Globale Optimierung mit parallelen Evolutionsstrategien

Diplomarbeit

an der

Universität Dortmund

Fachbereich Informatik

vorgelegt von

Günter Rudolph

im Juli 1990

Betreuer: Univ.-Prof. Dr.-Ing. H.-P. Schwefel

Inhaltsverzeichnis

1 Einleitung **7**

2 Multi-Prozessorsysteme **9**
2.1 Vorbemerkungen 9
2.2 Parallele Rechnerarchitekturen 9
2.2.1 Klassifikation 9
2.2.2 Transputersysteme 12
2.2.3 Das Programmiermodell des Transputers: CSP 14
2.3 Das verteilte Betriebssystem Helios 16
2.3.1 Allgemeines 16
2.3.2 Konzeption des Betriebssystems 16
2.3.2.1 Das Client/Server-Modell 16
2.3.2.2 Die Bereitstellung der Infrastruktur 17
2.3.3 Das Programmiermodell unter Helios 18

3 Das globale Optimierungsproblem **23**
3.1 Vorbemerkungen 23
3.2 Definition 23
3.3 Lösbarkeit 25
3.4 Klassifikation der Lösungsmethoden 31
3.4.1 Volumenorientierte Verfahren 32
3.4.1.1 Rastersuche 32
3.4.1.2 Verallgemeinerte Branch and Bound - Verfahren 33
3.4.1.3 Überdeckungsverfahren 33
3.4.1.4 Einfache Zufallssuche 34
3.4.1.5 Adaptiv stochastische Automaten 35
3.4.1.6 Cluster-Verfahren 36
3.4.1.7 Bayes'sche Methoden 38
3.4.2 Pfadorientierte Verfahren 40
3.4.2.1 Prognostizierende Verfahren 40
3.4.2.2 Explorative Verfahren 43

4 Parallele Evolutionsstrategien **53**
4.1 Vorbemerkungen 53
4.2 Parallelisierungsmöglichkeiten 53
4.3 Ein Evolutionsmodell zur globalen Optimierung 58

4.3.1 Zentrale Ideen 58
4.3.2 Einführung evolutionsbiologischer Termini 59
4.3.3 Formulierung des Modells 60
4.4 Implementierungsdetails 61
4.4.1 Benutzte Fremdsoftware 61
4.4.2 Synchronisation und Kommunikation 62
4.4.3 Parameter der Evolutionsstrategie 64

5 Test und Analyse **67**
5.1 Vorbemerkungen 67
5.2 Auswahl der Testfunktionen 67
5.3 Testplan 74
5.3.1 Konvergenzgeschwindigkeit 74
5.3.2 Konvergenzsicherheit 74
5.3.3 Varianten der parallelen Evolutionsstrategie 75
5.4 Testergebnisse 75
5.4.1 Zur Konvergenzgeschwindigkeit 75
5.4.2 Zur Konvergenzsicherheit 78

6 Zusammenfassung und Ausblick **89**

A Literaturverzeichnis **91**

Abbildungsverzeichnis

2.1 Architektur des Transputers T800 13
2.2 Der physikalische Zugang zum Transputersystem 14
2.3 Die Topologie des Transputersystems 15
2.4 Beipiel für eine Systemkonfiguration unter Helios 17
2.5 Parallel-Konstruktoren unter Helios 19
2.6 Logische Vernetzung in Beispiel 2.1 20
2.7 Physikalische Vernetzung in Beispiel 2.1 20
2.8 Abbildung von der logischen auf die physikalische Vernetzung . 21

3.1 Lokale und globale Minima 25
3.2 Konvexe und nichtkonvexe Mengen 27
3.3 Beziehungen zwischen verschiedenen Arten von Konvexität . . . 28
3.4 Funktionen unterschiedlicher Konvexitätsklassen 29
3.5 Approximation einer Umgebung 30
3.6 Klassifikation der Lösungsmethoden 31
3.7 Idee des Tunnelungs-Verfahrens 41

4.1 Parallele Evolutionsstrategie mit gemeinsamem Speicher 54
4.2 Parallele Evolutionsstrategie nach dem Master/Worker-Modell . 55
4.3 Zeitdiagramm für das synchrone Master/Worker-Konzept . . . 55
4.4 Zeitdiagramm für $t_{ZF} >> t_G$ 57
4.5 Asynchrone, dezentrale Selektion auf einer Mesh-Topologie . . . 58
4.6 Selbstähnliches Netzwerk 62
4.7 Prozeßschaltbild 63

5.1 Funktionsgraph von Testproblem 1 71
5.2 Funktionsgraph von Testproblem 2, 3 und 4 72
5.3 Funktionsgraph von Testproblem 5 73
5.4 Konvergenzverlauf aller Varianten für Problem 1 76
5.5 Schrittweiten von Variante 2 bei Testproblem 1 77
5.6 Konvergenzverlauf aller Varianten für Problem 2 78
5.7 Konvergenzverlauf aller Varianten für Problem 3 79
5.8 Konvergenzverlauf aller Varianten für Problem 4 80
5.9 Schrittweiten der Variante 6 bei Problem 2 81
5.10 Schrittweiten der Variante 8 bei Problem 2 82
5.11 Konvergenzverlauf aller Varianten für Problem 5 84
5.12 Schrittweiten der Variante 2 bei Problem 5 85

5.13 Schrittweiten der Variante 6 bei Problem 5 85
5.14 Schrittweiten der Variante 3 bei Problem 5 86
5.15 Schrittweiten der Variante 4 bei Problem 5 87

Kapitel 1

Einleitung

Nahezu ständig sieht sich der Mensch verschiedensten Entscheidungsproblemen ausgesetzt, von denen viele von derartig geringer Komplexität sind, daß sie als solche gar nicht mehr wahrgenommen werden. Kann jedoch ein Entscheidungsproblem nicht mehr *ad hoc* gelöst werden, so bedient man sich meist eines formalen Kalküls, um die Struktur des Problems analysieren zu können. Vielfach werden in diesem Fall mathematische Modelle aufgestellt, damit die bisher entwickelte mathematische Theorie zur Lösung des Problems genutzt werden kann.

Insbesondere nach dem 2. Weltkrieg wurde dieser Themenkomplex Gegenstand der wissenschaftlichen Forschung: So entstanden aus unterschiedlichen Wissenschaftszweigen heraus Disziplinen wie etwa die *Unternehmensforschung (Operations Research)* und die *Systemanalyse*. Die Systemanalyse läßt sich grob in die Bereiche

- Modellierung,
- Simulation und
- Optimierung

unterteilen. Bei der Modellierung versucht man, den Zusammenhang zwischen Ein- und Ausgabegrößen eines realen Systems durch ein formales (mathematisches oder algorithmisches) Modell zu beschreiben. Ist nun ein adäquates Modell gefunden, so kann man simulieren, *was geschieht, wenn* bestimmte Größen, sogenannte Parameter, verändert werden. Schließlich läßt sich dann auch betrachten, was zu tun ist, um ein bestimmtes Ziel zu erreichen - also das inverse ‚*was ist - wenn*‘ Problem. Letzteres ist Gegenstand der Optimierung.

Es gibt mehrere Möglichkeiten, zwischen Optimierungsmodellen zu unterscheiden:

- statisch - dynamisch,
- diskret - kontinuierlich,
- deterministisch - stochastisch,

- linear - nichtlinear,
- konvex - nichtkonvex,
- restringiert - nichtrestringiert.

Die in dieser Arbeit betrachteten Optimierungsprobleme sind von deterministischer, statischer sowie kontinuierlicher Natur. Das Interesse an dieser Klasse von Optimierungsproblemen wird zum einen durch ihre weite Verbreitung in den verschiedensten Disziplinen motivert und zum anderen dadurch, daß dieses Problem im allgemeinen als ungelöst betrachtet werden muß.

Die Schwierigkeit bei diesen Problemen liegt darin begründet, daß für die meisten (wenn nicht für alle) Verfahren, die bisher entwickelt worden sind, die Konvergenz zu einem ‚globalen Optimum' nicht zugesichert werden kann. Aus dieser Situation heraus wurden dann Verfahren entwickelt, die nach probabilistischen oder heuristischen Gesichtspunkten arbeiten. Zu solchen Verfahren sind auch die ‚Evolutionsstrategien' zu zählen: Sie verstehen den Prozeß der biologischen Evolution als einen Optimierungsprozeß und versuchen, diesen nachzuahmen. Bei numerischen Tests wie auch im experimentellen Einsatz konnte ihre Leistungsfähigkeit empirisch nachgewiesen werden.

Neben der Leistungsfähigkeit macht ein weiterer Aspekt die ‚Evolutionsstrategien' interessant: Durch ihre inhärente Parallelität sind sie für die nunmehr aufkommenden Parallelrechner als Studienobjekt „wie geschaffen". So ist es das Ziel dieser Arbeit, eine parallele Version der ‚Evolutionsstrategie' zur ‚globalen Optimierung' zu entwickeln, die sowohl einen Geschwindigkeitsvorteil als auch eine Qualitätsverbesserung hinsichtlich des Resultates gegenüber der sequentiellen Version bietet.

Da parallele Algorithmen je nach unterliegender Rechnerarchitektur sehr verschieden ausfallen können, soll im Kapitel 2 zunächst eine Klassifikation und Beschreibung der verschieden Architekturen vorgenommen werden. Dabei wird insbesondere auf das am Lehrstuhl XI installierte System eingegangen. In Kapitel 3 wird das ‚globale Optimierungsproblem' definiert und dessen Lösbarkeit diskutiert. Ein Überblick über vorhandene Ansätze schließt dieses Kapitel ab. Das Kapitel 4 beschäftigt sich mit dem Entwurf einer ‚parallelen Evolutionsstrategie', die insbesondere zur Lösung des ‚globalen Optimierungsproblems' eingesetzt werden soll. Der Testbericht und verschiedene Analysen der Testergebnisse finden sich in Kapitel 5. Das Kapitel 6 schließlich faßt die Resultate noch einmal zusammen und schließt mit einem Ausblick auf weitere mögliche Entwicklungen.

Kapitel 2

Multi-Prozessorsysteme

2.1 Vorbemerkungen

Die Datenverarbeitung der letzten Jahrzehnte wurde von der seriellen Rechnerarchitektur des von Neumannschen Modells und den dieser Architektur angepaßten höheren Programmiersprachen bestimmt, obwohl „viele Probleme ihrem Ursprung nach paralleler Natur sind“[1]. Daß aber erst in jüngerer Zeit das Interesse an der parallelen Datenverarbeitung gestiegen ist, liegt zum einen an der nunmehr erfolgten Bereitstellung erschwinglicher Rechner mit innovativen Rechnerarchitekturen und zum anderen an der Schwierigkeit, den Parallelismus zu entdecken, da unser algorithmisches Verständnis durch über 100 Jahre sequentieller Mathematik, etwa 60 Jahre sequentiellem Algorithmenentwurf und über 30 Jahre sequentieller Programmierung geprägt ist[2].

Da die für serielle Rechner angelegten Programmstrukturen nicht ohne weiteres auf parallele Rechnerarchitekturen übertragbar sind, werden nun neue Betriebssysteme, neue Programmiersprachen und Compiler sowie neue Anwendungsprogramme und Algorithmen erforderlich. Diese werden, je nach unterliegender Rechnerarchitektur, recht unterschiedlich ausfallen.

Deshalb werden in Abschnitt 2.2 zunächst verschiedene Rechnerarchitekturen vorgestellt, bevor auf ein spezielles System eingegangen wird. Ein für dieses System geeignetes Betriebssystem und das daraus resultierende Programmiermodell wird im Mittelpunkt von Abschnitt 2.3 stehen.

2.2 Parallele Rechnerarchitekturen

2.2.1 Klassifikation

Die seriellen Rechner des von Neumannschen Modells sind dadurch gekennzeichnet, daß an einer einzigen Stelle, nämlich der Steuereinheit, die Kontrolle vor-

[1] Hoßfeld (1981), S. 1.
[2] Vgl. Hoßfeld (1981), S. 1.

handen ist, welche Instruktion als nächstes ausgeführt wird. Die Daten werden sukzessiv aus einem globalen Speicher geholt. Da immer nur eine Instruktion zu einem Zeitpunkt ausgeführt wird, kann die Rechenleistung durch eine hohe Speicherzugriffszeit oder durch langsame Ein- und Ausgabegeräte beeinträchtigt werden. Deshalb wurden verschiedene Methoden entwickelt, diese Engpässe[3] zu umgehen: z.B. Cache-Speicher oder Pipelining[4].

Die ersten Supercomputer wurden auf der Basis solcher fortgeschrittenen Rechnerarchitekturen entwickelt. Auf diese Weise konnten etwa Vektoroperationen in Zeiten durchgeführt werden, die mit skalaren Operationen vergleichbar sind. Nichtsdestoweniger existiert bezüglich der Geschwindigkeit auch hier eine physikalisch bedingte oberere Schranke für solche (teuren) Supercomputer. Abweichend von der soeben skizzierten Architektur wurden mehrere Wege beschritten:

Die ersten Computer mit anderer Architektur besaßen ein ein- oder zweidimensionales Feld von Prozessoren, die jeweils mit ihrem nächsten Nachbarn verbunden waren. Der sogenannte Host-Computer kontrollierte dann die Programmausführung, indem er den Feldprozessoren die nächsten auszuführenden Instruktionen übermittelt. Solche Feldrechner sind gut geeignet für spezielle Probleme (z.B. die Lösung partieller Differentialgleichungen), nicht aber für allgemeine Anwendungen.

Deshalb wurden Rechnerarchitekturen entwickelt, die eine grobkörnigere Art von Parallelität (engl. coarse grained parallelism) unterstützen: Jeder Prozessor erhält, zusammen mit höherer Rechenleistung, die Kontrolle über seine Berechnungen. Solche Systeme, die auch allgemeine Berechnungen unterstützen, werden auch Multiprozessorsysteme genannt.

Eine andere Entwicklungslinie stützt sich auf die VLSI-Technologie ab: Hier werden oft alle Rechenelemente auf einen Chip plaziert. Systolische Arrays sind ein Beispiel für solche Spezialrechner, bei denen Teile des Algorithmus in die Hardware verlegt werden und die etwa bei der Durchführung von sogenannten schnellen Fourier-Transformationen zum Einsatz kommen.

Zwischen diese Extremfällen existieren zahlreiche Varianten, die aber hier nicht aufgeführt werden sollen. Zur Klassifikation dieser Varianten können mehrere Indikatoren herangezogen werden:

- Typ und Anzahl der Prozessoren
- Ebene des globalen Kontrollmechanismus
- Synchrone oder asynchrone Ausführung
- Prozessor-Verbindungen

[3] Als Metapher wird hier auch der Begriff Flaschenhals (engl. bottleneck) verwendet.
[4] Vgl. im folgenden Bertsekas und Tsitsiklis (1989), S. 4ff.

Bezüglich des Typs und der Anzahl der Prozessoren unterscheidet man zwischen *massiv parallelen* Systemen mit mehreren tausend Prozessoren und Systemen mit wenigen - dafür aber sehr leistungsstarken - Prozessoren in der Größenordnung von 10. Jedes dieser Systeme steht in einer bestimmten Weise unter einer globalen Kontrolle: So wird im einen Extremfall der globale Kontrollmechanismus benutzt, um jeden Prozessor sukzessiv über seine nächste auszuführende Operation zu unterrichten, während im anderen Extremfall die globale Kontrolle lediglich dafür sorgt, daß ein Programm und die Daten in die Prozessoren geladen werden, woraufhin jeder Prozessor selbständig seine Arbeit verrichtet. Eine auf Flynn (1966) zurückgehende Klassifikation unterscheidet hier zwischen SIMD- und MIMD-Systemen[5]. Solche SIMD-Systeme sind *ex definitione* Systeme mit synchroner Ausführung, d.h., das gesamte System wird von einer zentralen Korntolleinheit getaktet, so daß die Operationen der verschiedenen Prozessoren synchronisiert werden. Fehlt diese globale Kontrollinstanz, so spricht man von asynchroner Ausführung. Ein weiterer wichtiger Aspekt berührt die Frage, wie die Prozessoren Informationen austauschen. Hier gibt es zwei unterschiedliche Konzepte: Im einen Fall existiert ein gemeinsamer Speicher (shared memory), auf den alle Prozessoren zugreifen können. Die Kommunikation wird dann dadurch bewerkstelligt, daß ein Prozessor in den gemeinsamen Speicher schreibt und ein anderer Prozessor diese Stelle ausliest. Im anderen Fall besitzt jeder Prozessor einen lokalen Speicher, auf den nur er zugreifen kann. Die Kommunikation erfolgt über das Versenden von Nachrichten (message passing) über physikalische Verbindungen (engl. link) zwischen den Prozessoren. Die Ausprägung der Prozessorvernetzung nennt man die Topologie eines Netzwerkes[6].

Systeme mit dem message passing - Konzept werden auch verteilte Systeme genannt. Dabei muß man jedoch die sogenannten eng gekoppelten (closely coupled) Systeme gegenüber den lose gekoppelten (loosely coupled) Systemen wie etwa lokale Rechnernetze (LAN) abgrenzen: eng gekoppelte Systeme wurden dazu entworfen, gemeinsam eine Aufgabe zu lösen und sind deshalb - im Gegensatz zu lose gekoppelten Systemen - zum einen räumlich eng zusammengefaßt und zum anderen mit zuverlässigen Kommunikationsverbindungen ausgestattet.

Um gleichzeitig Teile eines gemeinsamen Problems in kooperativer Weise bearbeiten zu können, müssen die Prozessoren bzw. die Prozesse ihre Aktivitäten aufeinander abstimmen (sich synchronisieren) und Informationen austauschen (also kommunizieren). Viele Synchronisationsmechanismen, die bisher entwickelt wurden (busy waiting, spin locks, Unterbrechungssperren, Semaphore, unteilbare Maschinenoperationen und darauf aufbauend Monitore oder bedingte kritische Abschnitte[7]), beruhen auf der Existenz eines gemeinsamen Speichers. In verteilten Systemen dagegen erfolgt die Synchronisation ausschließlich über Nachrichten. Typische Synchronisationsmuster treten auf, wenn zwei oder mehrere

[5]SIMD: Single Instruction Multiple Data; MIMD: Multiple Instruction Multiple Data.

[6]Zur Bewertung von Topologien anhand von Kriterien wie Durchmesser, Konnektivität, Flexibilität etc. sei auf Bertsekas und Tsitsiklis (1989), S. 39ff., verwiesen.

[7]Vgl. dazu etwa Nehmer (1985).

Prozesse

- gewisse Aktionen nicht gleichzeitig ausführen dürfen,
- gewisse Aktionen gleichzeitig ausführen müssen und
- bzgl. der Ausführung gewisser Aktionen eine bestimmte Reihenfolge einhalten müssen[8].

Die Kommunikation in verteilten Systemen kann synchron oder asynchron erfolgen[9]: Im ersten Fall ist eine Sendeoperation beendet, wenn der Empfänger die Nachricht akzeptiert hat. Durch die Blockierung des Senders wird der Parallelitätsgrad reduziert, und es kann leicht zu sogenannten *Deadlocks* (Verklemmungen) kommen[10]. Im zweiten Fall wird der Sender nicht blockiert: Der Sendeauftrag wird abgesetzt, und der Prozeß setzt seine Programmausführung fort. Implementierungstechnisch läßt sich dies durch einen Nachrichtenpuffer realisieren. Falls jedoch der Puffer voll ist, muß der Sender verzögert werden, und es kann auch hier zu Blockierungen kommen.

Synchrone und asynchrone Kommunikation sind wechselseitig simulierbar: Eine asynchrone Kommunikation wird bei synchroner Kommunikation durch die Zwischenschaltung eines Pufferprozesses simuliert, der (fast) ständig empfangsbereit ist. Eine synchrone Kommunikation kann durch asynchrone Kommunikation simuliert werden, indem der Sender nach dem Versenden auf eine Quittung wartet, die der Empfänger nach Eintreffen der Nachricht explizit zurückschickt.

Ein Beispiel für ein MIMD-System mit asynchroner Ausführung und dem message passing - Konzept bei synchroner Kommunikation sind die sogenannten Transputersysteme, die im folgenden Abschnitt behandelt werden sollen.

2.2.2 Transputersysteme

Im Jahre 1985 brachte die Firma INMOS die ersten Transputer (T414) auf den Markt. Transputer sind leistungsfähige Prozessoren der VLSI-Technologie und speziell für Multiprozessorsysteme ausgelegt.

Intern[11] besteht ein Transputer aus einem 32 bit-Prozessor, einem Speicher (bis 4 kB), einem Kommunikationssystem (Link-Interface) und dem sogenannten „External Memory Interface“, welches den Zugang zu zusätzlichem lokalen Speicher bereitstellt. Diese Komponenten sind über einen 32 bit-Bus untereinander verbunden. Der seit 1987 auf dem Markt verfügbare Typ T800 beherbergt außerdem noch eine 64 bit-Fließkommaeinheit (FPU), die dem T800 eine maximale Leistung von 4.3 MFLOPS beschert. Abbildung 2.1 zeigt die vereinfachte Architektur des T800.

[8] Vgl. Mattern (1989), S. 9.

[9] Vgl. dazu etwa Mattern (1989), S. 10ff.

[10] Deadlocks treten etwa dann auf, wenn zwei Prozesse wechselseitig senden oder wechselseitig empfangen wollen.

[11] Vgl. im folgenden INMOS (1988), S. 27.

