch die Werte im Puffer zu ungenau repräsentiert wird. Dies macht den Selektionprozeß zu sehr abhängig von stochastischen Schwankungen und dadurch chaotisch.
     - Eine zu große Pufferlänge vermindert den Selektionsdruck, da es länger dauert, bis die Akzeptanzgrenze sinkt.
     - Die Pufferlänge sollte im Bereich von $\mu \ldots n \cdot \mu$ mit kleinem n (z. B. $n = 2$) liegen, mehr dazu in Abschnitt 5.4.1.

- Akzeptanzrate r_p: wählbar zwischen 0.0 und 1.0; nahe an diesen Grenzwerten ist die Abweichung von der tatsächlichen Akzeptanzrate etwas größer, weiter weg von den Rändern des Wertebereiches näher am tatsächlichen Wert. Die Bedeutung und der Einfluß sollten relativ klar sein. Der Richtwert liegt im Bereich zwischen 0.1 bis 0.2.

2. Es gibt zwei Parameter, die Einstellung ist also etwas komplexer als mit nur einem. Die Pufferlänge hat allerdings keinen allzu großen Spielraum und Einfluß. Den Hauptparameter zur Regelung des Selektionsdruckes stellt die Akzeptanzrate dar.

3. Die Bandbreite an erreichbarem Selektionsdruck ist sehr breit. Alleine die Akzeptanzrate kann von 0.0 bis 1.0 eingestellt werden, was sehr hohem und sehr niedrigem Selektionsdruck entspricht.

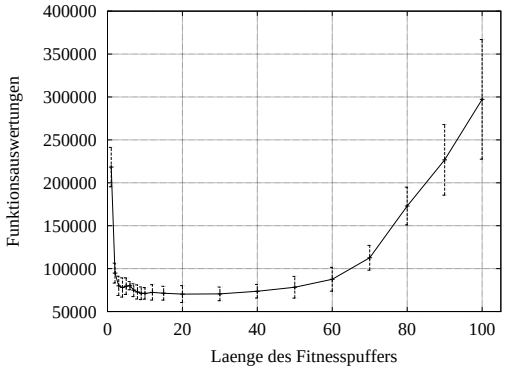
In den nächsten beiden Abschnitten wird der Einfluß der beiden Parameter Pufferlänge n_p und Akzeptanzrate r_p näher untersucht. Es wurden die folgenden 6 Funktionen und Strategien getestet:

Funktion	Dimension	Strategie	Konvergenz-grenze	max. Anz. Funktions-auswertungen
f_2	20	$(10+1)$ Median	$1 \cdot 10^{-20}$	1000000
f_6	20	$(10+1)$ Median	$1 \cdot 10^{-20}$	1000000
f_{15}	20	$(10+1)$ Median	$1 \cdot 10^{-20}$	1000000
q_2	20	$(10+1)$ Median	$1 \cdot 10^{-20}$	1000000
f_9	20	$(20+1)$ Median	$1 \cdot 10^{-10}$	1500000
f_{24}	4	$(20+1)$ Median	$3.07486 \cdot 10^{-4}$	1500000

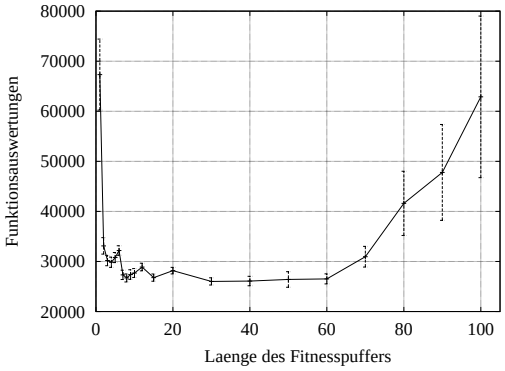
Bei allen Beispielen wurde pro Meßpunkt im Schaubild über jeweils 40 Messungen gemittelt. In den Schaubildern ist außer dem Mittelwert immer die Standardabweichung über die 40 Läufe dargestellt. Bei den ersten vier Funktionen mit $\mu = 10$ werden die Pufferlängen 1 bis 10, 12, 15, 20 und 30 bis 100 (in 10er Schritten) getestet. Bei den letzten beiden Funktionen mit $\mu = 20$ werden die Pufferlängen 1, 5, 10, 20, 40, 50, 60, 70, 80, 100, 120, 150 und 200 getestet.

5.4.1 Einfluß der Pufferlänge

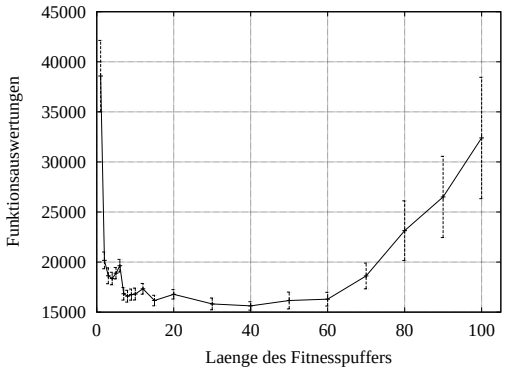
Im Fitneßpuffer soll eine repräsentative Stichprobe der Fitneßverteilung enthalten sein. Seine Länge n_p muß also groß genug gewählt werden, damit die Stichprobe dieser Anforderung genügt. Andererseits wird eine zu große Pufferlänge wahrscheinlich zu lange Zeit brauchen, um die Akzeptanzgrenze anzupassen. Die Ergebnisse bei den sechs Testfunktionen sind in Abbildung 5.9 gezeigt.



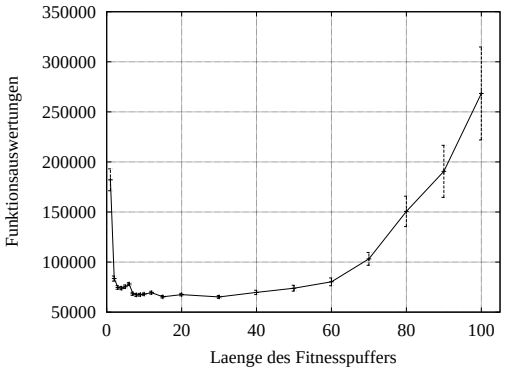
(a) Funktion f_2



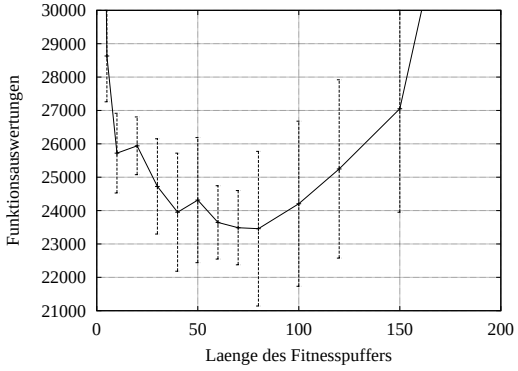
(b) Funktion f_6



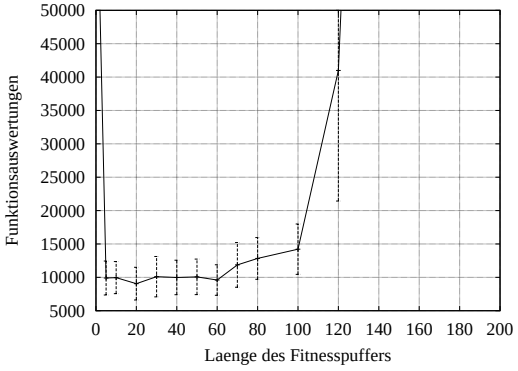
(c) Funktion f_{15}



(d) Funktion q_2



(e) Funktion f_9



(f) Funktion f_{24}

Abbildung 5.9: Abhängigkeit der Zahl der Funktionsauswertungen bis zum Abbruchkriterium von der Länge n_p des Fitneßpuffers. Die vertikalen Balken geben die Standardabweichung an.

An verschiedenen Stellen der Kurven sind Spitzen zu erkennen, besonders deutlich bei den Funktionen f_6, f_{15} und q_2. Dies tritt genau beim Übergang zwischen den Pufferlängen 6 und 7 und zwischen 12 und 15 auf. Die Spitzen kommen zustande, weil bei diesen Übergängen die Berechnung des Index k nach Gleichung 5.13 mit $r_p = 0.15$ unterschiedliche Werte liefert:

n_p	k
$1, \ldots, 6$	1
$7, \ldots, 13$	2
$14, \ldots, 19$	3
$20, \ldots, 26$	4
$\ldots$	

Es wird also zur Bestimmung von f_{lim} bei Pufferlängen von 1 bis 6 die beste im Puffer vorhandene Fitneß verwendet. Bei Pufferlängen 7 bis 13 wird die zweitbeste Fitneß verwendet, usw.

Dies könnte z. B. geglättet werden, durch eine Interpolation zwischen den im Puffer gespeicherten Fitneßwerten, falls $n_p \cdot r_p$ bzw. $n_p \cdot (1 - r_p)$ keine ganze Zahl ergibt. Dadurch wäre auch bei kleinen Pufferlängen die Akzeptanzrate feiner regulierbar. Dies wurde aber in dieser Arbeit noch nicht realisiert.

Wie erwartet sind sehr kleine oder große Pufferlängen nicht vorteilhaft. Stattdessen führt ein mittelgroßer Wert im Bereich $10 \leq n_p \leq 60$ ($\mu = 10$) bzw. $40 \leq n_p \leq 100$ ($\mu = 20$) zu den kleinsten Anzahlen von Funktionsauswertungen. Der Verlauf innerhalb dieser Grenzen ist sehr flach. Dies ist ein Vorteil, da es bedeutet, daß die Wahl der Pufferlänge in einem großen Bereich relativ robust ist.

Als grobe Empfehlung zur Wahl von n_p, die aus diesen Versuchen und dem Verständnis des Median-Selektionsalgorithmus abgeleitet ist, kann folgendes gesagt werden:

- Es sollte $n_p \geq 10$ gewählt werden, damit der Fitneßpuffer eine einigermaßen repräsentative Stichprobe der Fitneßwerte enthalten kann.
- $n_p < \lambda$: die Pufferlänge kann und sollte kleiner als die Anzahl der Nachkommenindividuen λ gewählt werden, wie sie in einer (μ, λ)-ES gewählt werden würde, um einen Selektionsdruck r_p zu erreichen. Bei einem üblichen Selektiondruck $\frac{\mu}{\lambda} = \frac{1}{5}$ sollte $n_p < 5\mu$ gewählt werden.
- $n_p > \mu$: dies sollte zumindest für kleine Populationsgrößen gelten. Wahrscheinlich macht es erst für größere Populationen Sinn, $n_p < \mu$ zu wählen, da wahrscheinlich analog zu obigen Schaubildern eine Art Sättigungsgrenze erreicht wird, ab welcher eine Vergrößerung des Fitneßpuffers die Performance des Algorithmus nur noch verschlechtert.

5.4.2 Einfluß der Akzeptanzrate

Der Parameter r_p simuliert die Rate $\frac{\mu}{\lambda}$ der akzeptierten Individuen in einer (μ, λ) Evolutionsstrategie. Bekannte und häufig verwendete Werte hierfür sind: $\frac{\mu}{\lambda} = \frac{1}{5} = 0.2$ [Smith et al. 96]; $\frac{\mu}{\lambda} \approx \frac{1}{6} \approx 0.1667$ [Bäck 92a]; $\frac{\mu}{\lambda} = \frac{15}{100} = 0.15$ [Ostermeier et al. 93].

Die Rahmenbedingungen der Messungen sind dieselben, wie im letzten Abschnitt bei den Messungen zur Pufferlänge. Hier wurde die Akzeptanzrate zwischen 0.05 und 1.0 in Schritten von 0.05 gemessen. Die Pufferlänge betrug für die ersten vier Funktionen $n_p = 30$ (f_2, f_6, f_{15} und q_2) und für die letzten beiden Funktionen $n_p = 60$ (f_9 und f_{24}).

Die Ergebnisse bei den sechs Testfunktionen sind in den Abbildungen 5.10 und 5.11 gezeigt.

Spätestens ab einem Wert von ca. $r_p \geq 0.35$ ist bei allen getesteten Funktionen ein Anstieg der benötigten Funktionsauswertungen zu beobachten. Bei höheren Akzeptanzraten nimmt auch die Konvergenzzuverlässigkeit schnell ab. Ab $r_p \geq 0.5$ konvergiert bei allen Funktionen kein einziger Lauf mehr. Dies bedeutet, daß für zu große Akzeptanzraten zu viele Individuen mit schlechten Fitneßwerten selektiert werden, so daß keine ausreichende Verbesserung mehr stattfindet. Dies ist auch bei einer generationenbasierten $(\mu\,\lambda)$-Evolutionsstrategie der Fall, wenn das Verhältnis $\frac{\mu}{\lambda}$ zu groß ist.

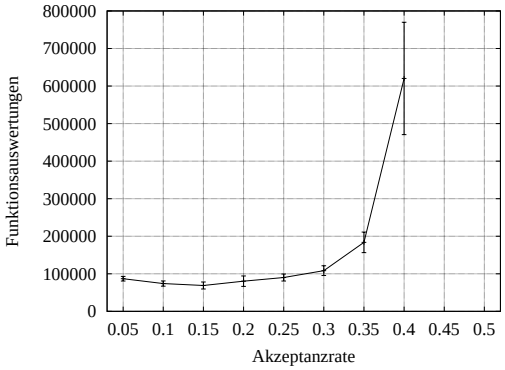
Unterhalb von $r_p \leq 0.3$ (Ausnahme f_{24}: $r_p \leq 0.25$) ist bei den meisten Funktionen ein relativ flacher Verlauf der benötigten Funktionsauswertungen zu erkennen. In diesem Bereich ist die Einstellung des Parameters also relativ robust. Das absolute Minimum liegt bei den getesteten Funktionen im Bereich $0.1 \leq r_p \leq 0.2$, bei dreien davon bei $r_p = 0.15$.

Die abschließende Empfehlung zur Wahl des Parameters r_p lautet also $r_p = 0.15$ bzw. im Bereich $0.1 \leq r_p \leq 0.2$.

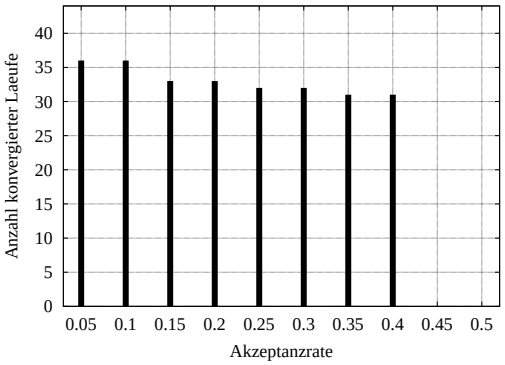
5.5 Vergleichstests mit Benchmarkfunktionen

Für die Tests zur Steady-State-Evolutionsstrategie mit Median-Selektion wurde insbesondere solche Funktionen herangezogen, die besondere Forderungen an die Selbstadaption stellen. Insbesondere sind dies Funktionen, die meistens nur mit der leistungsfähigen Kovarianzmatrix-Adaption in akzeptabler Geschwindigkeit optimiert werden können. Es wird hier von Bedingungen ausgegangen, wie sie bei einer Anwendung in der industriellen Praxis vorgefunden werden können:

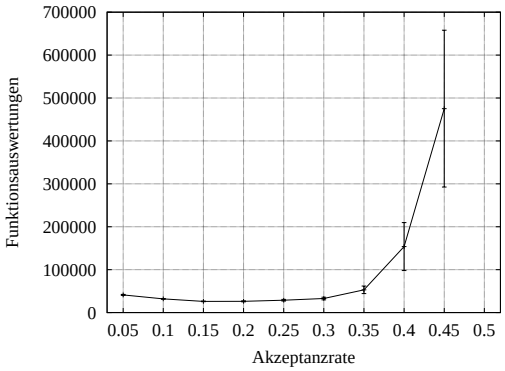
- die zu optimierende Funktion ist multimodal, so daß eine nicht zu kleine Populationsgröße benötigt wird ($\mu = 20$),



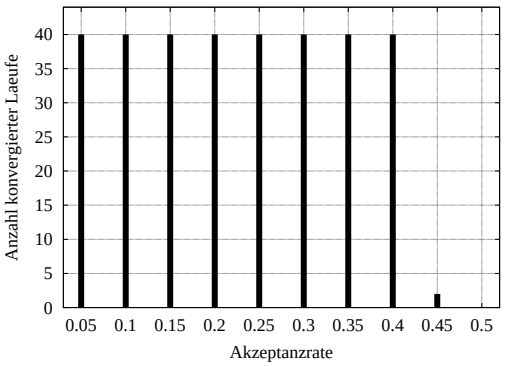
(a) Funktion f_2



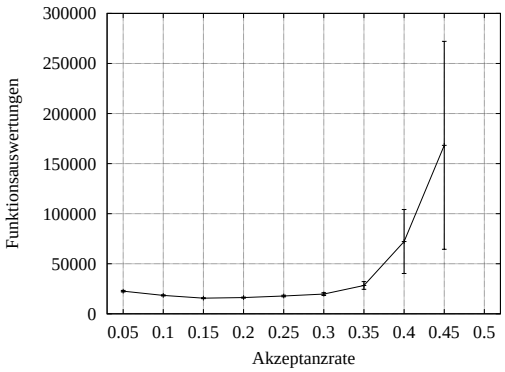
(b) Konvergenz bei Funktion f_2



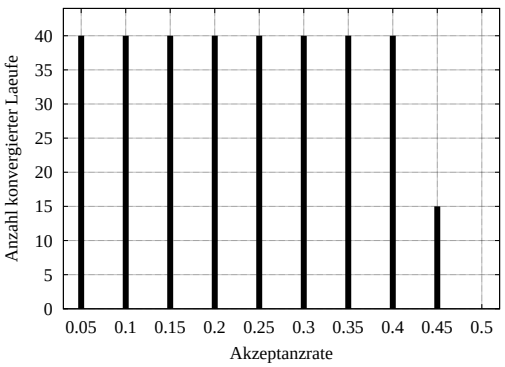
(c) Funktion f_6



(d) Konvergenz bei Funktion f_6

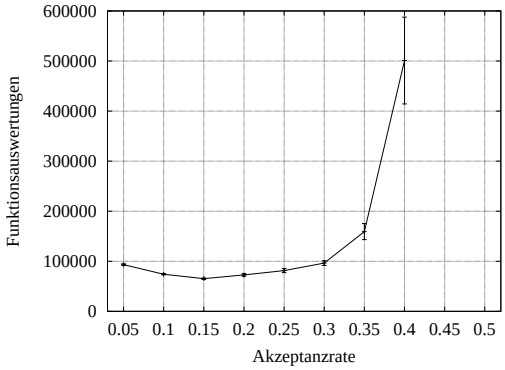


(e) Funktion f_{15}

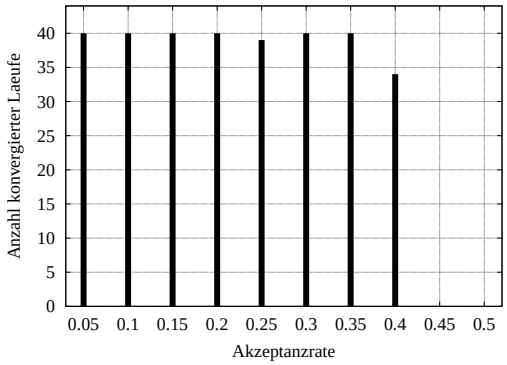


(f) Konvergenz bei Funktion f_{15}

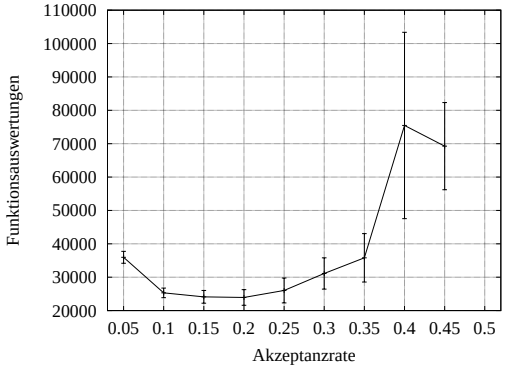
Abbildung 5.10: Einfluß der Akzeptanzrate r_p auf die Konvergenzgeschwindigkeit und die Konvergenzzuverlässigkeit.



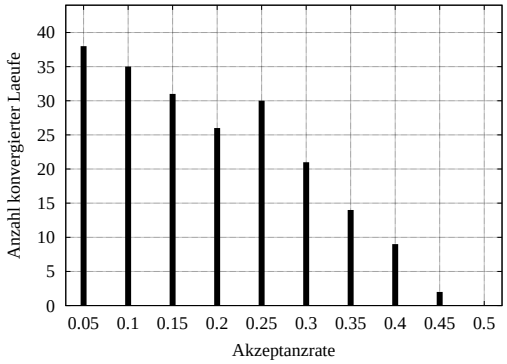
(a) Funktion q_2



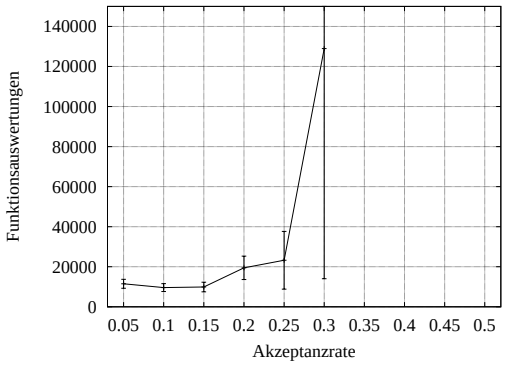
(b) Konvergenz bei Funktion q_2



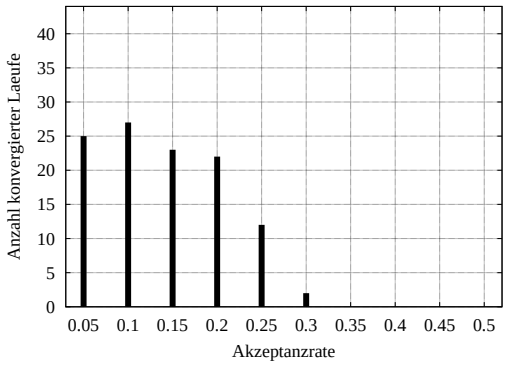
(c) Funktion f_9



(d) Konvergenz bei Funktion f_9



(e) Funktion f_{24}



(f) Konvergenz bei Funktion f_{24}

Abbildung 5.11: Einfluß der Akzeptanzrate r_p auf die Konvergenzgeschwindigkeit und die Konvergenzzuverlässigkeit.

- die Fitneßfunktionsauswertung dauert relativ lange, so daß es sich lohnt, das Modell der parallelen Fitneßevaluierung anzuwenden.

Diese Bedingungen treffen auf die im folgenden getesteten Standard-Benchmarkfunktionen nicht immer zu, die Testbedingungen bleiben aber für diesen Fall ausgelegt, da die Strategien dafür entworfen wurden.

Die getesteten Strategien sind:

$(20, 70)$ **"Komma"-ES:** $\lambda = 70$ ist ein Kompromiß zwischen einerseits kleiner Nachkommenpopulation, damit die Dauer einer Generation nicht allzu groß ist und andererseits aber ausreichend groß, um einen genügend hohen Selektionsdruck zu erreichen. Da die Komma-Strategie sich nicht sehr zur Parallelisierung eignet, wurde sie nur auf einem Prozessor gemessen.

$(20 + 1)$ **Standard Steady-State-ES:** Ersetzung des schlechtesten Individuums, wenn das neue eine bessere Fitneß besitzt.

$(20 + 1)$ **Steady-State-ES mit Median-Selektion:** Ersetzung des ältesten Individuums, Pufferlänge $n_p = 40$ und Akzeptanzrate $r_p = 0.15$.

$(20 + (1, 5))$ **Steady-State-ES mit lokaler Tournament-Selektion:** ebenfalls Ersetzung des ältesten Individuums, da dies einer Komma-ES am nächsten kommt und weil Ersetzung des schlechtesten Individuums der Standard Steady-State-Strategie zu ähnlich ist, um signifikant andere Ergebnisse zu liefern.

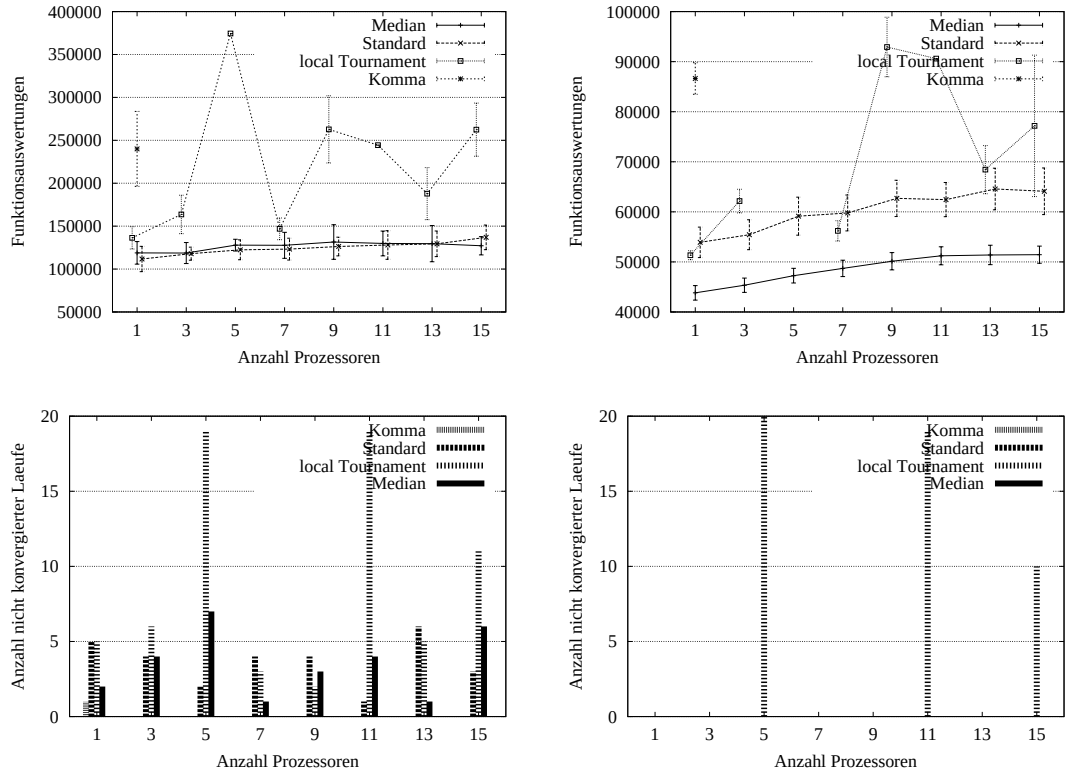
Diese Strategien wurden so einheitlich auf allen Testfunktionen verwendet, um die Tauglichkeit der Strategien ohne spezielles Parametertuning oder Meta-Optimierung zu zeigen. Die Dimension der Probleme wurde einheitlich auf $n = 20$ gesetzt, bis auf Funktion f_{24}, welche auf $n = 4$ festgelegt ist.

5.5.1 Funktion f_2: Generalized Rosenbrock's Function

Die Formel der Funktion f_2 ist in Anhang C, Gleichung C.1 aufgeführt. Außerdem wurde diese Funktion bereits für das Optimierungsbeispiel in diesem Kapitel verwendet. Die Funktion ist zwar unimodal, stellt aber aufgrund von Abhängigkeiten zwischen Variablen benachbarter Indizes besondere Anforderungen an die Selbstadaption.

Die Funktion wurde mit der Dimension $n = 20$ berechnet. Es wurde bei Erreichen des Konvergenzwertes von 10^{-20} oder nach maximal 400000 Funktionsauswertungen gestoppt.

Die Vergleichsmessungen für Funktion f_2 sind in Abbildung 5.12(a) zu sehen. Die Ergebnisse der Standard Steady-State-ES und der Steady-State-ES mit Median-Selektion



(a) Funktion f_2

(b) Funktion f_6

Abbildung 5.12: Vergleichsmessungen.

unterscheiden sich nicht relevant, Differenzen liegen innerhalb der statistischen Schwankungen. Im Vergleich zur Komma-ES, welche auf einem Prozessor durchschnittlich ca. 240000 Funktionsauswertungen benötigen, ist das Einsparungspotential durch eine Steady-State-ES enorm. Diese benötigen nur ca. 119000 (Median) bzw. 111000 (Standard) Funktionsauswertungen was eine Reduzierung auf 50% (Median) bzw. 46% (Standard) der Funktionsauswertungen im Vergleich zur Komma-ES darstellt. Dies liegt an der Abhängigkeit des Selektionsdrucks von der Populationsgröße μ bei der Komma-Selektion. Der Selektionsdruck spiegelt sich im Verhältnis $\frac{\mu}{\lambda}$ wieder. Für eine gegebene Populationsgröße μ und einen gewünschten Selektionsdruck, muß also die Größe der Nachkommenpopulation λ entsprechend hoch gewählt werden. Alle λ Individuen werden ausgehend von derselben Elternpopulation erzeugt. Innerhalb einer Generation ist also kein beliebig großer Fortschritt möglich. Ab einer gewissen Größe λ ist, abhängig von der Fitneßfunktion, effektiv keine Steigerung des Fortschritts in einer Generation mehr möglich. In diesem Fall ist es effizienter, mehrere kleine Schritte zu machen. Dies wird durch Steady-State-Algorithmen realisiert.

Implementiert man die Komma-Evolutionsstrategie mit paralleler Fitneßevaluation, so

wird man sich sofort dieses Mißstandes klar werden: Individuen werden zur Evaluation an Slaveprozessoren versendet und die Ergebnisse von ihnen empfangen. Diese müssen nun bis zur Selektion in der Nachkommenpopulation gespeichert werden. Es werden während der Generation laufend weitere Individuen aus den μ Elternindividuen erzeugt und zur Evaluierung verschickt, obwohl vielleicht schon längere Zeit ein besseres Individuum evaluiert wurde, dieses aber bis zur Selektion ungenutzt in der Nachkommenpopulation verweilt. Derselbe Effekt tritt auch in der sequentiellen Version der Komma-Evolutionsstrategie auf. Aus diesem Grund sind Steady-State-Algorithmen den generationenbasierten generell vorzuziehen. Allerdings kann durch Effekte des Selektionsdruckes die Diversität der Population verändert werden. Insbesondere bei Genetischen Algorithmen, bei denen der Crossoveroperator auf hohe Diversität angewiesen ist, kann sich dies negativ auswirken.

Der erläuterte Effekt bei der Komma-Selektion tritt auch bei den folgenden Testfunktionen auf und wird dort deshalb nicht mehr näher erörtert.

Die Steady-State-Evolutionsstrategie mit lokaler Tournamentselektion liefert zwar teilweise bessere Ergebnisse als die Komma-ES und reicht bei manchen Prozessorzahlen in die Nähe der anderen beiden Steady-State-Algorithmen, aber die Konvergenzrate und die Ergebnisse sind sehr großen Schwankungen unterworfen, sie liefert oft viel schlechtere Ergebnisse als die anderen beiden Steady-State-Algorithmen. Der Selektionsdruck des lokalen $(1, 5)$-Tournament ist wahrscheinlich groß genug, allerdings bilden die 5 Nachkommenindividuen vermutlich keinen ausreichenden großen Pool, um mit großer Wahrscheinlichkeit erfolgreiche Mutationen zu enthalten. Durch dieses Verhalten disqualifiziert sich diese Strategie für den praktischen Einsatz.

Mit steigender Prozessorenanzahl steigt bei der Standard- und bei der Steady-State-ES mit Median-Selektion die Anzahl der Funktionsauswertungen leicht an. Dies liegt an der zunehmenden Überlappung von Individuenerzeugung und Integration der evaluierten Individuen. Wird ein Individuum in die Elternpopulation integriert, können erst die ab jetzt erzeugten Individuen darauf basierend erzeugt werden und weiteren Fortschritt erzeugen. Es sind zu diesem Zeitpunkt aber auf allen Prozessoren noch Individuen in Bearbeitung, die noch auf den älteren Elternindividuen basieren und erst in der folgenden Zeit integriert werden können. Je mehr Prozessoren verwendet werden, desto größer ist dieser Überlappungseffekt. Dieser Effekt macht den Speedup durch die parallele Fitneßevaluation aber erst bei großen Anzahlen von Prozessoren zunichte (abhängig von der Dauer einer Fitneßevaluation).

5.5.2 Funktion f_6: Schwefel's Function 1.2 (Doublesum)

Die Formel der Funktion f_6 ist in Anhang C, Gleichung C.2 aufgeführt.

Die Funktion wurde mit der Dimension $n = 20$ berechnet. Es wurde bei Erreichen des Konvergenzwertes von 10^{-20} oder nach maximal 400000 Funktionsauswertungen gestoppt.

Die Vergleichsmessungen für Funktion f_6 sind in Abbildung 5.12(b) zu sehen. Die Steady-State-Evolutionsstrategie mit lokaler Tournamentselektion zeigt hier dasselbe Verhalten wie bei der Funktion f_2. Teilweise werden zwar gute Ergebnisse erzielt, aber bei manchen Prozessorzahlen konvergiert kein Lauf oder nur sehr wenige. Die Komma-Evolutionsstrategie benötigt auch hier eine deutlich größere Zahl Auswertungen als die Steady-State-Strategien. Die Steady-State-Algorithmen benötigen nur ca. 51% bzw. 62%.

Bei dieser Funktion benötigt die Steady-State-Evolutionsstrategie mit Median-Selektion (Mittelwert bei einem Prozessor: 43819) signifikant weniger Funktionsauswertungen, als die Standard Steady-State-ES (Mittelwert bei einem Prozessor: 53908). Dies ist eine Einsparung von ca. 18.7% der Funktionsauswertungen. Dies kann über alle Anzahlen von Prozessoren ungefähr gehalten werden. Die 95%-Vertrauensgrenze des Mittelwerts bei Median-Selektion und einem Prozessor liegt bei 677, d. h. mit einer Wahrscheinlichkeit von 95% weicht der gemessene Mittelwert von 43819 um maximal 677 vom tatsächlichen Mittelwert der Grundgesamtheit dieses Experimentes ab. Dies bestätigt die statistische Signifikanz der Ergebnisse.

Sowohl beim Standard Algorithmus, als auch unter Verwendung von Median-Selektion konvergieren alle Läufe.

5.5.3 Funktion f_9: Ackley's Function

Die Formel der Funktion f_9 ist in Anhang C, Gleichung C.3 aufgeführt.

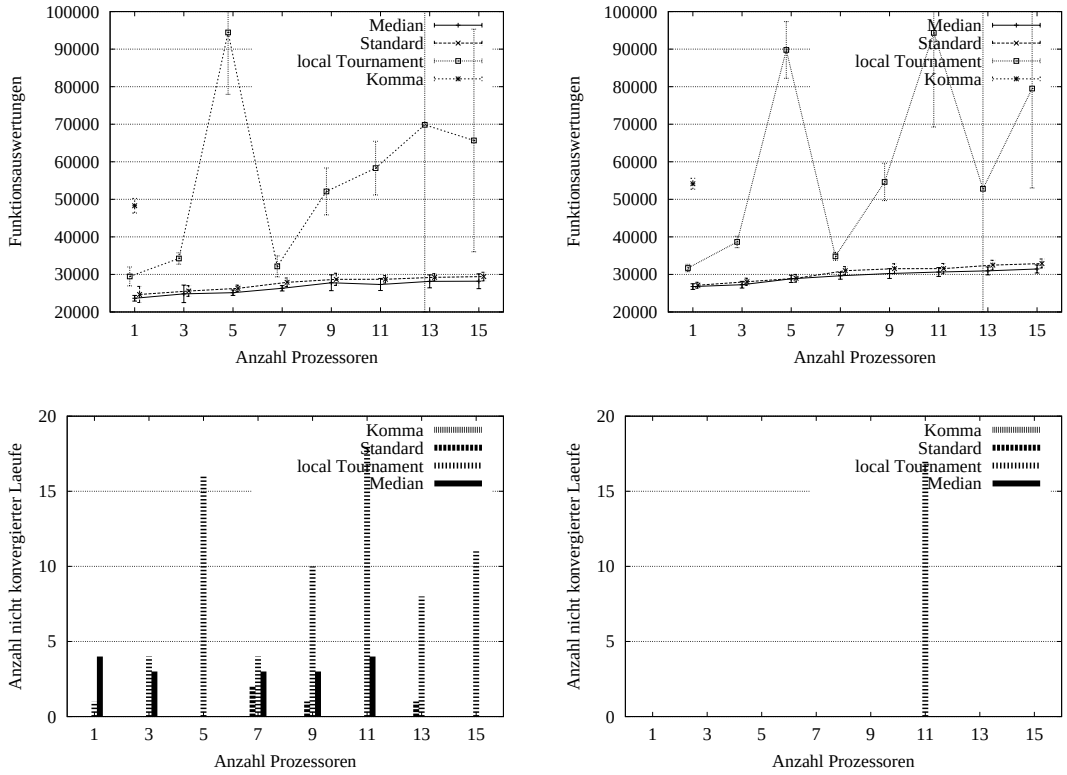
Die Funktion wurde mit der Dimension $n = 20$ berechnet. Es wurde bei Erreichen des Konvergenzwertes von 10^{-10} oder nach maximal 150000 Funktionsauswertungen gestoppt.

Die Vergleichsmessungen für Funktion f_9 sind in Abbildung 5.13(a) zu sehen. Die Evolutionsstrategie mit Median-Selektion und die Standard Steady-State-ES benötigen auch hier im Vergleich mit der Komma-ES nur halb so viele Funktionsauswertungen. Die Werte der Median-Selektion liegen durchweg etwas unter den Werten für die Standard Steady-State-ES, allerdings ist die Differenz (Median-Selektion ist ca. 4% besser als Standard Steady-State) nur bei einigen Knotenzahlen außerhalb der 95%-Vertrauensgrenzen.

5.5.4 Funktion f_{15}: Weighted Sphere Model

Die Formel der Funktion f_{15} ist in Anhang C, Gleichung C.5 aufgeführt.

Die Funktion wurde mit der Dimension $n = 20$ berechnet. Es wurde bei Erreichen des Konvergenzwertes von 10^{-20} oder nach maximal 150000 Funktionsauswertungen gestoppt.



(a) Funktion f_9

(b) Funktion f_{15}

Abbildung 5.13: Vergleichsmessungen.

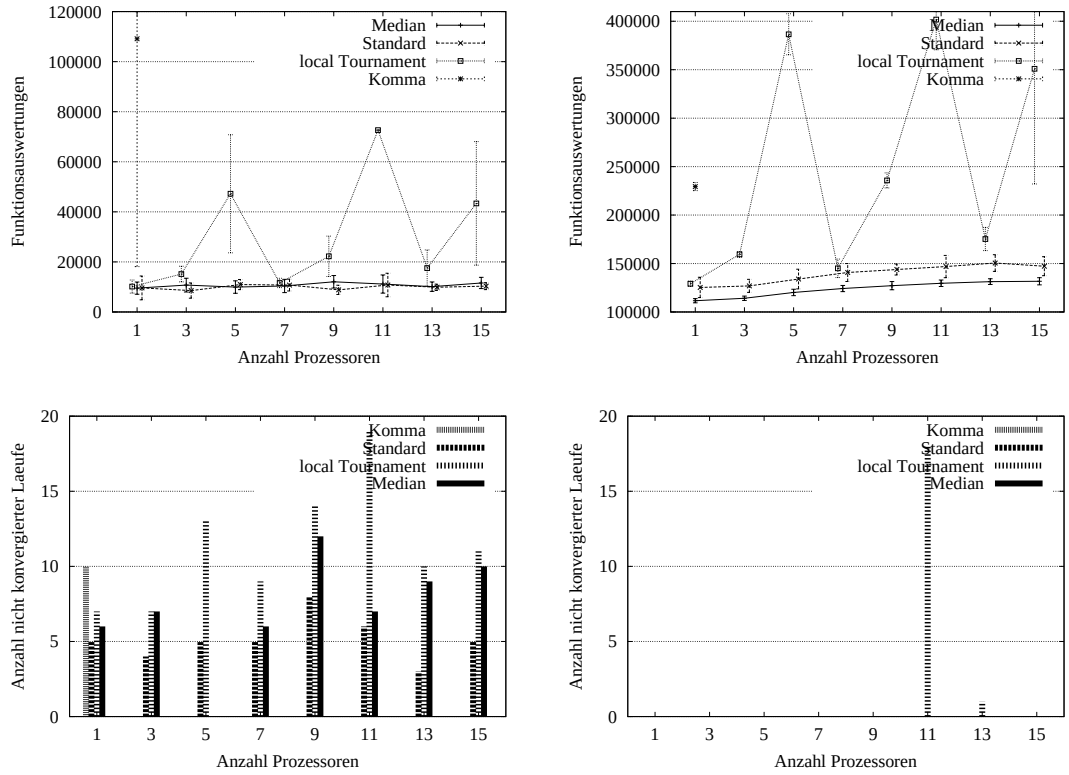
Die Vergleichsmessungen für Funktion f_{15} sind in Abbildung 5.13(b) zu sehen. Bei dieser Funktion ergibt sich ein ähnliches Bild wie bei der vorigen. Median-Selektion ist bei einigen Prozessorzahlen leicht besser als Standard Steady-State. Diese beiden Strategien benötigen auch hier im Vergleich mit der Komma-ES nur halb so viele Funktionsauswertungen.

5.5.5 Funktion f_{24}: Kowalik

Die Formel der Funktion f_{24} ist in Anhang C, Gleichung C.6 aufgeführt.

Die Funktion wurde mit der Dimension $n = 4$ berechnet. Es wurde bei Erreichen des Konvergenzwertes von $3.07486 \cdot 10^{-4}$ oder nach maximal 200000 Funktionsauswertungen gestoppt.

Die Vergleichsmessungen für Funktion f_{24} sind in Abbildung 5.14(a) zu sehen. Die Evolutionsstrategie mit Median-Selektion und die Standard Steady-State-ES liefern



(a) Funktion f_{24}

(b) Funktion q_1

Abbildung 5.14: Vergleichsmessungen.

hier effektiv dieselben Ergebnisse. Die Einsparungen zur Komma-ES sind hier besonders groß. Die Steady-State-Strategien reduzieren die Anzahl der Funktionsauswertungen im Vergleich auf unter 10%. Diese multimodale Funktion scheint der Komma-ES große Schwierigkeiten zu bereiten.

5.5.6 Funktion q_1: Achsenparalleler Hyperellipsoid

Die Formel der Funktion q_1 ist in Anhang C, Gleichung C.7 aufgeführt.

Die Funktion wurde mit der Dimension $n = 20$ berechnet. Es wurde bei Erreichen des Konvergenzwertes von 10^{-20} oder nach maximal 500000 Funktionsauswertungen gestoppt.

Die Vergleichsmessungen für Funktion q_1 sind in Abbildung 5.14(b) zu sehen. Diese Funktion wurde speziell als Testfunktion für Schrittweitenadaptionsmethoden entworfen. Deswegen zeigt sich hier ein deutlicher Vorsprung der Median-Selektion vor

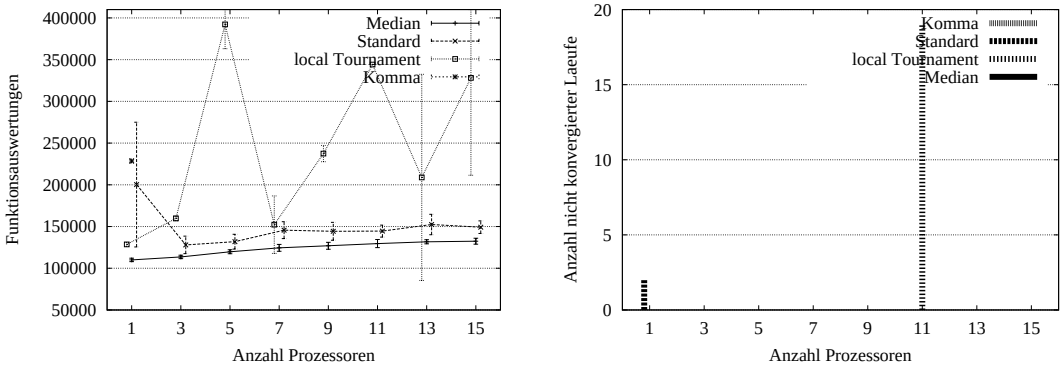


Abbildung 5.15: Vergleichsmessungen für Funktion q_2.

der Standard Steady-State-ES. Sie benötigt im Vergleich nur ca. 89% der Funktionsauswertungen. Auch hier reduzieren die Steady-State-Algorithmen im Vergleich zur generationenbasierten Komma-ES die Auswertungen auf beinahe die Hälfte.

5.5.7 Funktion q_2: Zufällig gedrehter Hyperellipsoid

Die Formel der Funktion q_2 ist in Anhang C, Gleichung C.8 aufgeführt.

Die Funktion wurde mit der Dimension $n = 20$ berechnet. Es wurde bei Erreichen des Konvergenzwertes von 10^{-20} oder nach maximal 500000 Funktionsauswertungen gestoppt.

Die Vergleichsmessungen für Funktion q_2 sind in Abbildung 5.15 zu sehen. Da diese Funktion bis auf die Drehung des Koordinatensystems dieselbe ist wie q_1, das verwendete Adaptionsverfahren CMA aber in der Lage ist, Koordinatensystemdrehungen zu kompensieren, sind die Ergebnisse qualitativ dieselben wie bei Funktion q_1. Auch hier ist die Median-Selektion gegenüber der Standard Steady-State-ES im Vorteil.

5.6 Zusammenfassung

Die Median-Selektion wurde speziell für Steady-State-Evolutionsstrategien entwickelt um die wichtige Eigenschaft der Selbstadaption zu erhalten. Der verbreitete Standard Steady-State-Algorithmus hat hier ein Defizit. In den durchgeführten Experimenten hat sich gezeigt, daß bei einigen Funktionen die Steady-State-Evolutionsstrategie mit Median-Selektion der Standard Steady-State-Evolutionsstrategie überlegen ist. Sie benötigt teilweise weniger Funktionsauswertungen, weil sie die Mutationsschrittweiten besser an die optimierte Funktion adaptieren kann.

Ein weiterer Vorteil der Median-Selektion ist die Entkopplung des Selektionsdruckes von der Nachkommenpopulationsgröße λ. Bei der (μ, λ)-Evolutionsstrategie wird der Selektionsdruck durch das Verhältnis $\frac{\mu}{\lambda}$ bestimmt, so daß bei großem μ auch λ entsprechend groß gewählt werden muß. Bei der Median-Selektion wird der Selektionsdruck hauptsächlich durch die Akzeptanzrate r_p bestimmt. Die Pufferlänge n_p kann auch kleiner als λ in einer entsprechenden Komma-Strategie gewählt werden. Dadurch kann sich die Fitneßakzeptanzgrenze schneller adaptieren.

Das als alternative Lösung getestete Verfahren der lokalen Tournament-Selektion hat sich als nicht geeignet herausgestellt. Der Selektionsdruck läßt sich hierbei zwar auch leicht einstellen, doch die im Verhältnis zur Elternpopulation sehr kleine Nachkommenpopulation reicht anscheinend nicht aus, um mit genügend großer Wahrscheinlichkeit in jedem Schritt erfolgreiche Mutationen zu enthalten.

Bei steigender Prozessorenzahl konnte aufgrund des Effekts der Überlappung der Auswertungen eine leicht ansteigende Zahl der Funktionsauswertungen bei den Steady-State-Algorithmen beobachtet werden. Dies vermindert bei steigender Prozessorenzahl - neben dem Aufwand für die Kommunikation - die Effizienz der Parallelisierung.

Es konnte gezeigt werden, daß der Einsatz von Steady-State-Algorithmen auch ohne Parallelisierung im Vergleich zur generationenbasierten (μ, λ)-Evolutionsstrategie mit den verwendeten Populationsgrößen bis zur Hälfte der Funktionsauswertungen eingespart werden können.

Insgesamt ist die Steady-State-Evolutionsstrategie mit Median-Selektion geeignet, die Standard Steady-State-Evolutionsstrategie zu ersetzen. Bei keiner der getesteten Funktionen ergaben sich dadurch relevante Verschlechterungen, bei einigen Funktionen aber deutliche Verbesserungen.

Kapitel 6

Parallele Meta-Optimierung

Ein auf dem Gebiet der Evolutionären Algorithmen schon lange bekanntes und in der Literatur oft behandeltes Problem ist die Wahl der Parametereinstellungen des evolutionären Optimierers - hier einer Evolutionsstrategie - so daß die Optimierung in möglichst kurzer Zeit zu einem möglichst guten Ergebnis kommt. Im Extremfall können falsche Parametereinstellungen sogar dazu führen, daß überhaupt keine gute Lösung gefunden wird. Speziell wenn die Optimierungsstufe in ein System eingebunden ist, in dem sie sehr oft ähnliche Instanzen derselben Problemstellung lösen muß, zahlt es sich aus, besonders gute Parameter für die Optimierungsstufe vorab zu bestimmen. In diesem Falle summiert sich der Gewinn sowohl im Hinblick auf die Lösungsqualität, als auch die Geschwindigkeit, mit der die Lösung gefunden wurde, auf.

Ein Anwender von Evolutionsstrategien ist in der Regel im Fachgebiet der speziellen Optimierungsanwendung erfahren, aber nicht notwendigerweise sehr gut mit dem Optimierungsverfahren an sich vertraut. Es ist also zumindest eine Empfehlung für gute Parametereinstellungen nötig. Wie in Abschnitt 6.1.1 angesprochen, ist es sehr fraglich, ob Standardempfehlungen auf neue Fitneßfunktionen übertragbar sind. Insbesondere ist fraglich, ob die Standardempfehlungen zu sehr guten Ergebnissen führen.

Bereits eine kanonische $(\mu/\rho^{+},\lambda)^{\gamma}$-Evolutionsstrategie beinhaltet schon die Parameter μ (Elternpopulationsgröße), ρ (Anzahl zu rekombinierender Individuen), λ (Nachkommenpopulationsgröße) und γ (Anzahl Generationen). Hinzu kommt noch die Art der Rekombination, ob diskret oder intermediär. Diese kann sogar für Objekt- und Strategievariablen unterschiedlich gewählt werden. Auch die Art der Schrittweitenadaption muß noch angegeben werden, um den Algorithmus festzulegen. Wie man z. B. in Abschnitt 4.3.1 sieht, gibt es im Programm VEES 24 Parameter, welche den Algorithmus beeinflussen[1]. Die Auswirkungen der Veränderung eines einzelnen Parameters kann qualitativ noch abschätzbar sein. Aber schon die Interaktion von nur zwei gleichzeitig veränderten Parametern kann vom Anwender nicht mit ausreichender Genauigkeit abschätzbare Auswirkungen auf die Performance des Algorithmus haben. Deshalb ist

[1] Nicht alle gleichzeitig, da einige Parameter nur in Abhängigkeit von anderen angewendet werden.

eine (möglichst automatische) Methode notwendig, die die Parameter des Optimierungsalgorithmus so einstellt, daß eine gute Performance erreicht wird.

Dies kann auf verschiedene Arten geschehen:

1. Setzen der Parameter auf Standardwerte, die z. B. aus Erfahrung oder der Literatur gewonnen wurden.
2. *Online*-Optimierung der Parameter, also dynamische Adaption während des eigentlichen Optimierungslaufes.
3. *Offline*-Optimierung der Parameter, welche während des Algorithmus statisch sind.

Die unter Punkt 2 genannte Methode soll der Klarheit halber mit *Selbstadaption*, wie bei der Mutationsschrittweite, bezeichnet werden. Die Möglichkeit unter Punkt 3 wird als *Meta-Optimierung* bezeichnet. Wird diese wiederum durch einen Evolutionären Algorithmus durchgeführt, so wird dieser als *Meta-EA* (oder *meta-level EA*) bezeichnet, der zu optimierende EA als *Basis-EA*.

Ein Meta-EA ist also ein EA, dessen Individuen jeweils einen Parametersatz mit Einstellungen (μ, λ, m, α, δ, ...) für einen Basis-EA darstellen. Bei der Evaluation wird der Basis-EA mit diesen Einstellungen gestartet. Die Fitneßfunktion des Meta-EA bewertet die Leistung des Basis-EA.