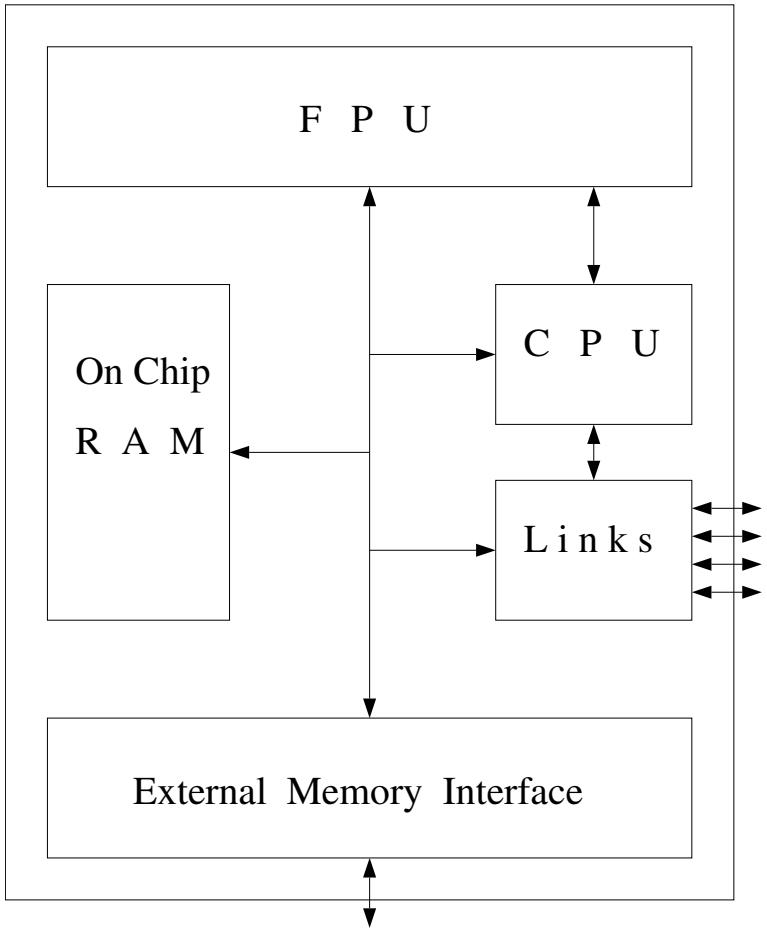

Abbildung 2.1: Architektur des Transputers T800

Über das Link-Interface kann der T800 mit bis zu vier weiteren Transputern über bi-direktionale Verbindungen, den sogenannten Links, kommunizieren. Je nach Konfiguration können über einen Link bis zu 2.35 MByte/sek. bi-direktional übertragen werden[12].

Typischerweise sind Transputer-Netzwerke mit einem herkömmlichen seriellen Rechner, dem Host-Computer, verbunden, der etwa das File-System und die Peripherie zur Verfügung stellt. Das am Lehrstuhl XI installierte System wird mit einer SUN 3/260 als Vorrechner betrieben (Abb. 2.2). Der physikalische Zugang zum Transputersystem verläuft über den sogenannten VME-Bus zu den vier Transputern T800/20 auf einem VMTM-Board der Firma Parsytec. Über jeden dieser vier Transputer kann man in das eigentliche Transputernetz mit 24 Transputern (T800 mit je 1 MB lokalen Speicher) gelangen. Abbildung 2.3 zeigt die physikalische Vernetzung der Transputer[13]. Der mittlere Abstand[14] zwischen

[12]Vgl. INMOS (1988), S. 45.
[13]Entnommen aus Hermes und Hoffmeister (1990).
[14]Vgl. Hermes und Hoffmeister (1990), S. 2.

zwei beliebigen Transputern beträgt bei dieser Topologie 2.1025.

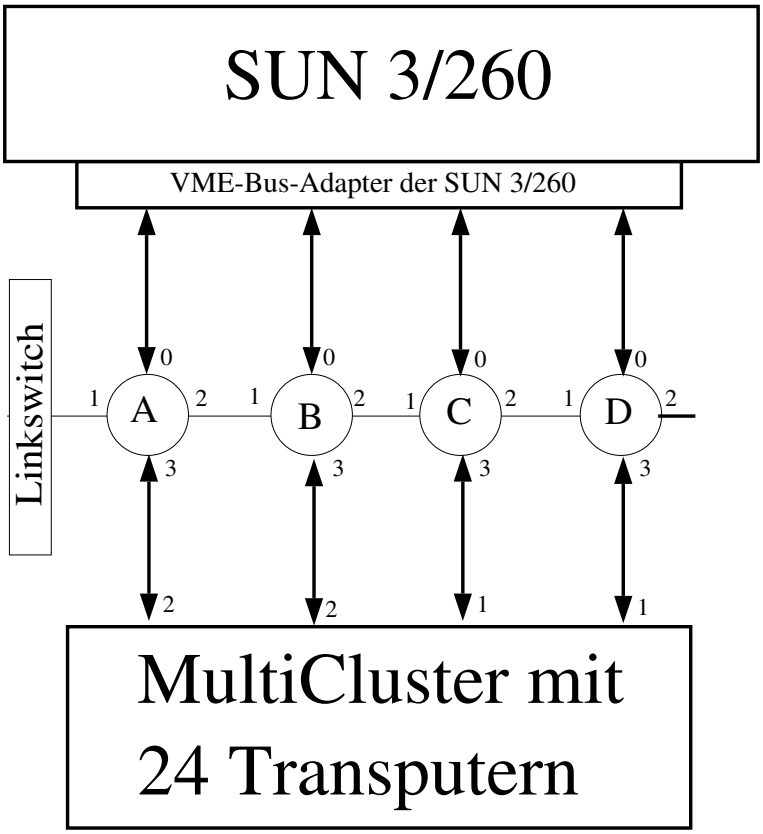

Abbildung 2.2: Der physikalische Zugang zum Transputersystem

2.2.3 Das Programmiermodell des Transputers: CSP

Das Programmiermodell für den Transputer wird durch die Sprache Occam definiert. Die formale Grundlage für Occam wurde durch die Arbeiten von Hoare (1978) über kommunizierende sequentielle Prozesse (engl. communicating sequential processes: CSP) bereitgestellt.

Die kleinste Aktionseinheit in Occam ist der Prozeß. Ausgehend von den drei primitiven Prozessen - Zuweisung, Eingabe und Ausgabe - werden Prozesse durch die Kombination mehrerer Sub-Prozesse konstruiert. Eine Zuweisung berechnet den Wert eines Ausdrucks und setzt die Variable auf diesen Wert, während Ein- und Ausgabe zur Kommunikation zwischen den Prozessen benötigt werden. Ein Paar von parallelen Prozessen kommuniziert über einen unidirektionalen Kanal, der die beiden Prozesse verbindet: Der eine Prozeß verschickt eine Nachricht (Ausgabe) und der andere Prozeß empfängt diese Nachricht (Eingabe). Zur weiteren Strukturierung stehen die üblichen, aus anderen Hochsprachen bekannten, Kontrollstrukturen zur Verfügung.

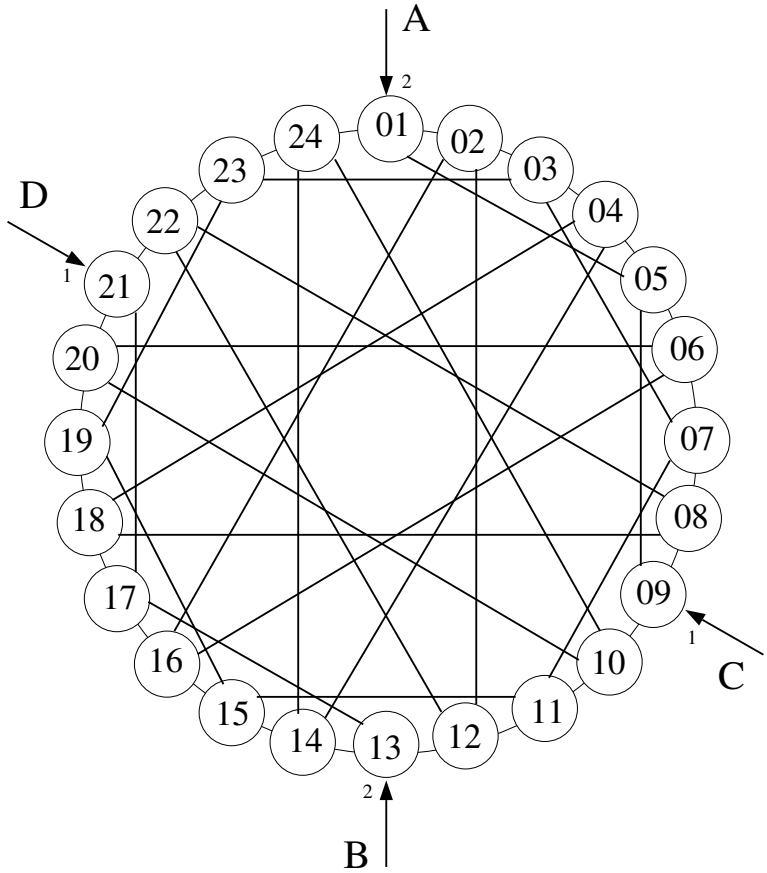

Abbildung 2.3: Die Topologie des Transputersystems

Das Schlüsselkonzept von Occam besteht darin, daß die Kommunikation synchronisiert und ungepuffert ist. Falls ein Kanal für die Eingabe in einen Prozeß benutzt wird, ein anderer Prozeß diesen Kanal jedoch zur Ausgabe benutzt, dann kann nur dann eine Kommunikation (und damit eine Synchronisation) zustande kommen, wenn beide Prozesse dazu bereit sind: Angenommen, ein Prozeß A verschickt über einen Kanal K eine Nachricht an einen Prozeß B. Falls Prozeß B noch nicht bereit ist, den Kanal K auszulesen, so wird Prozeß A suspendiert. Möchte umgekehrt Prozeß B den Kanal auslesen und Prozeß A ist noch nicht bereit zu senden, so wird Prozeß B solange suspendiert, bis Prozeß A sendet.

Sind nach diesem Mechanismus die beiden Prozesse synchronisiert worden, so wird die Nachricht des ausgebenden Prozesses in den empfangenden Prozeß kopiert, woraufhin beide Prozesse ihre Arbeit fortsetzen. Laufen dabei die kommunizierenden Prozesse auf verschiedenen Prozessoren, so wird die Nachricht über den entsprechenden Link geschickt. Sind dagegen die Prozesse auf dem gleichen Prozessor plaziert, erfolgt lediglich ein Umkopieren der Daten im Speicher. Der *Kanal* ist also als eine Abstraktion von den *Links* anzusehen.

Dem Programmierer fällt die Aufgabe zu, sowohl die Prozesse den einzelnen Prozessoren als auch die Verbindungskanäle den einzelenen Links explizit zuzuordnen. Ändert sich die Topologie des Systems oder werden weitere Prozessoren ii das System integriert, so sind Änderungen im Programmcode erforderlich.

Die Sprache Occam kann als ein maschinennaher Ansatz für parallele Applikationen angesehen werden, der es dem Benutzer ermöglicht, eine optimale Performanz auf Kosten eines hohen Programmieraufwandes zu erzielen[15].

Ein vom Programmiermodell her ähnlicher, aber auf einer höheren Abstraktionsebene angesiedelter Ansatz wurde bei der Entwicklung des Betriebssystems Helios durch die Firma Perihelion Ltd. beschritten, welches im folgenden näher beleuchtet werden soll.

2.3 Das verteilte Betriebssystem Helios

2.3.1 Allgemeines

Helios wurde als ein Betriebssystem entworfen, das auf einer Vielzahl von Systemkonfigurationen arbeiten soll. Die einzige Annahme[16], die bezüglich der Hardware gemacht wird, besteht darin, daß sich die Hardware aus einer Menge von Transputern zusammensetzt, die über ihre Links untereinander verbunden sind. Es werden keine Annahmen bezüglich der Topologie oder zusätzlicher Hardware getroffen.

Je nach Konfiguration nutzt Helios die Peripherie durch direkte Verbindungen an den Transputern oder über einen geeigneten I/O-Prozessor, der in der Regel identisch mit dem Prozessor des Host-Computers (IBM PC oder SUN Workstation) ist. Im letzteren Fall kann das System über den Host-Computer mit einem lokalen Netzwerk (LAN : Local Area Network) verbunden sein, so daß es sich als ein Prozessor-Pool zum allgemeinen Gebrauch nutzbar macht.

Das Betriebssystem selbst besteht im Grunde genommen aus einer geringen Anzahl von Systemsoftware-Komponenten, die den Prozessen die Infrastruktur bereitstellen, sich gegenseitig lokalisieren und miteinander kommunizieren zu können.

2.3.2 Konzeption des Betriebssystems

2.3.2.1 Das Client/Server-Modell

Das Betriebssystem Helios basiert auf dem sogenannten Client/Server-Modell: Jedes Programm ist entweder ein Client oder ein Server[17].

[15] Vgl. Veer (1990), S. 2.
[16] Vgl. im folgenden Perihelion (1988), S. 145f.
[17] Dabei kann ein Server wiederum ein Client eines anderen Servers sein.

Ein Server kontrolliert den Zugriff auf Systemressouren wie Disks, Bildschirme und Tastaturen, während ein Client ein Programm ist, welches den Zugriff auf Systemressourcen durch die Versendung einer Anforderung an den entsprechenden Server erhält: Ein Client, der von einer Datei lesen möchte, sendet eine Anforderung an den I/O-Server. Dieser greift auf die Datei zu und schickt die angeforderten Daten an den Client zurück.

Helios Server sind über das Netzwerk verteilt und sind üblicherweise auf dem Prozessor plaziert, der der kontrollierten Ressource am nächsten ist. So würde z.B. der X Window-Server der Systemkonfiguration, die in Abb. 2.4 dargestellt ist, auf dem Prozessor 01 laufen.

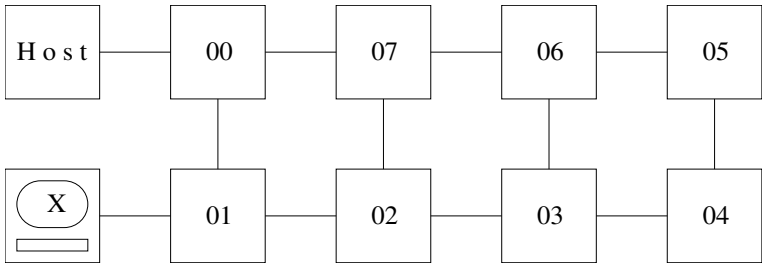

Abbildung 2.4: Beipiel für eine Systemkonfiguration unter Helios

2.3.2.2 Die Bereitstellung der Infrastruktur

Beim Systemstart[18] wird auf dem Host-Computer der sogenannte I/O-Server gestartet. Dieser lädt den Helios Nukleus in den sogenannten Root-Transputer und sorgt ansonsten für die Verbindung zum Transputersystem. Der Nukleus ist das Minimalsystem, das auf jedem Transputer unter der Kontrolle von Helios laufen muß. Seine Aufgabe besteht in der Kontrolle der Ressourcen des einzelnen Prozessors und dessen Integration ins Gesamtsystem. Der Nukleus des Root-Transputers lädt und startet den sogenannten Netzwerk-Server. Dieser liest aus einer Datei (Resource Map File) die Anzahl und die physikalischen Verbindungen der vorhandenen Prozessoren, damit er den Nukleus in jeden Transputer im Netz laden kann.

Nachdem der Netzwerk-Server einen weiteren Server, den sogenannten Task Force Manager (TFM), der für die Verteilung einer Applikation über das Netzwerk zuständig ist, gestartet hat, ist der Systemstart abgeschlossen und die Infrastruktur für parallele Applikationen bereitgestellt.

[18] Vgl. im folgenden Veer (1990), S. 3f.

2.3.3 Das Programmiermodell unter Helios

Die kleinste Einheit von Parallelität, um die sich Helios kümmern muß, wird eine Task genannt[19]. Eine Task ist ein eigenständiges, ablaufendes (aktives) Programm (Bsp.: `ls` , `more` , `fgrep` , `cc` usw.). Helios ermöglicht es nun, mehrere Tasks auf einem Prozessor oder aber verteilt auf mehreren Prozessoren laufen lassen.

Die Kommunikation zwischen Tasks erfolgt über Kommunikationskanäle, den sogenannten Pipes. Diese werden durch Deskriptoren repräsentiert. Auf diese Weise können die `read` und `write` Anweisungen in Standardsprachen wie C, FORTRAN oder Pascal verwendet werden. Der Aufbau einer Pipe und das Weiterreichen der Daten durch das Netzwerk wird vom Nukleus realisiert und bleibt dem Anwendungsprogramm verborgen.

Eine Gruppe von Tasks, die in einer Applikation zusammenarbeiten, wird eine Task Force genannt. So wäre etwa

```
ls | more
```

die Beschreibung einer Task Force, bei der die Task `ls` über eine Pipe (`|`) mit der Task `more` kommuniziert. Um auch komplexere Task Forces beschreiben zu können, wird unter Helios eine speziell dafür entworfene Sprache bereitgestellt: die sogenannte Component Distribution Language (CDL). Neben dem unter dem Betriebssytem UNIX bekannten Pipe-Konstruktur `|` existieren noch drei weitere Konstruktoren, um Kommunikationstopologien auszudrücken: `<>` ,`|||` und `^^` (vgl. Abb. 2.5).

Der Pipe-Konstruktor `|` definiert eine unidirektionale Pipeline zwischen zwei Tasks. Der sogenannte Subordinate-Konstruktor `<>` definiert eine bidirektionale Pipeline, also eine Pipe von A nach B und eine Pipe von B nach A. Der sogenannte Farm-Konstruktor `|||` wird hauptsächlich in Verbindung mit einem sogenannten Replikator `[ ]` verwendet, um eine Prozessor-Farm zu konstruieren. Dazu wird automatisch eine weitere Komponente, der sogenannte Load-Balancer (lb), in die Applikation eingefügt. Der Load-Balancer sorgt für die ausgewogene Nutzung der sogenannten Worker-Komponenten (W). Der sogenannte Parallel-Konstruktor `^^` definiert keinerlei Kommunikation zwischen den Tasks: Jedoch ist es durch weitere Sprachkonstrukte der CDL möglich, zwischen zwei Tasks „per Hand“ eine Verbindung zu ziehen. Dabei wird die Kommunikationstopologie explizit auf Deskriptoren abgebildet, während bei den zuvor vorgestellten Konstruktoren eine Abbildung auf Deskriptoren implizit gemäß bestimmter Konventionen geschieht. Auf weitere Einzelheiten der CDL soll aber hier nicht weiter eingegangen werden[20].

[19] Innerhalb einer Task ist es möglich, mehrere sogenannte Threads quasi-parallel laufen zu lassen. Threads sind vergleichbar mit Occam-Prozessen und kommunizieren unter Helios über gemeinsamen Speicher und Semaphore. Um ihre Verwaltung braucht sich Helios nicht zu kümmern: Dies erledigt der Transputer selbst mit seinem mikro-codierten Prozeß-Scheduler.

[20] Vgl. dazu Veer (1990): The CDL Guide.

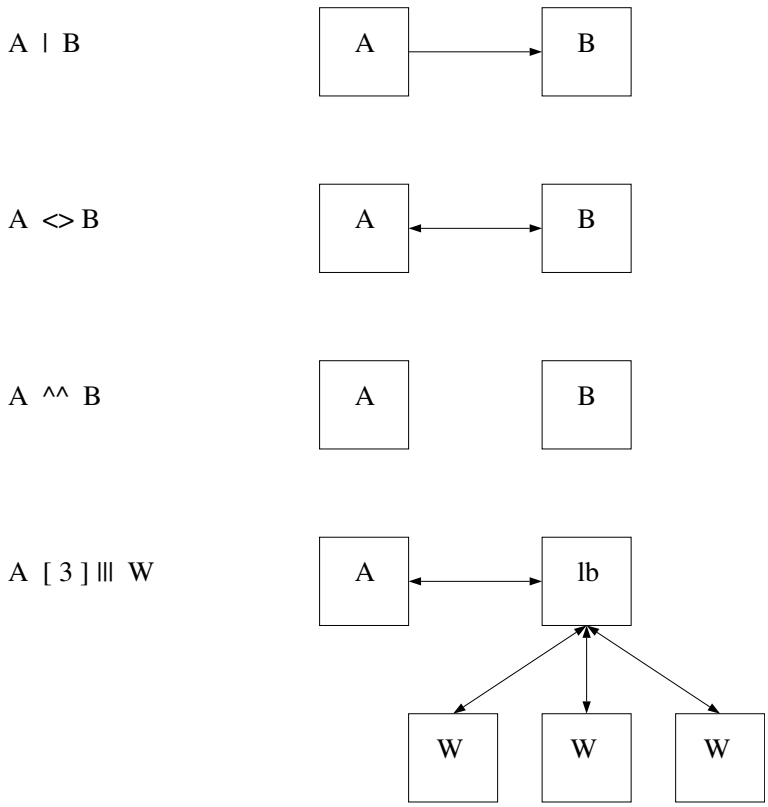

Abbildung 2.5: Parallel-Konstruktoren unter Helios

Unter Verwendung der CDL-Beschreibung einer Task Force sorgt der Task Force Manager für die Verteilung der einzelnen Tasks über das Netzwerk und für die Bereitstellung der spezifizierten Verbindungen (Pipes) zwischen ihnen. Durch die Verwendung von Pipes als Verbindungskanäle ist es möglich, von der unterliegenden physikalischen Topologie des Netzwerkes zu abstrahieren. Das folgende Beispiel diene der Illustration:

Beispiel 2.1: Eine Task Force sei durch eine CDL-Beschreibung wie folgt definiert (vgl. auch Abb. 2.6):

```
(A <| loop) | B | C | D | (E >| loop)
```

Dabei geben die dem Pipe-Konstruktor `|` vorangestellten Zeichen `<` und `>` die Richtung der Pipe und `loop` den Namen einer Pipe an: A liest und E beschreibt die Pipe `loop`. Die Task Force soll nun vom Task Force Manager auf eine physikalische Topologie wie in Abb. 2.7 abgebildet werden. Da konventionsgemäß die erstgenannte Task (hier: A) der CDL-Beschreibung die Verbindung zum Host-Computer besitzt, wird die Komponente A vom Task Force Manger (TFM) auf den Root-Transputer 00 gesetzt[21]. Die Tasks B bis E werden auf die Transputer

[21]Das gilt natürlich nur dann, wenn zu diesem Zeitpunkt keine weitere Task Force läuft.

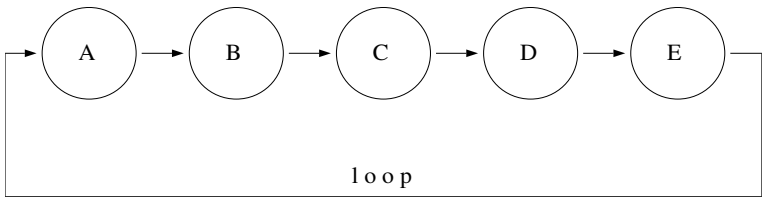

Abbildung 2.6: Logische Vernetzung in Beispiel 2.1

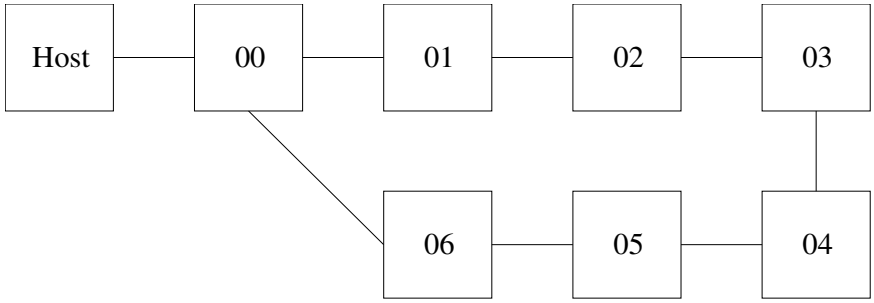

Abbildung 2.7: Physikalische Vernetzung in Beispiel 2.1

01 bis 04 verteilt. Der TFM sorgt dafür, daß eine Pipe von der Task A nach B usw. bereitgestellt wird. Jede Verbindung zwischen den Tasks wird durch den Nukleus N der entsprechenden Transputer realisiert: Beschreibt also die Task D die Pipe nach Task E, so laufen die Daten über den Nukleus von Transputer 03 zum Nukleus von Transputer 04 und von dort zur Task E. Reicht etwa die Task E diese Daten über die Pipe `loop` an die Task A weiter, so sorgt der TFM dafür, daß die Daten auf kürzestem Wege, nämlich über den Nukleus der Transputer 04, 05 und 06 zum Nukleus von Transputer 00 und von dort in die Task A gelangen.

Stünden nur zwei Transputer zur Verfügung, dann wären z.B. die Tasks A,B und C auf den einen und die Tasks D und E auf den anderen Transputer geladen worden, sofern der lokale Speicher groß genug ist. Entsprechend würden auch die Pipes physikalisch anders abgebildet. Darum muß sich der Anwendungsprogrammierer jedoch nicht kümmern: Sofern die Applikation korrekt programmiert ist, wird sie auch auf anderen physikalischen Topologien lauffähig sein.

Diese Abstraktion wird allerdings durch Performanz-Verluste erkauft: Während man etwa unter Occam die Kontrolle über den physikalischen Datenfluß besitzt, hat man unter Helios nur mit Tricks[22] einen Einfluß darauf. Nutzen z.B. mehrere

[22]Programmiert man in Occam, dann muß man die physikalische Vernetzung kennen; bei Helios ist das nicht nötig. Nur wenn man die Vernetzung kennt, dann sind auch über die CDL Tricks möglich.

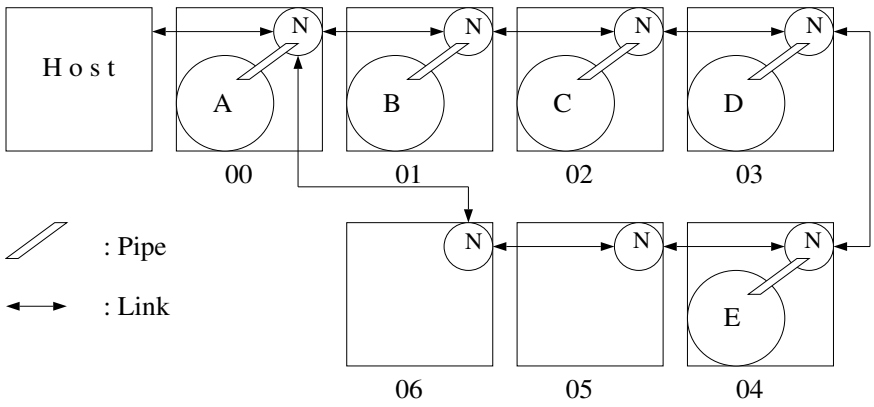

Abbildung 2.8: Abbildung von der logischen auf die physikalische Vernetzung

Tasks durch ihre logische Vernetzung dieselben physikalischen Links des Transputers zur Kommunikation, dann wird natürlich der Durchsatz vermindert.

Auf der anderen Seite stehen aber auch Vorteile: So kann vorhandene Software, die etwa in ANSI C oder FORTRAN 77 geschrieben ist, problemlos nach Helios portiert und braucht nicht in Occam re-implementiert zu werden. Weiterhin bereitet das Konzept der Kommunikation über Pipes für Programmier, die aus der UNIX/C - Welt kommen, keinerlei Verständnisschwierigkeiten.

Eine typische Vorgehensweise bei der Programmierung unter Helios besteht etwa darin, Teile eines sequentiellen Programms nach Helios zu portieren und darum eine Schale zu entwickeln, die die Kommunikation und Synchronisation leistet. Auf diese Weise dürfte die Entwicklungszeit für eine parallele Anwendung gegenüber Occam deutlich kürzer ausfallen.

Kapitel 3

Das globale Optimierungsproblem

3.1 Vorbemerkungen

Zunächst wird das ‚globale Optimierungsproblem' in Abschnitt 3.2 formal eingeführt, bevor in Abschnitt 3.3 dessen Lösbarkeit diskutiert wird. Hier werden weitere Begriffe eingeführt, um eine Klasse von Problemen charakterisieren zu können, die gegenüber dem allgemeinen ‚globalen Optimierungsproblem' als leicht zu lösen einzustufen sind. Schließlich werden in Abschnitt 3.4 im Anschluß an einer Klassifikation der Lösungsmethoden für das ‚globale Optimierungsproblem' einige typische Vertreter der verschiedenen Lösungsansätze vorgestellt.

3.2 Definition

Definition 3.1:
Der Wert $f(\underline{x}^*)$ einer Funktion $f : M \subseteq D_f \subseteq R^n \to R$ mit $M \neq \emptyset$ und $\underline{x}^* \in M$ heißt ein globales Minimum oder ein globaler Minimalwert[1] von f, falls gilt:

$$\forall \underline{x} \in M : \; f(\underline{x}^*) \leq f(\underline{x}) \; .$$

Die Aufgabe, einen Punkt $\underline{x}^* \in M$ zu bestimmen, für den die Funktion f den globalen Minimalwert annimmt, heißt das globale Optimierungsproblem[2]. Die Funktion f heißt dann eine Zielfunktion und die Teilmenge M ihres Definitionsbereiches D_f der zulässige Bereich. Jedes Element des zulässigen Bereichs heißt ein zulässiger Punkt oder eine zulässige Lösung. Wenn die Zielfunktion für einen zulässigen Punkt den globalen Minimalwert annimmt, so nennt man diesen Punkt eine globale Minimalstelle oder eine optimale Lösung oder kurz eine Lösung[3]. □

[1] Vgl. Luenberger (1973), S. 110; Dixon u.a. (1975), S. 30; Göpfert u.a. (1986), S. 167; Eiselt u.a. (1987), S. 485; Rinnooy Kan und Timmer (1989), S. 632.

[2] Vgl. Dixon u.a. (1975), S. 30; Rinnooy Kan und Timmer (1989), S. 632; Törn und Žilinskas (1989), S. 1f.

[3] Vgl. Göpfert u.a. (1986), S. 167.

Formal läßt sich das globale Optimierungsproblem wie folgt beschreiben:

$$\text{Bestimme ein } \underline{x}^* \in M \text{ mit } f(\underline{x}^*) = \min\{f(\underline{x}) \mid \underline{x} \in M \subseteq R^n\} \,. \tag{3.1}$$

Bemerkung 3.1:
Wegen der Beziehung

$$\min\{f(\underline{x}) \mid \underline{x} \in M \subseteq R^n\} \;=\; -\max\{-f(\underline{x}) \mid \underline{x} \in M \subseteq R^n\}$$

stellt die Definition 3.1 keine Beschränkung der Allgemeinheit dar[4].

□

Im folgenden soll der zulässige Bereich genauer spezifiziert werden:

Definition 3.2:
Sei $M := \{\underline{x} \in R^n \,|\, g_i(\underline{x}) \leq 0\,, i = 1(1)m\}$ der zulässige Bereich von (3.1). Dann heißen die Funktionen g_i die Restriktionen oder die Nebenbedingungen von (3.1). Eine Nebenbedingung g_i heißt in einem Punkt $\hat{\underline{x}} \in R^n$

a) erfüllt, falls $g_i(\hat{\underline{x}}) \leq 0$,

b) aktiv, falls $g_i(\hat{\underline{x}}) = 0$,

c) inaktiv, falls $g_i(\hat{\underline{x}}) < 0$ und

d) verletzt, falls $g_i(\hat{\underline{x}}) > 0$ [5].

Gilt für den zulässigen Bereich $M := R^n$, so heißt das globale Optimierungsproblem (3.1) frei oder unrestringiert, sonst restringiert[6].

□

Bemerkung 3.2:
Wegen der Äquivalenzen

$$\begin{aligned} h(\underline{x}) \leq b \in R \;&\Leftrightarrow\; g(\underline{x}) := h(\underline{x}) - b \leq 0 \\ h(\underline{x}) \geq 0 \;&\Leftrightarrow\; g(\underline{x}) := -h(\underline{x}) \leq 0 \\ h(\underline{x}) = 0 \;&\Leftrightarrow\; g_1(\underline{x}) := -h(\underline{x}) \leq 0 \text{ und } g_2(\underline{x}) := h(\underline{x}) \leq 0 \end{aligned}$$

stellt auch die gewählte Darstellungsform der Nebenbedingungen keine Beschränkung der Allgemeinheit dar. Bei manchen Optimierungsverfahren ist allerdings die Behandlung des Falles $h(\underline{x}) = 0$ schwierig.

□

[4] Vgl. Göpfert u.a. (1986), S. 167; Eiselt u.a. (1987), S. 484.
[5] Vgl. Luenberger (1973), S. 220; Gill u.a. (1989), S. 187.
[6] Vgl. Göpfert u.a. (1986), S. 167; Eiselt u.a. (1987), S. 484.

3.3 Lösbarkeit

Die meisten Verfahren, die bisher im Bereich der Optimierung entwickelt worden sind, sind im allgemeinen nicht in der Lage, das globale Optimierungsproblem (3.1) zu lösen[7], da sie in der Regel lediglich das dem Startpunkt der Suche nächstliegende ‚lokale Minimum' bestimmen können:[8]

Definition 3.3:
Der Wert $f(\underline{x}^*)$ einer Funktion $f : M \subseteq R^n \rightarrow R$ mit $M \neq \emptyset$ und $\underline{x}^* \in M$ heißt ein lokales Minimum oder ein lokaler Minimalwert[9] von f, falls eine ε-Umgebung $U_\varepsilon(\underline{x}^*)$ von $\underline{x}^*$ existiert, so daß gilt:

$$\forall \underline{x} \in M \cap U_\varepsilon(\underline{x}^*) : \; f(\underline{x}^*) \leq f(\underline{x}) \; .$$

□

Aus dieser Definition läßt sich leicht folgern, daß jedes globale Minimum auch ein lokales Minimum ist. Die folgende Abbildung möge dies verdeutlichen:

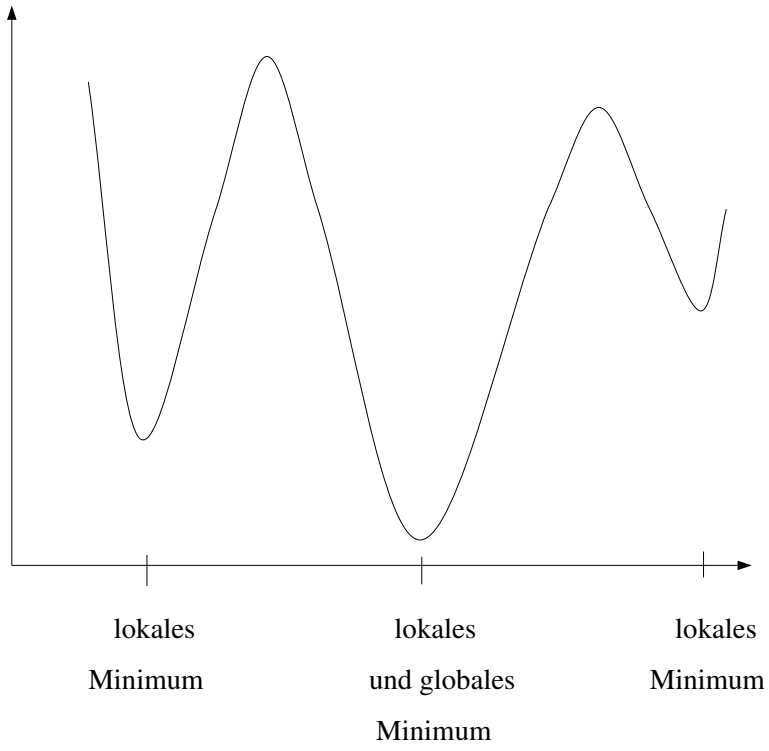

Abbildung 3.1: Lokale und globale Minima

[7]Vgl. Rinnooy Kan und Timmer (1989), S. 631.
[8]Selbst das kann bereits schwierig sein.
[9]Vgl. Luenberger (1973), S. 110; Göpfert u.a. (1986), S. 168; Eiselt u.a. (1987), S. 485.

Die Umkehrung dieser Folgerung gilt jedoch im allgemeinen nicht. Dazu müssen bestimmte Anforderungen an die Zielfunktion und den zulässigen Bereich gestellt werden. In diesem Zusammenhang sind verschiedene Formen von ‚Konvexität' von Bedeutung:

Definition 3.4:[10]
Eine Menge $M \subseteq R^n$ heißt eine konvexe Menge, falls für jedes $\underline{x}_1 \in M$ und $\underline{x}_2 \in M$ mit $\lambda \in (0;1) \subset R$ gilt:

$$\lambda\,\underline{x}_1 + (1-\lambda)\,\underline{x}_2 \ \in M\ .$$

Eine Funktion $f : M \subseteq R^n \to R$ über einer konvexen Menge M heißt eine streng konvexe Funktion, falls für jedes $\underline{x}_1 \in M$ und $\underline{x}_2 \in M$ mit $\underline{x}_1 \neq \underline{x}_2$ und $\lambda \in (0;1) \subset R$ gilt:

$$f(\lambda\,\underline{x}_1 + (1-\lambda)\,\underline{x}_2) < \lambda\,f(\underline{x}_1) + (1-\lambda)\,f(\underline{x}_2)\ .$$

Eine Funktion $f : M \subseteq R^n \to R$ über einer konvexen Menge M heißt eine konvexe Funktion, falls für jedes $\underline{x}_1 \in M$ und $\underline{x}_2 \in M$ mit $\underline{x}_1 \neq \underline{x}_2$ und $\lambda \in (0;1) \subset R$ gilt:

$$f(\lambda\,\underline{x}_1 + (1-\lambda)\,\underline{x}_2) \leq \lambda\,f(\underline{x}_1) + (1-\lambda)\,f(\underline{x}_2)\ .$$

Eine Funktion $f : M \subseteq R^n \to R$ über einer konvexen Menge M heißt eine stark quasikonvexe Funktion, falls für jedes $\underline{x}_1 \in M$ und $\underline{x}_2 \in M$ mit $\underline{x}_1 \neq \underline{x}_2$ und $\lambda \in (0;1) \subset R$ gilt:

$$f(\lambda\,\underline{x}_1 + (1-\lambda)\,\underline{x}_2) < \max\{f(\underline{x}_1),\ f(\underline{x}_2)\}\ .$$

Eine Funktion $f : M \subseteq R^n \to R$ über einer konvexen Menge M heißt eine streng quasikonvexe Funktion, falls für jedes $\underline{x}_1 \in M$ und $\underline{x}_2 \in M$ mit $f(\underline{x}_1) \neq f(\underline{x}_2)$ und $\lambda \in (0;1) \subset R$ gilt:

$$f(\lambda\,\underline{x}_1 + (1-\lambda)\,\underline{x}_2) < \max\{f(\underline{x}_1),\ f(\underline{x}_2)\}\ .$$

Eine Funktion $f : M \subseteq R^n \to R$ über einer konvexen Menge M heißt eine quasikonvexe Funktion, falls für jedes $\underline{x}_1 \in M$ und $\underline{x}_2 \in M$ mit $\underline{x}_1 \neq \underline{x}_2$ und $\lambda \in (0;1) \subset R$ gilt:

$$f(\lambda\,\underline{x}_1 + (1-\lambda)\,\underline{x}_2) \leq \max\{f(\underline{x}_1),\ f(\underline{x}_2)\}\ .$$

□

Eine Menge M heißt also konvex, wenn die gesamte Verbindungslinie zwischen zwei beliebigen Punkten aus M in M liegt (Abb. 3.2). Die (strenge) Konvexität einer Funktion sagt aus, daß der Funktionsgraph einer Funktion f für zwei beliebige Punkte aus M nicht oberhalb (echt unterhalb) der Verbindungslinie zwischen den zugehörigen Funktionswerten der beiden Punkte liegt.

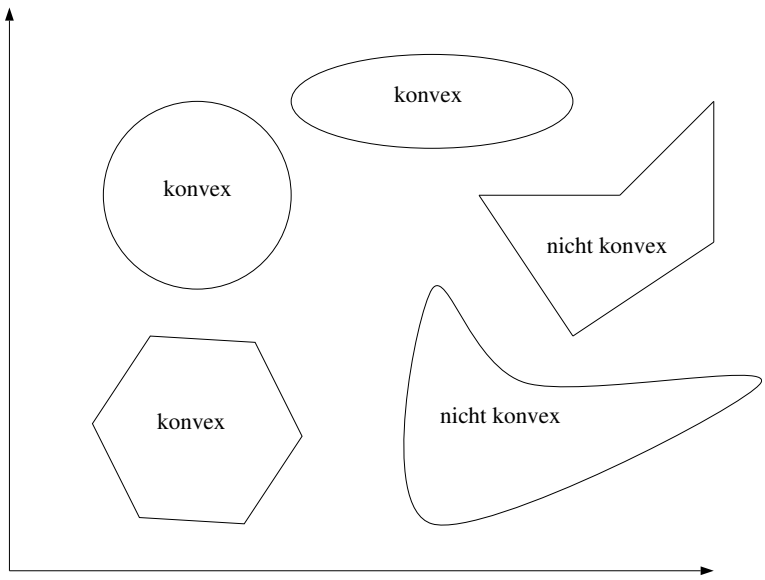

Abbildung 3.2: Konvexe und nichtkonvexe Mengen

Zwischen diesen Eigenschaften von Funktionen bestehen gewisse Implikationen, die in Form eines Diagramms[11] in der Abb. 3.3 dargestellt sind.

Aus diesem Diagramm läßt sich ablesen, daß etwa jede konvexe Funktion auch eine streng quasikonvexe Funktion[12] ist, aber nicht umgekehrt. Die Abb. 3.4 gibt Beispiele für Funktionen an, die unterschiedlichen Konvexitätsklassen angehören.

Eng verwandt mit den erweiterten Konvexitätsbegriffen ist der Begriff der ‚Unimodalität' einer Funktion:

Definition 3.5:
Eine Funktion $f : M \subseteq R^n \rightarrow R$ über einer nichtleeren Menge M soll unimodal heißen, wenn sie genau einen lokalen Minimalwert besitzt und die Menge der lokalen Minimalstellen zusammenhängend ist[13]. Eine Funktion heißt multimodal, wenn sie nicht unimodal ist.

□

Die Unterscheidung der Funktionen in Konvexitätsklassen liegt darin begründet, daß jede Klasse gewisse „angenehme" Eigenschaften besitzt: z.B. kann für kon-

[10]Vgl. Bazaraa und Shetty (1979), S. 507f. ; eine Vielzahl weiterer Arten von Konvexität findet sich bei Göpfert u.a. (1986), S. 54ff.

[11]Nach Bazaraa und Shetty (1979), S. 108.

[12]Die Implikation, daß jede streng quasikonvexe Funktion auch eine quasikonvexe Funktion ist, gilt nur, wenn zusätzlich für die Funktion die Eigenschaft der sogenannten ‚Unterhalbstetigkeit 'angenommen wird: Vgl. dazu Bazaraa und Shetty (1979), S. 503.

[13]Der Begriff der Unimodalität wird in der Literatur nicht einheitlich definiert: vgl. dazu Schwefel (1977), S. 30; Bazaraa und Shetty (1979), S. 317; Göpfert u.a. (1986), S. 55.

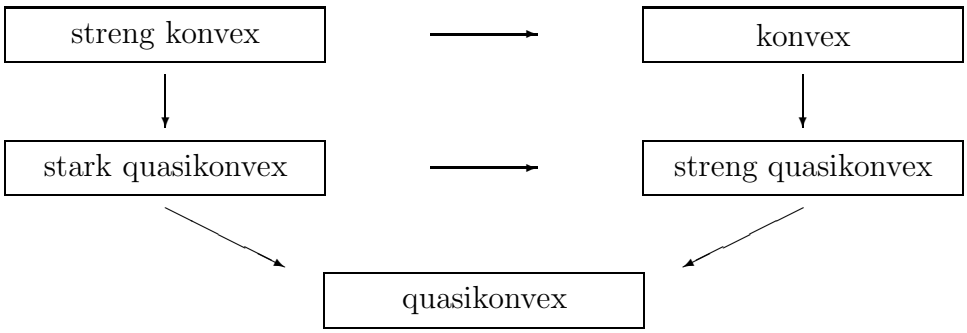

Abbildung 3.3: Beziehungen zwischen verschiedenen Arten von Konvexität

vexe Funktionen ihre Stetigkeit garantiert werden. Eine für die Optimierungstheorie bedeutendere Aussage macht jedoch das folgende

Theorem 3.1: [14]
Sei $f : M \subseteq R^n \to R$ eine streng quasikonvexe oder unimodale Funktion. Dann ist jede lokale Minimalstelle auch eine globale Minimalstelle. Ist f darüberhinaus auch stark quasikonvex, so ist die globale Minimalstelle eindeutig.