Im nächsten Abschnitt werden die drei oben genannten Möglichkeiten der Parametereinstellungen gegeneinander abgewägt und dazu einige Arbeiten aus der Literatur vorgestellt, die sich mit einem der Gebiete beschäftigen. Da sich die Meta-Optimierung als die vielversprechendste Methode herausgestellt hat, wird im restlichen Teil dieses Kapitels die Meta-Optimierung durch Meta-EAs behandelt. Da diese Problemstellung außer kontinuierlichen auch ganzzahlige und Aufzählungsparameter beinhaltet, wird auf die Parametertypen in den Abschnitten 6.3.1 und 6.3.2 eingegangen. Die Problemklasse, die sowohl Parameter mit kontinuierlichen als auch diskreten Wertebereichen beinhaltet, wird in der Literatur als *mixed-integer*-Anwendung bezeichnet. Dies wird als Erweiterung in einen Evolutionären Algorithmus integriert, der auf der Evolutionsstrategie aufbaut. Diese ES enthält erstmals alle drei der genannten Parameterarten mit Selbstadaption der Mutationsschrittweiten bzw. -Wahrscheinlichkeiten. Auch der Einsatz eines solchen Algorithmus als Meta-ES ist neu. In Abschnitt 6.3.4 wird dann die Fitneßfunktion für die Meta-ES definiert, mit der die Performance eines Basis-EA bewertet wird. Abschließend werden in Abschnitt 6.4 einige Tests an Standardfunktionen präsentiert. Weitere Testergebnisse mit komplexeren Anwendungen sind in Kapitel 7 enthalten.

Da sich der Rechenaufwand beim Meta-EA im Verhältnis zum Basis-EA potenziert, kann hier durch Parallelisierung viel Zeit gespart werden. Hierzu kann insbesondere parallele Fitneßevaluation zusammen mit der im vorigen Kapitel konstruierten

Median-Selektion verwendet werden, da die vorgestellte Meta-ES auf allen Parameterarten Selbstadaptionsmechanismen beinhaltet.

6.1 Diskussion

6.1.1 Verwendung von Standardparameterwerten

Die Verwendung von Standardparameterwerten bei EAs, die aus einer Menge von Fitneßfunktionen gewonnen wurden, führt bei einer neuen Fitneßfunktion möglicherweise zu suboptimalen Ergebnissen, da die Performance der Optimierung durch die verwendete Fitneßfunktion wesentlich beeinflußt wird. Dies wird von vielen Autoren bestätigt [Tusonn et al. 98], [Bramlette 91], [Grefenstette 86], [SV96]. Trotzdem gibt es viele Arbeiten, die sich damit beschäftigen, Empfehlungen für gute Parameterwerte für eine breite Menge von Optimierungsproblemen zu finden. Oder es werden für bestimmte Probleme theoretisch optimale Werte für Mutations- und Crossoverraten oder Populationsgrößen hergeleitet. Diese Werte gelten allerdings meist nur unter ganz bestimmten Annahmen, z. B. werden hauptsächlich binärcodierte GAs verwendet, manchmal nur mit Crossover ohne Mutation oder nur Mutation ohne Crossover. Außerdem handelt es sich fast immer um ganz bestimmte, sehr einfache Funktionen.

Die Erfahrungs- oder Literaturwerte basieren meist auf anderen Fitneßfunktionen, so daß die Ergebnisse nicht unbedingt auf eine völlig andere Anwendung übertragbar sind. Als grobe Richtlinien können diese Werte dienen, aber Verbesserungen können mit großer Wahrscheinlichkeit erzielt werden.

6.1.2 Selbstadaption

Selbstadaption wird bei Evolutionsstrategien schon als Standard für die Mutationsschrittweiten verwendet (s. 2.4.3) und war fast von Anfang an fester Bestandteil dieser Algorithmen. Dies macht auch die Stärke von ES aus, welche ohne Schrittweitenadaption kaum so leistungsfähig wären (Stichwort *Evolutionsfenster* [Rechenberg 94]). Deshalb gibt es auch einige Arbeiten, die selbstadaptive Mutationsraten bei GAs untersuchen, z. B. [Smith et al. 96] (siehe auch 5.3.1) und [Bäck et al. 96].

Außer der Mutationsrate werden als Kandidaten zur Selbstadaption in den meisten Arbeiten die Parameter Crossoverrate (beim kanonischen GA), Populationsgröße und Anwendungswahrscheinlichkeiten mehrerer verschiedener, problemspezifischer Mutations- und Crossover-Operatoren verwendet. Wobei sich problemspezifische Mutationsoperatoren meist durch unterschiedlich starke Veränderung der Individuen auszeichnen. Dadurch entspricht die zeitlich abhängige Verwendung derselben einer groben Mutationsschrittweitenregelung. Ähnlich ist es mit Crossoveroperatoren, die bei

GAs als Hauptoperator für die Erzeugung neuer, veränderter Individuen verwendet werden.

[SV96] stellen ein Verfahren zur Selbstadaption der Populationsgröße vor. In Anlehnung an die verschiedenen Spezies, die die Natur hervorgebracht hat, arbeiten sie mit konkurrierenden Subpopulationen unterschiedlicher Größe. Das relativ komplexe Adaptionsschema verwendet ein *quality criterion* um eine Subpopulation zu bewerten, ein *gain criterion*, um Populationen zu bestrafen oder zu belohnen, ein *evaluation interval* (Zeitfenster) in dem eine Population ihre Performance beweisen kann und ein *migration interval*, nach dem das global beste Individuum in alle Subpopulationen kopiert wird. Das Verfahren soll hauptsächlich bei multimodalen Funktionen Vorteile bringen, wo zu Beginn mit großen Populationen global gesucht wird und später mit kleineren Populationen eher lokal entlang des Gradienten gesucht wird.

[Schnecke et al. 96] arbeiten auch mit mehreren, explizit parallelen Populationen, allerdings mit gleicher, fester Populationsgröße. Die adaptierten Parameter sind hier die Crossover- und Mutationsrate, der Schwellwert für truncation selection und die Anwendungswahrscheinlichkeiten für die verschiedenen, auf die abstrakte Lösungsrepräsentation angepaßten Mutations- und Crossoveroperatoren. Alle 30 Generationen werden die Populationen bewertet und die Parameter jeder Population an diejenigen der nächstbesseren angepaßt. Die Parameter der besten Population werden verstärkt, d. h. wenn dort eine relativ kleine Crossoverrate verwendet wird, so wird sie noch mehr verkleinert. Allerdings geben die Autoren nicht an, wie festgestellt wird, was eine relativ kleine Crossoverrate ist. Auch wie die Anpassung der Parameter stattfindet, wird nicht genannt.

[Tusonn et al. 98] beschäftigen sich grundlegender mit Selbstadaption und der Frage, wann sie lohnend angewendet werden kann. Für jegliche Selbstadaption ist es nötig, die Leistung des adaptierbaren Operators zu bewerten, um damit ein Feedback zu erhalten, den Operator anzupassen. Dazu fordern sie drei Voraussetzungen:

- Die Bewertung der Operator-Leistung muß eine korrekte Anpassung des Operators anzeigen.
- Die Operator-Leistung muß einen klaren Bezug zu den Parametern des Operators haben, so daß genügend Feedback zur Adaption des Parameters geliefert wird.
- Das Adaptionsverfahren muß leistungsfähig genug sein, während der Optimierung schnell genug die richtige Art der Parameteränderung herauszufinden. Der Zusatzaufwand für die Adaption darf den potentiellen Nutzen nicht zunichte machen.

Ein Zusatzaufwand ist z. B. bei der Selbstadaption der Mutationsschrittweiten durch das Ausprobieren größerer und kleinerer Schrittweiten vorhanden, obwohl letztlich eines von beiden in der konkreten Situation vorzuziehen ist. Im schlimmsten Fall kann

ein selbstadaptiver Algorithmus sogar schlechter sein als ein Algorithmus mit konstanten Parametern. Sie halten es für schwieriger, einen guten zeitabhängigen Verlauf eines Parameters zu finden, als einen guten konstanten Wert. Sie bewerten dynamische Parameteradaption nicht als universell nützlich, da nicht unbedingt bessere Ergebnisse als durch handoptimierte Parametereinstellungen erzielt werden. Die drei Voraussetzungen sind teilweise schwer zu erfüllen. Dennoch bezweifeln sie nicht, daß die effektivsten Operatoren solche sind, die ihre Parameter über die Zeit verändern.

Die Nachteile der Selbstadaption sind:

- Sie beginnt bei jedem Optimierungslauf von neuem und die Suche verbraucht somit auch jedesmal Ressourcen und verlangsamt damit den Algorithmus gegenüber einem optimalen zeitlichen Parameter-Verlauf.
- Die Parametereinstellung am Ende der Optimierung kann auch nicht unbedingt als Startwert für weitere Optimierungen benutzt werden. Betrachtet man beispielsweise die Schrittweitenadaption bei Evolutionsstrategien, so wird diese gegen Ende der Optimierung meist kleine Schrittweiten bevorzugen. Diese wären aber am Anfang einer Optimierung sehr ungünstig, da hier der Lösungsraum eher breiter durchsucht werden muß.
- Es muß ein geeignetes Adaptionsschema gefunden werden, das meist speziell auf den Parameter zugeschnitten ist.

Aus der Mutationsschrittweitenadaption ist bekannt, daß eine einfache mutative Regelung mit separaten Schrittweiten pro Objektvariable nicht mit kleinen Populationen funktioniert [Schwefel 87]. [Ostermeier et al. 94] schätzt dafür Populationsgrößen im Bereich von $10 \cdot n$, wobei n die Problemdimension darstellt. Abhilfe schaffen hier nur komplexere Verfahren. Was in den oben genannten Arbeiten noch nicht untersucht wurde, ist die Frage, ob für die Selbstadaption der dort genannten Parameter, wie z. B. Operatorwahrscheinlichkeiten, ebenso eine bestimmte Mindestpopulationsgröße benötigt wird. Die Zahl der parallelen Populationen wurde dort pragmatisch gewählt.

Auch Mühlenbein stellt die Leistung von Selbstadaption in Frage. Er benutzt als Beispiel das einfache Onemax-Problem[2]. Die theoretisch beste Mutationsrate würde sich über die Zeit verringern und ist abhängig von der Zahl (noch falscher) 0-Bits im String. Diese Zahl ist aber im praktischen Fall nicht bekannt, da die Fitneßfunktion als Black-Box angesehen wird. Laut Mühlenbein ist der Unterschied zwischen fester und variabler Mutationsrate bei diesem Problem sehr gering.

Bei der Selbstadaption müssen die momentanen Parameterwerte eine gewisse Zeit lang getestet werden. In der Literatur werden hierbei sehr unterschiedliche Intervalle verwendet: bei [SV96] sind es nur 4 Generationen, [Schnecke et al. 96] verwendet 30

[2] auch Bitsum genannt. Es geht dabei um die Maximierung der Anzahl der 1-Bits in einem Binärstring einer bestimmten Länge l.

Generationen und [Tusonn et al. 98] ein Gap G von 200 bis zu 2000 Generationen. Hier stellte sich dieser Parameter bei drei der fünf Testprobleme als robust heraus, bei zwei der Testprobleme jedoch hatte dieser einen Einfluß auf die Adaptionsfähigkeit. Einen größeren Einfluß hatten dort die Anfangswerte für die Operatorwahrscheinlichkeiten.

[Grefenstette 86] ist der Meinung, daß die beschränkte Zahl der Funktionsauswertungen keine signifikante Änderungen an der Suchstrategie zulassen würde. Das Testintervall muß lange genug sein, um ausreichend Feedback über die Tauglichkeit des Operators zu liefern. Da dies in einer einzigen Population kaum möglich ist, werden aus diesem Grunde oft mehrere, meist auf verschiedenen Prozessoren parallel berechnete Populationen verwendet.

Die Selbstadaption vieler interagierender Parameter ist also fraglich und muß evtl. auch für jeden Parameter unterschiedlich realisiert werden. Die Arbeiten auf diesem Gebiet haben auch nur einen oder relativ wenige Parameter adaptiert. Inwieweit die adaptiv gewichtete Verwendung mehrerer Crossoveroperatoren als unterschiedliche Operatoren zu zählen sind, ist nicht klar.

Aus all den hier genannten Gründen wurde in dieser Arbeit die vorher schon genannte Meta-Optimierung gewählt, welche im nächsten Abschnitt behandelt wird.

6.1.3 Meta-Optimierung

Eine Meta-Optimierung der Parameter der Evolutionsstrategie kann grundsätzlich mit jedem geeigneten Optimierungsverfahren durchgeführt werden. Evolutionäre Algorithmen bieten sich hierfür aber aus den folgenden Gründen besonders an:

- der Suchraum ist höchstwahrscheinlich multimodal und besitzt viele lokale Optima,
- die Fitneßfunktion ist nicht als mathematischer Term formuliert und deshalb auch nicht ableitbar,
- besonderes Vorwissen über den Suchraum, welches sich ausnutzen ließe, ist nicht viel bekannt,
- die Fitneßfunktion ist verrauscht, d. h. sie liefert Werte, welche stochastischen Schwankungen unterworfen sind,
- es werden Parameter unterschiedlicher Art vermischt (kontinuierliche und ganze Zahlen, ungeordnete Aufzählungen).

Im nächsten Abschnitt werden zunächst einige Arbeiten über Meta-Optimierung vorgestellt. Dabei wurden bisher fast ausschließlich Genetische Algorithmen verwendet.

Für die zu optimierenden Parameter wurden dabei immer nur relativ kleine Wertebereiche zugelassen (z. B. maximal 16 verschiedene Werte). Diese Beschränkung wird in dieser Arbeit aufgehoben, indem eine Evolutionsstrategie als Meta-EA verwendet wird, die um die Fähigkeit erweitert wird, neben kontinuierlichen auch ganzzahlige und Aufzählungsparameter zu handhaben. Es werden für alle drei Parameterarten Selbstadaptionsmechanismen verwendet, um den vergrößerten Lösungsraum auch effizient durchsuchen zu können. Arbeiten, die einen selbstadaptiven Optimierungsalgorithmus für alle drei Parameterarten vorstellen, sind dem Autor nicht bekannt. Auch die Meta-Optimierung einer Evolutionsstrategie ist neu.

6.2 Verwandte Arbeiten über Meta-Optimierung

In der folgenden Vorstellung von Arbeiten aus dem Gebiet der Meta-Optimierung wird auf die hier dargelegten Punkte besonderen Wert gelegt:

- die Art des Meta-EA (GA/ES/ . . .),
- die Art des Basis-EA (GA/ES/ . . .),
- die evolvierten Parameter des Basis-EA und deren Wertebereich,
- die Anwendung, welche mit dem Basis-EA optimiert wird,
- die Wahl der Parameter für den Meta-EA,
- die Fitneßfunktion für den Meta-EA.

[Grefenstette 86] ist die erste Arbeit über Meta-GAs (laut [Cicirello et al. 00]). Sowohl für den Basisalgorithmus, als auch auf der Meta-Ebene wird ein GA verwendet. Die evolvierten Parameter des Basis-GA sind: Populationsgröße $N \in \{10, \ldots, 160\}$ in 10er-Schritten, Crossoverrate $C \in \{0.25, \ldots, 1.00\}$ in Schritten von 0.05, Mutationsrate M mit 8 Werten von 0.0 bis 1.0 mit exponentiellem Verlauf, Generation Gap (Anteil der Population, der ersetzt wird) $G \in \{0.3, \ldots, 1.0\}$ in Schritten von 0.1, Scaling Window W mit 8 Werten (Faktor bei der prop. Selektion), Selektionsmethode $S \in \{\mathrm{pure}, \mathrm{elitist}\}$. Insgesamt ergeben sich 2^{18} Parameterkombinationen. Der Basis-GA wird auf 5 verschiedene Funktionen jeweils 5000 Auswertungen lang angewendet. Dafür wird wahlweise die online- oder offline-Performance mit der Performance von Zufallssuche auf der Funktion gemittelt. Zum Schluß des Meta-Laufes wird das beste Individuum jeder Generation (insgesamt 20) nochmals über 5 verschiedene Anfangswerte des Zufallszahlengenerators gemessen und daraus der beste ausgewählt. Der Meta-GA hat eine Populationsgröße von 50 und läuft 20 Generationen lang, wodurch sich 1000 Läufe des Basis-GA ergeben. Der Meta-GA selbst wird mit Standardparametern versorgt.

Als Ergebnis bringt der Meta-GA mit online-Performance eine 3.09%ige Verbesserung (statistisch relevant) des GA-Ergebnisses im Vergleich zum GA mit Standardparametern. Bei offline-Performance ergibt sich eine 3.0%ige Verbesserung, allerdings nicht statistisch relevant.

Auch in [Cicirello et al. 00] wird ein Meta-GA verwendet, um gute Steuerparameter für einen Basis-GA herauszufinden. Um die Zahl der Basis-GA-Läufe zur Performancebestimmung zu reduzieren, verwenden sie ein Neuronales Netz, welches die Fitneß von Parametersätzen schätzt. Dazu wird das Netz anhand zufällig erzeugter Parametersätze trainiert. Das Training des Netzes findet vor der Meta-Optimierung statt und der Meta-GA verwendet dann ausschließlich die vom Netz gelieferten Fitneßwerte. Der Aufwand zum Training des Netzes ist 1000 Läufe des Basis-GAs plus 250 Läufe als Validierungssatz. Der Meta-GA benötigt aber 5000 Evaluierungen, wodurch sich eine Beschleunigung von Faktor 4 durch die Verwendung des NNs ergibt.

Die evolvierte Parametermenge besteht aus: P - Populationsgröße, C - Crossoverrate, U - Crossoverwahrscheinlichkeit, M - Mutationsrate, T - Stopkriterium (anhand der Diversität) und S - anwendungsspezifische Operatorwahrscheinlichkeit. Als Anwendung wird das *largest common subgraph* Problem verwendet (ein NP-vollständiges Problem, welches in einigen geometrischen Anwendungen auftritt). Die Fitneß des Parametersatzes eines GA stellt die Fitneß der besten gefundenen Lösung des Basis-GA dar, welche wiederum die Größe des gefundenen Graphen ist (Maximierungsproblem). Alle Parameter werden binär mit 4 Bit dargestellt, wodurch 16 verschiedene Werte zustandekommen. Die Populationsgröße kann dabei die Werte 10 bis 160 in 10er-Schritten annehmen. Mutations- und Crossoverrate werden als 2^{-b} interpretiert, wobei b die dekodierte Binärzahl ist. Ein Individuum hat eine Gesamtlänge von 24 Bit, wodurch sich 2^{24} mögliche Parameterkombinationen ergeben. Von den drei zufällig aus der letzten Population gezogenen Parametersätze ist keiner in der Trainingsmenge des NN enthalten. Diese Sätze führen alle zu einer besseren Lösung des Basis-GA als von Hand angepaßte Parameter. Letztere finden die Lösung allerdings schneller.

[Bäck 96] widmet ein Kapitel seines Buches der Meta-Evolution (siehe auch [Bäck 94a]). Er hat den Eindruck, daß die Parametereinstellungen für EAs meist auf Erfahrungswerten beruhen und nur zum kleinen Teil auf theoretischen Erkenntnissen (z. B. die Mutationsrate im GA). Er weist darauf hin, daß Fogel [Fogel et al. 91] den Begriff *Meta-EP* für die Selbstadaption von Mutationsschrittweiten benutzt, dies ist aber nicht das, was man üblicherweise (und auch in dieser Arbeit) darunter versteht. Ganzzahlige und kontinuierliche Parameter werden oft getrennt optimiert, was aber Separabilität voraussetzt. Als Meta-EA verwendet Bäck eine Evolutionsstrategie mit MSR mit separaten Schrittweiten plus Erweiterungen: bei ganzzahligen Parametern wird der Mutationsvektor $\vec{\Delta}$ (aus Gleichung 2.27) vor der Addition auf eine ganze Zahl abgerundet. Die Anfangsschrittweite für ganzzahlige und kontinuierliche Parameter beträgt jeweils $1/60$ des Wertebereichs. Aufzählungsparameter werden mit einer konstanten Mutationswahrscheinlichkeit neu aus dem Wertebereich gewählt. Als Fitneßfunktion des Meta-EA kommt für ihn die Konvergenzzuverlässig-

keit oder -geschwindigkeit in Frage. Ersteres sei aber zu schwer und allgemein nicht lösbar, daher wird die Konvergenzgeschwindigkeit in der Form verwendet, daß die Fitneß des Meta-EA gleich der Fitneß des besten Individuums der letzten Population des Basis-EA ist. Als Basis-EA wird ein GA mit folgenden Parametern verwendet: Crossoverwahrscheinlichkeit $p_c \in [0,1]$, Mutationsrate $p_m \in [0,0.5]$, maximaler Erwartungswert $\eta^+ \in [1,2]$ für die lineare Ranking-Selektion, Tournamentgröße $q \in \{1,\ldots,20\}$ für die Tournament-Selektion, Elternpopulationsgröße $\mu \in \{1,\ldots,100\}$, Nachkommenpopulationsgröße $\lambda \in \{1,\ldots,100\}$, Anzahl der Crossoverpunkte $z \in \{1,\ldots,8\}$, Selektions-Operator Proportional/Linear Ranking/Tournament/(μ,λ), Rekombinationsoperator z-Punkt/uniform/diskretes/kein Crossover, Elitist-Flag e. Hierbei ist zu beachten, daß z. B. abhängig vom Crossoveroperator andere Parameter wie die Anzahl der Crossoverpunkte wirkungslos sein können. Dadurch wird effektiv die Dimension des Problems beeinflußt. Variablen werden bei Überschreiten der Grenzen des Wertebereiches auf die Grenzen zurückgesetzt (wie bei *clip*, s. Abschnitt 4.3.3). Der Basis-GA darf maximal 10^4 Funktionsauswertungen machen und wird wahlweise anhand seiner online- oder offline-Performance bewertet. Ein Parametersatz wird jeweils zweimal mit unterschiedlichen Werten des Zufallszahlengenerators evaluiert und das Ergebnis zum Fitneßwert gemittelt.
Der Meta-EA wird wegen des hohen Rechenaufwands nach dem Modell der parallelen Fitneßevaluation auf mehreren Prozessoren ausgeführt. Der Meta-EA führt eine $(4,30)^{50}$-Strategie aus, wobei er 1504 Läufe des Basis-GA ausführt. Die Parameter des Meta-EA wurden auch hier nach Erfahrungswerten gewählt, da diese weitaus robuster seien als die des Basis-GA.

In [Bramlette 91] wird ein stark modifizierter GA vorgestellt, der sowohl als Basis-GA, wie auch als Meta-GA verwendet wird. Dabei wird auf die Binärdarstellung verzichtet und stattdessen auf Integerwerten operiert. Entsprechend werden angepaßte Crossover- und Mutationsoperatoren (exogen gesteuerte Mutation, mit geringer werdenden Werten) verwendet. Eine weitere Modifikation findet bei der Initialisierung statt, wo eine Zufallssuche verwendet wird, außerdem gibt es eine Multistart-Option. Insgesamt ist das Ziel, beim Basis-GA mit möglichst wenig Funktionsauswertungen auszukommen (max. 1000). Die Fitneßfunktion des Basis-GA ist eine 10-dimensionale, ganzzahlige Funktion.
Die evolvierten Parameter sind: drei Parameter, die zu den Modifikationen gehören, Populationsgröße in Bereich $\{4,\ldots,100\}$, Anzahl Generationen, Mutationswahrscheinlichkeit, Crossoverwahrscheinlichkeit und die Zahl der Wiederholungen bei Multistart, wenn die max. Zahl Funktionsauswertungen noch nicht erreicht wurde. Die Fitneßfunktion für den Meta-GA bildet aus den 100 besten Individuen von 100 Läufen des Basis-GA den Durchschnittswert. Die Parameter für den Meta-GA wurden auch hier auf Erfahrungswerte gesetzt (u. a. Populationsgröße 30 Individuen, 200 Generationen). Ein Übertragen der für den Basis-GA gefundenen Parameter auf den Meta-GA brachte keinen Erfolg. Daraus wurde eine starke Abhängigkeit der Parameter einerseits von der zu optimierenden Fitneßfunktion und andererseits von der Anzahl der

Funktionsauswertungen geschlossen.

In [Caldwell et al. 91] wird ein Meta-GA verwendet, um die Crossover- und Mutationsrate für einen Basis-GA zu finden der beim praktischen Einsatz dann mit diesen Parametern laufen soll. Fitneßfunktion des GAs ist eine interaktive Funktion zur Generierung von Phantombildern von Personen. Nähere Angaben zum Meta-GA werden nicht gemacht.

6.3 Die Meta-Evolutionsstrategie

Der Meta-EA wurde in dieser Arbeit durch eine erweiterte Evolutionsstrategie realisiert. Durch die Selbstadaption kann der Wertebereich der Parameter ohne Einschränkungen gewählt werden, da sich die Suche nach einiger Zeit auf lohnende Bereiche konzentrieren kann und der Suchbereich selbstständig verfeinert wird. Es sollen dann z. B. nicht nur 10er-Schritte in der Populationsgröße zugelassen werden, sondern jede ganze Zahl.

Die Evolutionstrategie ist jedoch zunächst nur zur Optimierung kontinuierlicher Variablen ausgelegt. Die Variablen der Meta-ES sind aber die Parameter der Basis-ES. Diese können vom folgenden Typ sein:

kontinuierliche Parameter: z. B. die Adaptionsrate α, die Anfangsschrittweite δ.
Diese Parameter können Werte aus dem reellen Zahlenraum $\boldsymbol{R}$ annehmen. Da Evolutionsstrategien speziell für kontinuierliche Parameter entworfen wurden, sind hierfür keine speziellen Mechanismen erforderlich.
Die kontinuierlichen Parameter werden hier mit x_i $(0 < i < n_x)$ bezeichnet. n_x ist dabei die Anzahl der kontinuierlichen Parameter.

ganzzahlige Parameter: z. B. die Populationsgrößen μ und λ, die Anzahl der zu rekombinierenden Individuen ρ.
Diese Parameter können Werte aus der Menge der ganzen Zahlen $\boldsymbol{Z}$ annehmen. Es existiert ein sinnvolles Distanzmaß zwischen den Werten, so daß eine Mutationsschrittweite definiert werden kann.
Die ganzzahligen Parameter werden hier mit d_i $(0 < i < n_d)$ bezeichnet. n_d ist dabei die Anzahl ganzzahliger Parameter.