□

Sind also alle Voraussetzungen des Theorems 3.1 erfüllt, dann werden aus den hinreichenden Kriterien für ein lokales Minimum auch hinreichende Kriterien für ein globales Minimum. In solchen Fällen genügt also die Anwendung eines sogenannten lokalen Optimierungsverfahrens, um das globale Optimum bestimmen zu können.

Kann jedoch die Unimodalität der Zielfunktion über ihren zulässigen Bereich nicht zugesichert werden, was bei den meisten praktischen Problemen der Fall ist[15], dann bieten solche Verfahren keine Lösung des globalen Optimierungsproblems.

Die relative Schwierigkeit der globalen Optimierung, verglichen mit der lokalen Optimierung, liegt darin begründet, daß im Falle der stetigen Differenzierbarkeit der Zielfunktion sämtliches Wissen, ob ein Punkt ein lokales Minimum ist, in den ersten und ggf. zweiten partiellen Ableitungen in diesem Punkt steckt. Liefert das Kriterium für ein lokales Minimum[16] kein positives Ergebnis, dann sichert die stetige Differenzierbarkeit der Funktion zu, daß ein benachbarter Punkt mit einem niedrigeren Funktionswert gefunden werden kann. Auf diese

[14] Vgl. Bazaraa und Shetty (1979), S. 104; Göpfert u.a. (1986), S. 55.

[15] Vgl. dazu Archetti und Frontini (1978), S. 179ff. ; Dixon und Szegö (1978), S. 2; Törn und Žilinskas (1989), S. 202ff.

[16] Gemeint sind hier die lokalen Kuhn/Tucker-Bedingungen: vgl. etwa Göpfert u.a. (1986), S.116.

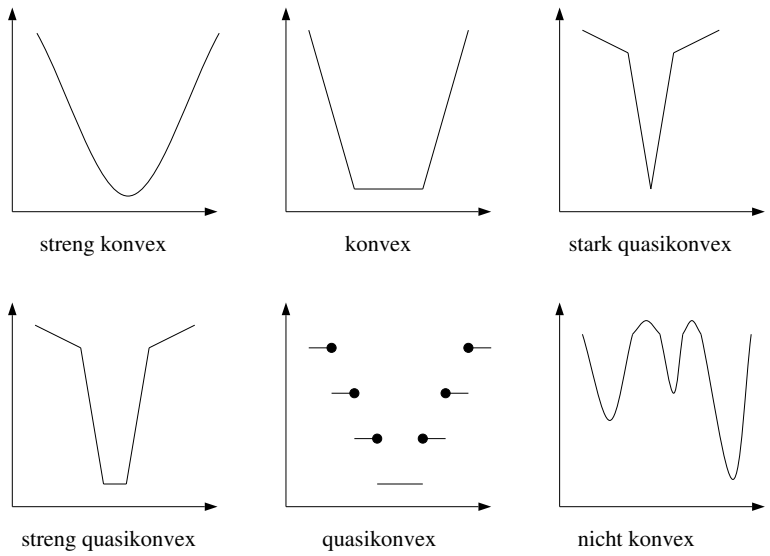

Abbildung 3.4: Funktionen unterschiedlicher Konvexitätsklassen

Weise läßt sich eine Folge von Punkten konstruieren, die zum lokalen Minimum konvergiert. Solche lokalen Kriterien sind nun aber ganz offensichtlich nicht hinreichend, um globale Optimalität zu gewährleisten[17]. Wünschenswert wäre also ein allgemeines und konstruktives Kriterium für ein globales Minimum, welches jedoch bisher noch nicht gefunden werden konnte[18].

Darüberhinaus existieren noch weitere Schwierigkeiten, das globale Minimum zu bestimmen, da numerische Verfahren wegen der begrenzten Genauigkeit von Digitalrechnern niemals mehr als approximative Antworten liefern können. Man könnte also das Problem (3.1) als gelöst betrachten, falls für ein $\varepsilon > 0$ entweder ein Element einer ε - Umgebung von $\underline{x}^*$

$$A_{\underline{x}}(\varepsilon) := U_\varepsilon(\underline{x}^*) = \{\underline{x} \in M |\ ||\underline{x} - \underline{x}^*|| \leq \varepsilon\} \tag{3.2}$$

oder ein Element der sogenannten Niveaumenge[19]

$$A_f(\varepsilon) := L_{f^*+\varepsilon} = \{\underline{x} \in M |\ f(\underline{x}) \leq f(\underline{x}^*) + \varepsilon\} \tag{3.3}$$

bestimmt werden kann, wobei $\underline{x}^*$ die globale Minimalstelle bezeichnet.

Ein Nachteil der ersten Möglichkeit (3.2) besteht darin, daß der Abstand zweier Punkte zwar beliebig klein, der Unterschied zwischen den zugehörigen Zielfunktionswerten aber sehr groß sein kann. Die Abb. 3.5 möge dies verdeutlichen.

Für die zweite Möglichkeit (3.3) existiert keine allgemeine Methode, ein Element dieser Menge in einer endlichen Abzahl von Schritten zu finden. Tatsächlich läßt

[17] Vgl. Rinnooy Kan und Timmer (1989), S. 633.
[18] Vgl. Törn und Žilinskas (1989), S. 2.
[19] Vgl. etwa Göpfert u.a. (1986), S. 110.

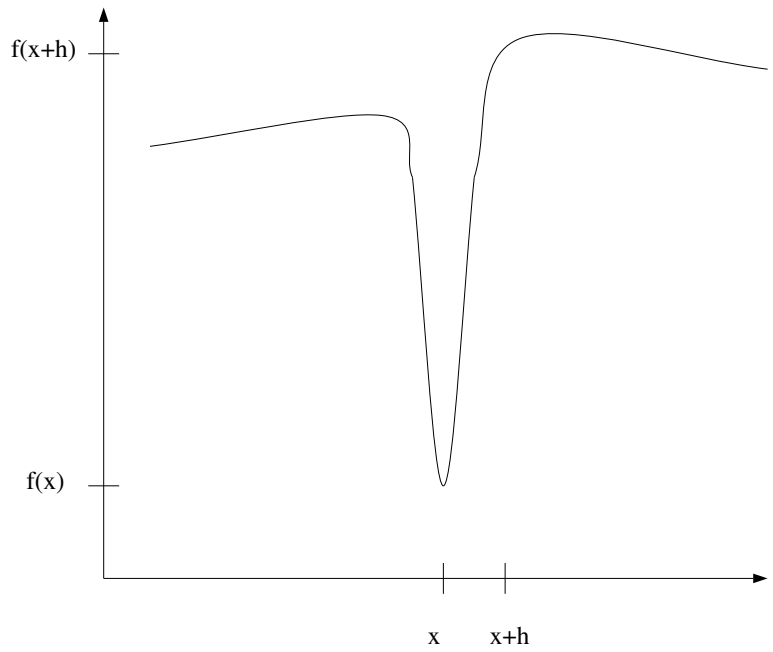

Abbildung 3.5: Approximation einer Umgebung

sich beweisen:

Theorem 3.2:
Das Problem, ein Element der Menge (3.3) eines beliebigen Problems (3.1) in einer endlichen Anzahl von Schritten zu bestimmen, ist unlösbar.

Beweis:[20]
Sei die Zielfunktion des Problems (3.1) stetig und der zulässige Bereich beschränkt und abgeschlossen. Angenommen, die Zielfunktion sei bereits k mal ausgewertet worden. Dann ist es immer noch möglich, daß der globale Minimalwert der Funktion f von dem bisher besten gefundenen Wert beliebig weit abweicht. Die zuvor gemachte Behauptung folgt durch Induktion über k.

□

Um das globale Optimierungsproblem überhaupt theoretisch angehen zu können, werden deshalb meist, je nach unterliegender Lösungsidee, bestimmte Annahmen getroffen, so daß wenigstens für Spezialfälle des allgemeinen Problems (3.1) Lösungsverfahren zur Verfügung stehen. Im folgenden Abschnitt sollen die verschiedenen Lösungsideen kurz umrissen werden.

[20] Vgl. Törn und Žilinskas (1989), S. 6.

3.4 Klassifikation der Lösungsmethoden

Eine Klassifikation der Lösungsmethoden für das globale Optimierungsproblem sollte sich einerseits dadurch auszeichnen, daß alle Methoden abgedeckt werden und andererseits, daß die resultierenden Klassen nicht überlappend und leicht unterscheidbar sind. Eine solche Unterteilung kann, je nach verwendeten Unterteilungscharakteristika sehr unterschiedlich ausfallen: Törn und Zilinskas [21] diskutieren aus diesem Grunde sechs verschiedene Klassifikationen, bevor sie ihre eigene motivieren. Im folgenen soll die in dieser Arbeit vorgenommene Klassifikation begründet werden (Abb. 3.6).

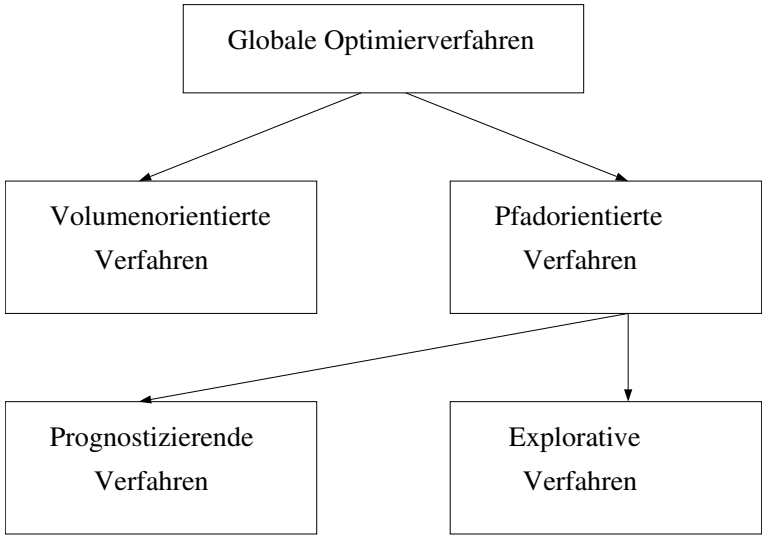

Abbildung 3.6: Klassifikation der Lösungsmethoden

Das erste Unterscheidungscharakteristikum betrifft die unterliegende Suchphilosophie: Bei den *volumenorientierten* Verfahren wird davon ausgegangen, daß eigentlich der gesamte zulässige Bereich abgesucht werden muß, um das globale Optimum zu finden. Die unvermeidliche Konsequenz aus dieser Ansicht spiegelt sich in der Forderung nach einem beschränkten Suchraum wider. Aus dieser Forderung motiviert sich auch die Namensgebung dieser Klasse: einem beschränkten Raum kann nämlich ein endliches Volumen zugeordnet werden.

Bei den *pfadorientierten* Verfahren geht man dagegen davon aus, daß ein solcher Aufwand von $\sim c^n$ nicht betrieben werden muß : Verfolgt man nämlich einen Weg im R^n, dann ist dieser im ungünstigsten Fall $\sim \sqrt{n}$ lang, wobei n die Dimension des Suchraumes angibt. Man startet also von einem beliebigen Punkt aus und verfolgt einen „geeigneten“ Pfad, um zum globalen Minimum zu gelangen. Im wesentlichen existieren zwei verschiedene Ansätze, um einen solchen Pfad bestimmen zu können:

[21] Vgl. Törn und Zilinskas (1989), S. 16ff.

Prognostizierende Verfahren sagen den nächsten Optimierschritt anhand eines expliziten inneren Modells von der Zielfunktion vorher. Treffen diese Prognosen im weiteren Verlauf des Verfahrens nicht zu (nämlich weil das Modell nicht auf die Zielfunktion zutrifft), wird das Verfahren meist abgebrochen.

Demgegenüber besitzen die *explorativen* Verfahren kein explizites inneres Modell: Vielmehr probieren sie mehrere Wege aus. Dabei wird nicht verlangt, daß jeder Versuch erfolgreich ist. Erst wenn im weiteren Verlauf erkannt wird, daß ein bestimmter Pfad erfolglos ist, wird ein alternativer Weg eingeschlagen. Auf diese Weise soll vermieden werden, daß das Verfahren vorzeitig zu einem lokalen Minimum konvergiert.

3.4.1 Volumenorientierte Verfahren

3.4.1.1 Rastersuche

Bei diese Methode[22] wird ein gleichmäßiges Raster in den zulässigen Bereich gelegt und die Zielfunktion an jedem dieser Rasterpunkte ausgewertet. Der Aufwand für dieses Verfahren steigt exponentiell mit der Dimension des Suchraumes, wie eine kurze Analyse zeigt:

Sei der zulässige Bereich M_R durch eine Hyperkugel mit dem Radius R gegeben. Weiterhin lasse sich M_R durch eine Vereinigung von N durchschnittsfremden Hyperkugeln M_r mit dem Radius $r < R$ darstellen:

$$M_R := \bigcup_{i=1}^{N} M_r^{(i)} \ , \ M_r^{(i)} \cap M_r^{(j)} = \emptyset \ \ \forall i \neq j. \tag{3.4}$$

Da sich das Volumen einer Hyperkugel M_R mit dem Radius R durch

$$\mu(M_R) = \frac{R^n \pi^{\frac{n}{2}}}{\Gamma(\frac{n}{2}+1)} \tag{3.5}$$

angeben läßt, folgt aus (3.4) für das Volumen:

$$\mu(M_R) = \sum_{i=1}^{N} \mu(M_r^{(i)}) = N\mu(M_r). \tag{3.6}$$

Durch einfache Umformung von (3.6) und Einsetzen von (3.5) ergibt sich für die Anzahl der erforderlichen Rasterpunkte N:

$$N = \frac{\mu(M_R)}{\mu(M_r)} = \frac{R^n \pi^{\frac{n}{2}}}{\Gamma(\frac{n}{2}+1)} \frac{\Gamma(\frac{n}{2}+1)}{r^n \pi^{\frac{n}{2}}} = \left(\frac{R}{r}\right)^n. \tag{3.7}$$

Wenn r den Abstand von der globalen Minimalstelle bezeichnet, dann sind $N = \left(\frac{R}{r}\right)^n$ Auswertungen der Zielfunktion nötig, um die gewünschte Genauigkeit r zu erlangen. Da für realistische Fälle $r << R$ gelten dürfte, wird der Aufwand dramatisch mit der Dimension des Suchraumes anwachsen.

[22] Vgl. dazu etwa Schwefel (1977), S. 47; Törn und Žilinskas (1989), S. 44.

3.4.1.2 Verallgemeinerte Branch and Bound - Verfahren

Die schlechte Effizienz der Rastersuche liegt darin begründet, daß der weitaus größte Teil der Rasterpunkte, die ausgewertet werden, weit vom Optimum entfernt liegen. Die Idee der Verfahren, die sich auf ein verallgemeinertes Branch and Bound - Verfahren abstützen, besteht darin, solche Gebiete des zulässigen Bereichs von der weiteren Betrachtung auszuschließen, die das globale Minimum nicht enthalten können. Somit erhält man folgenden Grundalgorithmus[23]:

`repeat`

- Teile M in Untermengen M_i auf.
- Bestimme eine untere Schranke $s_u(M_i) = \min\{f(\underline{x})|\underline{x} \in M_i\}$ für jede Untermenge M_i.
- Bestimme die obere Schranke $s_o = f(\underline{\tilde{x}})$, wobei $f(\underline{\tilde{x}})$ den bisher besten gefundenen Zielfunktionswert repräsentiert.
- Entferne diejenigen Untermengen M_i, für die $s_u(M_i) \geq s_o$ gilt, von der Menge M, da diese das globale Minimum nicht enthalten können:

$$M \leftarrow M \backslash \bigcup_i \{M_i | s_u(M_i) \geq s_o\}.$$

`until` $\mu(M) < \varepsilon$ mit $\mu(.)$ als Lebesgue-Maß.

Die Probleme bei diesen Verfahren sind darauf zurückzuführen, daß für jede Untermenge M_i eine untere Schranke bestimmt werden muß. Demnach muß also auch eine untere Schranke für das globale Minimum angegeben werden. Diese Aufgabe ist aber von derselben Schwierigkeit wie die Bestimmung des globalen Minimums selbst. Darum benötigen alle Verfahren diesen Typs weitere Annahmen bezüglich der Zielfunktion.

3.4.1.3 Überdeckungsverfahren

Bei den sogenannten Überdeckungsverfahren wird das globale Optimierungsproblem in das Problem umformuliert, den zulässigen Bereich mit unterschiedlich großen Hyperkörpern zu überdecken. Dabei handelt es sich um die Verallgemeinerung eines Verfahren von Evtushenko (1971). Letztlich zeigt sich aber auch bei diesem Verfahren, daß die Anzahl der zu ermittelnen Zentren der Hyperkörper exponentiell mit der Dimension des Suchraumes wächst[24].

[23] Vgl. dazu Rinnooy Kan und Timmer (1989), S. 635; Törn und Žilinskas (1989), S. 35ff.
[24] Vgl. Rinnoy Kan und Timmer (1989), S. 640f.

3.4.1.4 Einfache Zufallssuche

Während bei der Rastersuche ein äquidistantes Raster in den zulässigen Bereich gelegt wurde, so verteilt man bei der einfachen Zufallssuche die Versuchspunkte gemäß einer Gleichwahrscheinlichkeitsverteilung. Offensichtlich gibt es keinen Grund anzunehmen, daß dieses Verfahren besser als die Rastersuche abschneiden könnte. Tatsächlich läßt sich zeigen, daß dieses Verfahren eher schlechter abschneidet[25]:

Sei der zulässige Bereich M_R durch eine Hyperkugel mit dem Radius R und eine Umgebung $M_r(\underline{x}^*) := \{\underline{x} \in M : ||\underline{x} - \underline{x}^*|| \leq r\}$ von der globalen Minimalstelle mit dem Radius r gegeben. Wenn p die Wahrscheinlichkeit angibt, daß ein zufällig gewählter Punkt in der Umgebung M_r liegt, so ergibt sich mit (3.5) :

$$p = \frac{\mu(M_r)}{\mu(M_R)} = \left(\frac{r}{R}\right)^n. \tag{3.8}$$

Die Wahrscheinlichkeit p_N, daß nach N zufällig gewählten Punkten mindestens einer in der Umgebung M_r liegt, läßt sich mit (3.8) durch

$$p_N = 1 - (1-p)^N = 1 - \left[1 - \left(\frac{r}{R}\right)^n\right]^N \tag{3.9}$$

angeben. Mit der Abschätzung

$$\ln(1+y) \approx y \text{ für } y << 1$$

und der Zusicherung $r << R$ läßt sich (3.9) nach N auflösen:

$$N = \frac{\ln(1-p_N)}{\ln\left[1 - \left(\frac{r}{R}\right)^n\right]} \approx \frac{\ln(1-p_N)}{-\left(\frac{r}{R}\right)^n} = -\ln(1-p_N)\left(\frac{R}{r}\right)^n. \tag{3.10}$$

Da für die Rastersuche gemäß (3.7)

$$N = \left(\frac{R}{r}\right)^n$$

Versuchspunkte errechnet wurden, schneidet die einfache Zufallssuche schlechter ab, sobald für die Wahrscheinlichkeit p_N gilt:

$$-\ln(1-p_N) > 1 \Leftrightarrow p_N > 1 - \frac{1}{e} \approx 0.63.$$

Dabei hat die zufällige Reihenfolge der Rastersuch-Proben keinen negativen Einfluß auf die Effizienz, wohl aber die (unnötige) Wiederholung schon zuvor durchgeführter oder nahe beieinanderliegender Proben.

[25] Vgl. dazu Schwefel (1977), S. 108f.

3.4.1.5 Adaptiv stochastische Automaten

Es wurden zahlreiche Vorschläge gemacht, den Aufwand für die einfache Zufallssuche zu verringern. Ein großer Teil dieser Methoden läßt sich in die Klasse der sogenannten adaptiven stochastischen Automaten zusammenfassen[26]. Sämtliche Methoden dieser Art trachten danach, den Suchraum durch Ausnutzung von Information aus vorangegangenen Versuchen auf solche Gebiete einzuschränken, die erfolgversprechend erscheinen[27]. Dabei wird der Mechanismus, der die Reduktion des Suchraumes steuert, als ein stochastischer Automat aufgefaßt:

(1) Zunächst wird der Suchraum M in N gleich große Hyperquader M_i aufgeteilt:

$$M := \bigcup_{i=1}^{N} M_i \text{ mit } M_i \cap M_j = \emptyset \text{ für } i \neq j.$$

Jeder dieser Hyperquader M_i wird durch einen Zustand s_i des stochastischen Automaten repräsentiert. $p_i^{(t)}$ gibt die Wahrscheinlichkeit für den Zustand s_i des Automaten zum Zeitpunkt t an und wird mit

$$p_i^{(0)} = \frac{1}{N}$$

initialisiert.

(2) Aufgrund der Wahrscheinlichkeitsdichteverteilung $p^{(t)}$ der Zustände zum Zeitpunkt t wird nun ein Zustand s_k gewählt, d.h., s_k ist eine Realisation der Zufallsvariablen $s^{(t+1)}$ zum Zeitpunkt (t+1) gemäß der Verteilung $p^{(t)}$. Der Automat weist also zum Zeitpunkt (t+1) den Hyperquader M_k als erfolgversprechend aus.

(3) Falls gilt: $(t+1) = t_{max}$, gehe nach (5).

(4) Berechne:

$$z_k^{(t+1)} = \left(\frac{h}{f(\underline{x})}\right)^{\gamma} \text{ mit } \underline{x} \in M_k \text{ Zentrum von } M_k. \tag{3.11}$$

$$\bar{z}_i^{(t+1)} = \begin{cases} \alpha\, \bar{z}_i^{(t)} + (1-\alpha) z_i^{(t+1)} & , \text{ wenn } i = k \text{ mit } \alpha \in (0;1) \\ \bar{z}_i^{(t)} & , \text{ wenn } i \neq j. \end{cases} \tag{3.12}$$

$$p_i^{(t+1)} = \frac{\bar{z}_i^{(t+1)}}{\sum_{j=1}^{N} \bar{z}_j^{(t+1)}} \tag{3.13}$$

Gehe nach (2).