Aufzählungsparameter: z. B. die Selektionsarten 'Plus' oder 'Komma', die Rekombinationsarten keine/intermediär/dominant/kontinuisiert.
Diese Parameter bestehen aus einer ungeordneten Menge von Werten. In der Regel handelt es sich um eine relativ kleine Menge. Es kann kein sinnvolles Distanzmaß und keine Ordnung auf den Werten definiert werden. Deshalb ist hier ein spezielles Mutations- und Adaptionsschema notwendig.

Im Folgenden werden die Parameter mit den nachstehenden Symbolen bezeichnet:

$$\begin{array}{lll} \text{kontinuierliche Parameter :} & x_i \in \mathbb{R} & \text{mit } 0 \leq i \leq n_x \\ \text{ganzzahlige Parameter :} & d_i \in \mathbb{Z} & \text{mit } 0 \leq i \leq n_d \\ \text{Aufzählungsparameter :} & a_i \in \{a_{i,1}, \ldots a_{i,n_{a,i}}\} & \text{mit } 0 \leq i \leq n_a \end{array} \tag{6.1}$$

Die Meta-ES muß also erweitert werden, so daß neben den vorhandenen kontinuierlichen Parametern auch ganzzahlige und Aufzählungsparameter optimiert werden können. Da die Selbstadaption die wichtigste Eigenschaft der Evolutionsstrategie ist, soll auch für die beiden neuen Parameterarten ein Selbstadaptionsmechanismus realisiert werden.

Die ursprüngliche Version der Meta-ES [Scheer 99, Koch et al. 00] stützte sich auf die Arbeit von [Bäck 96], der ganzzahlige Variablen in Form von kontinuierlichen Variablen repräsentiert, die vor der Anwendung gerundet werden. Außerdem existiert für Aufzählungsparameter eine feste Mutationsrate, wie in einem GA. Soll ein Aufzählungsparameter mutiert werden, so wird aus dem Wertebereich ein zufälliger Wert neu gewählt.

In dieser Arbeit wird für ganzzahlige Parameter eine spezielle Mutationsverteilung verwendet, die [Rudolph 94] vorschlägt. Auch für diese Verteilung kann ein Selbstadaptionsmechanismus realisiert werden. Für Aufzählungsparameter existiert eine Mutationsrate, die nach der Methode von [Bäck et al. 95] adaptiert wird. Die Meta-ES unterscheidet sich damit von bisherigen Arbeiten in den folgenden Punkten:

- Als Basis-EA wird eine ES verwendet.
- Als Meta-EA wird eine ES verwendet. Dies ist sonst nur bei Bäck der Fall.
- Die Meta-ES wird um ganzzahlige Parameter erweitert.
- Der Wertebereich der kontinuierlichen und ganzzahligen Parameter ist in der Genauigkeit nicht eingeschränkt.
- Erstmals werden alle drei Parameterarten in einer ES vereint. [Bäck et al. 95] verwendet nur kontinuierliche und Aufzählungsparameter, er setzt sie auch nicht zur Meta-Optimierung ein.
- Alle drei Parameterarten sind selbstadaptiv.

Die nächsten beiden Abschnitte 6.3.1 und 6.3.2 erläutern nun die Mutations- und Selbstadaptionsmechanismen für ganzzahlige und Aufzählungsparameter.

6.3.1 Behandlung von ganzzahligen Parametern

Um mit einer Evolutionsstrategie ganzzahlige Variablen zu evolvieren, gibt es zunächst zwei naheliegende Möglichkeiten:

1. Die Variablen werden als kontinuierlich gespeichert und erst vor der Auswertung der Fitneßfunktion auf ganze Werte gerundet.

2. Die Variablen $\vec{x}$ werden als ganzzahlig gespeichert, aber vom Mutationsvektor $\vec{\Delta}$ werden vor Addition die Nachkommastellen abgeschnitten:

$$\vec{x}' = \vec{x} + \vec{\Delta} \tag{6.2}$$

   Dies wird in [Bäck et al. 95] auf diese Art realisiert.

Diese beiden Methoden wirken sich wahrscheinlich unterschiedlich auf die Selbstadaption aus. Bei Methode 1 können auch mehrere aufeinanderfolgende Mutationen mit Änderungen kleiner als 1 eine Änderung der Variablen bewirken. Bei Methode 2 kann nur eine Mutation um mindestens 1 die Variablen ändern und sich somit selektionsrelevant auswirken.

[Rudolph 94] stellt fest, daß die benutzte Mutationsverteilung üblicherweise an die Art des Suchraumes angepaßt ist. Für kontinuierliche Suchräume wird die Normalverteilung verwendet und wenn der Suchraum aus binären Zeichenketten besteht, wird oft eine Binominalverteilung verwendet. Daher erscheint es sinnvoll, auch für den ganzzahligen Raum eine angepaßte Mutationsverteilung zu benutzen. Aus mehreren Möglichkeiten sucht er eine auf der geometrischen Verteilung basierende Verteilung heraus, die maximale Entropie besitzt. Dies bedeutet, daß sie so nahe an der Gleichverteilung orientiert ist, wie möglich, jedoch ohne andere geforderte Eigenschaften wie z. B. Symmetrie zu verlieren.

Die Experimente von Rudolph zeigen, daß die benutzte Verteilung auf den fünf getesteten, ganzzahligen Optimierungsaufgaben sehr schnell arbeitet und in 80% der Fälle in der Lage ist, das globale Optimum zu finden. Daher wird in dieser Arbeit diese Mutationsverteilung benutzt.

Die von Rudolph vorgeschlagene Möglichkeit ist, den Mutationsvektor als Differenz zweier unabhängig erzeugter, geometrisch verteilter Zufallszahlen zu bilden:

$$\vec{\Delta} = G_1 - G_2 \tag{6.3}$$

Eine geometrisch verteilte Zufallszahl G läßt sich mittels einer Transformation aus einer im Intervall $[0, 1)$ gleichverteilten Zufallszahl U berechnen:

$$G = \left\lfloor \frac{\log(1-U)}{\log(1-p)} \right\rfloor \tag{6.4}$$

wobei p ein Parameter der Verteilung ist, welcher sich aus der gewünschten mittleren Schrittweite S berechnen läßt:

$$\begin{array}{rcl} S & = & n \cdot \frac{2(1-p)}{p(2-p)} \\ \Leftrightarrow \quad p & = & 1 - \frac{S/n}{\sqrt{1+(S/n)^2}+1} \end{array} \tag{6.5}$$

Die Schrittweite S ist der Erwartungswert der Verteilung des Mutationsvektors $\vec{\Delta}$:

$$S = E[\left\| \vec{\Delta} \right\|_1] \tag{6.6}$$

Die Norm $\|\|_1$ wird l_1-Norm genannt und ist definiert als:

$$\|k\|_1 = \sum_{i=1}^{n} |k_i| \tag{6.7}$$

Die Schrittweite S wird nach unten auf den Wert 1 begrenzt, da kleinere Schrittweiten im Ganzzahlbereich nicht sinnvoll sind.

6.3.2 Behandlung von Aufzählungsparametern

Auf den Aufzählungsparametern a_i besteht keine Ordnung und es kann kein Abstandsmaß definiert werden. Deshalb ist es sinnvoll, bei einer Mutation jeden beliebigen Wert mit gleicher Wahrscheinlichkeit zu wählen. Ob ein Aufzählungsparameter mutiert wird, wird von einer Mutationsrate p_j ($1 \leq j \leq n_p$) festgelegt. Die Mutation wird folgendermaßen durchgeführt:

$$a_i' = \begin{cases} a_i & \text{falls} \quad u_i > p' \\ \widetilde{X_i} & \text{falls} \quad u_i \leq p' \end{cases} \tag{6.8}$$

wobei u_i eine gleichverteilte Zufallszahl $U(0,1)$ ist und $\widetilde{X_i}$ eine im ganzzahligen Indexbereich von d_i verteilte Zufallszahl, hierfür wird eine Gleichverteilung verwendet.

Die Mutationsrate muß aber nicht fest sein, sondern kann ebenfalls adaptiert werden.

[Bäck et al. 95] benutzt zur Adaption der Mutationsrate p_j eine logistische Transformation, die von [Obalek 94] vorgeschlagen wurde:

$$p_j' = \left(1 + \frac{1-p_j}{p_j} \cdot e^{-\tau \cdot N_j}\right)^{-1} \tag{6.9}$$

$$1 \leq j \leq n_w$$

mit N_j als $N(0,1)$-normalverteilte Zufallsvariable. $1 \leq n_w \leq n_d$ ist die Anzahl der Schrittweiten für den diskreten Teil (maximal n_d, was die Anzahl der diskreten Parameter darstellt).

Es wird in der Implementierung eine Begrenzung durchgeführt, so daß P_j aufgrund der Rechengenauigkeit des Computers nicht auf 0 oder 1 konvergiert. Es gilt: $p'_j := \varepsilon_p$ falls $p'_j < \varepsilon_p$ und $p'_j := 1 - \varepsilon_p$ falls $p'_j > 1 - \varepsilon_p$. Dabei ist $\varepsilon_p = 10^{-40}$. Laut [Bäck et al. 95] wirkt sich dies nicht auf die Verteilungsdichtefunktion aus. Es wird eine Lernrate $\tau \approx \frac{1}{\sqrt{2\sqrt{n_a}}}$ vorgeschlagen, wobei n_a die Länge des Vektors der diskreten Variablen ist. Die Autoren schreiben allerdings, daß der Adaptionsprozeß nur für $n_w = 1$ gut funktioniert, für größere n_w gibt es Probleme. Deshalb wird hier auch nur $n_w = 1$ verwendet.

6.3.3 Selbstadaption des gemischten Vektors

Zur Selbstadaption der Mutationsschrittweiten für die ganzzahligen Parameter wird eine mutative Schrittweitenregelung mit separaten Schrittweiten wie in Abschnitt 2.27 benutzt. Dabei werden die Parameter τ_1 und τ_2 nach den Gleichungen 2.28 und 2.29 benutzt. Die Gesamtdimension $n = n_x + n_g$ ist jetzt die Summe der Anzahl der reellwertigen und ganzzahligen Parametern. Die Anfangsschrittweite für die kontinuierlichen und die ganzzahligen Parameter ist jeweils 0.1 des Intervalles, in dem die Parameter evolviert werden.

6.3.4 Fitneßfunktion des Meta-Algorithmus

Die Meta-ES benötigt zur Bewertung der Parametersätze der Basis-ES eine Fitneßfunktion f_{meta}. Eine naheliegende und auch in den vorgestellten Arbeiten im Abschnitt 6.2 häufig verwendete Möglichkeit ist es, die Fitneß des besten Individuums des Basis-EA zu verwenden. Dieser Wert wird im folgenden mit $f_{\text{basis}}(\vec{x}_{best})$ bezeichnet. Um die Rechenzeit bei der Fitneßauswertung zu begrenzen, wird üblicherweise eine maximale Anzahl an Funktionsauswertungen $n_{\text{fe,max}}$ festgelegt, die der Basis-EA rechnen darf.

Allein die erreichte Fitneß als Bewertungskriterium reicht in manchen Fällen aber nicht aus. Findet beispielsweise die Basis-ES mit zwei verschiedenen Parametersätzen jeweils das Optimum (oder eine ausreichende Annäherung), jedoch ohne daß die maximale Zahl der Funktionsauswertungen ausgeschöpft werden mußte, so ist es sinnvoll, die tatsächlich benötigte Zahl der Funktionsauswertungen n_{fe} mit in die Bewertung einzubeziehen. Denn die Reduzierung der Funktionsauswertungen ist ein zweites sinnvolles Kriterium bei der Meta-Optimierung. Dies ist besonders wichtig, wenn es bei der Optimierungsanwendung nicht sehr schwer ist, das Optimum zu finden, dieses aber möglichst schnell gefunden werden soll.

Um die beiden Fitneßkriterien vereinen zu können, ist die Angabe einer maximalen Fitneß $f_{\text{basis,max}}$ (bei Maximierung), bzw. einer minimalen Fitneß $f_{\text{basis,min}}$ (bei Minimierung) nötig, die in der Anwendung des Basis-EA vorkommt oder auch gerade nicht mehr vorkommt. Diese Angabe ist für die Benchmarkfunktionen problemlos möglich, in den meisten Fällen auch für selbst konstruierte Fitneßfunktionen.

Die Fitneßfunktion für die Meta-ES soll nun so konstruiert werden, daß im Falle des Ausschöpfens der maximalen Anzahl Funktionsauswertungen durch eine Basis-ES alleine die erreichte Fitneß als Bewertung dient. Wird dagegen von einer Basis-ES die vorgegebene Fitneß $f_{\text{basis,min}}$ bzw. $f_{\text{basis,max}}$ mit weniger Auswertungen n_{fe} erreicht, so soll diese Zahl als Vergleichskriterium herangezogen werden. Die beiden Kriterien sollen so aneinandergefügt werden, daß jede Basis-ES, die die vorgegebene Fitneß in weniger als der maximalen Zahl Funktionsauswertungen erreicht, besser bewertet wird als eine ES, welche die vorgegebene Fitneß nicht erreicht.

Zunächst der Fall, daß die Fitneßfunktion des Basis-EA minimiert werden soll. Es gelten dann die Grundvoraussetzungen:

$$f_{\text{min}} \leq f_{\text{basis}}(\vec{x}) \tag{6.10}$$

$$0 < n_{\text{fe}} \leq n_{\text{fe,max}} \tag{6.11}$$

Nun wird der Wertebereich der Funktionsauswertungen so transformiert, daß er ohne Überschneidungen an den Wertebereich der Basis-Fitneßfunktion angefügt werden kann:

$$-n_{\text{fe,max}} < n_{\text{fe}} - n_{\text{fe,max}} \leq 0 \tag{6.12}$$

$$f_{\text{min}} - n_{\text{fe,max}} < n_{\text{fe}} - n_{\text{fe,max}} + f_{\text{min}} \leq f_{\text{min}} \tag{6.13}$$

Dann lautet die zu *minimierende* Fitneßfunktion des Meta-EA:

$$f_{\text{meta}} = \begin{cases} n_{\text{fe}} - n_{\text{fe,max}} + f_{\text{min}} & \text{falls } n_{\text{fe}} < n_{\text{fe,max}} \\ f_{\text{basis}}(\vec{x}) & \text{falls } n_{\text{fe}} = n_{\text{fe,max}} \end{cases} \tag{6.14}$$

Der Term $n_{\text{fe}} - n_{\text{fe,max}} + f_{\text{min}}$ liefert immer größere Werte als $f_{\text{basis}}(\vec{x})$, d. h. die beiden Fälle überschneiden sich nicht bei den Fitneßwerten.

Bei Maximierung der Fitneßfunktion des Basis-EA gelten zunächst die Grundvoraussetzungen:

$$f_{\text{max}} \geq f_{\text{basis}}(\vec{x}) \tag{6.15}$$

$$0 < n_{\text{fe}} \leq n_{\text{fe,max}} \tag{6.16}$$

Hier muß zuerst die Richtung der Werte bei den Funktionsauswertungen umgedreht werden, da nun maximiert werden soll, aber weniger Funktionsauswertungen besser sind:

$$0 > -n_{\text{fe}} \geq -n_{\text{fe,max}} \tag{6.17}$$

$$n_{\text{fe,max}} > n_{\text{fe,max}} - n_{\text{fe}} \geq 0 \tag{6.18}$$

$$n_{\text{fe,max}} + f_{\max} > n_{\text{fe,max}} - n_{\text{fe}} + f_{\max} \geq f_{\max} \tag{6.19}$$

Dann lautet die zu *maximierende* Fitneßfunktion des Meta-EA:

$$f_{\text{meta}} = \begin{cases} n_{\text{fe,max}} - n_{\text{fe}} + f_{\max} & \text{falls } n_{\text{fe}} < n_{\text{fe,max}} \\ f_{\text{basis}}(\vec{x}) & \text{falls } n_{\text{fe}} = n_{\text{fe,max}} \end{cases} \tag{6.20}$$

Ein Detail ist aber bisher noch nicht berücksichtigt: je nach Basis-ES kann die maximale Anzahl Funktionsauswertungen $n_{\text{fe,max}}$ nie ganz genau ausgenutzt werden. Beispielsweise kann eine $(1,5)^{\gamma}$-ES niemals genau $n_{\text{fe,max}} = 1000$ Funktionsauswertungen ausführen, bei $\gamma = 199$ werden $n_{\text{fe}} = 996$ Funktionsauswertungen durchgeführt und bei $\gamma = 200$ bereits $n_{\text{fe}} = 1001$ Auswertungen benötigt. Deshalb muß bei der Fallunterscheidung in den Fitneßfunktionen ein der Strategie angepaßter Vergleichswert verwendet werden:

$$n'_{\text{fe,max}} = \left\lfloor \frac{n_{\text{fe,max}}}{\lambda} \right\rfloor \cdot \lambda \tag{6.21}$$

6.3.4.1 Mehrfachauswertung

Da eine Evolutionsstrategie ein stochastischer Algorithmus ist, ist auch die Fitneßfunktion der Meta-Optimierung, welche die Ausführung einer Evolutionsstrategie beinhaltet, stochastischen Schwankungen unterworfen (Rauschen). Grundsätzlich sind Evolutionäre Algorithmen in der Lage, auch verrauschte Fitneßfunktionen zu optimieren. Die Stärke des Rauschens spielt dabei aber sicherlich auch eine Rolle. Eine Möglichkeit, das Rauschen zu verringern, ist es, die Funktion an einer Stelle mehrfach auszuwerten und das Ergebnis zu mitteln. Die Anzahl der Auswertungen wird hier mit n_{eval} bezeichnet. Auf die Rolle der Mehrfachauswertung bei der Meta-Optimierung wird bei den Versuchen weiter eingegangen.

6.3.5 Parameter der Meta-Evolutionsstrategie

Es stellt sich die Frage, wie die Parameter der Meta-ES gewählt werden sollen. In den in Abschnitt 6.2 vorgestellten Arbeiten werden diese mit Standardwerten belegt, z. B.:

> *The metalevel GA (. . .) used the standard parameter settings (. . .). Past experience has shown that these parameters yield a fairly good search for a variety of problems (. . .).* [Grefenstette 86]
>
> *The control parameters for the higher-level Genetic Algorithm were given intuitively-chosen values.* [Bramlette 91]

Bäck ist der Meinung, daß der Meta-EA robuster auf Parameteränderungen reagiert:

> *(. . .) the problem is shifted one level upwards this way, but the metalevel parameterization is likely to be more robust than the parameterizations of Genetic Algorithms themselves.* [Bäck 96]

Was ist mit *robust* gemeint? Es bedeutet, daß für eine breite Menge an Parametereinstellungen, der EA qualitativ ungefähr gleichwertige Lösungen findet. Wie schnell diese gefunden werden, kann dabei sehr wohl von den Parametern abhängen. Letzteres ist ja genau der Ansatzpunkt der Meta-Optimierung. Da es keine sinnvollen Alternativen gibt, müssen die Meta-Parameter also nach Standard- und Erfahrungswerten gewählt werden.

In den folgenden Versuchen wurde deshalb eine $(20 + 1)$-Steady-State-ES verwendet (Standard oder Median-Selektion). $\mu = 20$ sollte eine ausreichende Anzahl Elternindividuen sein, um nicht zu früh in lokale Optima zu geraten.

Die Rekombinationsarten wurden nach der Empfehlung von [Schwefel 95] (s. Abschnitt 2.4.2) diskrete Rekombination auf den Objektvariablen aller drei Typen und intermediäre Rekombination auf den Strategievariablen gewählt. Für die Aufzählungsparameter muß schon alleine deshalb dominante Rekombination verwendet werden, da intermediär auf diesem Typ überhaupt nicht sinnvoll definiert werden kann.

6.4 Ergebnisse

6.4.1 Evolvieren von λ

Beim ersten Versuch mit der Meta-ES sind die Bedingungen so gewählt, daß das Ergebnis voraussagbar ist. Hiermit soll gezeigt werden, daß die Meta-ES grundsätzlich funktioniert. Es wird die Anzahl der Nachkommenindividuen λ einer $(1, \lambda)$-ES mit Kovarianzmatrixadaption auf einigen Funktionen evolviert. λ kann dabei Werte im Bereich $\lambda \in \{1, \ldots, 100\}$ annehmen.

Die Meta-ES läuft mit folgenden Parametern: $(20/3 + 1)$-ES mit Median-Selektion, $n_p = 40$, $r_p = 0.15$. Jedes Parameterset wird $n_{\text{eval}} = 5$ mal mit unterschiedlichen Werten für den Zufallszahlengenerator getestet und die resultierenden Fitneßwerte f_{meta}

gemittelt. Dies dient dazu, die Zuverlässigkeit der Ergebnisse für einen Parametersatz zu erhöhen, da die Fitneßfunktion verrauscht ist. Versuche mit $n_{\text{eval}} = 1$ lieferten keine zufriedenstellenden Ergebnisse. Dabei konnte zwar der Optimierungslauf nachvollzogen werden, welcher zur Berechnung von f_{meta} verwendet wurde, weitere Läufe wichen von diesem Ergebnis jedoch stark ab.

Funktion	n	$n_{\text{fe,max}}$	λ
f_1	10	3000	5
f_2	10	10000	6
f_6	10	3000	6
f_9	10	3000	6
f_{15}	10	3000	7
q_1	10	10000	6
q_2	10	10000	5

In der Literatur wird üblicherweise bei $\mu = 1$ der Wert λ im Bereich $\{5, \ldots, 10\}$ gewählt. Die Versuchsergebnisse bestätigen dies.

Bei allen Versuchen war eine Konvergenzgrenze von $f_{\text{min}} = 1 \cdot 10^{-10}$ vorgegeben. Hierbei hat sich der Einsatz der Fitneßfunktion f_{meta} bewährt. In den ersten Generationen waren noch Individuen mit einem Fitneßwert $f_{\text{meta}} > f_{\text{min}}$ enthalten. Die Parametersätze haben also unter Ausnutzung aller Funktionsauswerungen $n_{\text{fe,max}}$ die Konvergenzgrenze nicht erreicht. Erst in späteren Generationen besaßen alle Individuen eine Fitneß $f_{\text{meta}} < f_{\text{min}}$, wodurch dann Strategien evolviert wurden, welche die Zahl der Funktionsauswertungen minimieren.

6.4.2 Funktion f_{10} - Traveling Salesman Problem

Die Basis-ES optimiert hier eine Instanz des *Traveling Salesman Problem* (TSP). Dies ist ein kombinatorisches Problem, bei dem die Besuchsreihenfolge von n gegebenen Städten gesucht ist, welche die kürzeste Rundreise bildet. Dies ist ein populäres Beispiel aus der Klasse der NP-vollständigen Probleme. Es kann also nach heutigem Wissen nur optimal gelöst werden mit einem Aufwand von der Größenordnung von vollständigem Durchsuchen des Lösungsraumes, welcher exponentiell mit der Anzahl n der Städte wächst. Die Instanz, welche hier verwendet wird, ist Krolak's 100-Städte TSP (kroA100) aus der TSPLIB95 [Reinelt 95] (siehe auch Anhang C). Die hierfür kürzeste bekannte Rundreise hat eine Länge von 21285.

Um das kombinatorische Problem für eine ES auf ein Parameteroptimierungsproblem abzubilden, wird die Sort-Order-Kodierung verwendet. Hierbei legt jede Variable eines Individuums durch ihren Wert relativ zu den anderen Variablen den Besuchszeitpunkt der ihr zugeordneten Stadt fest. Es findet eine Sortierung der Variablen statt,

Parameter	Bereich	Typ	Beschreibung
Strategie	{generationenbasiert, Steady-State}	diskret	Strategietyp
s	{Plus, Komma}	diskret	Selektion
μ	$\{1, \ldots, 500\}$	ganz	Anzahl Elternindividuen
λ	$\{1, \ldots, 2000\}$	ganz	Anzahl Nachkommenindividuen
ρ	$\{1, \ldots, 100\}$	ganz	Anzahl Rekombinanten
r_{obj}	{dominant, intermediär kontinuisiert}	diskret	Rekombination der Objektvariablen
r_{strat}	{dominant, intermediär}	diskret	Rekombination d. Strategievar.
m	{MSR, MSR separat, KSR, DE, keine}	diskret	Mutationsoperator/ Adaptionsverfahren
α	$[0.333, 3.0]$	kont.	Modifikationsfaktor (bei MSR)
δ	$[0, 1]$	kont.	relative Anfangsschrittweite
n_p	$\{1, \ldots 200\}$	ganz	Pufferlänge (Median-Selektion)
r_p	$(0, 1.0]$	kont.	Akzeptanzrate (Median-Selektion)
rep_{ind}	{ältestes Ind., schlechtestes Ind.}	diskret	Ersetzungsstrategie
rep_{cond}	{immer, wenn besser}	diskret	Ersetzungsbedingung

Tabelle 6.1: Durch die Meta-Optimierung evolvierbare Parameter und ihre Wertebereiche.

wobei die zugeordneten Städte mitgetragen werden. Die Besuchsreihenfolge der Städte ergibt sich dann durch die Reihenfolge der Variablen nach der Sortierung. Dies ist sicher nicht das beste Verfahren zur Lösung des TSP. Allerdings ist das TSP ein anspruchsvolles Problem, und bietet damit genug Potential für Verbesserungen durch die Meta-Optimierung.

Von der Meta-ES werden alle Parameter der Basis-ES evolviert. Diese sind in Tabelle 6.1 aufgelistet. Es ist $n_{\text{fe,max}} = 100000$ als maximale Anzahl Funktionsauswertungen vorgegeben. Die Kovarianzmatrixadaption wurde hier als Adaptionsverfahren nicht zugelassen, da diese eine Zeitkomplexität der Größenordnung $O(n^2)$ in Abhängigkeit von den Objektvariablen besitzt. Da hier $n = 100$ ist, bedeutet dies im Verhältnis zu den anderen Verfahren bereits einen sehr hohen Zeitaufwand.

6.4.2.1 Empirisch ermittelte Parameter

Die empirische Vorgehensweise, einen Parametersatz zu ermitteln, der das gegebene Problem gut löst, ist die folgende: Ausgehend von einem Anfangs-Parametersatz variiert der Experimentator jeweils einen der Parameter über einen als sinnvoll erachteten

Wertebereich. Der dabei beste auftretende Wert wird dann für die darauffolgenden Tests der restlichen Parameter verwendet. Die Parameter sind dabei nach der vermuteten Relevanz absteigend geordnet. Dieses Vorgehen ist bereits an drei Punkten vom Experimentator abhängig:

1. die Bestimmung des Anfangs-Parametersatzes,
2. der Wertebereich und die Schrittweite der Tests
3. die Ordnung der Parameter nach Relevanz.

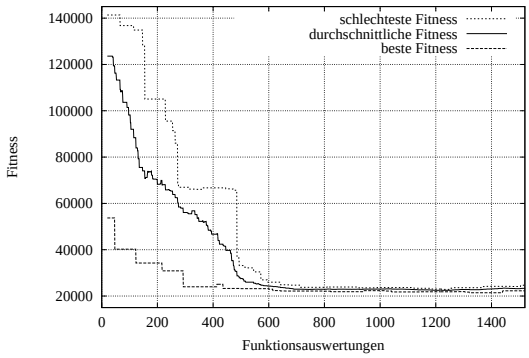
Hier wurde der Anfangs-Parametersatz wie folgt gewählt: $(75/10, 500)^{199}$, MSR, $\alpha = 1.3$, $\delta = 0.1$, dominante Rekombination auf den Objektvariablen und intermediäre Rekombination auf den Strategievariablen. Die Parameter $\mu = 75$, $\lambda = 500$ und $\gamma = 199$ wurden auf ihren Werten belassen. Das Verhältnis μ zu λ entspricht dabei dem bekannten Wert 0.15.

Variiert wurden die folgenden Parameter in den angegebenen Wertebereichen. Zur Platzersparnis werden statt Schaubildern hier nur die Ergebnisse angegeben:

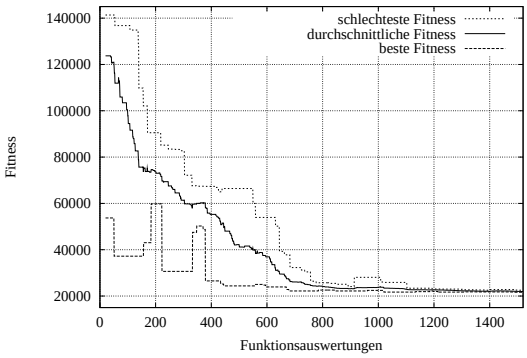
1. Selektion Plus/Komma: $(75/10, 500)$, $(75/10 + 500)$
   $\rightarrow$Ergebnis: Komma-Selektion ist deutlich besser.
2. Schrittweitenadaption: keine, MSR, MSR separat, KSR, CMA und Differential Evolution
   $\rightarrow$Ergebnis: KSR.
3. Anzahl Individuen bei der Rekombination ρ: 1 bis 10, 15, 20, 30, 40, 50, 75
   $\rightarrow$Ergebnis: $\rho = 10$. Ohne Rekombination werden sehr schlechte Ergebnisse erzielt, die Ergebnisse bei $\rho \geq 2$ unterscheiden sich kaum.
4. $\delta = 0.05$ bis 0.5 in Schritten von 0.05
   $\rightarrow$Ergebnis: nur $\delta = 0.05$ liefert schlechte Ergebnisse, weil mit dieser Anfangsschrittweite zu lokal gesucht wird, ab $\delta = 0.1$ ist bis auf stochastische Schwankungen kein Unterschied in der Performance zu erkennen. $\delta = 0.2$ liefert dennoch den besten Wert und wird deshalb hier verwendet.
5. Rekombination bei Objekt- und Strategievariablen: dominant oder intermediär, bei den Objektvariablen auch kontinuisiert (insgesamt 6 Kombinationen).
   $\rightarrow$Ergebnis: dominante Rekombination auf den Objektvariablen und intermediäre auf den Strategievariablen. Dies ist die von [Schwefel 95] empfohlene Standardeinstellung.

Der empirische Parametersatz lautet also: $(75/10, 500)^{199}$, KSR, $\delta = 0.2$, Rekombination dominant/intermediär auf Objekt-/Strategievariablen.

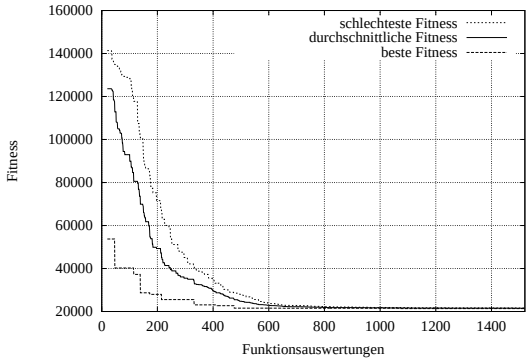
Ein Test dieses Parametersatzes über 40 Läufe liefert eine durchschnittliche Weglänge von 23395. Die kürzeste Weglänge die dabei auftritt ist 21842.



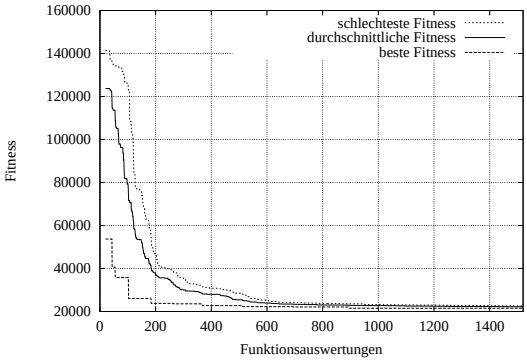
(a) Median-Selektion



(b) Median-Selektion mit doppelter Auswertung



(c) Standard Steady-State



(d) Standard Steady-State mit doppelter Auswertung

Abbildung 6.1: Fitneßverlauf der Meta-Optimierungsläufe für das TSP.

6.4.2.2 Meta-Optimierung

Es wurden insgesamt vier Meta-Läufe durchgeführt. Ein Standard Steady-State-Algorithmus und eine Strategie mit Median-Selektion jeweils einmal mit einfacher und doppelter Auswertung (bezeichnet mit $2\times$) der Basis-ES. Die verwendeten Strategien sind:

1. eine $(20/5 + 1)^{1500}$-ES mit Median-Selektion, $n_p = 40$ und $r_p = 0.15$,
2. eine $(20/5 + 1)^{1500}$ Standard Steady-State-ES.

Die Läufe wurden in einer Konfiguration mit 16 Prozessoren durchgeführt, wobei 15 zur Durchführung der Basis-ES benutzt wurden. Ein Lauf mit einfacher Auswertung

dauert hierbei ca. 98 Minuten, ein Lauf mit zweifacher Auswertung etwa doppelt so lange.

In Abbildung 6.1 sind die Fitneßverläufe der Meta-ES dargestellt. In der oberen Zeile sind die Diagramme für die Steady-State-ES mit Median-Selektion und in der unteren Reihe für die Standard Steady-State-ES. Die Diagramme in der linken Spalte sind die Läufe mit einfacher Fitneßauswertung, in der rechten Spalte mit doppelter Auswertung. Allein im Fitneßverlauf bestehen zwischen einfacher und doppelter Auswertung keine nennenswerten Unterschiede. Die beste auftretende Fitneß ist bei doppelter Auswertung sogar schlechter. Wie im folgenden gezeigt wird, ist die doppelte Auswertung zumindest mit Median-Selektion dennoch vorzuziehen.
Die Meta-ES mit dem Standard Steady-State-Algorithmus konvergiert hier schneller zu guten Fitneßwerten als die Meta-ES mit Median-Selektion. Bei dieser ist im Verlauf die Differenz zwischen bestem und schlechtestem Individuum etwas größer als bei der Standard Steady-State-ES. Die Diversität der Individuen in der Population scheint also größer zu sein, was bei manchen Anwendungen von Vorteil sein kann. Der Grund hierfür ist, daß bei Standard Steady-State schlechte Individuen so früh wie möglich durch bessere ersetzt werden. Bei der Median-Selektion dagegen ist die Ersetzung nur vom Alter des Individuums abhängig. Somit verbleiben auch schlechtere Individuen längere Zeit in der Population.

Als Ergebnis der Meta-Optimierung wurden die Parametersätze aus Tabelle 6.2 evolviert. Bei allen vier Strategien wurde eine generationenbasierte Evolutionsstrategie mit Komma-Selektion evolviert. Die Zahl der Elternindividuen ist hier durchweg größer als bei den empirischen Parametern. Dadurch wird bei der vorgegebenen maximalen Zahl von $n_{\mathrm{fe,max}} = 100000$ Funktionsauswertungen auch die Zahl der zur Verfügung stehenden Generationen geringer. Bei der empirischen Ermittlung von Parametern ist es schwierig, den richtigen Kompromiß zwischen Populationsgröße und der Anzahl der Generationen zu wählen. Offensichtlich ist es bei dieser Anwendung besser, mit größeren Populationen, dafür aber weniger Generationen zu rechnen.

Die Werte 0.13 und 0.16 beim Verhältnis $\frac{\mu}{\lambda}$ liegen im Rahmen des üblichen (0.1 bis 0.2). Alleine der Wert 0.24 ein wenig größer als erwartet aus, funktioniert aber anscheinend noch gut genug.

Der Parameter ρ wurde durchweg auf einen Wert von ca. 50 evolviert, obwohl der Wertebereich bis 100 zugelassen war. Eine weitere Erhöhung scheint also negative Auswirkungen auf das Ergebnis der Basis-ES zu haben. Eine mögliche Erklärung hierfür wäre, daß größere Werte für ρ die Diversität der erzeugten Nachkommen reduzieren, da sie über zu viele Individuen mitteln. Wenn ρ in der Größenordnung von μ gewählt wird, sind mit großer Wahrscheinlichkeit im Rekombinationspool immer auch relativ schlechte Individuen aus der Elternpopulation enthalten. Als Rekombinationsart wurde bei drei der Parametersätze die empfohlene Kombination aus dominanter/intermediär für die Objekt- und Strategievariablen evolviert. Nur in einem Fall wurde die Kombination dominant/dominant evolviert. Dieser Satz weicht allerdings auch

Meta-Strategie:	Median	Median $2\times$	Standard	Standard $2\times$
Strategie	generat.	generat.	generat.	generat.
s	Komma	Komma	Komma	Komma
μ	264	134	108	155
λ	1073	1059	655	983
ρ	50	54	48	46
r_{obj}	dominant	dominant	dominant	dominant
r_{strat}	dominant	intermediär	intermediär	intermediär
m	MSR	KSR	KSR	KSR
α	0.728950602	-	-	-
δ	0.217932345	0.164158122	0.215377374	0.039471554
n_p, r_p	-	-	-	-
Ersetzungsstrategie	-	-	-	-
Ersetzungsbed.	-	-	-	-
Fitneß f_{meta}	21413	21624	21368	21541
durchschn. Fit. (40)	23921	22696	22949	24526
μ/λ	0.24	0.13	0.16	0.16

Tabelle 6.2: Mit der Meta-Optimierung evolvierte Parametersätze.

bei anderen Parametern etwas von den anderen ab: beim Verhältnis $\frac{\mu}{\lambda}$ und beim Adaptionsverfahren. Dieses wurde dort als MSR gewählt. Der Faktor $\alpha = 0.728950602$ bzw. $1/\alpha \approx 1.37$ liegt im normalen Bereich. Bei den Parametersätzen mit KSR als Adaptionsverfahren, wird α nicht verwendet.

Aus der Erfahrung heraus sollte die Anfangsschrittweite δ nicht wesentlich kleiner als 0.1 sein. Hier fällt nur der Parametersatz in der vierten Spalte aus dem Rahmen.

Die Parameter n_p, r_p, Ersetzungsstrategie und Ersetzungsbedingung sind hier nicht verwendet, da kein Steady-State-Algorithmus evolviert wurde.

Zur Evaluation der Ergebnisse wurden der empirische und die vier evolvierten Parametersätze jeweils über 40 verschiedene Anfangswerte für den Zufallszahlengenerator getestet. Das Ergebnis ist in Abbildung 6.2 dargestellt, die Mittelwerte sind auch in Tabelle 6.2 in der vorletzten Zeile aufgeführt.

Man kann folgendes erkennen:

- Die Anwendung ist sehr verrauscht, die Ergebnisse der Basis-Evolutionsstrategie auf dem TSP schwanken sehr stark. Die Fitneßwerte, welche von der Meta-Fitneßfunktion vergeben werden, sind deutlich besser als die nachträglich evaluierten Durchschnittswerte.

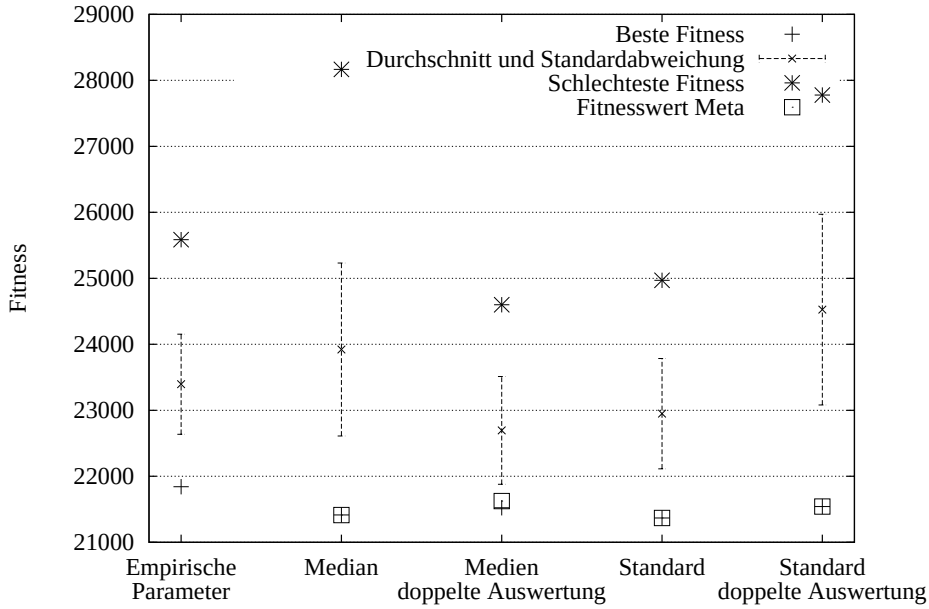


Abbildung 6.2: Vergleich des empirischen Parametersatzes mit den Ergebnissen der vier Meta-Optimierungen.

- Bei Standard Steady-State ist die Abweichung des Meta-Fitneßwertes vom evaluierten Durchschnittswert etwas größer als bei den beiden Steady-State-Algorithmen mit Median-Selektion. Wenn bei Standard Steady-State ein Individuum zufällig mit einem Anfangswert für den Zufallszahlengenerator evaluiert wird, welcher ein gutes Ergebnis liefert, dann kann dieses im weiteren nur sehr schwer durch ein anderes Individuum ersetzt werden. Tatsächlich kann man im Schaubild sehen, daß der Fitneßwert des besten der 40 Läufe genau der von der Meta-Fitneßfunktion vergebene Wert ist. Das Individuum wurde also mit demselben Anfangswert evaluiert. Dies konnte auch durch die während des Laufes mitprotokollierten Daten bestätigt werden.

- Bei der Median-Selektion bringt die doppelte Auswertung des Parametersatzes eine deutliche Steigerung des evaluierten Durchschnittswertes. Die Abweichung zum Meta-Fitneßwert ist hier auch deutlich geringer. Mehrfachauswertung ist also auch hier von Vorteil. Dieses Verhalten wäre auch bei der Standard Steady-State-ES zu erwarten gewesen. Die Messung liefert hier jedoch das umgekehrte Bild.

- Der von der Median-Selektion mit doppelter Auswertung evolvierte Parametersatz weist eine *geringfügig* besseren Durchschnittswert auf, als der von der Standard Steady-State-ES evolvierte. Die Differenz ist allerdings aufgrund der großen Varianzen nicht relevant.

6.5 Zusammenfassung

Bei einer Erörterung des Problems der Parametereinstellungen von Evolutionsstrategien wurde die Meta-Optimierung der Selbstadaption vorgezogen, da sie verspricht, weniger Probleme aufzuwerfen und bessere Ergebnisse zu liefern.

Bishere Ansätze zur Meta-Optimierung hatten die folgenden Einschränkungen:

- der Wertebereich der Parameter wurde durch die Wahl von diskreten Stufen eingeschränkt,
- die Parameter wurden meist nicht selbstadaptiv evolviert,
- ganzzahlige Parameter wurden nicht mit auf ganze Zahlen spezialisierten Operatoren mutiert,
- es wurden noch nie alle drei Parametertypen zugleich optimiert,
- es wurden immer nur die Parameter von Genetische Algorithmen, nicht aber von Evolutionsstrategien optimiert.

Die hier vorgestellte Meta-Evolutionsstrategie hebt diese Einschränkungen auf. Es werden kontinuierliche, ganzzahlige und Aufzählungsparameter gleichzeitig optimiert. Dabei wurden eine Evolutionsstrategie um selbstadaptive Mutationsverfahren für ganzzahlige und Aufzählungsparameter erweitert. Eine Diskretisierung des Wertebereichs war deswegen nicht mehr notwendig. Als Basis-EA wurde erstmals eine Evolutionsstrategie verwendet. Dabei hat sich gezeigt, daß die Meta-ES funktioniert und daß die evolvierten Parametersätze bessere Ergebnisse erzielen als empirisch ermittelte Parameter.

Es hat sich herausgestellt, daß eine Mehrfachauswertung der Basis-ES die Zuverlässigkeit der Ergebnisse steigert. Mehrfachauswertung ist notwendig, da die Leistung der Basis-ES ein sehr verrauschtes Fitneßmaß bildet.

Kapitel 7

Anwendungen

Die in den vorigen beiden Kapiteln entwickelten Verfahren wurde bereits mit einigen der Standard-Benchmarkfunktionen getestet. Diese sind jedoch synthetisch. Deswegen sollen die Verfahren in diesem Kapitel mit realen Anwendungen getestet werden. Dazu wird die Meta-Optimierung auf eine Problemstellung, die in der holzbearbeitenden Industrie auftritt, angewendet. Die parallele Steady-State-Evolutionsstrategie mit Median-Selektion wird zur Optimierung einer optischen Linse und bei der Konformationsoptimierung von Molekülen eingesetzt.

7.1 Spannmittelplazierung zur Werkstückfixierung

Eine Anwendung aus der holzbearbeitenden Industrie ist die Plazierung von Vakuum-Spannelementen zur Fixierung eines Holzplatten-Werkstückes auf dem Bearbeitungstisch [Koch et al. 99, Ott et al. 98, Ott 98]. Sie soll hier als Beispiel verwendet werden, um die Tauglichkeit der Meta-Evolutionsstrategie zu testen.

Ein rohes Werkstück in Form einer Holzplatte soll durch einen CNC-gesteuerten Roboterarm mit Werkzeugen bearbeitet werden (Fräsen, Bohren, Leimen, etc.). Dazu wird es mithilfe von Vakuum-Spannelementen auf dem Bearbeitungstisch fixiert. Diese Spannelemente (meist rechteckig) saugen sich mit der Unterseite am Bearbeitungstisch und mit der Oberseite an der Holzplatte fest. Die Optimierungsaufgabe besteht darin, eine bestimmte Anzahl der Spannelemente so unter dem Werkstück zu plazieren, daß einerseits das Werkstück während der Bearbeitung nicht verrutscht oder flattert und andererseits keines der Spannelemente durch die Werkzeuge beschädigt wird. Die Bearbeitungsspur der Werkzeuge in der Ebene des Werkstückes liegt als Polygonzug vor, welcher aus einer CNC-Simulation gewonnen wird.

Je ein Spannelement wird in den Individuen der Evolutionsstrategie durch drei kontinuierliche Variablen beschrieben: die x- und y-Koordinate des Mittelpunktes und der Drehwinkel, welcher die Orientierung angibt.

Es wurde versucht, die von Experten gegebenen natürlichsprachlichen Anforderungen an die Plazierung in einer Fitneßfunktion zu implementieren. Die geforderten Kriterien sind:

- Plazierung der Spannelemente auf dem Bearbeitungstisch,
- gleichmäßige Verteilung der Spannelemente über das ganze Werkstück,
- Plazierung der Spannelemente möglichst nahe an der Bearbeitungsspur (da hier die größten Kräfte auftreten),
- keine Überschneidung der Bearbeitungsspur mit einem Spannelement,
- keine Überschneidung von Spannelementen untereinander.

Auf die Details der Fitneßfunktion kann an dieser Stelle nicht näher eingegangen werden, sie sind in [Koch et al. 99] zu finden. Grob setzt sich die Funktion aus den folgenden Teilen zusammen:

$$F_{\text{Plazierung}} = \alpha \cdot I - \beta \cdot U + \gamma \cdot O_1 \tag{7.1}$$

Der Teilterm I berechnet den Schnitt zwischen zwei Spannelementen (kleiner ist besser), U die Gleichverteilung über das Werkstück (größer ist besser) und O_1 die Konturnähe (kleiner ist besser). Insgesamt muß die Fitneßfunktion minimiert werden. Es sind rechenaufwendige geometrische Algorithmen enthalten, die Schnitte zwischen Polygonen berechnen. Die Eingabedaten bestehen oft aus Polygonen mit 200 oder mehr Punkten.

Die absoluten Fitneßwerte haben keine direkte Aussagekraft. Sie können nur zum Vergleich zwischen Plazierungskonfigurationen unter denselben Rahmenbedingungen verwendet werden, also bei demselben Werkstück und derselben Anzahl zu plazierender Spannelemente.

Da die Maschinen nicht für die Massenfertigung gleichartiger Teile gedacht sind, sondern oftmals nur ein einziges Teil derselben Art herstellen, muß die Spannmittelplazierung jedesmal von neuem erfolgen. Dieser Vorgang unterliegt gewissen Zeitbeschränkungen. Die Plazierung findet in zwei möglichen Anwendungsszenarien statt: entweder als Vorschlag zur manuellen Plazierung; dort kommt es auf möglichst kurze Antwortzeiten an, um die Interaktivität der Software zu erhalten. Oder bei automatisch bestückten Maschinen; hierbei darf die zur Plazierung benötigte Zeit die Maschineneffizienz nicht vermindern. Deshalb wurde als Rahmenbedingung eine maximale Ausführungszeit von 5 Sekunden auf aktueller PC-Hardware vorgegeben.

In Abbildung 7.1 ist das Spannmittelplazierungsproblem schematisch dargestellt. Der bemaßte Rahmen stellt den Arbeitstisch der Bearbeitungsmaschine dar. Das schwarze, gerahmte Rechteck ist die Form der unbearbeiteten Holzplatte. Die Bearbeitungsspur

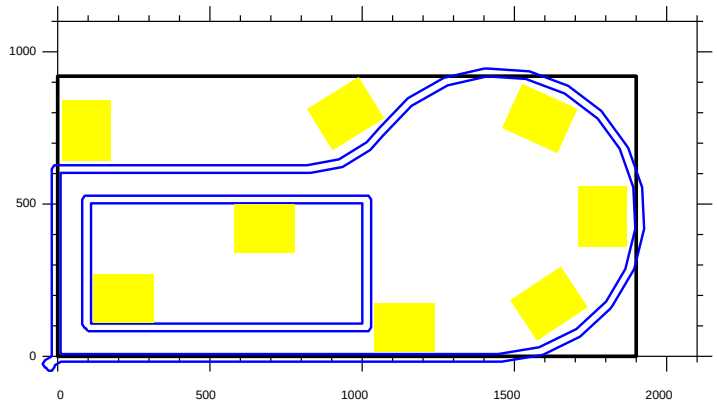


Abbildung 7.1: Schematische Darstellung des Spannmittelplazierungsproblems. Die Maßeinheit der Achsen ist Millimeter.

ist mit Doppellinien dargestellt. Das gezeigte Werkstück ist eine Küchenarbeitsplatte mit einer Aussparung für die Spülbecken. Die plazierten Spannelemente sind als graue Rechtecke eingezeichnet. Diese Konfiguration, in der acht rechteckige Spannelemente plaziert werden sollen, ist der Ausgangspunkt für die Versuche zur Meta-Optimierung. Da die Position jedes Spannelements durch zwei Koordinaten und einen Winkel beschrieben wird, ergibt sich eine Problemdimension von 24.

Der beim Einsatz des Systems verwendete Standard-Parametersatz ist eine $(4/4 + 20)^{200}$-Evolutionsstrategie mit dem Adaptionsverfahren MSR mit separaten Schrittweiten. Die benötigte Optimierungszeit beträgt ca. 4.4 Sekunden auf einem Pentium III Prozessor mit 700 MHz. Dabei wird, gemittelt über 40 Läufe, eine durchschnittliche Fitneß von 2.31 erreicht.

7.1.1 Meta-Optimierung

Auf das vorgestellte Problem wird nun die Meta-Optimierung angewendet. Dabei sollen Parameter für die Evolutionsstrategie evolviert werden, welche:

- die Lösungsqualität verbessern,
- die Zeit zur Lösungsfindung verringern.

Es werden alle möglichen Parameter der Evolutionsstrategie evolviert. Eine Liste der Parameter und ihrer Wertebereiche ist in Abbildung 6.1 aufgeführt. Bei den Adaptionsverfahren wurde hier die Kovarianzmatrixadaption ebenfalls zugelassen, da sie aufgrund der Dimension der Anwendung (24) noch praktikabel ist.