(5) Der aktuelle Zustand s_k repräsentiert den Hyperquader M_k, der durch den Automaten als wahrscheinlichster Kandidat für das globale Minimum vorhergesagt wird.

[26] Vgl. dazu Schwefel (1977), S. 110f.; Törn und Žilinskas (1989), S. 70ff.
[27] Vgl. McMurty und Fu (1966), S. 379ff.

Erläuterungen zu (3.11) bis (3.13):

(3.11) Die Funktion $f(\underline{x})$ wird über M als positiv angenommen:

$$f(\underline{x}) > 0 \quad \forall \underline{x} \in M.$$

h ist eine beliebige positive Konstante und die Wahl der Konstanten γ ist abhängig davon, wie „steil“ das „Tal“ für das globale Minimum abfällt. Dazu sind aber *a priori* - Informationen über den Verlauf der Zielfunktion notwendig. Es ist also an dieser Stelle festzuhalten, daß die Wahl der Konstanten γ für das gesamte Verfahren kritisch ist.

(3.12) $\bar{z}_k$ bezeichnet den durch α gewichteten Durchschnitt aus vorangegangenen Versuchen im Zustand s_k. Alle anderen Durchschnitte bleiben unverändert.

(3.13) Hier werden die neuen Wahrscheinlichkeiten p_i für die Zustände s_i ermittelt.

Dieses Verfahren wurde unter anderem von Hill (1969) dahingehend modifiziert, daß er

a) den resultierenden Hyperquader M_k als neuen Suchraum auffaßt und den Algorithmus von neuem startet und daß er

b) während jeden Zustandes einen sogenannten Gradientensuchschritt durchführt und am Ende des Algorithmus im verbleibenden Hyperquader eine lokale Suche startet.

Er selbst merkt jedoch an, daß die praktische Anwendbarkeit des Verfahrens auf Probleme der Dimensionalität ≤ 8 beschränkt sei[28].

3.4.1.6 Cluster-Verfahren

Ähnlich wie die stochastischen Automaten versuchen die sogenannten Cluster-Verfahren den Suchraum geeignet einzuschränken. Dazu verwenden sie als Steuerungsmechanismus jedoch keinen stochastischen Automaten, sondern setzen dafür Algorithmen aus dem Bereich der Datenanalyse, nämlich Cluster- bzw. Formationsanalyse-Algorithmen[29] ein.

Interpretiert man eine Sequenz von N Versuchspunkten mit den zugehörigen Zielfunktionswerten als zu analysierendes Datenmaterial, so lassen sich mit Hilfe der Clusteranalysetechniken die N Versuchspunkte derart um lokale Minima gruppieren, daß die Versuchspunkte in jedem Cluster möglichst homogen und die Cluster untereinander möglichst heterogen sind. Durch die Verwendung von Clusteranalysetechniken wird also versucht, die Struktur der Zielfunktion bezüglich ihrer lokalen Minima zu analysieren, um nur noch in solchen Gebieten weiterzusuchen, wo besonders ‚gute‘ lokale Minima vermutet werden.

[28] Vgl. Hill (1969), S. 8.
[29] Vgl. dazu etwa Späth (1983).

Ausgehend vom ersten Cluster-Suchverfahren von Becker und Lago (1970) wurden zahlreiche Varianten entwickelt[30]. Im folgenden soll der Grundalgorithmus wiedergegeben werden:

(1) Plaziere N Versuchspunkte im Suchgebiet M.

(2) Konzentriere die Stichprobe der N Versuchspunkte wie folgt:

   (a) Betrachte nur die $N' < N$ Punkte mit den niedrigsten Funktionswerten oder

   (b) verschiebe die Punkte durch die Anwendung eines lokalen Suchverfahrens oder

   (c) verwende eine Kombination aus (a) und (b).

(3) Verwende einen Clusteranalysealgorithmus, um die Versuchspunkte um lokale Minima zu gruppieren.

(4) Abbruch,

   (a) falls nur noch ein Cluster übrig ist oder

   (b) falls keine neuen lokalen Minima entdeckt werden oder

   (c) falls das Gebiet des Suchraumes M, das eine Verbesserung erbracht hat, ein „geringes“ Maß hat oder

   (d) falls ein Kriterium greift, das auf der Verteilung der erwarteten Anzahl von lokalen Minima basiert.

(5) Treffe Vorbereitungen für die nächste Iteration und gehe dann nach (2):

   (a) Betrachte nur einige, die besten, Cluster als neue Probleme und bestimme ein Suchgebiet aus den Punkten in jedem Cluster, oder

   (b) wähle eine Anzahl von Punkten aus jedem Cluster oder

   (c) behalte die Clusterstruktur sowie die besten Punkte je Cluster bei und plaziere neue Versuchspunkte in M.

(6) Abschließende Berechnungen

   (a) Verwende eine lokale Suche vom besten Cluster aus oder

   (b) verwende eine lokale Suche von demjenigen Punkt je Cluster, der den niedrigsten Funktionswert besitzt.

[30] Für einen Überblick vgl. Törn und Žilinskas (1989), S. 98ff.

3.4.1.7 Bayes'sche Methoden

Diese Methoden trachten nicht danach, die maximale Abweichung vom Extrempunkt in einer begrenzten Anzahl von Schritten zu minimieren, sondern vielmehr nach der Minimierung der mittleren Abweichung vom Extrempunkt[31]. Dazu muß *a priori* eine Wahrscheinlichkeitsdichteverteilung bekannt sein, die ein akzeptables Modell für diejenige Klasse von Funktionen darstellt, der die Zielfunktion angehört.

Die prizipielle Idee für diese Methoden entspringt der intuitiv offensichtlichen Vorstellung, daß eine Rastersuche über ein ungleichmäßig verteiltes Raster, welches in der Umgebung von guten gefundenen Punkten feiner ist, effizienter zur Lösung führt, als eine Suche über ein gleichmäßig verteiltes Raster[32]. Um solche ungleichmäßigen Raster rational konstruieren zu können, wird ein Modell von der Zielfunktion benötigt.

Žilinskas (1978) schlägt vor, einige allgemeine Annahmen über die Kompliziertheit der Zielfunktion durch ein Axiomensystem zu formalisieren[33]. Er zeigt, daß diese Axiome durch eine Familie von Zufallsvariablen $Y_{\underline{x}}$, $\underline{x} \in M$, mit den Dichten $p_{\underline{x}}(.)$ repräsentiert werden können. Die Wahl eines neuen Raster- bzw. Versuchspunktes für eine Zielfunktionsauswertung kann dann als eine Wahl zwischen den Dichten $p_{\underline{x}}(.)$ auf der Basis des akzeptierten statistischen Modells von der Zielfunktion interpretiert werden. Žilinskas' theoretische Untersuchungen sichern zwar die Existenz und die Eindeutigkeit der Dichten $p_{\underline{x}}(.)$ zu, jedoch ergibt sich daraus keine theoretische Grundlage für ihre konstruktive Herleitung. Eine explizite Form von $p_{\underline{x}}(.)$ ist aber notwendig, um einen Optimieralgorithmus konstruieren zu können.

Durch ein psychologisches Experiment[34] motiviert Žilinskas die Verwendung von normalverteilten Dichten $p_{\underline{x}}(.)$ als statistisches Modell von der Zielfunktion. Zur Charakterisierung der Dichten genügt somit die Kenntnis des Erwartungswertes und der Varianz der Zufallsvariablen $Y_{\underline{x}}$. Als Schätzer für den Erwartungswert wird der gewichtete mittlere Wert $m_k(.)$ aus k Versuchspunkten gewählt[35]:

$$m_k(\underline{x}; (\underline{x}_i, y_i); i = 1(1)k) = \sum_{i=1}^{k} y_i \, w_i^k(\underline{x}; \underline{x}_j; j = 1(1)k),$$

wobei die

$$w_i^k(\underline{x}; \underline{x}_j; j = 1(1)k) = \begin{cases} 0 & \text{, falls } i \notin I(\underline{x}) \\ \frac{d(\underline{x}, \underline{x}_i)}{\sum\limits_{j \in I(\underline{x})} d(\underline{x}, \underline{x}_j)} & \text{, sonst} \end{cases}$$

[31] Vgl. Mockus (1975), S. 167f.
[32] Vgl. Žilinskas (1980), S. 139.
[33] Vgl. Žilinskas (1978), S. 256ff.
[34] Vgl. Žilinskas (1978), S. 260ff.; Törn und Žilinskas (1989), S. 141ff.
[35] Vgl. Žilinskas (1980), S. 140.

die Gewichte mit einem Abstandsmaß

$$d(\underline{x}\,,\,\underline{x}_i) = \frac{e^{-c\,||\underline{x}-\underline{x}_i||^2}}{||\underline{x}-\underline{x}_i||}$$

mit c = 3.3 darstellen und $I(\underline{x})$ die Indexmenge der 5 nächsten Nachbarn von $\underline{x}$ ist. Als Schätzer für die Varianz wird

$$s_k^2(\underline{x};(\underline{x}_i,y_i);i=1(1)k) = \gamma_k \sum_{i=1}^{k} a\,||\underline{x}-\underline{x}_i||\; w_i^k(\underline{x};\underline{x}_j;j=1(1)k)$$

mit $a>0,\ \gamma_k>0$ verwendet[36].

Für eine rationale Wahl eines neuen Versuchspunktes ist dann eine sogenannte Präferenzrelation zwischen den Paaren (m,s) aus den Erwartungswert- und Varianzschätzern notwendig. Die von Žilinskas axiomatisch begründete Präferenzrelation läßt sich wie folgt interpretieren:

- Die Wahl eines neuen Versuchspunktes, bei dem der zu erwartende Zielfunktionswert m vergleichsweise groß ist, kann nur im Fall großer Unsicherheit s rational sein.
- Die Auswertung der Zielfunktion in einem Punkt, bei dem die Varianz s gleich Null ist, liefert kein neues Wissen über die Zielfunktion und ist deshalb nicht rational.
- Es ist rational, einen Punkt auszuwerten, der einen niedrig zu erwartenden Zielfunktionswert m und eine große Varianz s besitzt, weil so die Unsicherheit s verringert wird.

Der nächste Versuchspunkt ist also derjenige, der aufgrund der Präferenzrelation allen anderen Punkten mit den Paaren (m,s) vorgezogen wird. Zur Bestimmung dieses Punktes wird eine sogenannte Nutzenfunktion

$$U(m,s) = \int_{-\infty}^{\infty} u(t)\,p_{m,s}(t)\;dt$$

mit

$$u(t) = \begin{cases} 1 & \text{für } t < f(\tilde{\underline{x}}) \\ 0 & \text{sonst} \end{cases}$$

konstruiert[37], die proportional zu der Wahrscheinlichkeit $P_k(\underline{x})$ ist, daß eine Realisation der Gauß'schen Zufallsvariable $Y_{\underline{x}}$ kleiner als der bisher beste gefundene Zielfunktionswert $f(\tilde{\underline{x}})$ ist. Man wählt also als nächsten Punkt einen solchen mit maximalem Nutzen:

$$\underline{x}_{k+1} := \hat{\underline{x}} = \{\underline{x} \in M \mid P_k(\hat{\underline{x}}) \geq P_k(\underline{x})\,\}.$$

[36]Vgl. Törn und Žilinskas (1989), S. 147.
[37]Vgl. Törn und Žilinskas (1989), S. 148ff.

Die Maximierung der Nutzenfunktion U(.) wird bei Žilinskas durch die Kombination aus Rastersuche und lokalen Techniken durchgeführt[38], so daß eine sinnvolle Anwendung dieses Verfahrens nur dann gegeben ist, wenn die Auswertung der Zielfunktion extrem aufwendig ist, da durch die Rastersuche bei der Maximierung der Nutzenfunktion viele Hilfsberechnungen ausgeführt werden müssen. Žilinskas berichtet von erfolgreichen Anwendungen auf Probleme einer Dimensionalität $n \leq 15$.

3.4.2 Pfadorientierte Verfahren

3.4.2.1 Prognostizierende Verfahren

Prognostizierende Methoden sagen anhand eines explizitem inneren Modells der Zielfunktion den Weg zu einem neuen, besseren Versuchspunkt voraus. Trifft dann die Prognose im weiteren Verlauf nicht zu, so wird das Verfahren meist abgebrochen.

3.4.2.1.1 Trajektorien-Verfahren Das globale Minimum einer Funktion f kann dann gefunden werden, wenn es möglich ist, alle lokalen Minima zu identifizieren. Für ein lokales Minimum gilt die notwendige Bedingung

$$\nabla f(\underline{x}) = \underline{0} \text{ mit } \underline{x} \in M. \tag{3.14}$$

Um dieses nichtlineare Gleichungssystem (3.14) zu lösen, schlug Branin (1972) vor, den Trajektorien der Differentialgleichungen

$$\dot{\underline{x}} = \pm\left[\nabla^2 f(\underline{x})\right]^{-1} \nabla f(\underline{x}) \tag{3.15}$$

zu folgen, da diese durch die Nachbarschaft der meisten (jedoch nicht aller) lokalen Minmalstellen verlaufen[39]. So gibt Treccani (1975), S. 109ff., das Beispiel einer konvexen, quadratischen Funktion an, die auf einer Teilmenge des zulässigen Bereichs das globale Minimum nicht annimmt und aus dem die Trajektorie von (3.15) nicht entkommen kann[40]. Weitere Schwierigkeiten entstehen, wenn die Zielfunktion f durch ein Computerprogramm gegeben ist, da dann über die Stabilität der numerischen Integration von (3.15) keine Aussage mehr gemacht werden kann[41].

[38] Vgl. Törn und Žilinskas (1989), S. 134.

[39] Diese Formel läßt sich durch eine Taylor-Reihenentwicklung begründen.

[40] Diskretisiert man Gleichung (3.15), so erhält man gerade das NEWTONsche Iterationsverfahren zur Lösung eines nichtlinearen Gleichungssystems. In Abhängigkeit von den Startwerten strebt dieses Verfahren gegen verschiedene sogenannte Fixpunkte. Schon bei sehr einfachen rationalen Funktionen läßt sich zeigen, daß die Grenzen zwischen den Einzugsgebieten der Fixpunkte keine glatten Kurven, sondern komplizierte, ineinander verwobene selbstähnliche Strukturen darstellen, die eine nichtganzzahlige fraktale Dimension besitzen. Vgl. dazu Leven u.a. (1989), S. 143ff.

[41] Vgl. Törn und Žilinskas (1989), S. 59.

Für dieses Verfahren wurden eine Reihe von Modifikationen vorgeschlagen[42], um die Trajektorien aus solchen Bereichen abzulenken, aus denen sie nicht entkommen können. Das kann man zum einen dadurch erreichen, daß man (3.15) mit einem Störterm versieht, der stark oszillierend wirkt, wenn $|| \nabla f(\underline{x}) ||$ klein wird, also wenn man sich einem stationären Punkt nähert[43]. Bevor dann die Trajektorie abgelenkt wird, wird eine lokale Suche durchgeführt, die das lokale Minimum identifiziert (falls es ein solches ist). Zum anderen kann man denselben Effekt erzielen, indem man einen stochastischen Störterm verwendet. Diese Verfahren sind dann schon der Klasse der explorativen Methoden zuzurechnen, die später vorgestellt werden.

3.4.2.1.2 Tunnelungs-Verfahren Die sogenannten Tunnelungs-Verfahren gehen von der intuitiven Vorstellung aus, daß die Topologie der Zielfunktion mit einem Gebirge vergleichbar ist (Abb. 3.7) und immer bessere lokale Minima („tiefere Täler“) mittels einer „Durchtunnelung“ der „Berge“ gefunden werden können[44].

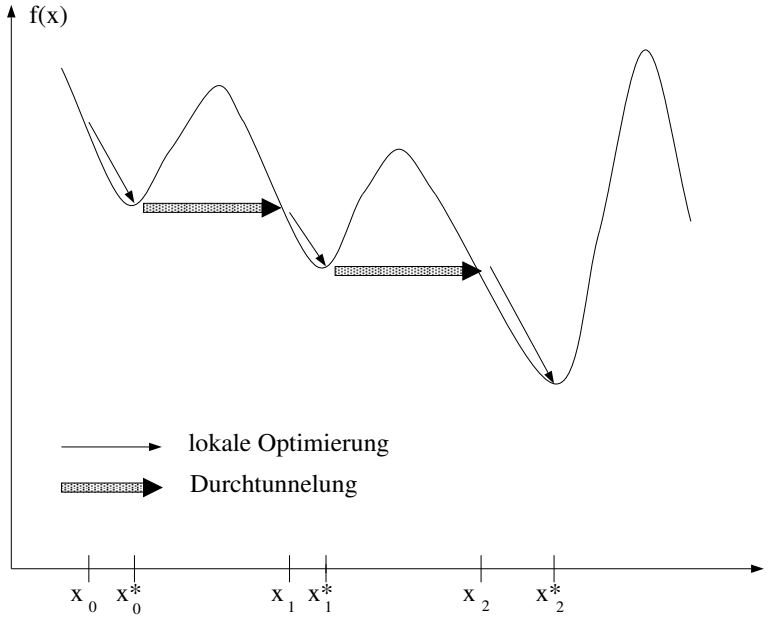

Abbildung 3.7: Idee des Tunnelungs-Verfahrens

Der Algorithmus besteht aus lokalen Optimierungs- und einer Durchtunnelungsphasen: Man startet bei einem $x_0 \in M$ mit einem lokalen Optimierverfahren und

[42]Vgl. dazu Dixon u.a. (1975), S. 35ff. und 44ff.

[43]Vgl. Griewank (1981).

[44]Dieser Ansatz wird Vilkow u.a. (1975) zugeschrieben. Eine Verallgemeinerung auf den multivariaten Fall stammt von Levy und Montalvo (1985).

gelangt so zu einer lokalen Minimalstelle x_0^*. Von hier aus wird der „Berg“ der Zielfunktion bis zu einem Punkt x_1 „durchtunnelt“. Von diesem Punkt aus startet erneut eine lokale Minimierung, die in der lokalen Minimalstelle x_1^* endet. Nun wiederholt sich der Prozeß, bis kein neues lokales Minimum mehr gefunden werden kann. Das zuletzt gefundene lokale Minimum ist dann das globale Minimum.

Die Durchtunnelungsphase wird dadurch modelliert, daß eine Nullstelle des Polynoms

$$T_\alpha(\underline{x}) = \frac{f(\underline{x}) - f(\underline{x}^*)}{||\underline{x} - \underline{x}_m||^{\alpha_0} \prod_{i=1}^{k} ||\underline{x} - \underline{x}_i^*||^{\alpha_i}} \tag{3.16}$$

gefunden werden muß, die den gleichen Funktionswert wie das bisherige lokale Minimum besitzt. Diese Nullstelle ist dann der Startpunkt für die nächste lokale Minimierungsphase. $\underline{x}^*$ bezeichnet die zuletzt gefundene lokale Minimalstelle und die $\underline{x}_i^*$ solche lokalen Minimalstellen, die den gleichen Zielfunktionswert wie $\underline{x}^*$ besitzen. Die Multiplikatoren

$$\prod_{i=1}^{k} ||\underline{x} - \underline{x}_i^*||^{-\alpha_i}$$

stellen Strafkosten für eine Annäherung an bekannte lokale Minima dar, und durch den Multiplikator $||\underline{x} - \underline{x}_m||^{-\alpha_0}$ wird durch eine geignete Wahl von $\underline{x}_m$ verhindert, daß ein sogenannter stationärer Punkt von

$$\frac{f(\underline{x}) - f(\underline{x}^*)}{\prod_{i=1}^{k} ||\underline{x} - \underline{x}_i^*||^{\alpha_i}}$$

angelaufen wird, der keine Nullstelle von (3.16) ist. Wird in der Tunnelungsphase kein neuer Startpunkt entdeckt, so werden die Exponenten α_i erhöht.

Törn und Žilinskas (1989), S. 62, kritisieren an dieser Methode, daß die Hyperoberfläche der Tunnelungsfunktion (3.16) sehr flach wird, wenn die α_i erhöht werden. Dadurch bewegen sich die Werte von (3.16) sämtlich nahe um Null, d.h., die Nullstellensuche muß mit sehr kleinen Schrittweiten durchgeführt werden.

Ähnlich argumentieren auch Rinnooy Kan und Timmer (1989), S. 649, indem sie bemerken, daß die Tunnelungsmethode das globale Optimierungsproblem lediglich dahingehend umformuliert, den Nachweis zu erbringen, daß ab einem bestimmten Iterationsschritt keine weitere Nullstelle existiert.

Da es darüberhinaus schwierig ist, eine Nullstelle im mehrvariaten Fall zu finden, wird von Yao (1989), S. 1225, vorgeschlagen, die Nullstellensuchen dadurch zu ersetzen, dem dynamischen Fluß der Differentialgleichungen

$$\dot{\underline{x}} = -\nabla f(\underline{x}) \; [\, ||\underline{x} - \underline{x}^*||^\alpha + k \, \max\{0\,,\, f(\underline{x}) - f(\underline{x}^*)\}]$$

mit $k > 1$ zu folgen. Dazu müssen allerdings die ersten partiellen Ableitungen sowie *a priori* - Information bzgl. f zur Abschätzung der Konstanten k bekannt sein.

3.4.2.2 Explorative Verfahren

Eine wesentliche Eigenschaft der explorativen Methoden ist ihre Fähigkeit, mehrere alternative Wege ausprobieren zu können: Dadurch ist es möglich, aus der Nähe lokaler Minima wieder zu entkommen.