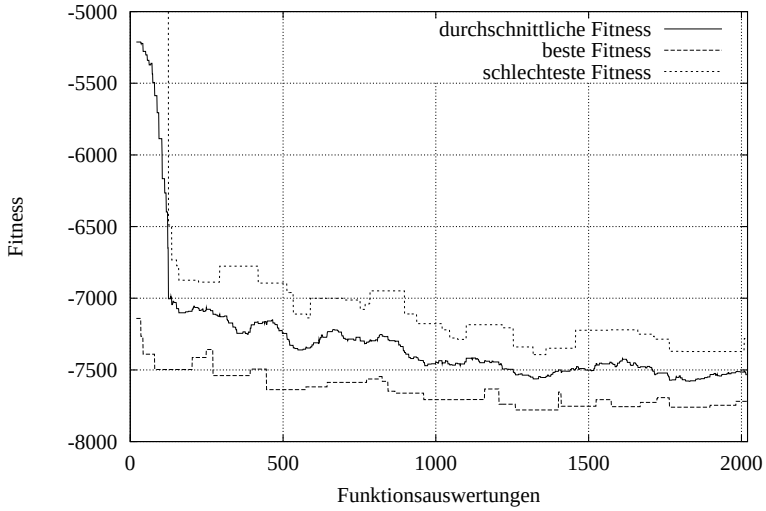
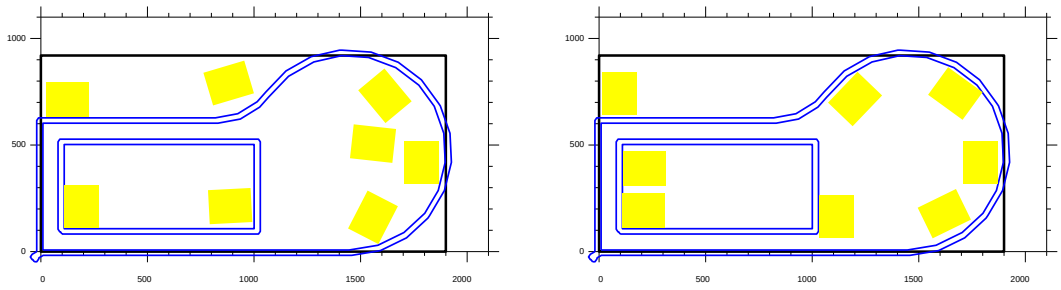


Abbildung 7.2: Fitneßverlauf bei der Meta-Optimierung der Spannmittelplazierung.

Strategie	Steady-State
s	lokale Tournamentselektion
μ	3
λ	20
ρ	1
r_{obj}	-
r_{strat}	-
m	Differential Evolution
α	-
δ	-
n_p, r_p	-
Ersetzungsstrategie	schlechtestes Individuum
Ersetzungsbedingung	immer ersetzen
Fitneß	-7779.5
durchschnittliche Fitneß (40)	1.67

Tabelle 7.1: Evolvierter Parametersatz für die Spannmittelplazierung. Parameter, deren Werte aufgrund von Abhängigkeiten nicht relevant sind, sind mit "-" gekennzeichnet.



(a) Beste Lösung mit dem Standard-Parametersatz

(b) Beste Lösung mit dem evolvierten Parametersatz

Abbildung 7.3: Vergleich der Lösungsqualität.

Für die Meta-Optimierung wurde eine $(20/5 + 1)^{2000}$-Steady-State-ES mit Median-Selektion mit $n_p = 40$ und $r_p = 0.15$ verwendet. Bei der Fitneßberechnung der Basis-ES wurden wie beim Standard-Parametersatz maximal $n_{\text{fe,max}} = 8004$ Funktionsauswertungen zugelassen, und jeder Parametersatz wurde $n_{\text{eval}} = 3$ mal ausgewertet, um die Verrauschtheit der Fitneßwerte zu verringern. Bei der Basis-ES war als Abbruchkriterium das Erreichen einer Fitneß von $f_{\min} = 2.5$ vorgegeben. Die Meta-Optimierung wurde parallel auf 16 Prozessoren durchgeführt und dauerte 10 Minuten und 54 Sekunden auf dem Testsystem.

Als Ergebnis wurde der Parametersatz in Tabelle 7.1 evolviert. Dabei handelt es sich um eine $(3 + (1, 20))$-Steady-State-Evolutionsstrategie mit lokaler Tournamentselektion und Ersetzung des schlechtesten Individuums, mit der Ersetzungsbedingung *immer ersetzen*. Zur Schrittweitenadaption wurde das *Differential Evolution* Mutationsverfahren evolviert (siehe Abschnitt 2.4.3.8).

Bei einer Messung über 40 Läufe mit der durchschnittlichen Fitneß 2.31 des Standardparametersatzes als Konvergenzlimit werden beim Einsatz des evolvierten Parametersatzes durchschnittlich nur 1206 Funktionsauswertungen benötigt. Im Gegensatz zu den vorgegebenen 8004 Funktionsauswertungen des Standard-Parametersatzes stellt dies eine enorme Verbesserung dar. Das vorrangige Ziel aber ist die Steigerung der Fitneß. Eine Messung der durchschnittlichen Fitneß dieses Parametersatzes über 40 Läufe, ermittelt über alle 8004 Funktionsauswertungen (ohne Konvergenzlimit) ergab einen Wert von 1.67. Dies stellt ebenfalls eine Verbesserung dar. Wie stark diese ist, läßt sich allerdings aus dem Zahlenwert nicht ablesen. Dies muß anhand der Plazierungen optisch beurteilt werden.

In Abbildung 7.3 sind die beiden Plazierungsergebnisse mit jeweils dem besten Fitneßwert dargestellt. Mit den empirischen Parametern wird ein Spannelement zu nahe bei drei anderen plaziert, welche die rechte Rundung abdecken. Mit den evolvierten

Parametern können sich die Spannelemente weiter und in einem besseren Winkel an die Konturen annähern. Dies liegt vermutlich an dem verwendeten Mutationsverfahren Differential Evolution, welches bessere Adaptionseigenschaften als MSR aufweist. Der Vergleich anhand dieser beiden Konfigurationen darf allerdings nicht zu sehr verallgemeinert werden. Um eine allgemeinere Aussage zu erhalten, müßte über alle Läufe, über mehrere Werkstücke und über verschiedene Anzahlen und Formen von Spannelementen verglichen werden. Dies steht allerdings nicht im Zentrum dieser Arbeit und würde den gegebenen Rahmen sprengen. Aus dem gezeigten Beispiel läßt sich aber das Potential für die Nützlichkeit der Meta-Optimierung am Beispiel einer Anwendung aus der industriellen Praxis erkennen.

7.2 Konformationsanalyse eines Peptids

Bei der Konformationsanalyse geht es darum, die 3D-Struktur eines Moleküls vorherzusagen, welche dieses im Raum einnimmt. Als Molekül wird hier ein Peptid verwendet. Dies sind Moleküle, welche aus kurzen Ketten von Aminosäuren bestehen. Als Konformation wird dabei die Faltung des Kettenmoleküls im Raum bezeichnet. Variabel sind hierbei die Winkel zwischen Atomen mit Einfachbindungen. Das hier verwendete Peptid ist *Met-Enkephalin*, das aus den Aminosäuren Tyrosin, Glycin, Phenylalanin und Methionin in dieser Reihenfolge besteht: Tyr-Gly-Gly-Phe-Met. Dieses Peptid ist ein Standard zum Testen von Strukturvorhersage-Programmen.

Die Aminosäurensequenz, aus der ein Peptid besteht, läßt sich in Experimenten relativ leicht bestimmen. Zur Bestimmung der Konformation sind jedoch aufwendige Versuche notwendig (Röntgenkristallographie oder NMR - Nuclear Magnetic Resonance). Die native Konformation, in welcher das Peptid seine biologische Funktion ausübt, ist diejenige mit dem niedrigsten Energiepotential. Das Energiepotential wird von den vier Grundkräften bestimmt. Um die aufwendigen Versuche zu vermeiden, versucht man, die Bestimmung der nativen Konformation im Rechner durchzuführen. Hierfür gibt es verschiedene Modelle, die das Energiepotential im Kraftfeld des Moleküls berechnen. Ein exaktes Modell, welches alle Kräfte in einem Molekül berechnet, wäre zu rechenaufwendig. Deshalb sind die Modelle auf spezielle Molekülarten oder Randbedingungen optimiert. Für die Tests dieser Arbeit wurde das *oplsaa*-Kraftfeld aus dem Softwarepaket *Tinker* [Ponder 98] benutzt, welches sich speziell für Peptide und andere organische Moleküle eignet. Es wurden die Teilprogramme *protein* zur Generierung der Proteinbeschreibung aus der Aminosäuresequenz und das Programm *analyze* zur Energieberechnung benutzt.

Jede Aminosäure in einer Peptidkette ist an zwei Peptidbindungen zur restlichen Kette beteiligt. Diese beiden Winkel pro Aminosäure bilden die Variablen für die Evolutionsstrategie.

Das Peptid Met-Enkephalin sieht in Form einer Beschreibungsdatei für das Programm *protein* folgendermaßen aus:

```
Met-Enkephalin (Tyr-Gly-Gly-Phe-Met)
tyr   -86  157  180 -165    87
gly  -165   79  180
gly    64  -94  180
phe   -80  -30  180  180    80
met   -79  146  180  -66  -179  -179
```

In der linken Spalte sind die Aminosäuren angegeben, die beiden linken Zahlenspalten sind die mit der Evolutionsstrategie variierten Winkel der Peptidbindungen, die restlichen Spalten geben die Winkel zu den verbleibenden Seitengruppen an. Das dadurch gegebene Optimierungsproblem wird also mit 10 Winkel-Variablen beschrieben. Als Fitneßfunktion wird direkt der Ausgabewert der Energieberechnung durch das *oplsaa*-Kraftfeld verwendet. Die Energie soll minimiert werden. Auf weitere Einzelheiten soll an dieser Stelle verzichtet werden, diese sind in [Scheer 99] zu finden.

7.2.1 Steady-State-Optimierung mit Median-Selektion

Es wurden die folgenden Strategien getestet:

- $(40 + (1, 10))^{500}$-Evolutionsstrategie mit lokaler Tournamentselektion, Ersetzungsstrategie: ältestes ersetzen, Ersetzungsbedingung: immer ersetzen.
- $(40 + 1)^{5000}$ Standard Steady-State-Evolutionsstrategie,
- $(40 + 1)^{5000}$-Evolutionsstrategie mit Median-Selektion, $n_p = 80$, $r_p = 0.15$, Ersetzungsstrategie: ältestes Individuum.

Bei allen Strategien wurde die Kovarianzmatrixadaption eingesetzt. Für jede Strategie wurden die Ergebnisse über 40 Läufe gemittelt. Dabei waren maximal $n_{\mathrm{fe,max}} = 5040$ Funktionsauswertungen pro Lauf zugelassen.

Der Vergleich der dabei erreichten Fitneßwerte ist in Abbildung 7.4 gezeigt. Die Strategie mit lokaler Tournamentselektion liefert auch hier die schlechtesten Ergebnisse, sie hat sich generell als nicht sinnvolle Möglichkeit herausgestellt. Die 95%-Vertrauensgrenze für den Mittelwert bei der Standard Steady-State-Strategie liegt bei ± 11.9, d. h. mit einer Wahrscheinlichkeit von 95% weicht der gemessene Mittelwert (-195.5) um nicht mehr als 11.9 vom wirklichen Mittelwert der Grundgesamtheit ab. Der gemessene Mittelwert der Strategie mit lokaler Tournamentselektion liegt bei -182.7 und unterscheidet sich somit signifikant davon. Die durchschnittlich erreichte Fitneß beim Verfahren mit Median-Selektion (-195.2) unterscheidet sich dagegen nicht signifikant. Die Standard Steady-State-Evolutionsstrategie erreicht im Mittel also dieselbe Fitneß wie die Evolutionsstrategie mit Median-Selektion.

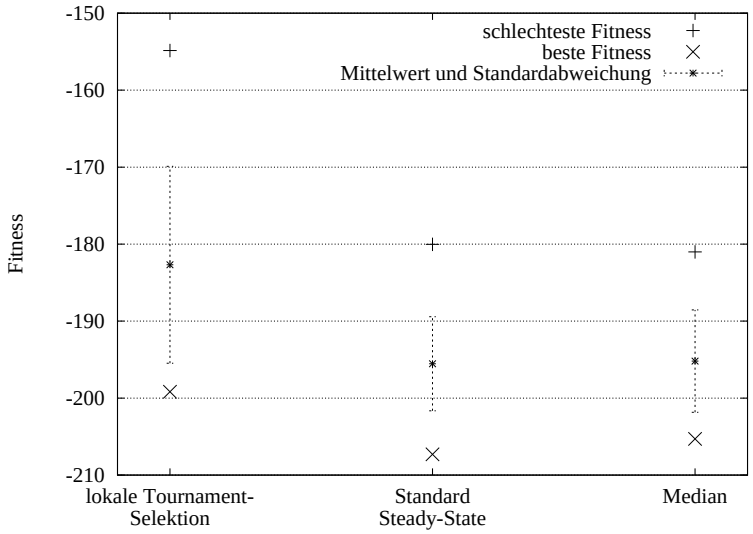


Abbildung 7.4: Vergleich der drei Strategien bei der Energieminimierung des Peptids.

Dies läßt darauf schließen, daß diese Anwendung keine so hohen Ansprüche an die Schrittweitenadaption stellt, wie manche der Benchmarkfunktionen oder die Linsenoptimierung im nächsten Abschnitt. Es ist durchaus positiv zu bewerten, daß die Strategie mit Median-Selektion keine schlechteren Ergebnisse liefert, als der Standard Steady-State-Algorithmus.

Eine Funktionsauswertung, welche die Energie für eine Peptidkonformation berechnet, dauert hier ca. 0.2 Sekunden. Dies ist länger, als die Kommunikationszeit, um die Daten und Fitneß des Individuums über das Netzwerk zu versenden. Dadurch kann die parallele Fitneßevaluation effizient arbeiten. In Abbildung 7.5 ist die Gesamtzeit der Optimierung (Steady-State-ES mit Median-Selektion) über der Anzahl der Prozessoren aufgetragen. Da die Implementierung nach dem Master-Slave-Modell arbeitet, steigt die Ausführungszeit mit zwei Prozessoren gegenüber einem Prozessor leicht an. Bei nur einem Prozessor führt dieser sowohl den Algorithmus als auch die Funktionsauswertungen durch. Bei zwei Prozessoren führt nur einer den Algorithmus und der andere die Funktionsauswertungen durch. Hier addiert sich dann die Kommunikationszeit. Im weiteren Verlauf mit mehr als zwei Prozessoren ist dann deutlich der Verlauf proportional zum Kehrwert der Anzahl der Prozessoren $\frac{1}{n_{\mathrm{proc}}}$ zu erkennen. Die Parallelität kann bei dieser Anwendung mit einer Steady-State-Evolutionsstrategie also sinnvoll ausgenutzt werden.

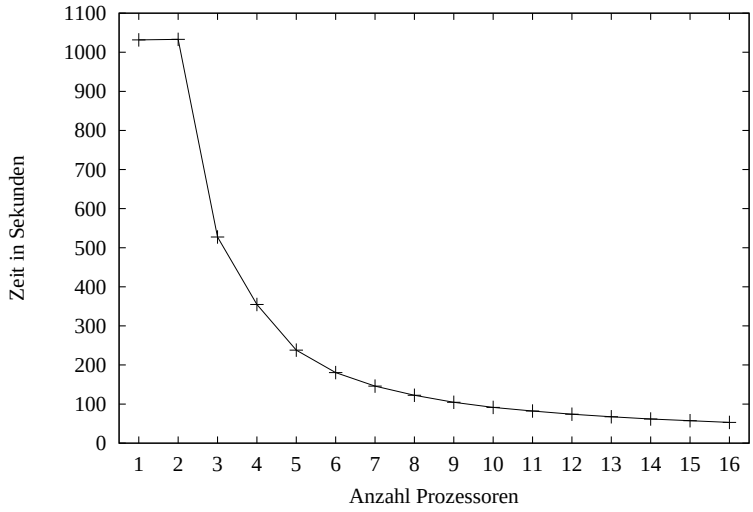


Abbildung 7.5: Skalierung der parallelen Fitneßevaluation über die Anzahl verwendeter Prozessoren.

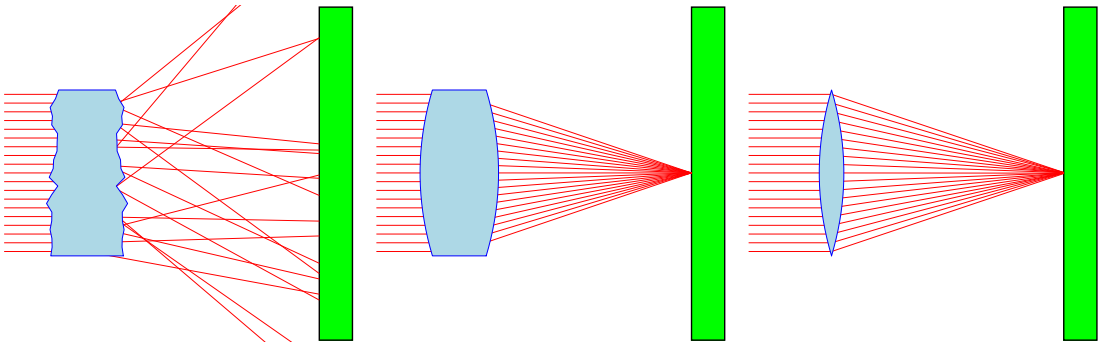
7.3 Optimierung einer Linse

Besonders für die Evolutionsstrategie geeignet ist die Optimierung eines Glaskörpers zur scharf fokussierenden Linse. Diese Anwendung wird in [Rechenberg 94] beschrieben. Der Glaskörper mit Radius R wird durch seine Dicke an n Stellen charakterisiert, welche direkt als n kontinuierliche Variablen x_1 bis x_n in den Individuen der Evolutionsstrategie repräsentiert werden. Dadurch setzt sich der Glaskörper aus $n-1$ Prismen der Höhe $h = 2R/(n-1)$ zusammen. Als Ziel soll die Linse derart geformt werden, daß alle Strahlen durch alle Prismen im Mittelpunkt der Fokusfläche konzentriert werden. Die Entfernung der Fokusfläche von der Linse wird mit b bezeichnet. Die Ablenkungen der Strahlen durch die Prismen werden mit einer vereinfachten Formel zur Brechung am dünnen Prisma berechnet. Die Fitneßfunktion ist die Summe der quadrierten Abweichungen der Strahlen vom Fokuspunkt. Als zweites Optimierungskriterium soll noch die Gesamtdicke der Linse minimiert werden. Dies wird durch Addition der dünnsten und dicksten Stelle der Linse realisiert:

$$f_{\text{Linse}}(\vec{x}) = \sum_{i=2}^{n} \left(R - \frac{h}{2} - h(i-1) - \frac{b}{h}(\varepsilon - 1)(x_i - x_{i-1}) \right)^2 + \min_{i=1\ldots n} x_i + \max_{i=1\ldots n} x_i \tag{7.2}$$

$$\text{Brechungsindex } \varepsilon = 1.6,\ \text{Radius } R = 25\text{mm},\ \text{Entfernung } b = 75\text{mm} \tag{7.3}$$

Die Fitneßfunktion zur Optimierung der Linse ist zwar unimodal, stellt aber durch die Verfolgung zweier Optimierungskriterien (Fokussierung und Linsendicke) dennoch eine Herausforderung für Optimierungsverfahren dar. Durch die implizite Gewichtung



(a) Generation 1 (b) Generation 450 (c) Generation 1060

Abbildung 7.6: Optimierung einer Linse mit einer $(1, 10)$ Evolutionsstrategie mit Kovarianzmatrixadaption.

der beiden Ziele ist die Fehlersummierung der zuerst minimierte Term. Dadurch wird eine Linsenform erreicht, die zwar sehr gut fokussiert, aber noch relativ dick ist (Abbildung 7.6(b)). Weitere Verbesserungen von diesem Punkt an können nur durch sehr spezielle Mutationen erzielt werden, die mit allen Variablen gleichzeitig die Dicke verringern. Andernfalls gewinnt der Term zur Fehlersummierung zu stark an Gewicht und liefert eine verschlechterte Fitneß. Solche gerichtete Mutationen treten nur mit sehr geringer Wahrscheinlichkeit auf, deshalb ist im Fitneßverlauf in Abbildung 7.7 ca. zwischen 4000 und 7000 Funktionsauswertungen ein Plateau zu erkennen. Von den in Abschnitt 2.4.3 vorgestellten Mutations- und Adaptionsverfahren sind nur die Kovarianzmatrix-Adaption (CMA) und das Differential Evolution-Verfahren in der Lage, die benötigte Mutationsrichtung zu lernen[1]. Nach der Adaptionsphase ist in Abbildung 7.7 ab 7000 Funktionsauswertungen wieder ein deutlicher Fortschritt zu erkennen, bis die Linse schließlich eine minimale Dicke unter Beibehaltung der Fokussierung erreicht hat (Abbildung 7.6(c)).

In den folgenden Tests war die Dimension der Anwendung $n = 20$. Als Adaptionsverfahren kam aus den eben genannten Gründen CMA zum Einsatz.

7.3.1 Steady-State-Optimierung mit Median-Selektion

Es wurden die folgenden Strategien getestet:

- $(1, 10)$-Evolutionsstrategie (nur 1 Prozessor),

[1]Differential Evolution benötigt allerdings eine Populationsgröße $\mu > 1$ und deutlich mehr Funktionsauswertungen.

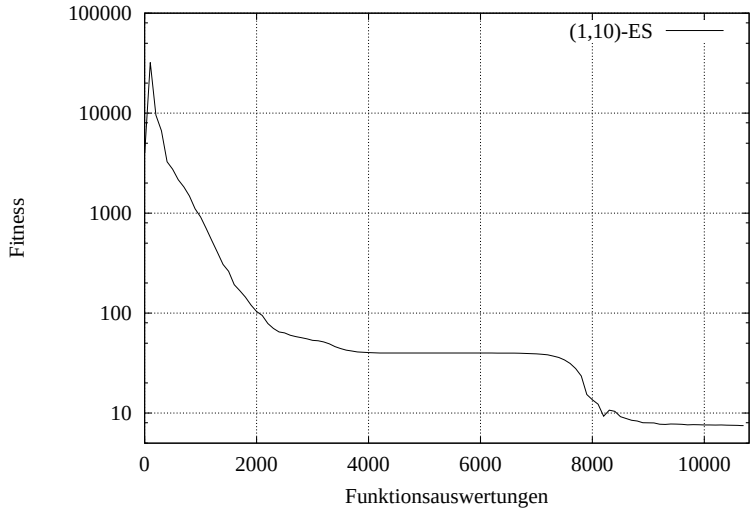


Abbildung 7.7: Beispielhafter Fitneßverlauf bei der Linsenoptimierung mit einer $(1, 10)$ Evolutionsstrategie mit CMA.

- $(1 + 1)$ Standard Steady-State-Evolutionsstrategie,
- $(1 + 1)$-Evolutionsstrategie mit Median-Selektion, $n_p = 10$, $r_p = 0.1$, Ersetzungsstrategie: ältestes Individuum ersetzen.

Die $(1, 10)$-ES mit Kovarianzmatrixadaption ist die minimale generationenbasierte Evolutionsstrategie, die in der Lage ist, die Linse zu optimieren. Deshalb dient sie hier als Vergleichsstrategie.

Für jede Strategie wurden 40 Läufe durchgeführt. Abbruchkriterium war das Erreichen einer Fitneß von 7.55. Bei diesem Fitneßwert ist die Linse minimal dünn und fokussiert sehr gut (siehe Abbildung 7.6(c)). Dabei war eine maximale Zahl von Funktionsauswertungen von $n_{\mathrm{fe,max}} = 400000$ zugelassen, ansonsten wurde ein Lauf als nicht konvergiert gezählt.

Die Ergebnisse des Vergleichs sind in Abbildung 7.8 zu sehen. Links sind jeweils die Durchschnitte und die Standardabweichungen für die 40 Läufe dargestellt. Rechts ist die Anzahl der nicht konvergierten Läufe zu sehen. Die sequentielle $(1, 10)$-Evolutionsstrategie dient als Ausgangspunkt dieses Vergleiches und benötigt im Durchschnitt 12339 Funktionsauswertungen, um eine dünne, fokussierende Linse zu evolvieren. Dabei konvergieren alle 40 Läufe. Die $(1 + 1)$ Standard Steady-State-Evolutionsstrategie benötigt deutlich mehr Funktionsauswertungen, mehr als ca. 20000. Sie hat Schwierigkeiten, die für diese Anwendung wichtige Adaption der Mutationsschrittweiten schnell genug durchzuführen. In einigen Fällen gelingt ihr dies überhaupt nicht innerhalb der vorgegebenen Zahl von Funktionsauswertungen. Die $(1+1)$-Evolutionsstrategie mit Median-Selektion dagegen bewältigt diese Aufgabe in allen Versuchen.

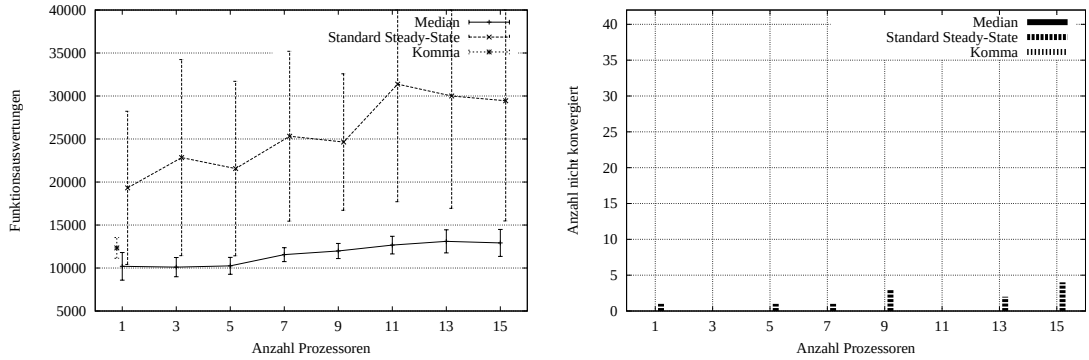


Abbildung 7.8: Vergleich der Evolutionsstrategien bei der Linsenoptimierung.

Durch die sofortige Integration bereits evolvierter Individuen in die Elternpopulation kann sie sogar einige Funktionsauswertungen gegenüber der $(1,10)$-ES einsparen. Sie benötigt im Mittel nur noch 10194 Funktionsauswertungen (auf einem Prozessor). Dies ist im Vergleich zur $(1,10)$-ES (100%) eine Verbesserung um 17.3%. Bei der Parallelisierung steigt mit zunehmender Zahl von Prozessoren die Zahl der benötigten Funktionsauswertungen durch die überlappende Auswertung an.

Kapitel 8

Bewertung und Ausblick

In dieser Arbeit wurden zwei neue parallele Verfahren und das Softwaresystem EvA vorgestellt, die speziell für den praktischen Einsatz von Evolutionsstrategien und die dabei auftretenden Problemstellungen geeignet sind.

Das EvA-Softwarepaket bildet ein einfach zu bedienendes System mit graphischer Benutzeroberfläche, welches auch von nicht-Experten schnell verwendet werden kann. Bei vielen anderen im Internet erhältlichen Systemen und Programmen ist dagegen noch relativ viel eigene Programmierarbeit erforderlich. Das Modul für Evolutionsstrategien umfaßt viele algorithmische Varianten und Verfahren. Insbesondere die modernen und leistungsfähigen Schrittweitenadaptionsverfahren sind enthalten. Ebenso ist eine Parallelisierung nach dem Inselmodell und die parallele Fitneßevaluation für rechenintensive Anwendungen implementiert.

Beim Einsatz für eine neue Optimierungsanwendung ist die Formulierung und Einbindung einer Fitneßfunktion die zentrale Aufgabe. Dies ist bei EvA sehr flexibel gehalten. Es können verschiedene Methoden der Constraint-Behandlung angewendet werden und es existiert auch die Möglichkeit, Benutzereingaben komfortabel über graphische oder textuelle Menüpunkte zu verarbeiten.

Es existiert ebenfalls eine Infrastruktur zu Einbindung des Optimierungsmoduls als Komponente in ein komplexeres System, wie es in der Praxis oft vorkommt. Damit erzielt EvA eine herausragende Position, was die Integration und Vollständigkeit der Software betrifft.

In dieser Arbeit wurde erstmals ein Selektionsverfahren entwickelt, welches sich speziell für Steady-State-Evolutionsstrategien eignet. Steady-State-Algorithmen werden besonders immer dann eingesetzt, wenn mit paralleler Fitneßevaluation gearbeitet wird, da bei generationenbasierten Algorithmen die evaluierten Individuen nicht sofort weiterverarbeitet werden. Es ist allerdings bekannt, daß bei elitärer Selektion, wie sie beim Standard Steady-State-Algorithmus auftritt, der Selbstadaptionsprozeß empfindlich gestört werden kann. Eine leistungsfähige Selbstadaption der Mutationsschrittweiten ist aber die herausragendste Eigenschaft von Evolutionsstrategien. Manche

Optimierungsprobleme, wie z. B. die Optimierung einer optischen Linse, sind erst dadurch überhaupt vollständig lösbar. Es konnte an vielen Beispielen gezeigt werden, daß die neue Median-Selektion für Steady-State-Evolutionsstrategien gerade bei Optimierungsanwendungen, die hohe Anforderungen an die Selbstadaption stellen, die Standard Steady-State-Evolutionsstrategie in der Leistungsfähigkeit übertrifft. Sind die Anforderungen an die Selbstadaption nicht sehr hoch, so haben beide Verfahren ungefähr dieselbe Leistung. Deswegen wird die Steady-State-Evolutionsstrategie mit Median-Selektion als uneingeschränkter Ersatz für die Standard Steady-State-Evolutionsstrategie vorgeschlagen.

Es konnte ebenso gezeigt werden, daß es auch ohne Parallelisierung auf nur einem Prozessor vorteilhaft ist, eine Steady-State-Evolutionsstrategie zu verwenden. Hier können selbst bei der minimalen Populationsgröße von $\mu = 1$ im Vergleich zur generationenbasierten (μ, λ)-Evolutionsstrategie Funktionsauswertungen eingespart werden.

Für die Zukunft ist eine Erweiterung der Median-Selektion vorstellbar, bei der nicht mehr nur die direkt im FIFO-Puffer vorkommenden Fitneßwerte verwendet werden. Hier ist es denkbar, auch zwischen zwei Fitneßwerten zu interpolieren, damit bei kleinen Pufferlängen die dadurch repräsentierte Verteilung geglättet wird.

Mit der erweiterten Meta-Evolutionsstrategie wurde erstmals ein selbstadaptiver Algorithmus zur Meta-Optimierung vorgestellt, welcher alle drei Arten von Parametern spezialisiert behandelt: kontinuierliche, ganzzahlige und Aufzählungsparameter. Im realisierten Verfahren wurde im Gegensatz zu bisherigen Arbeiten, der Wertebereich der Parameter nicht durch willkürlich eingeführte, diskrete Stufen eingeschränkt. Durch Integration eines speziellen Adaptionsverfahrens für ganzzahlige Parameter, konnte hier die volle Genauigkeit des Ganzzahlbereichs ausgeschöpft werden. Mit der Verwendung einer Evolutionsstrategie für die kontinuierlichen Parameter, liegt auch hier keine Beschränkung der Genauigkeit vor. Erstmals wurde ein solcher Algorithmus zur Meta-Optimierung einer Evolutionsstrategie eingesetzt. In ersten Experimenten hat sich dieser der empirischen Wahl von Parametern überlegen gezeigt. Für die Anwendung der Spannmittelplazierung in der Holzindustrie konnten Parameter evolviert werden, welche die Aufgabenstellung schneller lösen und bessere Fitneßwerte erzielen. Die Geschwindigkeitssteigerung ist für dieses Problem besonders wichtig, da sehr oft ähnliche Problemstellungen aus dieser Klasse gelöst werden müssen.

Für die Meta-Optimierung wurde eine neue Fitneßfunktion entwickelt, welche die Leistung einer Evolutionsstrategie beurteilt. Damit werden als erstes Kriterium Strategien evolviert, die möglichst gute Fitneßwerte erzielen. Wird eine bestimmte Fitneßgrenze erreicht, so werden als zweites Kriterium Strategien evolviert, welche diese Grenze möglichst schnell erreichen. Diese Fitneßfunktion hat sich bei der Spannmittelplazierung bewährt.

Die Schrittweitenadaption wurde mit einer globalen Schrittweite für alle Aufzählungsparameter realisiert. Eine Erweiterung auf separate Schrittweiten bietet sich hier für die Zukunft an. Dazu sind noch Untersuchungen über das Zusammenspiel der Ad-

aptionsverfahren für die drei unterschiedlichen Parametertypen notwendig. Inwieweit dabei eine Korrelation möglich und sinnvoll ist, stellt eine weitere offene Frage dar.

Gerade die Meta-Optimierung bietet ein Szenario, für das die Steady-State-Evolutionsstrategie mit Median-Selektion geeignet ist: lang dauernde Fitneßauswertungen und Einsatz von Selbstadaptionsmethoden. Durch die Verrauschtheit der Fitneßfunktion bei der Meta-Optimierung öffnet sich hier ein weiteres Anwendungsgebiet für die Median-Selektion. Sie könnte sich als vorteilhaft gegenüber der Standard Steady-State-Evolutionsstrategie erweisen, da die Individuen nur eine begrenzte Lebensdauer besitzen. Dadurch können nur zufällig als gut bewertete Individuen nicht unbegrenzt lange überleben und damit das Ergebnis verfälschen. Ihre Variablenwerte müssen sich stattdessen über die Fortpflanzung durch Rekombination in neueren Individuen mehrfach bewähren. Dieser Punkt bietet Raum für weitere Untersuchungen.

Insgesamt konnte durch diese Arbeit in mehreren Aspekten ein Beitrag zum sinnvollen, praktischen Einsatz von parallelen Evolutionsstrategien geleistet werden.

Literaturverzeichnis

[Alba et al. 00] Enrique Alba, José M^a Troya. *Cellular Evolutionary Algorithms: Evaluating the Influence of Ratio*. In Schoenauer et al. [Schoenauer et al. 00], Seiten 29–38.

[Bäck 91] Thomas Bäck. *Extendend Selection Mechanisms in Genetic Algorithms*. In Belew und Booker [Belew et al. 91], Seiten 92–99.

[Bäck 92a] Thomas Bäck. *The Interaction of Mutation Rate, Selection and Self-Adaptation Within a Genetic Algorithm*. In Männer und Manderick [Männer et al. 92], Seiten 85–94.

[Bäck 92b] Thomas Bäck. *A User's Guide to GENEsYs 1.0*. Technical report, University of Dortmund, Department of Computer Science, System Analysis Research Group, 1992.

[Bäck 94a] Thomas Bäck. *Parallel Optimization of Evolutionary Algorithms*. In Davidor et al. [Davidor et al. 94], Seiten 418–427.

[Bäck 94b] Thomas Bäck. *Selective Pressure in Evolutionary Algorithms: A Characterization of Selection Mechanisms*. In *Proceedings of the First IEEE Conference on Evolutionary Computation*, Seiten 57–62, Piscataway NJ, 1994. IEEE Press. `http://ls11-www.informatik.uni-dortmund.de/people/baeck/papers/wcci94-s%el.ps.gz`.

[Bäck 96] Thomas Bäck. *Evolutionary Algorithms in Theory and Practice*. Oxford University Press, 1996.

[Bäck 97] Thomas Bäck, editor. *Proceedings of the 7th International Conference on Genetic Algorithms (ICGA)*, San Francisco, California, 1997. Morgan Kaufmann Publishers, Inc. ISBN 1-55860-487-1.

[Bäck et al. 91] Thomas Bäck, Frank Hoffmeister, Hans-Paul Schwefel. *A survey of evolutionary strategies*. In Belew und Booker [Belew et al. 91], Seiten 2–9.

[Bäck et al. 95] Thomas Bäck, Martin Schütz. *Evolution Strategies for Mixed-Integer Optimization of Optical Multilayer Systems*. In J. R. McDonnell, R. G. Reynolds, D. B. Fogel, editors, *Evolutionary Programming IV: Proceedings of the Fourth Annual Conference on Evolutionary Programming*, Seiten 33–51, Cambridge, MA, 1995. MIT Press.

[Bäck et al. 96] Thomas Bäck, Martin Schütz. *Intelligent Mutation Rate Control in Canonical Genetic Algorithms*. In Zbigniew Michalewicz, Maciek W. Rás, editors, *Proceedings of the Ninth International Symposium on Foundations of Intelligent Systems*, Band 1079 von *LNAI*, Seiten 158–167, Berlin, 1996.

[Bäck et al. 97] Thomas Bäck, David B. Fogel, Zbigniew Michalewicz, editors. *Handbook of Evolutionary Computation*. IOP Publishing and Oxford University Press, New York, Bristol, 1997.

[Baumann 95] Roland Baumann. *Verteilte Genetische Algorithmen auf dem MIMD–Parallelrechner Intel Paragon*. Diplomarbeit Nr. 1227, Universität Stuttgart, Institut für Parallele und Verteilte Höchstleistungsrechner, 1995.

[Belding 95] T. C. Belding. *The distributed genetic algorithm revisited*. In L. Eschelman, editor, *Proceedings of the Sixth International Conferenc on Genetic Algorithms*, Seiten 114–121, San Mateo, CA, 1995. Morgan Kaufmann.

[Belew et al. 91] Richard K. Belew, Lashon B. Booker, editors. *Proceedings of the Fourth International Conference on Genetic Algorithms*, University of California, San Diego, July 13-16 1991. Morgan Kaufmann Publishers, San Mateo, California.

[Bramlette 91] Mark F. Bramlette. *Initialization, Mutation and Selection Methods in Genetic Algorithms for Function Optimization*. In Belew und Booker [Belew et al. 91], Seiten 100–107.

[Bräunl 93] Thomas Bräunl. *Parallele Programmierung – eine Einführung*. Vieweg, Braunschweig, Wiesbaden, 1993. ISBN 3-528-05142-6.

[Caldwell et al. 91] Craig Caldwell, Victor S. Johnston. *Tracking a Criminal Suspect through Face Space" with a Genetic Algorithm*. In Belew und Booker [Belew et al. 91], Seiten 416–421.

[Cerny 85] V. Cerny. *Thermodynamical Approach to the Traveling Salesman Problem: An Efficient Simulation Algorithm*. *Journal of Optimisation Theory an Applications*, (45):41–51, 1985.

[Cicirello et al. 00] Vincent A. Cicirello, Stephen F. Smith. *Modeling GA Performance for Control Parameter Optimization*. In Whitley et al. [Whitley et al. 00], Seiten 235–242.

[CP97] Erick Cantú-Paz, David E. Goldberg. *Predicting Speedups of Idealized Bounding Cases of Parallel Genetic Algorithms*. In Bäck [Bäck 97], Seiten 113–120. ISBN 1-55860-487-1.

[Darwin 29] Charles Darwin. *The Origin of Species by Means of Natural Selection or the Preservation of Favored Races in The Struggle for Life*. The Book League of America, New York, 1929. originally published by Murray, London, 1859.

[Davidor et al. 94] Yuval Davidor, Hans-Paul Schwefel, Reinhard Männer, editors. *Parallel Problem Solving from Nature – PPSN III*, Band 3 von *Lecture Notes in Computer Science*, Jerusalem, Israel, October 1994. Springer.

[Dueck 93] G. Dueck. *New Optimization Heuristics. The Great Deluge Algorithm and the Record-to-Record Travel*. *Journal of Computational Physics*, (104):86–92, 1993.

[Dueck et al. 90] G. Dueck, T. Scheuer. *Threshold Accepting: A General Purpose Optimization Algorithm Appearing Superior to Simulated Annealing*. *Journal of Computational Physics*, (90):161–175, 1990.

[Dueck et al. 93] G. Dueck, T. Scheuer, H.-M. Wallmeier. *Toleranzschwelle und Sintflut: neue Ideen zur Optimierung*. *Spektrum der Wissenschaft*, (3):42–51, 1993.

[Evo95] *Verbundprojekt EvoAlg - Grundlagen und Anwendungen Evolutionärer Algorithmen*. In Marius van der Meer Gottfried Wolf, Ralph Schmidt, editor, *Berichte vom Statusseminar bmb+f*, Seiten 259–292. Humboldt-Universität zu Berlin, Siemens AG München, Informatik Centrum Dortmund, Braunschweig, 7. + 8. November 1995.

[Flynn 66] M. J. Flynn. *Very High Speed Computing Systems*. In *Proceedings of the IEEE*, Band 54, Seiten 1901–1909, 1966.

[Fogel 92] D. B. Fogel. *Evolving Artificial Intelligence*. Doktorarbeit, University of California, San Diego, 1992.

[Fogel 95] D. B. Fogel. *Evolutionary Computation. Toward a New Philosophy of Machine Intelligence*. IEEE Press, 1995.

[Fogel et al. 66] L. J. Fogel, A. J. Owens, M. J. Walsh. *Artificial Intelligence through Simulated Evolution*. John Wiley & Sons, New York, 1966.

[Fogel et al. 91] D. B. Fogel, L. J. Fogel, W. Atmar. *Meta-evolutionary programming*. In R. R. Chen, editor, *Proceedings of the 25th Asilomar Conference on Signals, Systems and Computers*, Seiten 540–545, Pacific Grove, CA, 1991.

[Goldberg 89] David E. Goldberg. *Genetic algorithms in search, optimization and machine learning*. Addison–Wesley, 1989.

[Görzig 95] Steffen Görzig. *Massiv Parallele Evolutionsstrategien auf dem Parallelrechner MasPar MP-1*. Diplomarbeit Nr. 1291, Universität Stuttgart, Institut für Parallele und Verteilte Höchstleistungsrechner, 1995.

[GPL] *GNU General Public License*. `http://www.gnu.org/copyleft/gpl.html`.

[Grefenstette 86] John J. Grefenstette. *Optimization of Control Parameters for Genetic Algorithms*. *IEEE Transactions on Systems, Man and Cybernetics*, 16(1):122–128, February 1986.

[Hansen 00] Nikolaus Hansen. *Invariance, Self-Adaptation and Correlated Mutations in Evolution Strategies*. In Schoenauer et al. [Schoenauer et al. 00], Seiten 355–364.

[Hansen et al. 95a] Nikolaus Hansen, Andreas Ostermeier, Andreas Gawelczyk. *On the Adaptation of Arbitrary Normal Mutation Distributions in Evolution Strategies: The Generating Set Adaptation*. In L. J. Eshelman, editor, *Proceedings of the Sixth International Conference on Genetic Algorithms (ICGA 6)*, Seiten 57–64, 1995.

[Hansen et al. 95b] Nikolaus Hansen, Andreas Ostermeier, Andreas Gawelczyk. *Über die Adaptation von allgemeinen, Koordinatensystem-unabhängigen, normalverteilten Mutationen in der Evolutionsstrategie: Die Erzeugendensystemadaptation*. Technical report, Technische Universität Berlin, Berlin, Februar 1995. `ftp://ftp-bionik.fb10.tu-berlin.de/pub/papers/Bionik/tr-02-95.ps.Z`.

[Hansen et al. 96] Nikolaus Hansen, Andreas Ostermeier. *Adapting Arbitrary Normal Mutation Distributions in Evolution Strategies: The Covariance Matrix Adaptation*. In *Proceedings of the 1996 IEEE International Conference on Evolutionary Computation (ICEC '96)*, Seiten 312–317, Nagoya, Japan, 1996. IEEE. `ftp://ftp-bionik.fb10.tu-berlin.de/pub/papers/Bionik/CMAES.ps.Z`.

[Hansen et al. 97] Nikolaus Hansen, Andreas Ostermeier. *Convergence Properties of Evolution Strategies with the Derandomized Covariance Matrix Adaptation: The (my/my_I, lambda)-CMA-ES*. In Prof. Dr. Dr. h. c. Hans-Jürgen Zimmermann, editor, *Proceedings of the 5th European Congress on Intelligent Techniques and Soft Computing (EUFIT'97)*, Seiten 650–654, Aachen, Germany, September 1997. ftp://ftp-bionik.fb10.tu-berlin.de/pub/papers/Bionik/CMAES2.ps.Z.

[Hansen et al. 00] Nikolaus Hansen, Andreas Ostermeier. *Completely Derandomized Self-Adaptation in Evolution Strategies*. *Evolutionary Computation, Special Issue on Self-Adaptation*, 2000.

[Holland 62] John. H. Holland. *Outline for a logical theory of adaptive systems*. *Journal of the Association for Computing Machinery*, (3):297–314, 1962.

[Holland 75] John H. Holland. *Adaption in Neural an Artificial Systems*. Ann Arbor The University of Michigan Press, 1975.

[Holland et al. 78] John. H. Holland, J. S. Reitman. *Pattern-Directed Inference Systems*, Seiten 313–329. Academic Press, New York, 1978.

[Holland et al. 86] John H. Holland, K. J. Holyoak, R. E. Nisbett, P. R. Thagard. *Introduction: Processes of Inference, Learning and Discovery*. MIT Press, Cambridge, MA, 1986.

[Huhse et al. 00] Jutta Huhse, Andreas Zell. *Evolution Strategy with Neighborhood Attraction*. In H. Bothe, R. Rojas, editors, *Procee-*

dings of the Second ICSC Symposium on Neural Computation – NC 2000, Seiten 363–369. ICSC Academic Press, Canada/Switzerland, 2000.

[Hummler 95] Alex Hummler. *Massiv parallele Genetische Algorithmen auf dem Parallelrechner MasPar MP-1*. Diplomarbeit Nr. 1240, Universität Stuttgart, Institut für Parallele und Verteilte Höchstleistungsrechner, 1995.

[Kirkpatrick et al. 83] S. Kirkpatrick, C. D. Gelatt Jr., M. P. Vecchi. *Optimization by Simulated Annealing*. *Science*, (220):671–680, 1983.

[Koch et al. 99] Thomas E. Koch, Ralph Müller, Franz Schneider, Andreas Zell. *Positionierung von Spannmitteln zur Werkstückfixierung mittels Evolutionärer Algorithmen*. *Automatisierungstechnische Praxis atp*, 41(9):32–39, 1999. Oldenbourg Verlag.

[Koch et al. 00] Thomas E. Koch, Jürgen Wakunda, Volker Scheer, Andreas Zell. *A parallel, hybrid Meta Optimization for Finding better Parameters of an Evolution Strategy in Real World Optimization Problems*. In A. Wu, editor, *Proceedings of the Genetic and Evolutionary Computation Conference 2000 Workshop Program*, Seiten 17–19. Las Vegas, U.S.A., July 8 2000.

[Koza 92] John R. Koza. *Genetic Programming*. MIT Press, Cambridge/MA, 1992.

[Koza 94] John R. Koza. *Genetic Programming II*. MIT Press, Cambridge/MA, 1994.

[Männer et al. 92] Reinhard Männer, Bernard Manderick, editors. *Parallel Problem Solving from Nature – PPSN II*, Band 2, Amsterdam, Netherlands, September 1992. Elsevier Science Publishers.

[Michalewicz 91] Zbigniew Michalewicz. *Handling Constraints in Genetic Algorithms*. In Belew und Booker [Belew et al. 91], Seiten 151–157.

[MPIa] *MPICH*. `http://www-unix.mcs.anl.gov/mpi`. Mathematics and Computer Science Division, Argonne National Laboratory, 9700 South Cass Avenue, Argonne, Illinois, 60439-4844, USA.

[MPIb] *LAM (Local Area Multicomputer) MPI Implementation*. `http://www.mpi.nd.edu/lam/`. University of Notre Dame, IN, USA.

[Obalek 94] J. Obalek. *Rekombinationsoperatoren für Evolutionsstrategien*. Diplomarbeit, Universität Dortmund, Fachbereich Informatik, 1994.

[OG] *The Open Group*. `http://www.opengroup.org`. 1010 El Camino Real, Suite 380, Menlo Park, CA 94025/4345, USA.

[Ostermeier et al. 93] Andreas Ostermeier, Andreas Gawelczyk, Nikolaus Hansen. *A Derandomized Approach to Self Adaptation of Evolution Strategies*. Technical report, Technische Universität Berlin, Juli 1993. `ftp://ftp-bionik.fb10.tu-berlin.de/pub/papers/Bionik/tr-03-93.ps.Z`.

[Ostermeier et al. 94] Andreas Ostermeier, Andreas Gawelczyk, Nikolaus Hansen. *Step-Size Adaptation Based on Non-Local Use of Selection Information*. In Davidor et al. [Davidor et al. 94], Seiten 189–198.

[Ott 98] Hartmut Ott. *Evolutionäre Algorithmen für die Positionsoptimierung von Vakuumspannern zur Holzwerkstückfixierung*. Diplomarbeit Nr. WSI-97/14, Universität Tübingen, Wilhelm-Schickard-Institut für Informatik, 1998.

[Ott et al. 98] Hartmut Ott, Jürgen Wakunda, Andreas Zell. *Evolutionäre Algorithmen für die Positionsoptimierung von Vakuumspannern zur Holzwerkstückfixierung*. In M. Tietze J. Kuhl, V. Nissen, editor, *Tagungsband zum 4. Göttinger Symposium Soft Computing in Produktion und Materialwirtschaft*, Universität Göttingen, 12. März 1998. Culliver Verlag, Göttingen. ISBN 3-89712-134-4.

[Ousterhout 95] John K. Ousterhout. *Tcl und Tk. Entwicklung grafischer Benutzerschnittstellen für das X Window System*. Professional Computing. Addison–Wesley, 1995.

[Paredis 92] Jan Paredis. *Co-evolutionary Constraint Satisfaction*. In Männer und Manderick [Männer et al. 92], Seiten 46–55.

[Ponder 98] Jay William Ponder. *TINKER – Software Tools for Molecular Design, Version 3.6*, February 1998. `http://dasher.wustl.edu/tinker/`.

[Price et al. 95] Kenneth Price, Rainer Storn. *Differential Evolution - a Simple and Efficient Adaptive Scheme for Global Optimization over Continuous Spaces*. Technical Report TR-95-012, International Computer Science Institute (ICSI), University of California at Berkeley, March 1995.

[Price et al. 97a] Kenneth Price, Rainer Storn. *Differential Evolution*. *Dr. Dobb's Journal*, Seiten 18–24, April 1997.

[Price et al. 97b] Kenneth Price, Rainer Storn. *Differential Evolution - a Simple and Efficient Heuristic for Global Optimization over Continuous Spaces*. *Journal of Global Optimization*, 11:341–359, 1997.

[Rechenberg 73] Ingo Rechenberg. *Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Doktorarbeit, TU Berlin, F. f. Maschinenwesen, Oktober 1973. Ersch. auch in: Schriften zur Informatik 1971.

[Rechenberg 94] Ingo Rechenberg. *Evolutionsstrategie '94*, Band 1 von *Werkstatt Bionik und Evolutionstechnik*. frommann–holzboog, Stuttgart, 1994.

[Reinelt 95] Gerhard Reinelt. *TSPLIB95*. Interdisziplinäres Zentrum für Wissenschaftliches Rechnen (IWR), Heidelberg, 1995. `http://www.iwr.uni-heidelberg.de/iwr/comopt/software/TSPLIB95/`.

[Rudolph 94] Günter Rudolph. *An Evolutionary Algorithm for Integer Programming*. In Davidor et al. [Davidor et al. 94], Seiten 139–147.

[Saravanan et al. 96] N. Saravanan, David B. Fogel. *An Empirical Comparison of Methods for Correlated Mutations under Self-Adaptation*. In *Fifth Annual Conference on Evolutionary Programming (ACEP V)*, San Diego, California, 29.2.–2.3.1996.

[Sarma et al. 96] Jayshree Sarma, Kenneth De Jong. *An Analysis of the Effects of Neighborhood Size and Shape on Local Selection Algorithms*. In Voigt et al. [Voigt et al. 96], Seiten 236–244. ISBN 3-540-61723-X.

[Scheer 99] Volker Scheer. *Parallele Meta-Optimierung einer Konformationsanalyse*. Diplomarbeit nr. wsi-99/3, Universität Tübingen, Wilhelm-Schickard-Institut für Informatik, Lehrstuhl Rechnerarchitektur, 1999.

[Schnecke et al. 96] Volker Schnecke, Oliver Vornberger. *An Adaptive Parallel Genetic Algorithm for VLSI-Layout Optimization*. In Voigt et al. [Voigt et al. 96], Seiten 859–868. ISBN 3-540-61723-X.

[Schoenauer et al. 00] Marc Schoenauer, Kalyanmoy Deb, Günther Rudolph, Xin Yao, Evelyne Lutton, Juan Julian Merelo, Hans-Paul Schwefel, editors. *Proceedings of the 6th Internation Conference Parallel Problem Solving from Nature - PPSN VI*, Band 1917 von *Lecture Notes in Computer Science*, Paris, Fance, September 16-20 2000. Springer Verlag.