3.4.2.2.1 Rotierende Koordinaten Bei der einfachen Koordinaten- oder Gauß-Seidel Strategie wird jeweils längs der Koordinatenachsen gesucht. Ganz offensichtlich gibt es aber keinen Grund, diese Richtungen zu bevorzugen. So schlug Rosenbrock (1960) vor, die Suchschritte parallel zu den Achsen eines im R^n drehbaren Koordinatensystems durchzuführen. Das im Laufe des Verfahrens rotierende Koordinatensystem wird dabei wie folgt konstruiert: Die erste Achse wird in die am günstigsten erscheinende Richtung gelegt, und alle anderen Achsen werden durch ein Orthogonalisierungsverfahren (nach Gram-Schmidt oder Palmer) senkrecht untereinander ausgerichtet. War ein solcher Suchschritt erfolgreich, wird die Schrittweite erhöht und anderenfalls verringert.

Unter den zahlreichen Modifikationen sei noch die sogenannte DSC-Strategie von Swann (1964) erwähnt[45]. Bei diesem Verfahren wird die Idee der *rotierenden Koordinaten* mit eindimensionalen Suchen in den Koordinatenrichtungen kombiniert.

Mit dieser Art von Verfahren wurden gute Erfolge erzielt, allerdings macht sich der Aufwand für die Orthonormierung bei höherdimensionalen Problemen negativ bemerkbar.

3.4.2.2.2 Muster-Suche Die Vorgehensweise der Muster-Suche (eng. pattern search) von Hooke und Jeeves (1961) ist durch zwei Arten von Bewegungen gekennzeichnet[46]: Bei jeder Iteration wird ein Tastzyklus (exploratory move) durchgeführt, der aus einer vereinfachten Gauß-Seidel Variation mit je einem diskreten Schritt pro Koordinatenrichtung besteht. In der Annahme, daß der Vektor vom Start- zum Endpunkt des Tastzyklus in eine günstige Richtung zeigt, wird in ihr eine Extrapolation (pattern move) durchgeführt. Erst nach dem daran anschließenden Tastzyklus wird der Erfolg der Iteration überprüft. Dadurch wird bei einer nur allmählichen Änderung der optimalen Suchrichtung die Länge des Musterschrittes sukzessiv vergrößert, was sich besonders bei engen Schluchten auszahlt.

3.4.2.2.3 Complex-Strategie Bei der Complex-Strategie von Box (1965) handelt es sich um eine Modifikation der sogenannten Simplex-Strategie von Nelder und Mead (1965), die im übrigen nichts mit dem sogenannten Simplex-Verfahren von Dantzig (1966) für die *Lineare Programmierung (LP)* gemein hat. Bei diesem Verfahren wird ein Polytop mit $n+1 \leq N \leq 2n$ Ecken aufgespannt,

[45] Vgl. dazu Schwefel (1977), S. 63ff.
[46] Vgl. dazu Schwefel (1977), S. 54ff.

welches dann durch geeignete Expansions- und Kontraktionsregeln durch den Suchraum wandert.

3.4.2.2.4 Simulated Annealing Unter ‚statistische Mechanik' versteht man eine Sammlung von Methoden der Physik zur Analyse von Aggregateigenschaften einer großen Anzahl von Atomen eines Stoffes im festen, flüssigen oder gasförmigen Zustand[47]. Da diese Atome nicht einzeln beobachtet werden können, wird in Experimenten nur das wahrscheinlichste Gesamtverhalten des Systems im thermischen Gleichgewicht beobachtet. Von zentralem Interesse ist dabei, wie sich das System nahe seinem Gefrierpunkt verhält, weil dann die Eigenschaften seines Grundzustandes (energetisches Minimum) dominieren.

Bei Experimenten zeigte sich, daß ein bloßes Abkühlen des Stoffes noch kein Garant für das Auffinden eines Grundzustandes ist: Wird nämlich der Stoff zu schnell abgekühlt, so werden die zufälligen Schwankungen des Systems mit eingefroren. Um dies zu vermeiden, wird die Substanz zunächst geschmolzen und dann langsam schrittweise abgekühlt, wobei die Temperatur jeweils so lange konstant gehalten wird, bis das System in ein thermisches Gleichgewicht gerät.

Diesen langsamen Abkühlungsprozeß nennt man *Annealing*. Entsprechend wird dessen Simulation auf einem Rechner bzw. die Anwendung dieses Prinzips auf Optimierungsprobleme als *Simulated Annealing* bezeichnet. Die Analogien zwischen dem Auffinden eines Grundzustandes einer Substanz und dem globalen Optimierungsproblem sind in der Tab. 3.1 zusammengefaßt.

statist. Mechanik	**globale Optimierung**
Atomkonfiguration	zulässige Lösung
Energie des Systems	Zielfunktionswert
Schwankungen des Systems	Möglichkeit, aus lokalen Minimabereichen herauszulaufen
Grundzustand bei energetischem Minimum	globale Minimalstelle

Tabelle 3.1: Analogien zwischen stat. Mechanik und Optimierung

Das Ausmaß der zufälligen Schwankungen wird bei niedrigeren Temperaturen immer geringer. Theoretisch läßt sich dies aus der Sicht der statistischen Mechanik durch die sogenannte *Boltzmann-Verteilung* herleiten: Die Wahrscheinlichkeit, daß eine im Sinne der Optimierung schlechtere Konfiguration (mit höherer Energie: $\Delta E > 0$) akzeptiert wird, beträgt

$$P(\Delta E) = e^{-\frac{\Delta E}{bT}} ,$$

[47] Vgl. im folgenden Kirkpatrick u.a. (1983), S. 671ff.

wobei b die sogenannte Boltzmann-Konstante bezeichnet[48]. Der Grundalgorithmus lautet:

geg.: Temperatur T,
Abkühlungsfaktor $\chi_T \in (0;1)$,
Startwert $\underline{x}^{(0)}$

(1) $M = c\,n$ (n: Dimension, $c = 15$ nach Vanderbilt und Louie (1984))

(2) $\underline{x}^{(t+1)} = \underline{x}^{(t)} + \Delta\underline{x}$ „Nachfolgekonfiguration“

(3) $\Delta f = f(\underline{x}^{(t+1)}) - f(\underline{x}^{(t)})$

(4) falls $\Delta f < 0$, dann akzeptiere: $\underline{x}^* := \underline{x}^{(t+1)}$
sonst akzeptiere mit Wahrscheinlichkeit $e^{-\frac{\Delta f}{T}}$

(5) $M \leftarrow M - 1$

(6) falls $M > 0$, gehe nach (2) „Gleichgewichtsschleife“

(7) $T \leftarrow \chi_T\, T$

(8) falls $T > 0$, gehe nach (1) „Abkühlungsschleife“

(9) Ausgabe: $\underline{x}^*$

Ursprünglich war Simulated Annealing für die diskrete Optimierung konzipiert. Vanderbilt und Louie (1984), S. 259ff., modifizierten das Verfahren für die kontinuierliche Optimierung. Dazu mußten sie den Teilalgorithmus in Schritt (2), der die Nachfolgekonfiguration erzeugt, geeignet anpassen:

Die additive Änderung $\Delta\underline{x}$ des alten Versuchspunktes $\underline{x}^{(t)}$ kann durch

$$\Delta\underline{x} = \underline{Q}\,\underline{u} \tag{3.17}$$

berechnet werden, wobei $\underline{u}$ einen Zufallsvektor mit stochastisch unabhängigen, gleichverteilten Komponenten $u_i \sim G(-\sqrt{3};\sqrt{3})$ bezeichnet. Eine einfache Wahl für die Matrix $\underline{Q}$, die die Schrittweite und -richtung steuert, wäre etwa

$$\underline{Q} = a\,\underline{E} \;(\; \underline{E} : \text{Einheitsmatrix};\; a \in R\;)$$

oder auch

$$\underline{Q} = \text{diag}(a_1,\, a_2,\, \ldots,\, a_n) \quad \text{mit } a_i \in R.$$

Vanderbilt unf Louie (1984) schlagen vor, die Steuermatrix $\underline{\text{Q}}$ durch die Kovarianzmatrix $\underline{\text{K}}$ der letzten m Versuchspunkte adaptiv anzupassen[49]:

$$\underline{K} = \underline{Q}\,\underline{Q}^T \quad . \tag{3.18}$$

[48]Die Boltzmann-Konstante hat für die Optimierung keine Bedeutung und kann deshalb im weiteren Verlauf vernachlässigt werden.

[49]Die Matrix $\underline{\text{Q}}$ erhält man aus $\underline{\text{K}}$ durch die sogenannte Cholesky-Zerlegung; vgl. etwa Jordan-Engeln und Reutter (1976), S. 49.

Zur Bestimmung der Kovarianzmatrix $\underline{K}$ werden zunächst die ersten zwei Momente $\underline{a}$ und $\underline{S}$ des Wegsegmentes berechnet:

$$a_i^{(l)} = \frac{1}{M} \sum_{m=1}^{M} s_i^{(m;l)} \quad , \tag{3.19}$$

$$S_{ij}^{(l)} = \frac{1}{M} \sum_{m=1}^{M} |x_i^{(m;l)} - a_i^{(l)}| \, |x_j^{(m;l)} - a_j^{(l)}| \quad , \tag{3.20}$$

wobei $\underline{x}^{(m;l)}$ den Wert von $\underline{x}$ beim m-ten Schritt der l-ten „Abkühlungsschleife“ darstellen soll. Die neue Kovarianzmatrix $\underline{K}^{(l+1)}$ der nächsten „Abkühlungsschleife“ berechnet sich dann aus dem zweiten Moment des vorherigen Wegsegmentes durch[50]

$$\underline{K}^{(l+1)} = \frac{\chi_s}{\beta \, M} \underline{S}^{(l)} \quad \text{mit } \beta = 0.11 \quad , \tag{3.21}$$

wobei $\chi_S > 1$ einen „Wachstumsfaktor“ darstellt, so daß ein Zufallsweg in der (l+1)-ten „Abkühlungsschleife“ im Mittel ein um den Faktor $\sqrt{\chi_S}$ größeres Gebiet abdeckt als in der vorherigen Iteration.

Ein solcher reiner Zufallsweg (engl. random walk) ist dann möglich, wenn die Temperatur T sehr groß gewählt wurde: Die Versuchspunkte werden dann verhältnismäßig „weit“ auseinanderliegen, da bei höherer Temperatur nahezu jeder Punkt (auch eine Verschlechterung) akzeptiert wird. Ein solches Verhalten ist eher den volumenorientierten Verfahren zuzurechnen - klingt jedoch die Temperatur ab, so geht dieses Verfahren in eine pfadorientierte Methode über, weil dann die Anzahl der akzeptierten schlechteren Versuchspunkte allmählich geringer und somit die Weglänge einer „Abkühlungsschleife“ reduziert wird. Auf diese Weise paßt sich das Verfahren der Topologie der Zielfunktion an.

Vanderbilt und Louie berichten von zufriedenstellenden Ergebnissen bei Testproblemen mit einer Dimension ≤ 8.

3.4.2.2.5 Kriechende Zufallssuche und verwandte Konzepte Diese Verfahren versuchen, den Aufwand für die einfache Zufallssuche zu verringern. Die Idee besteht darin, von einem Startpunkt $\underline{x}^{(0)}$ aus zu starten und von diesem ausgehend zufällig einige Versuchspunkte zu plazieren, die mit großer Wahrscheinlichkeit in die unmittelbare Nähe von $\underline{x}^{(0)}$ fallen. Eine geeignete Wahrscheinlichkeitsdichteverteilung dafür bietet etwa die mehrdimensionale Normalverteilung mit dem Erwartungsvektor $\underline{x}^{(0)}$ und dem Standardabweichungsvektor[51] $\underline{\sigma}$. Jeder Versuchspunkt mit besserem Zielfunktionswert wird als Ausgangspunkt $\underline{x}^{(t+1)}$ einer erneuten zufälligen Suche mit der Verteilung $\mathrm{N}(\underline{x}^{(t+1)}, \underline{\sigma})$ verwendet.

Auf diese Weise wird das Wahrscheinlichkeitsdichtefeld allmählich durch den Suchraum verschoben, bzw. das Wahrscheinlichkeitsdichtefeld „kriecht“ durch

[50] Weitere Möglichkeiten finden sich bei Vanderbilt und Louie (1984), S. 264.

[51] Die Kovarianzen sind also sämtlich gleich Null.

den Suchraum. Solange dabei $\underline{\sigma} > \underline{0}$, kann selbst ein multimodales Problem jederzeit im Prinzip mit einem Schritt gelöst werden, gleich wie gering die Wahrscheinlichkeit dafür ist. Somit ist die (theoretische) Konvergenz stets gesichert[52]

Diese von Brooks (1958) vorgebrachte Idee wurde einer Vielzahl von Modifikationen unterzogen[53]. Einen typischen Algorithmus gibt folgende Iterationsvorschrift wieder:

$$\underline{x}^{(t+1)} = \begin{cases} \underline{x}^{(t)} + \underline{z}^{(t)} & \text{, falls } f(\underline{x}^{(t)} + \underline{z}^{(t)}) \leq f(\underline{x}^{(t)}) \text{ und } g_i(\underline{x}^{(t)} + \underline{z}^{(t)}) \leq 0 \quad \forall i \\ \underline{x}^{(t)} & \text{, sonst} \end{cases}$$

wobei $\underline{z}^{(t)}$ einen Zufallsvektor gemäß einer $N(\underline{0}, \underline{\sigma})$-Verteilung darstellt. Bei diesem Verfahren ist die Wahl der Standardabweichung σ von Bedeutung, da durch sie im Wesentlichen die Schrittweite des Verfahrens festgelegt wird. Je nach aktueller Position $\underline{x}^{(t)}$ und der Topologie der Zielfunktion muß die Standardabweichung angepaßt werden. Theoretische Untersuchungen dazu finden sich bei Rastrigin (1963), Schumer und Steiglitz (1968), Rechenberg (1973), Schwefel (1977), Born (1978), Rappl (1984), Marti (1986) sowie Beyer (1989).

Jedes dieser Verfahren kann man als eine vereinfachte Simulation der biologischen Evolution auffassen, wenn die zufällige Änderung $\underline{z}^{(t)}$ als eine *Mutation* des Genoms $\underline{x}^{(t)}$ eines Individuums und die Auswahl des im Sinne der Zielfunktion besseren Genoms als *Selektion* betrachtet. Der Ansatz, die Analogien zwischen dem Evolutions- und dem Optimierungsprozeß im Sinne von Optimier-Regeln bewußt auszunutzen, stammen von dem Team um Rechenberg (1973), von Bremermann (1970) und von Holland (1975).

Eine Weiterentwicklung der sogenannten zweigliedrigen Evolutionsstrategie von Rechenberg (1973) durch die Aufnahme eines *Populations-Konzeptes* in die Optimierungsstrategie führte zu der *mehrgliedrigen* Evolutionsstrategie von Schwefel (1977). Diese ist konzeptuell eng verwandt mit den unabhängig davon entwickelten *genetischen Algorithmen* von Holland (1975). Diese Weiterentwicklungen sollen Gegenstand des nächsten Abschnitts sein.

3.4.2.2.6 Mehrgliedrige Evolutionsstrategien Mehrgliedrige Evolutionsstrategien und genetische Algorithmen[54] unterscheiden sich von anderen Strategien zunächst durch die Aufnahme eines ‚Populationskonzeptes': Während etwa bei der sogenannten zweigliedrigen Evolutionsstrategie[55] von Rechenberg (1973) für einen Versuchspunkt $\underline{x}^{(n)}$ genau ein neuer Versuchspunkt $\underline{x}^{(n+1)}$ je Iteration erzeugt wird, so werden bei der mehrgliedrigen Evolutionsstrategie von Schwefel

[52] Vgl. Born (1978).

[53] Vgl. dazu Schwefel (1977), S. 111ff.

[54] Auf genetische Algorithmen wird hier nicht näher eingegangen; vgl. dazu Holland (1975) oder Goldberg (1989).

[55] Vgl. dazu den vorangegangenen Abschnitt.

(1977) aus $k > 1$ Versuchspunkten $l > k$ neue Versuchspunkte generiert[56] und aus diesen k Punkte für die nächste Iteration ausgewählt[57].

Da diese Verfahren die Prinzipien der biologischen Evolution bewußt als Optimier-Regeln auffassen, wird bei der Formulierung der Algorithmen auch auf die evolutionsbiologische Terminologie zurückgegriffen[58]:

Ein ‚Individuum' I wird durch ein Tripel

$$I = (\underline{x}, \underline{\sigma}, \underline{\alpha}) \in R^n \times R_+^{n_s} \times [-\pi, \pi]^{n_p}$$

mit $1 \leq n_s \leq n$ und $n_p = \frac{1}{2}(2n - n_s)(n_s - 1)$ dargestellt[59]. Jede Komponente des sogenannten Objektvariablenvektors $\underline{x}$ und der sogenannten Strategievariablenvektoren $\underline{\sigma}$ und $\underline{\alpha}$ werden als ‚Phäne' bzw. ‚Gene' des Individuums aufgefaßt.

Eine ‚Population' $P^{(g)}$ zur ‚Generation' g ($\equiv$ Iteration) besteht aus einer Menge von $(k + l)$ Individuen:

$$P^{(g)} := \{I^{(1)}, I^{(2)}, \ldots, I^{(k-1)}, I^{(k)}, I^{(k+1)}, \ldots, I^{(k+l)}\} \ .$$

Dabei werden die Individuen $I^{(1)}$ bis $I^{(k)}$ als ‚Eltern' und die Individuen $I^{(k+1)}$ bis $I^{(k+l)}$ als ‚Nachkommen' bezeichnet.

Zur Bestimmung der Population der nächsten Generation $P^{(g+1)}$ werden zunächst k Individuen aus der vorangegangenen Population geeignet ausgewählt. Dazu sind folgende Vorgehensweisen denkbar:

- Wähle die k besten Individuen aus den $(k + l)$ Individuen der Population $P^{(g)}$ aus[60].
- Wähle die k besten Individuen aus den l Nachkommen der Population $P^{(g)}$ aus[61].

Dieser Vorgang wird ‚Selektion' genannt. Während bei der ersten Variante ein Elter theoretisch unbegrenzt „leben" könnte, wird bei der zweiten Variante die „Lebenszeit" rigoros begrenzt: Auf diese Weise erhält der Algorithmus einen Mechanismus, zuvor durchgeführte Schritte vergessen zu können. Die letztere Variante kommt dem Vorbild der Natur näher, und tatsächlich zeigten empirische Tests[62], daß durch das zweite Selektionsschema eine kontinuierlichere Anpassung

[56] Die Bedingung $l > k$ ist nur bei der sogenannten Komma-Strategie erforderlich, sonst genügt schon $l \geq 1$.

[57] Auch die $\geq n + 1$ Ecken des Polygons der Complex-Strategie könnte man bereits als eine ‚Population' bezeichnen.

[58] Vgl. Schwefel (1989), S. 180f.

[59] Wegen einer einfacheren Darstellung wird im folgenden davon ausgegangen, daß die Beziehung $n_s = n$ gilt.

[60] PLUS-Strategie: $(k + l)$.

[61] KOMMA-Strategie: (k, l).

[62] Vgl. Schwefel (1987).

der Strategieparameter und damit insgesamt eine schnellere Zielannäherung erzielt wird.

Nachdem nun die k Eltern der neuen Generation bekannt sind, müssen noch die l Nachkommen erzeugt werden. Bei einem sogenannten ‚haploiden Vererbungsschema' werden die Gene eines Elters zur Erzeugung eines Nachkommen verwendet. Häufig ist jedoch in der Natur ein ‚diploides Vererbungsschema' anzutreffen: Hier gehen die Gene zweier Eltern in die Erzeugung eines Nachkommen ein. Dabei werden ihre Gene geeignet gemischt oder umorganisiert und bilden dann die Gene des Nachkommen. Diesen Vorgang nennt man ‚Rekombination'. Hier sollen zwei Varianten vorgestellt werden:

$$I_{x_j}^{(k+i)} := \begin{cases} I_{x_j}^{(e_1)} \text{ oder } I_{x_j}^{(e_2)} & \text{, diskrete Rekombination} \\ \frac{1}{2}\left(I_{x_j}^{(e_1)} + I_{x_j}^{(e_2)}\right) & \text{, intermediäre Rekombination} \end{cases} \tag{3.22}$$

für $i = 1(1)l$ und $j = 1(1)n$, wobei $e_1,\ e_2\ \in \{1,\ \ldots,\ k\}$ ganzzahlig gleichverteilte Zufallsvariable und $I_{x_j}^{(k+i)}$ die j-te Komponente des Objektvariablenvektors $\underline{x}$ des (k+i)-ten Individuums, also des i-ten Nachkommen bezeichnet. Für die Strategievariablenvektoren $\underline{\sigma}$ und $\underline{\alpha}$ wird eine entsprechende Notation verwendet.

Anschließend werden die Gene des Individuums einer ‚Mutation' unterworfen: Das sind geringfügige Veränderungen der einzelnen Gene. In der Natur entstehen Mutationen durch sogenannte Kopierfehler bei der Vererbung - bei den Evolutionsstrategien werden sie über eine Normalverteilung modelliert:

$$I_{\underline{x}}^{(k+i)} := I_{\underline{x}}^{(k+i)} + \underline{z}^{(i)} \quad ,$$

wobei $\underline{z}^{(i)}$ einen normalverteilten Zufallsvektor mit der Dichte

$$p_{\underline{z}^{(i)}}(\underline{z}) = \left[(2\,\pi)^n\ \det(\underline{A}^{-1})\right]^{-\frac{1}{2}} e^{-\frac{1}{2}\underline{z}^T\,\underline{A}\,\underline{z}} \quad ,$$

also mit dem Erwartungsvektor $\underline{0}$ und der Kovararianzmatrix $\underline{A}^{-1}$ repräsentiert. Die Kovararianzmatrix $\underline{A}^{-1}$ kann auf verschiedene Arten gebildet werden: Möchte man auf die Kovarianzen, also die Nichthauptdiagonalelemente von $\underline{A}^{-1}$ verzichten[63], so ergibt sich

$$\underline{A}^{-1} :=\ \mathrm{diag}(\left[I_{\sigma_1}^{(k+i)}\right]^2, \ldots, \left[I_{\sigma_n}^{(k+i)}\right]^2) \tag{3.23}$$

als Diagonalmatrix aus dem Quadrat der Komponenten des Strategievariablenvektors $\underline{\sigma}$ eines Individuums. Durch $\underline{\sigma}$ wird also ein Streuungshyperellipsoid für die Schrittweiten eines Individuums beschrieben.

Auch die Strategievariablen werden der Mutation und der Rekombination unterworfen. Während für die Rekombination eine zu (3.22) analoge Vorschrift gilt,

[63] Werden auch die Kovarianzen berücksichtigt, so dreht sich der Streuungshyperellipsoid im Raum; vgl. Schwefel (1981), S. 239.

wird die Mutation auf eine andere Art durchgeführt. Für die Standardabweichungen $\underline{\sigma}$ der Schrittweiten wird die Vorschrift

$$I_{\sigma_j}^{k+i} := I_{\sigma_j}^{k+i}\, e^{s_0\, s_j}$$

verwendet, wobei s_0 und s_j normalverteilte Zufallsvariable mit dem Erwartungswert 0 und fest eingestellter Standardabweichung τ_0 bzw. τ_j sind.

Duch die Anwendung der Exponentialfunktion auf $s_0\, s_j$ werden die Standardabweichungen der Schrittweiten durch die Multiplikation mit einer *logarithmisch normalverteilten* Zufallsvariablen mutiert. Dadurch wird gewährleistet[64], daß

1) die Übernahme der Gene der Eltern am wahrscheinlichsten,

2) eine kleine Änderung häufiger als eine große und

3) die Wahrscheinlichkeit einer Schrittweitenverdoppelung gleich der Wahrscheinlichkeit einer Schrittweitenhalbierung ist.

Möchte man auch die Kovarianzen berücksichtigen und einer Mutation unterziehen[65], so muß man dafür sorgen, daß die Kovarianzmatrix $\underline{A}^{-1}$ positiv definit bleibt[66], wenn das Koordinatensystem orthogonal verbleiben soll. Da dies nicht garantiert werden kann, schlägt Schwefel (1981) vor, die Variationen (linear) zu korrelieren[67]. Eine Variation $\Delta\underline{x}$ ist identisch mit der Realisation des Zufallsvektors $I_{\underline{\sigma}}$:

$$I_{\underline{\sigma}^r} := I_{\Delta\underline{x}},$$

wobei das hochgestellte r die Realisation kennzeichnet.

Jedes Achsenpaar des Streuungshyperellipsoids wird durch einen sogenannten Lagewinkel $\alpha \in [-\pi\,, \pi]$ ausgerichtet. Dazu werden, falls $n_s = n$,

$$n_p = \frac{n\,(n-1)}{2}$$

Winkel benötigt. Formal läßt sich dieser Vorgang durch die Multiplikation des Variationsvektors $\underline{\sigma}^r$ eines Individuums mit n_p *Rotationsmatrizen*[68] beschreiben:

[64] Vgl. Schwefel (1977), S. 166.
[65] Für die Rekombination ergibt sich eine zu (3.22) analoge Vorschrift.
[66] Vgl. Schwefel (1981), S. 239.
[67] Vgl. Schwefel (1981), S. 240f.
[68] Vgl. etwa Faddejew und Faddejewa (1973), S. 38f.

Durch die Multiplikation eines Vektors $\underline{\sigma}^r$ mit einer Rotationsmatrix

$$T_{pq}(\alpha) = \begin{pmatrix} 1 & 0 & & & & \cdots & & & & 0 \\ 0 & 1 & & & & & & & & \\ & & \ddots & & & & & & & \\ & & & 1 & & & & & & \\ & & & & \cos\alpha & & & -\sin\alpha & & \\ & & & & & 1 & & & & \\ \vdots & & & & & & \ddots & & & \vdots \\ & & & & & & & 1 & & \\ & & & & \sin\alpha & & & \cos\alpha & & \\ & & & & & & & & 1 & \\ & & & & & & & & \ddots & 0 \\ 0 & & & & & \cdots & & & 0 & 1 \end{pmatrix}$$

wird der Vektor $\underline{\sigma}^r$ bzgl. der p-ten und q-ten Achse sowie des Winkels α einer Koordinatentransformation unterzogen, wenn die trigonometrischen Funktionen jeweils in der p-ten bzw. q-ten Spalte sowie in der p-ten bzw. q-ten Reihe der Matrix stehen. Angewandt auf jedes Achsenpaar ergibt sich dann der (linear) korrelierte Vektor einer Realisation der Standardabweichungen aus[69]

$$I_{\underline{\sigma}^r}^{(k+i)} := \left[\prod_{p=1}^{n-1} \prod_{q=p+1}^{n} T_{pq}\left(I_{\alpha_{w(p,q)}}^{(k+i)} \right) \right] \cdot I_{\underline{\sigma}^r}^{(k+i)}$$

für $i = 1(1)l$, wobei sich der Index $w(p,q)$ des Lagewinkelvektors α durch

$$w(p,q) = \frac{1}{2}(2n-p)(p+1) - 2n + q \tag{3.24}$$

errechnet. Formel (3.24) ist die speichereffiziente Abbildung einer oberen Dreiecksmatrix ohne Hauptdiagonale.

Nun sind die realisierten Schrittweiten, die Variationen $\Delta\underline{x}$ miteinander (linear) korreliert, und die Kovarianzmatrix $\underline{A}^{-1}$ ergibt sich wie zuvor aus (3.23). Anstelle von Kovarianzen werden nun die Lagewinkel mutiert:

$$I_{\alpha_j}^{(k+i)} := I_{\alpha_j}^{(k+i)} + a_j^{(i)} \quad \text{mit } i = 1(1)l \text{ , } j = 1(1)n_p$$

wobei $a_j^{(i)}$ eine normalverteilte Zufallsvariable mit Erwartungswert 0 und fest vorgegebener Standardabweichung τ ist[70]. Numerische Test ergaben als eine gute Wahl für τ einen Drehwinkel zwischen 5° und 10°.

Untersuchungen aus jüngerer Zeit zeigten, daß auch eine Verwendung einer sogenannten doppelten Weibull-Verteilung anstelle der Normalverteilung zu guten Ergebnissen führt[71]. Neben den bisher genannten Konzepten aus der biologi-

[69] Diese formal aufwendige Berechnungsvorschrift läßt sich wegen der schwachen Besetzung und speziellen Struktur der Rotationsmatrizen T_{pq} durch wenige Zeilen Code implementieren: vgl. dazu das FORTRAN-Listing in Schwefel (1980), S. 43.

[70] Gemäß einer Behauptung von Schneider (1986) müßten die Lagewinkel α_j von j unabhängig gemacht werden, was jedoch einer näheren Untersuchung bedarf.

[71] Vgl. Mück (1989).

schen Evolution wurde auch mit dem Prinzip der ‚Polyploidie' und der ‚Dominanz/Rezessivität' speziell für Probleme mit mehrfacher Zielsetzung experimentiert. Darauf soll hier jedoch nicht weiter eingegangen werden[72].

Durch die Nachahmung der verschiedenen Prinzipien der biologischen Evolution wird der Algorithmus befähigt, neben einer geeigneten Skalierung der Variablen auch eine geeignete Metrik adaptiv erlernen zu können. Auf diese Weise wird die Konvergenzgeschwindigkeit erhöht, was im allgemeinen dem Streben nach globaler Konvergenzsicherheit diametral gegenübersteht. Auf der anderen Seite wird das Wissen über die Topologie der Zielfunktion nicht auf ein Individuum konzentriert sondern über die gesamte Population verteilt. Deshalb bestehen gute Chancen, nicht vom nächstliegenden lokalen Minimum angezogen zu werden, wenn die Population initial weit genug im Parameterraum gestreut ist. Zusammenfassend kann man sagen, daß die vorgestellte Evolutionsstrategie gute nichtlokale Eigenschaften besitzt - eine globale Konvergenzsicherheit kann jedoch, insbesondere für $n >> 1$, nicht garantiert werden.

Um der Lösung des globalen Optimierungsproblems wenigstens näher zu kommen, muß offensichtlich der schmale Grat zwischer schneller (lokaler) Konvergenz und globaler Konvergenzsicherheit aufgefunden werden[73]. Ob dies durch die Aufnahme weiterer Konzepte der biologischen Evolution in den Algorithmus gelingen oder zumindest die globale Konvergenzsicherheit erhöht werden kann, wird Gegenstand der nächsten Kapitel sein. Allein die ‚Hypothese', daß die Evolution auch selbst einen solchen Kompromiß gefunden hat, begründet das Prinzip der Nachahmung von Evolutionsprinzipien. Wie sonst dürften wir uns als ‚Krone der Schöpfung' wähnen ?

[72] Vgl. Kursawe (1990).

[73] Das gilt nicht nur für Evolutionsstrategien.

Kapitel 4

Parallele Evolutionsstrategien

4.1 Vorbemerkungen

Eine Beschleunigung eines Optimierlaufes mit der mehrgliedrigen Evolutionsstrategie kann durch zweierlei Maßnahmen erzielt werden:

1. Parallelisierung der Evolutionsstrategie selbst.
2. Parallelisierung der Zielfunktion und der Nebenbedingungen.

Betrachtet man den Aufwand, den man zur Parallelisierung betreiben muß, dann ist die erste Möglichkeit vorzuziehen: Der Aufwand zur Parallelisierung der Evolutionsstrategie selbst ist einmaliger Natur, eine Parallelisierung der Zielfunktion und der Nebenbedingungen ist dagegen für jeden neuen Anwendungsfall erneut durchzuführen.

Im Abschnitt 4.2 wird zunächst erörtert, welche Möglichkeiten zur Parallelisierung der Evolutionsstrategie denkbar sind, bevor in Abschnitt 4.3 ein *paralleles Evolutionsmodell zur globalen Optimierung* entwickelt wird. Abschnitt 4.4 ist der Vorstellung der Implementierungdetails des Evolutionsmodells gewidmet.

4.2 Parallelisierungsmöglichkeiten

Bereits bei der Entwicklung der mehrgliedrigen Evolutionsstrategie wurde erkannt, daß sie sich durch eine hohe inhärente Parallelität auszeichnet: Die Generierung der Nachkommen und ihre anschließende Bewertung anhand der Zielfunktion kann innerhalb einer Generation unabhängig voneinander und damit parallel auf verschiedenen Prozessoren durchgeführt werden. Erst vor Beginn einer neuen Generation müßte der parallele Algorithmus synchronisiert werden. Für dieses Konzept wäre ein Multiprozessorsystem mit gemeinsamen Speicher geeignet, aus dem die Daten der Eltern parallel ausgelesen werden können (Abb. 4.1).

Wird jedoch die Kommunikation auf dem Multiprozessorsystem durch die Versendung von Nachrichten realisiert, müßte dazu die gesamte Elternpopulation

zu jeder Generation an jeden Prozessor verschickt werden. Um diesen Kommunikationsaufwand zu verringern, könnte man wie folgt vorgehen: Ein sogenannter Master-Prozeß sorgt für die Generierung der Nachkommen und verschickt sie an die einzelnen sogenannten Worker-Prozesse, die die Auswertung der Zielfunktion und die Überprüfung der Nebenbedingungen durchführen. Nach der Bewertung werden die Nachkommen wieder auf dem Master zusammengefaßt und der Selektion unterzogen. Dafür wäre eine sogenannte Transputer-Farm geeignet (Abb. 4.2).

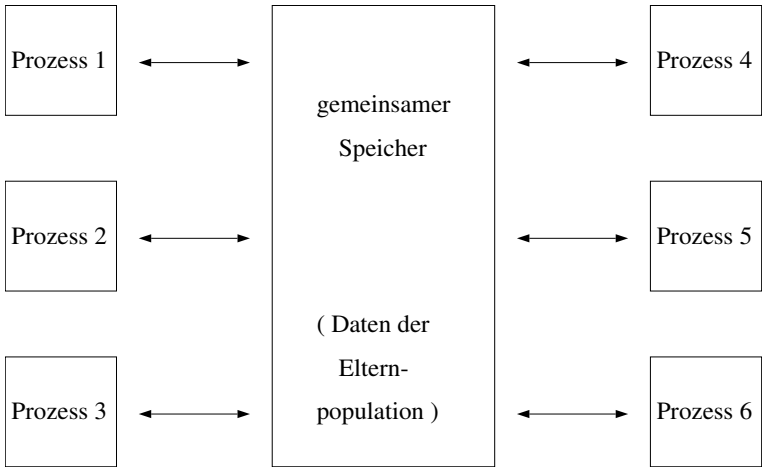

Abbildung 4.1: Parallele Evolutionsstrategie mit gemeinsamem Speicher

Diese Vorgehensweise dürfte insbesondere dann lohnenswert sein, wenn die Auswertung der Zielfunktion zeitaufwendig ist. Abbildung 4.3 stellt ein Zeitdiagramm für einen solchen Algorithmus mit $w = 5$ Workern[1] über zwei Generationen dar. Dabei bedeutet t_G die Zeit, die für die Generierung der Nachkommen und für ihre Versendung benötigt wird, und t_{ZF} die Zeit, die zur Auswertung der Zielfunktion und der Nebenbedingungen beansprucht wird. Die sogenannten Idle-Zeiten, in denen ein Prozessor beschäftigungslos ist, sind schraffiert eingezeichnet.

Im folgenden soll analysiert werden, wie stark die Beschleunigung bei diesem Ansatz ausfallen wird. Ein gebräuchliches Maß dafür ist der sogenannte *Speedup*:

$$S_P = \frac{T_1}{T_p} \quad . \tag{4.1}$$

Dabei bezeichnet T_1 die Zeit, die ein für einen seriellen Rechner konzipierter Algorithmus benötigt, während T_p die Zeit angibt, die ein paralleler Algorithmus für das gleiche Resultat[2] braucht, wenn er auf p Prozessoren läuft.

[1] Also etwa eine (1,5)-Strategie mit einem Elter und 5 Nachkommen

[2] Bei stochastischen Optimieralgorithmen ist diese Forderung kaum zu erfüllen. Deshalb wird hier nur ein qualitativ gleichwertiges Resultat gefordert, z.B. die gleiche Größenordnung der Zielannäherung.

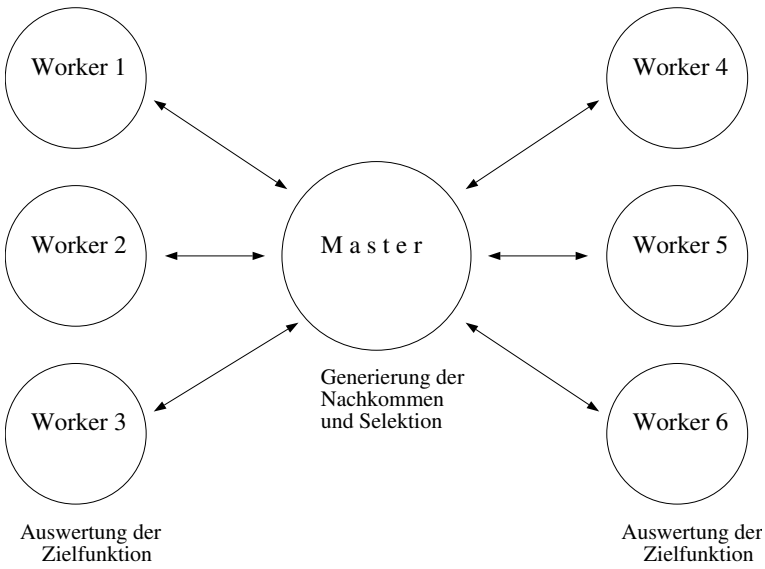

Abbildung 4.2: Parallele Evolutionsstrategie nach dem Master/Worker-Modell

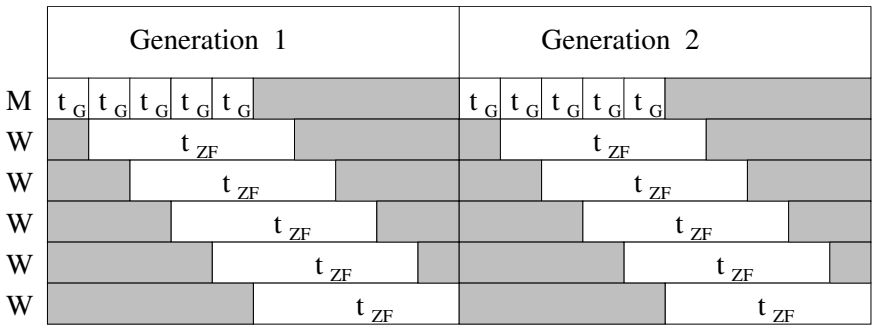

Abbildung 4.3: Zeitdiagramm für das synchrone Master/Worker-Konzept

Bei dem sequentiellen Algorithmus muß zunächst die Kommunikationszeit t_K von t_G abgezogen werden: $\hat{t}_G = t_G - t_K$. Dazu wird angenommen, daß die Kommunikation einen Anteil $\frac{1}{c}$ von t_G ausmacht ($c > 1$):

$$\hat{t}_G = t_G - t_K = t_G - \frac{1}{c}\, t_G = \frac{c-1}{c}\, t_G \quad . \tag{4.2}$$

Für jeden Nachkommen werden dann $(\hat{t}_G + t_{ZF})$ Zeiteinheiten benötigt. Je Generation wären das dann $w\,(\hat{t}_G + t_{ZF})$ Zeiteinheiten und für einen Lauf über g Generationen:

$$T_1 = g \cdot w\,(\hat{t}_G + t_{ZF}) \quad . \tag{4.3}$$

Zur Vereinfachung wird angenommen, daß die Auswertung der Zielfunktion ein Vielfaches mehr an Zeit beansprucht als die Generierung und Versendung eines

Nachkommen beim parallelen Algorithmus:

$$t_{ZF} = \beta \cdot t_G \ , \ \beta > 1 \ . \tag{4.4}$$

Nach Einsetzen von (4.2) und (4.4) in (4.3) ergibt sich:

$$\begin{aligned} T_1 &= g \cdot w \, (\hat{t}_G + t_{ZF}) \\ &= g \cdot w \, \left(\tfrac{c-1}{c} \, t_G + \beta \cdot t_G \right) \\ &= g \cdot w \cdot t_G \, \left(\tfrac{c-1}{c} + \beta \right) \end{aligned} \tag{4.5}$$

Wie aus Abb. 4.3 ersichtlich, werden beim parallelen Algorithmus je Generation $(w \cdot t_G + t_{ZF})$ Zeiteinheiten benötigt. Für einen Lauf über g Generationen auf $(w+1)$ Prozessoren erhält man somit:

$$T_{w+1} = g \, (w \cdot t_G + t_{ZF}) \quad . \tag{4.6}$$

Durch Einsetzen von (4.4) in (4.6) ergibt sich:

$$T_{w+1} = g \, (w \cdot t_G + t_{ZF}) = g \cdot t_G \, (w + \beta) \quad . \tag{4.7}$$

Nun läßt sich der Speedup (4.1) unter Verwendung von (4.5) und (4.7) berechnen:

$$S_{w+1} = \frac{T_1}{T_{w+1}} = \frac{g \cdot w \cdot t_G \, (\frac{c-1}{c} + \beta)}{g \cdot t_G \, (w + \beta)} = \frac{w \, (\frac{c-1}{c} + \beta)}{w + \beta} \quad .$$

Durch die Ersetzung $\beta = \gamma \cdot w$, $\gamma > 0$, folgt schließlich:

$$\begin{aligned} S_{w+1} &= \frac{w \, (\frac{c-1}{c} + \gamma \cdot w)}{w + \gamma \cdot w} = \frac{\frac{c-1}{c} + \gamma \cdot w}{1 + \gamma} \\ &= \frac{c-1}{c} \cdot \frac{1}{1+\gamma} + \frac{\gamma}{1+\gamma} w \ \longrightarrow \ w \ \text{ für } \gamma \to \infty \ . \end{aligned}$$

Im ersten Summanden stecken die Kommunikationskosten: Sie fallen jedoch kaum ins Gewicht, da für γ große Werte gefordert werden müssen, so daß dieser Term gegen Null strebt. Man erhält also für ein großes γ einen linearen Speedup von annähernd w, weil für ein großes $\gamma = \frac{\beta}{w} = \frac{t_{ZF}}{w \cdot t_G}$ der Masterprozeß entsprechend lange warten muß. Abbildung 4.4 diene der Veranschaulichung. Die sogenannte *Effizienz*

$$E_p = \frac{S_p}{p} = \frac{T_1}{p \cdot T_p}$$

des Algorithmus könnte dadurch verbessert werden, daß man den Algorithmus auf nur 5 Prozessoren fährt: Der Master generiert 4 Nachkommen und schickt sie an die Worker, woraufhin er einen fünften Nachkommen generiert und ihn selbst auswertet.

Alle diese Überlegungen werden allerdings hinfällig, wenn die Zeit t_{ZF} zur Auswertung der Zielfunktion nicht konstant ist: Das trifft z.B. dann zu, wenn hinter

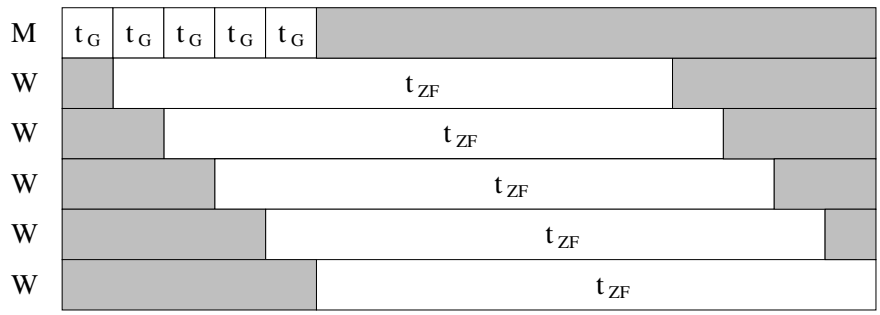

Abbildung 4.4: Zeitdiagramm für $t_{ZF} >> t_G$

der Zielfunktionauswertung eine Simulation steht, die in Abhängigkeit der Parameterwerte einen anderen Verlauf hat. In diesem Fall birgt die strenge Generationentaktung einen gravierenden Nachteil, da solange keine neue Generation begonnen werden kann, bis die am längsten dauernde Auswertung abgeschlossen ist. Um diesem Nachteil entgehen zu können, muß die synchrone Selektion zugunsten einer asynchronen Selektion aufgegeben werden.