[Schwefel 65] Hans-Paul Schwefel. *Kybernetische Evolution als Strategie der Experimentellen Forschung in der Strömungstechnik*. Diplomarbeit, Technische Universität Berlin, 1965.

[Schwefel 81] Hans-Paul Schwefel. *Numerical Optimization of Computer Models*. Wiley, Chichester, 1981.

[Schwefel 87] Hans-Paul Schwefel. *Collective Phenomena in Evolutionary Systems*. In P. Checkland, I. Kiss, editors, *Problems of Constancy and Change — The Complementarity of Systems Approaches to Complexity, Papers presented at the 31st Annual Meeting of the International Society for General System Research*, Band 2, Seiten 1025–1033, Budapest, 1.–5. Juni 1987. International Society for General System Research.

[Schwefel 92] Hans-Paul Schwefel. *Natural Evolution and Collective Optimum Seeking*. In A. Sydow, editor, *Computational Systems Analysis — Topics and Trends*, Seiten 5–14. Elsevier, Amsterdam, 1992. `ftp://lumpi.informatik.uni-dortmund.de/pub/ES/papers/sydow.ps.gz`.

[Schwefel 95] Hans-Paul Schwefel. *Evolution and optimum seeking*. Sixth-Generation Computer Technology Series. John Wiley & Sons, INC., 1995.

[Schwehm 97] Markus Schwehm. *Globale Optimierung mit massiv parallelen genetischen Algorithmen*. Dissertation, arbeitsberichte des immd, band 30, nr. 1, Universität Erlangen-Nürnberg, 1997.

[Smith et al. 96] Jim E. Smith, Terence C. Fogarty. *Self adaptation of mutation rates in a steady state genetic algorithm*. In *Proceedings of the 1996 IEEE Conference on Evolutionary Computation*, Seiten 318–323, New York, 1996. IEEE Press.

[SV96] Dirk Schlierkamp-Voosen, Heinz Mühlenbein. *Adaptation of Population Sizes by Competing Subpopulations*. In *Proceedings of the 1996 IEEE International Conference on Evolutionary Computation (ICEC '96)*, Seiten 330–335, Nagoya, Japan, 1996. IEEE.

[Tanese 97] R. Tanese. *Parallel genetic algorithm for a hypercube*. In J. J. Grefenstette, editor, *Proceedings of the Second International Conference on Genetic Algorithms*, Seiten 177–183, Hillsdale, NJ, 1897. Lawrence Erlbaum Associates.

[Tanese 89] R. Tanese. *Distributet genetic algorithms*. In J. D. Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, Seiten 434–439, San Mateo, CA, 1989. Morgan Kaufmann.

[Tusonn et al. 98] Andrew Tusonn, Peter Ross. *Adapting Operator Settings in Genetic Algorithms*. *Evolutionary Computation*, 6:161–184, Summer 1998.

[Voigt et al. 96] Hans-Michael Voigt, Werner Ebeling, Ingo Rechenberg, Hans-Paul Schwefel, editors. *Parallel Problem Solving from Nature – PPSN IV*, Lecture Notes in Computer Science, Berlin, Germany, September 22-27 1996. Springer Verlag. ISBN 3-540-61723-X.

[Wakunda 95] Jürgen Wakunda. *Verteilte Evolutionsstrategien auf dem Supercomputer Intel Paragon*. Diplomarbeit Nr. 1300, Universität Stuttgart, Institut für Parallele und Verteilte Höchstleistungsrechner, 1995.

[Wakunda et al. 97] Jürgen Wakunda, Andreas Zell. *EvA - A Tool for Optimization with Evolutionary Algorithms*. In *Proceedings of the 23rd EUROMICRO Conference*, Budapest, Hungary, September 1-4 1997.

[Wakunda et al. 00a] Jürgen Wakunda, Andreas Zell. *Median-Selection for Parallel Steady-State Evolution Strategies*. In Schoenauer et al. [Schoenauer et al. 00], Seiten 405–414.

[Wakunda et al. 00b] Jürgen Wakunda, Andreas Zell. *A new Selection Scheme for Steady-State Evolution Strategies*. In Whitley et al. [Whitley et al. 00], Seiten 794–801.

[Watson et al. 97] Andrew H. Watson, Dr. Ian C. Parmee. *Steady State Genetic Programming with Constrained Complexity Crossover Using Species Sub-Populations*. In Bäck [Bäck 97], Seiten 315–321. ISBN 1-55860-487-1.

[Whitley et al. 00] Darell Whitley, David Goldberg, Erick Cantú-Paz, Lee Spector, Ian Parmee, Hans-Georg Beyer, editors. *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO*

2000, Las Vegas, Nevada, July 10-12 2000. Morgan Kaufmann Publishers, San Francisco, California.

Anhang A

Abkürzungen und Mathematische Symbole

Abkürzungen

Abkürzung	Bedeutung
EA	Evolutionärer Algorithmus (auch Mehrzahl)
EC	Evolutionary Computation (ebenfalls ein Oberbegriff)
ES	Evolutionsstrategie
GA	Genetischer Algorithmus
EP	Evolutionary Programming
GP	Genetic Programming
SA	Simulated Annealing
TA	Threshold Accepting
GD	Great Deluge Algorithmus (dt.: Sintflut Algorithmus)
DE	Differential Evolution
NN	Neuronales Netz
FL	Fuzzy Logic
FOP	Free Optimization Problem
CSP	Constraint Satisfaction Problem
COP	Constraint Optimization Problem
TSP	Traveling Salesman Problem
GUI	Graphical User Interface

Mathematische Symbole

Generell gilt:

- Großbuchstaben bezeichnen Mengen, z. B. eine Population P, die Randbedingungen G.
- Alle Variablen n, meist mit einem Index versehen n_{index}, stellen eine Anzahl dar.
- Indizes werden mit i und j bezeichnet.

Symbol	Wertebereich	Bedeutung
$\mathbb{R}$		Menge/Raum der reellen Zahlen
$\mathbb{B}$	$\{0,1\}$	Menge der binären Zahlen
μ	$\mathbb{N}$	Anzahl der Individuen der Elternpopulation
λ	$\mathbb{N}$	Anzahl der Individuen der Nachkommenpopulation
ρ	$\{1,\ldots,\mu\}$	Anzahl der Individuen bei der Rekombination(Crossover)
γ	$\mathbb{N}$	Anzahl der Generationen
δ	$(0,\infty)$	Mutationsschrittweite (Standardabweichung)
n	$\mathbb{N}$	Anzahl der Objektvariablen, Dimension der Optimierungsanwendung
i	$\{1,\ldots,n\}$	Indexvariable im Bereich der Dimension n
j		Indexvariable
t	$\{1,\ldots,\gamma\}$	Zählvariable über die Anzahl der Generationen
q		Intervall in Generationen
$\vec{x}$	$\mathbb{R}^n$	Vektor der Objektvariablen eines Individuums
x_i	$\mathbb{R}$	Objektvariable eines Individuums
$\vec{s}$	$\mathbb{R}^n$	Vektor der Strategievariablen
$\vec{\Delta}$	$\mathbb{R}^n$	Mutationsvektor
n_g	$\mathbb{N}$	Anzahl der Randbedingungen g_j
g_j		Randbedingung, $j \in \{1,\ldots,n_g\}$
u_j	$\mathbb{R}$	Untergrenze des Wertebereichs der Objektvariable x_j
o_j	$\mathbb{R}$	Obergrenze des Wertebereichs der Objektvariable x_j
K_j	$\mathbb{R}$	Gewichtung der Randbedingung g_j bei Koevolution
z, $\vec{z}$		Zufallsvariable, Verteilung wie angegeben

e		Eulersche Konstante $e \approx 2,718281828$
$\emptyset$		leere Menge
$\mathbb{L}$		Lösungsraum
$\mathbb{M}$	$\mathbb{M} \subseteq \mathbb{L}$	Raum der gültigen Lösungen
$\mathbb{I}$		Raum der Individuen
I	$\mathbb{I}$	Individuum
P	$\mathbb{I}^{\mu}$, $\mathbb{I}^{\lambda}$	Population (Menge von Individuen),als Index bezogen auf die Population
G		Menge der Randbedingungen g_j
W_e	$[0,1]$	Erfolgswahrscheinlichkeit
φ		Fortschrittsgeschwindigkeit
$\widetilde{X_i}$		ganzzahlige, gleichverteilte Zufallszahl
$U(u,o)$		Gleichverteilung im Intervall $[u,o)$
$N(m,\sigma)$		Normalverteilung mit Mittelwert m undStandardabweichung σ
G_z		Geometrische Verteilung
α	ca. $[1,1.5]$	Modifikationsfaktor für mutative Schrittweitenregelung
ξ		realisierter Modifikationsfaktor
τ_1, τ_2	$(0,\infty)$	Gewichtungsfaktoren bei der Mutation
ω		Rotationswinkel
$R(\omega_{ij})$		Rotationsmatrix
β		Dämpfungsexponent für die globale Schrittweite
β_{scal}		Dämpfungsexponent für die separaten Schrittweiten
$\boldsymbol{C}$		Kovarianzmatrix
$\boldsymbol{B}$		Matrix, die $\boldsymbol{C}$ bestimmt
e_i	$\mathbb{R}$	Eigenwert i von $\boldsymbol{C}$, $i = \{1,\ldots,n\}$
$\vec{c}_i$	$\mathbb{R}^n$	Eigenvektor i von $\boldsymbol{C}$, $i = \{1,\ldots,n\}$
c_{ij}		Element in Zeile i, Spalte j von $\boldsymbol{C}$auch Elemente der Rotationsmatrix $R(\omega_{ij})$
$\vec{s}$, $\vec{s}_\delta$		kumulierte Mutationsschritte
c, c_{cov}, c_u	$[0,1]$	Gewichtungsfaktoren
ω	$[0,2\pi)$	Rotationswinkel
n_δ	$\{1,\ldots n\}$	Anzahl separater Schrittweiten
n_ω		Anzahl Rotationswinkel
F		Gewichtungsfaktor bei Rekombination und DE
p	$[0,1]$	Wahrscheinlichkeit
p_c	$[0,1]$	Crossoverrate (GA), üblich: $0,5 \leq p_c \leq 0,95$

p_m	$[0,1]$	Mutationsrate (GA)
$\phi,\ f(x)$		Zielfunktion
$F(x)$		Fitneßfunktion
$\hat{\vec{x}}$	$\mathbb{R}^n$	lokaler Minimumpunkt
$\hat{f} = f(\hat{\vec{x}})$	$\mathbb{R}$	lokales Minimum
$\vec{x}^*$	$\mathbb{R}^n$	globaler Minimumpunkt
$f^* = f(\vec{x}^*)$	$\mathbb{R}$	globales Minimum
$E,\ N$		Elter, Nachkomme (auch Indexbezeichnung für Individuen)
χ_n		Zufallsverteilung
$\hat{\chi}_n$		Erwartungswert der χ_n-Verteilung
$O(n),\ O(n^2), \ldots$		Komplexitätsklassen
r_p	$(0,1)$	Akzeptanzrate bei der Median-Selektion
n_p	$\{1, \ldots, \infty\}$	Pufferlänge bei der Median-Selektion

Anhang B

EA-Software

Hier ist die detaillierte Übersicht über die gesichtete EA-Software im Internet, auf die sich Abschnitt 4.1 bezieht.

In der Spalte *Lizenz* bedeutet *OS - komm.*, daß die Software Open Source ist, also im Quellcode vorliegt und dieser auch verändert werden darf. Allerdings wird eine kommerzielle Nutzung ausgeschlossen, bzw. muß mit dem Autor verhandelt werden.

Name	Art	Sprache	Plattform	Wartung
Alg.	GUI	Lizenz	Besonderheiten	
				Quelle
MAFRA	OO-Frw.	Java	JVM	aktuell
EA	keine GUI	OS - komm.	UML + Pattern-Design	
				`http://www.csm.uwe.ac.uk/~n2krasno/MAFRA/MAFRA.html`
EO 0.8.5	OO-Frw.	ANSI C+	Unix, Win	aktuell
EA (GA, ES)	gtk+	OS - komm.		
				`http://geneura.ugr.es/~jmerelo/EO.html`
PGA Pack 1.0	Library	C, (Fortran)	Unix	mittel
GA	keine GUI	GPL*	MPI-parallelisiert	
				`http://www-fp.mcs.anl.gov/CCST/research/reports_pre1998/comp_bio/stalk/pgapack.html`
JDEAL 1.5.2	OO-Frw.	Java	JVM	aktuell
EA (incl. ES)	keine GUI	OS - komm.	unterstützt parallele EAs	
				`http://laseeb.ist.utl.pt/sw/jdeal/home.html`
Sugal 2.1	Library, Prog.	ANSI C	Unix, Windows, OS/2	mittel
GA	Motif	OS - komm.		
				`http://www.trajan-software.demon.co.uk/sugal.htm`
EA Visualizer 1.4	OO-Frw., Prog.	Java	JVM	aktuell
EA (incl. ES)	GUI vorhanden	Shareware	graph. Algorithmus-Baukasten	
				`http://www.cs.uu.nl/people/peterb/computer/ea/eavisualizer/EAVisualizer.html`

Evolvuton 0.2.3	OO-Frw.	C++ mit STL	Unix	mittel
EA	keine GUI	GPL**	2 versch. Frameworks	
				`http://www.vrainn.com/evolvuton/`

BUGS	Prog.	C	Unix	alt
GA	GUI vorhanden	GPL		
				`ftp://ftp.aic.nrl.navy.mil/pub/galist/src/BUGS.tar.Z`

Genesis 5.0	Prog.	C	Unix, DOS	alt
GA	keine GUI	?		
				`ftp://www.aic.nrl.navy.mil/pub/galist/src/genesis.tar.Z`

dgenesis 1.0	Prog.	C	Unix	alt
GA	keine GUI	?	parall. Version von Genesis 5.0	
				`ftp://ftp.aic.nrl.navy.mil/pub/galist/src/dgenesis-1.0.tar.Z`

evoC 2.0	Library	ANSI C	Unix, DOS	alt
EA (incl. ES)	keine GUI	~GPL***		
				`http://lautaro.fb10.tu-berlin.de/evoC.html`

EvolC 2.1	Klassenbib.	C++	Unix	aktuell
GA, ES, EP	graph. Monitoring	?	Doku. u. Code in Französisch	
				`http://www.eark.polytechnique.fr/EvolC.html`

GENEsYs 1.0	Prog.	C	Unix, DOS	alt
GA	keine GUI	?	Benchmarkfunktionen f_1 bis f_{24}	
				`ftp://www.aic.nrl.navy.mil/pub/galist/src/GENEsYs-1.0.tar.Z`

GENOCOP 1.0	Prog.	C	Unix, Win?	alt
GA	keine GUI	OS - komm.	lineare Constraints	
				`ftp://www.aic.nrl.navy.mil/pub/galist/src/genocop.tar.Z`

REGAL 3.2	Prog.	C	Unix	alt
GA	GUI vorhanden	OS - komm.	parall. mit PVM	
				`ftp://ftp.di.unito.it/pub/MLprog/REGAL3.2`

GA Playground	Prog.	Java	JVM	mittel
GA	GUI vorhanden	?	Fitneßfkt. einfach einbindbar	
				`http://www.aridolan.com/ga/gaa/gaa.html`

GALOPPS 3.2.2	Library	C	Unix, Win?	mittel
GA	GUI vorhanden	GPL	PVM-Version existiert	
				`ftp://garage.cse.msu.edu/pub/GA/galopps/`

GAlib 2.4.5	OO-Framew.	C++	Unix, Win	aktuell
GA	nur f. Beispiele	free/teilw. GPL		
				`http://lancet.mit.edu/ga/`

* GPL-ähnliche Lizenz, d. h. Nutzung, kopieren, modifizieren und Weitergabe erlaubt, allerdings soll bei der Verwendung ein Hinweis auf die Software angegeben werden.

** GPL-Lizenz mit Ausnahme von militärischer Verwendung.

*** GPL-ähnliche Lizenz unter Ausschluß kommerzieller Nutzung. Veränderte Versionen dürfen nicht unter dem gleichen Namen oder nur als Patch vertrieben werden.

Kleinprogramme			
Name	Alg.	Sprache	Quelle
GAJIT	GA	Java	`http://www.angelfire.com/ca/Amnesiac/gajit.html`
GAC	GA	C	`ftp://www.aic.nrl.navy.mil/pub/galist/src/GAC.shar.Z`
GAS	GA	C++	`ftp://ftp.jate.u-szeged.hu/pub/math/optimization/GAS/`
GAucsd	GA	C	`ftp://www.aic.nrl.navy.mil/pub/galist/src/GAucsd14.sh.Z`
genetic2(n)	GA	C	`ftp://www.aic.nrl.navy.mil/pub/galist/src/genetic2.tar.Z`
libga100	GA	C	`http://www.cs.clemson.edu/GA/src/libga100/`
paragenesis	GA	C	`www.aic.nrl.navy.mil/pub/galist/src/paragenesis.tar.Z`
pga 2.5	GA	C	`http://www.aic.nrl.navy.mil/galist/src/pga-2.5.tar.Z`
sga-c(ube)	GA	C	`http://www.aic.nrl.navy.mil/galist/src/sga-c.tar.Z`
GAEiffel	GA	Eiffel	`http://www.cs.und.ac.za/~ikram/Projects/GAEiffel/`
?	GA	Fortran	`http://www.staff.uiuc.edu/~carroll/ga.html`
GAL, GECO, Koza-GPl	GA GP	LISP	`http://www.aic.nrl.navy.mil/galist/src/index.html`
GAOT	GA	Matlab	`http://www.eos.ncsu.edu/eos/service/ie/research/-kay_res/GAToolBox/gaot/`
EvolStrat(?)	ES	Scilab	`http://cubi210.fi-b.unam.mx/EvolStrat/`

Anhang C

Testfunktionen

Hier sind die Formeln der in dieser Arbeit verwendeten Testfunktionen f_x. Sie sind mit dieser Numerierung aus [Bäck 92b] übernommen worden.

C.1 Funktion f_2: Generalized Rosenbrock's Function

$$f_2(\vec{x}) = \sum_{i=1}^{n-1}(100 \cdot (x_{i+1} - x_i^2)^2 + (x_i - 1)^2) \tag{C.1}$$

$$-5,12 \leq x_i \leq +5,12$$

$$\min(f_2) = f_2(1, \ldots, 1) = 0$$

f_2 ist eine unimodale Funktion. Die Besonderheit besteht darin, daß durch den Teilterm $(x_{i+1} - x_i^2)^2$ jeweils die Variablen x_i und x_{i+1} eng miteinander korreliert sind. Deshalb müssen ab einem gewissen Abstand zum Optimum diese Variablenpaare koordiniert geändert werden, damit sich der Funktionswert verkleinern kann.

C.2 Funktion f_6: Schwefel's Function 1.2 (Doublesum)

$$f_6(\vec{x}) = \sum_{i=1}^{n}\left(\sum_{j=1}^{i} x_j\right)^2 = x^T \boldsymbol{A} x + \boldsymbol{b}^T x \tag{C.2}$$

$$-65.536 \leq x_i \leq +65.536$$

$$\min(f_6) = f_6(0, \ldots, 0) = 0$$

Dies ist ein quadratisches Minimierungsproblem. Die rechte Form ist die Funktion in Matrixschreibweise. Die Konditionszahl K der Matrix A ist ein Maß für die Schwierigkeit des Problems. Bei dieser Funktion wächst K in Abhängigkeit von der Dimension n des Problems mit $O(n^2)$ [Schwefel 95].

C.3 Funktion f_9: Ackley's Function

$$f_9(\vec{x}) = -a \cdot e^{\left(-b\sqrt{\frac{1}{n}\cdot\sum_{i=1}^{n} x_i^2}\right)} - e^{\left(\frac{1}{n}\cdot\sum_{i=1}^{n}\cos(c\cdot x_i)\right)} + a + e \tag{C.3}$$

$$a = 20 \quad ; \quad b = 0,2 \quad ; \quad c = 2\pi$$

$$-32.768 \leq x_i \leq +32.768$$

$$\min(f_9) = f_9(0, \ldots, 0) = 0$$

Diese Funktion ist multimodal. Sie besteht aus einer Exponentialfunktion, die mit Cosinus-Wellen moduliert wird. Ursprünglich war sie von Ackley nur für den zweidimensionalen Fall formuliert, wird hier aber in einer erweiterten, skalierbaren Version gezeigt [Bäck 96].

C.4 Funktion f_{10}: Krolak's 100 city TSP

$$f_{10}(\vec{x}) = \sum_{i=1}^{n} d\left(c_{\pi(i \bmod n)}, c_{\pi((i+1) \bmod n)}\right) \tag{C.4}$$

$$n = 100$$

$$\min(f_{10}) = 21285$$

Dies ist die Instanz *kroA100* des *Traveling Salesman Problem* (Problem des Handlungsreisenden) aus der TSPLIB95 [Reinelt 95]. Gesucht ist die kürzeste Rundreiseroute, die jede Stadt genau einmal besucht. Die Städte sind durch ihre Koordinaten in der Ebene gegeben, die Funktion d berechnet den Abstand zweier Städte.

C.5 Funktion f_{15}: Weighted Sphere Model

$$f_{15}(\vec{x}) = \sum_{i=1}^{n} i \cdot x_i^2 \tag{C.5}$$

$$-5.12 \leq x_i \leq +5.12$$

$$\min(f_{15}) = f_{15}(0, \ldots, 0) = 0$$

Dies ist eine Erweiterung der *Sphere Model*-Funktion $f_1(\vec{x}) = \sum_{i=1}^{n} x_i^2$ um einen Gewichtungsfaktor.

C.6 Funktion f_{24}: Kowalik

$$f_{24}(\vec{x}) = \sum_{i=1}^{11} \left(a_i - \frac{x_1(b_i^2 + b_i x_2)}{b_i^2 + b_i x_3 + x_4} \right)^2 \tag{C.6}$$

$$-5.0 \leq x_i \leq +5.0$$

$$n = 4$$

i	a_i	b_i^{-1}
1	0.1957	0.25
2	0.1947	0.5
3	0.1735	1
4	0.1600	2
5	0.0844	4
6	0.0627	6
7	0.0456	8
8	0.0342	10
9	0.0323	12
10	0.0235	14
11	0.0246	16

$$\min(f_{24}) \approx f_{24}(0.1928, 0.1908, 0.1231, 0.1358) \approx 0.0003075$$

Non-linear fitting problem [Schwefel 95]. Es sind 11 Datenpunkte gegeben, die durch eine Funktion mit 4 freien Variablen angenähert werden soll, so daß der Fehler (Euklidische Norm) minimiert wird.

C.7 Funktion q_1: Achsenparalleler Hyperellipsoid

$$\begin{aligned} q_1(\vec{x}) &= \sum_{i=1}^{n} (1000^{\frac{i-1}{n-1}} \cdot \langle \vec{x}, \vec{e}_i \rangle)^2 \\ &= \sum_{i=1}^{n} (1000^{\frac{i-1}{n-1}} \cdot x_i)^2 \end{aligned} \tag{C.7}$$

$$\min(q_1) = q_1(0, \ldots, 0) = 0$$

Die Flächen mit gleicher Fitneß bilden bei der Funktion q_1 [Hansen et al. 95b, Hansen et al. 96] einen Hyperellipsoiden, wobei benachbarte Achsen (bezgl. Index i) ein Verhältnis von 1.44 zueinander haben. $\left\langle \vec{a}, \vec{b} \right\rangle$ ist hierbei das Skalarprodukt und die Vektoren $\vec{e}_i$ sind die Einheitsvektoren, so daß $\langle \vec{x}, \vec{e}_i \rangle = x_i$ ergibt. Die Vektorschreibweise wurde hier verwendet, um deutlich zu machen, daß die Achsen des Hyperellipsoids parallel zu den Koordinatenachsen liegen. Funktion q_2 (nächster Abschnitt) ist eine Erweiterung von q_1, die Achsen sind hier nicht mehr nach dem Koordinatensystem ausgerichtet. q_1 selbst kann als Erweiterung der Funktion $f_1(\vec{x}) = \sum_{i=1}^{n} x_i^2$ vom Kreis/Kugel zur Ellipse/Ellipsoid angesehen werden. Alle drei Funktionen werden gerne bei Untersuchungen zur Selbstadaption von ES verwendet.

C.8 Funktion q_2: Zufällig gedrehter Hyperellipsoid

$$q_2(\vec{x}) = \sum_{i=1}^{n} (1000^{\frac{i-1}{n-1}} \cdot \langle \vec{x}, \vec{o}_i \rangle)^2 \tag{C.8}$$

$$\min(q_1) = q_1(0, \ldots, 0) = 0$$

Funktion q_2 unterscheidet sich von q_1 dadurch, daß die Achsen der Hyperellipsoiden nicht parallel zu den Achsen des Koordinatensystems ausgerichtet sind, sondern zufällig davon verdreht sind. Die Vektoren $\vec{o}_1, \ldots, \vec{o}_n$ bilden eine Orthonormalbasis mit zufälliger Orientierung im Raum.

Lebens- und Bildungsgang

20. Nov. 1995 bis 31. März 2001	**Wissenschaftlicher Mitarbeiter an der Universität Tübingen** Wilhelm-Schickard-Institut für Informatik, Lehrstuhl Rechnerarchitektur, Prof. Dr. Andreas Zell
Okt. 1990 bis Nov. 1995	**Studium der Informatik** an der Universität Stuttgart mit Nebenfach Elektrotechnik Studienarbeit: *Parallaxis-III-Compiler für Multiprozessorsysteme unter PVM.* Diplomarbeit: *Verteilte Evolutionsstrategien auf dem Supercomputer Intel Paragon.*
1. Juni 1989 bis 31. August 1990	**Ableistung des Wehrdienstes bei der Bundeswehr** Würzburg und Tauberbischofsheim
Aug. 1980 bis Juni 1989	**Justinus-Kerner Gymnasium Weinsberg** Erwerb der allgemeinen Hochschulreife
Aug. 1976 bis Juli 1980	**Grundschule Lehrensteinsfeld/Ellhofen**