Eine Möglichkeit besteht darin, ein Individuum direkt nach seiner Bewertung in die Elternpopulation einzubringen. Dabei kann es sinnvoll sein, als einen weiteren Parameter das „Alter“ eines Individuums einzuführen, der über die „Geschlechtsreife“ und den „Tod“ (Entfernen aus der Population) Auskunft geben könnte. Dadurch kommt es zu überlappenden Generationen[3].

Eine andere Möglichkeit besteht darin, die bisher zentrale Selektion zu dezentralisieren[4]. Dieser Ansatz führt unweigerlich zum Begriff der ‚Nachbarschaft‘ von Individuen: Neben der zeitlichen Komponente (Generationen) wird nun auch eine räumliche Komponente (Nachbarschaft) in das Modell eingebracht. Dazu stelle man sich z.B. vor, daß sich bei einer sogenannte Mesh-Topologie auf jedem Prozessor jeweils ein Individuum befindet. Durch die Prozessorverbindungen wird seine Nachbarschaft zu anderen Individuen definiert. Abbildung 4.5 diene der Illustration. Das Individuum I hat hier vier Nachbarn (N), die zu ihm in Konkurrenz stehen: Die Selektion wird hier also dezentral vollzogen.

Ähnliches läßt sich auch mit einem etwas grobkörnigeren Ansatz modellieren: Auf jedem Prozessor wird eine Teilpopulation plaziert, die durch den sequentiellen Algorithmus bewertet wird. Danach werden Individuen zwischen den Teilpopulationen ausgetauscht[5].

Der letztgenannte Ansatz soll in dieser Arbeit weiter verfolgt werden, da er leicht die Möglichkeit bietet, weitere Prizipien der biologischen Evolution nachzuah-

[3]Vgl. Hoffmeister und Schwefel (1988); Goldberg (1989), S. 209.

[4]Vgl. Gorges-Schleuter (1989); Manderick und Spiessens (1989).

[5]Ähnliche Ansätze finden sich bei Cohoon u.a. (1987), Pettey u.a. (1987), Tanese (1987, 1989) sowie Bormann (1989).

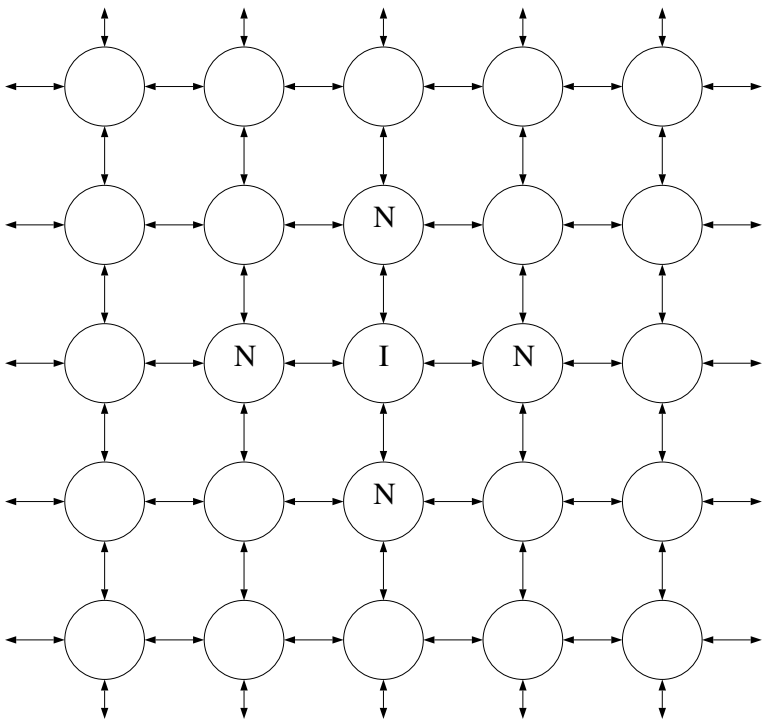

Abbildung 4.5: Asynchrone, dezentrale Selektion auf einer Mesh-Topologie

men. Dabei wird in erster Linie an die *globale Konvergenzsicherheit* und weniger an eine Beschleunigung des Verfahrens gedacht werden. Eine zentrale Frage wird also sein, ob durch paralleles Zusammenarbeiten die *Güte* des Verfahrens verbessert werden kann.

4.3 Ein Evolutionsmodell zur globalen Optimierung

Wie bereits erwähnt, steht die Erhöhung der globalen Konvergenzsicherheit bei der Entwicklung eines parallelen Evolutionsmodells im Mittelpunkt des Interesses. Im Unterabschnitt 4.3.1 werden zunächst die zentralen Ideen vorgestellt, bevor in Unterabschnitt 4.3.2 einige evolutionsbiologische Termini eingeführt werden. Auf dieser Grundlage schließlich wird in Unterabschnitt 4.3.3 das Modell formuliert.

4.3.1 Zentrale Ideen

Um die globale Konvergenzsicherheit zu erhöhen, kann man bei der sequentiellen Evolutionsstrategie nacheinander mehrere Läufe mit unterschiedlichen Startwerten und anderen Zufallsfolgen der Pseudozufallszahlengeneratoren durchführen. Auf einem Multi-Prozessorsystem könnten diese Läufe parallel ausgeführt wer-

den: Man erhält das gleiche Ergebnis (sowohl quantitativ als auch qualitativ) in kürzerer Zeit.

Wenn allerdings ein (Evolutions-) Prozeß vorzeitig zu einem lokalen Minimum konvergiert, trägt der ausführende Prozessor nichts mehr zur Suche bei. Deshalb wäre es sinnvoll, zwischen den (Evolutions-) Prozessen Informationen auszutauschen, damit die Exploration des Suchraumes nicht frühzeitig abgebrochen wird. Es ist jedoch (mathematisch gesehen) noch völlig offen,

- welche Art von Information und
- welcher Umfang an Information ausgetauscht werden soll, und
- wie soll oder kann diese Information genutzt werden ?

Da eine mathematische Durchdringung der Evolutionsstrategie erst in den Anfängen steht, ist eine Nachahmung weiterer Prinzipien der biologischen Evolution von experimenteller Natur. Solche Experimente können aber Hinweise darauf geben, welche Richtung eine theoretische Untersuchung einschlagen könnte.

4.3.2 Einführung evolutionsbiologischer Termini

Jede Trennung, die einen Genaustausch zwischen Populationen, den Genfluß, einer Art oder nahe verwandter Arten unmöglich macht, wird Isolation genannt. Dabei unterscheidet man zwischen reproduktiver und geographischer Isolation[6].

Bei der reproduktiven Isolation wird einer Fortpflanzung zwischen verschiedenen Populationen durch sogenannte Isolationsmechanismen entgegengewirkt, die entweder die Kreuzung als solche verhindern oder aber den Erfolg der Kreuzung vermindern. So kann es z.B. nicht zur Paarung kommen, wenn die Fortpflanzungsfähigkeit der potentiellen Paarungspartner zu verschiedenen Jahreszeiten besteht[7].

Von geographischer Isolation oder auch Separation spricht man, wenn der Genaustausch zwischen verschiedenen Populationen durch eine räumliche oder ökologische Trennung verhindert wird[8]: Beispielsweise könnte eine Population durch plötzlich auftretende geographische Barrieren (etwa: Ansteigen des Meeresspiegels) in Teilpopulationen aufgespalten werden. Jede dieser Teilpopulationen könnte sich nun anders entwickeln: Die Entwicklungslinie, die eingeschlagen wird, ist nun im hohen Maße von der Gesamtheit der Gene der Teilpopulation, ihrem sogenannten Genpool, abhängig. Auf diese Weise könnte jede Teilpopulation der Ausgangspunkt für die Entwicklung einer neuen Art sein.

[6] Vgl. Sedlag und Weinert (1987), S. 149.

[7] Weitere Beispiele finden sich bei Sedlag und Weinert (1987), S. 149, sowie bei Futuyma (1990), S. 126f.

[8] Vgl. Sedlag und Weinert (1987), S. 265.

Bestehen jedoch die geographischen Barrieren nicht auf Dauer, kann einer solchen Differenzierung durch die Wanderung (Migration) von Individuen zwischen den Teilpopulationen entgegengewirkt werden[9]. Andererseits kann durch Migration aber auch ein anderer Effekt entstehen: Waren nämlich die Teilpopulationen längere Zeit voneinander getrennt, so ist es möglich, daß die Variabilität des Genpools stark herabgesetzt ist, d.h., die genetische Entwicklung des Genpools ist zum Stillstand gekommen. Treffen nun Migranten aus Teilpopulationen, die eine andere Entwicklungslinie hatten, aufeinander, so wird die Variabilität des Genpools schlagartig wieder erhöht, und durch die Genrekombination von Individuen verschiedener Teilpopulationen können Nachkommen entstehen, die völlig anders geartet sind als ihre Eltern.

Diese soeben aufgeführten Prizipien der biologischen Evolution - *Separation und (periodische) Migration* - sollen im Evolutionsmodell zur globalen Optimierung nachgeahmt werden.

4.3.3 Formulierung des Modells

Auf jedem Prozessor wird je eine Teilpopulation gemäß der sequentiellen Evolutionsstrategie plaziert, die sich prizipiell unabhängig von den anderen Teilpopulationen entwickeln kann (Separation). Einer möglichen „Verarmung“ des Genpools, also dem vorzeitigen Konvergieren zu einem lokalen Minimum, soll durch Migration (Versendung von Individuen zwischen den Prozessen) entgegengewirkt werden. Zu untersuchen wären nun etwa folgende Fragen:

- Wieviele „Berührungszonen“ (Verbindungen) sollen zwischen den Teilpopulationen bestehen, also welche Topologie ist für das Transputernetzwerk vorzusehen ? Eine vollständige Vernetzung etwa würde die „regionale“ Information, die auf jedem Transputer vorhanden ist, zu schnell auslöschen. Hinweise darauf finden sich bei Baram (1989), der verschiedene Topologien von sogenannten *neuronalen Netzen* hinsichtlich ihrer Eignung als Assoziativ-Speicher untersucht. Eine zu geringe Konnektivität dagegen könnte der Konvergenzgeschwindigkeit des Gesamtverfahrens entgegenstehen. Bormann (1989) experimentierte bei einem ähnlichen Modell mit einem bidirektionalen Ring (2 Verbindungen), Gorges-Schleuter (1989) mit einer Helix-Struktur (3 Verbindungen), Manderick und Spiessens (1989) mit einem Torus (4 Verbindungen) und Tanese (1987, 1989) mit einem Hyperkubus. Baram (1989) arbeitet mit ‚selbstähnlichen‘ Topologien, wodurch es zusätzlich zu einer hierarchischen Strukturierung der Information kommt (Abb. 4.6).

- Wie häufig sollen Informationen ausgetauscht werden, also wie lange bleibt eine Teilpopulation isoliert ? Durch eine längere Isolation einer Teilpopulation kann es dort zu einer schnellen (lokalen) Konvergenz kommen. Das

[9] Vgl. Sedlag und Weinert (1987), S. 317.

könnte bei Problemen mit vielen lokalen Minima durchaus wünschenswert sein: Schließlich ist auch das globale Minimum ein lokales Minimum. Wird der Isolationszeitraum zu kurz gewählt, dann wäre es möglich, daß eine schnell aufgefundene, vermeintlich gute Information zu rasch im Netzwerk bekannt wird und so der Exploration des Suchraumes einengend gegenübersteht. Man kann erwarten, daß die Isolationsdauer in enger Beziehung zur gewählten Topologie steht.

- Welche Menge an Information und besonders welche Information soll zwischen den Prozessen ausgetauscht werden ? Zunächst möchte man meinen, daß das anhand der Zielfunktion bewertete „beste“ Individuum der ideale Informationsträger ist. Ist jedoch die Individuenanzahl einer Teilpopulation sehr hoch, dann kann die genetische Information des immigrierenden Individuums bei „ungünstiger Partnerwahl“ schnell verloren gehen. Immigriert dagegen eine Gruppe von „besten“ Individuen, dann ist die Wahrscheinlichkeit, daß die neue genetische Information Effekte in der Population hervorruft, wesentlich höher einzuschätzen.

Abschließend sollen hier die Annahmen aufgeführt werden, die dem Evolutionsmodell zugrunde liegen:

1. Die Anzahl der Teilpopulationen ist konstant.
2. Die Individuenzahl jeder Teilpopulation bleibt konstant.
3. Ein Individuum lebt nur eine Generation.
4. Die Rekombination wird wahlweise intermediär oder diskret durchgeführt.
5. Migration findet periodisch statt.
6. Die Anzahl der Migranten ist konstant.
7. Die Umwelt (bzw. die Zielfunktion) ist konstant.

4.4 Implementierungsdetails

4.4.1 Benutzte Fremdsoftware

Der Kern des parallelen Optimieralgorithmus besteht aus der mehrgliedrigen Evolutionsstrategie nach Schwefel (1977), die als Unterprogramm KORR in einer Fortran-Codierung am Lehrstuhl vorhanden ist[10]. Da der Source Code Debugger unter Helios nur die Sprache C unterstützt, wurde das Unterprogramm mit einem Fortran/C-Crosscompiler nach ANSI C konvertiert. Die Abschaltregel wurde dahingehend modifiziert, daß ein Rücksprung aus dem Unterprogramm durch den einzustellenden Parameter *Isolation* nach *Isolation* Generationen erfolgt, damit die Migranten ausgetauscht werden können.

[10]Vgl. Schwefel (1980).

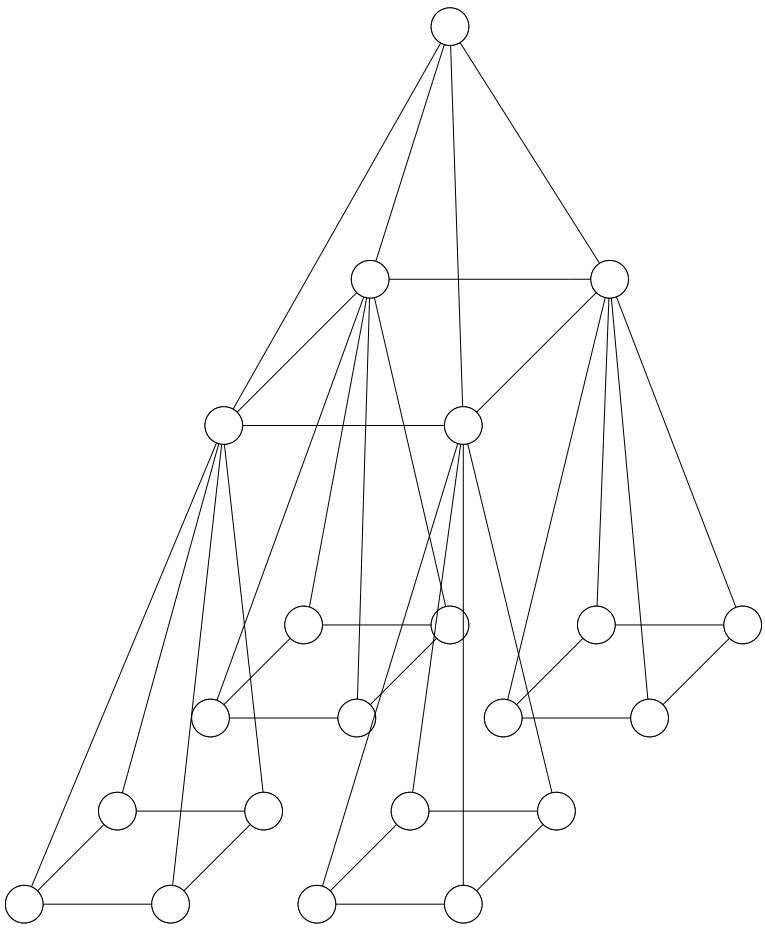

Abbildung 4.6: Selbstähnliches Netzwerk

4.4.2 Synchronisation und Kommunikation

Um den eigentlichen Optimieralgorithmus KORR wurde eine Schale entwickelt, die den Austausch der Individuen zwischen den Prozessoren/Populationen realisiert. Diese Schale wurde so ausgelegt, daß ein Wechsel der Topologie von einem bidirektionalen Ring über eine Helix bis zu einem Torus durch die Angabe eines Parameters software-technisch geschieht.

Um einem möglichen *Deadlock* vorzubeugen, wurden zur Kommunikation Pufferprozesse eingerichtet, die eine asynchrone Kommunikation simulieren. Für einen bidirektionalen Ring sind dazu je zwei Sender- und Empfangsprozesse notwendig. Diese Pufferprozesse wurden durch Threads realisiert: Sie laufen also quasiparallel auf dem gleichen Prozessor und kommunizieren mit dem eigentlichen Optimieralgorithmus über gemeinsamen Speicher und Semaphore.

Zusätzlich wird ein sogenannter Watchdog-Prozeß gestartet, dessen einzige Aufgabe darin besteht, einen weiteren Kanal hinsichtlich einer Terminierungsnachricht abzuhören. Solange diese Nachricht nicht eintrifft, wird dieser Prozeß vom

mikro-codierten Prozeß-Scheduler des Transputers suspendiert und belastet die CPU nicht. Trifft die Nachricht ein, wird eine globale Variable gesetzt, die durch einen Semaphor gesichert ist und die alle anderen Threads abfragen können.

Die Terminierungsnachricht selbst wird von einem globalen Kontrollprozeß (einer Task) verschickt, an den die Prozessoren/Populationsprozesse auch ihre Zwischenergebnisse senden und der die Zwischenergebnisse in Dateien archiviert. Abbildung 4.7 zeigt das Prozeßschaltbild für einen bidirektionalen Ring mit vier Populationsprozessen und dem Kontrollprozeß.

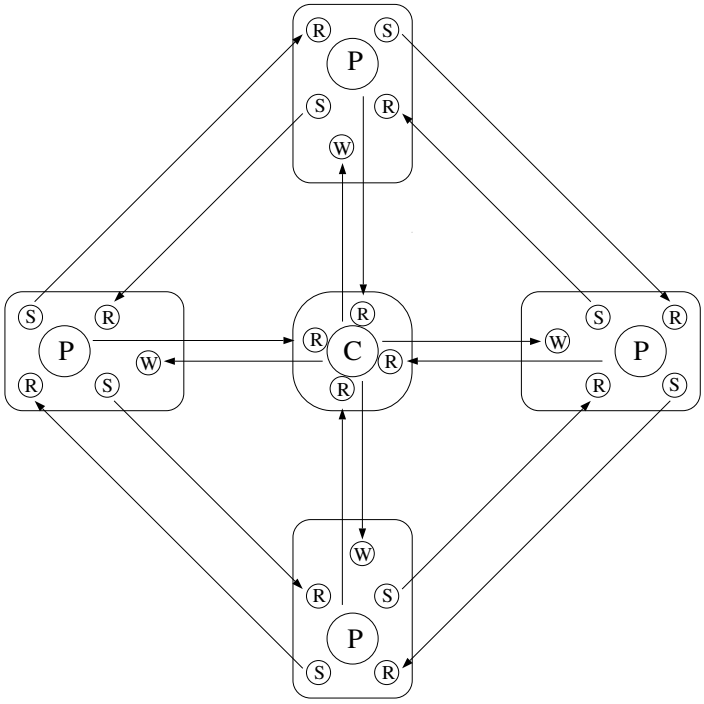

Abbildung 4.7: Prozeßschaltbild

Die Parameter der Evolutionsstrategie werden vom Kontrollprozeß beim Start aus einer Datei (`startup.file`) ausgelesen und an alle Populationsprozesse als Initialisierungspaket verschickt. Daraufhin allokieren die Populationsprozesse dynamischen Speicher und versuchen, ihre Threads auszusetzen. Anschließend

wird der Kontrollprozeß über den Erfolg oder Mißerfolg in Kenntnis gesetzt. Tritt mindestens ein Mißerfolg im Netzwerk auf, werden alle Prozesse davon unterrichtet, damit ein geordneter und sauberer Abbruch der Task Force möglich ist.

4.4.3 Parameter der Evolutionsstrategie

Da bei der Evolutionsstrategie viele Parameter einzustellen sind und eine Übergabe dieser Parameter über das Environment sehr umständlich bzw. fehlerträchtig wäre, werden die Parameter aus einer Datei ausgelesen. Im folgenden Beispiel für eine solche Datei ist zusätzlich angegeben, welche Parameter für die Testläufe konstant blieben.

```
# Parameter der Evolutionsstrategie je Subpopulation

> IELTER        =  30           # Anzahl der Eltern (konst.)
> BKOMMA        =   1           # Komma-Strategie (konst.)
> NACHKO        = 300           # Anzahl der Nachkommen (konst.)
> IREKOM        = 221           # diskrete Rekombination
> EPSILO[1]     =   1.0e-100    # Genauigkeiten (konst.)
> EPSILO[2]     =   1.0e-100    #
> EPSILO[3]     =   1.0e-100    #
> EPSILO[4]     =   1.0e-100    #
> DELTAS        =  2.236068e-01 # siehe Text
> DELTAI        =  3.976354e-01 # siehe Text
> N             =  10           # Dimension des Problems
> M             =  10           # Anzahl der Restriktionen
> NS            =  10           # Achsen des Streuungsellipsoids
> ISOLATION     =  10           # Isolationsdauer
> MIGRANTEN     =  1            # Anzahl der Migranten je Link
> MIGRATIONSTYP =  1            # nicht implementiert
> PROBLEM       =  5            # Problemnummer
> POPULATIONEN  =  24           # Anzahl der Populationen (konst.)
> LAUF-NR.      =  1            # Laufnummer zur Archivierung
> MAX. GEN      =  1600         # max. Anzahl der Generationen
```

Bei den Parametern `DELTAS` und `DELTAI` handelt es sich um Faktoren, die das Ausmaß der Schrittweitenänderung beschreiben. Gemäß der Theorie in Schwefel (1977), S. 168, sind diese Faktoren abhängig von der Dimension `N` des Problems:

$$\delta_s = \frac{\Phi^*}{\sqrt{2 \cdot n}} \,,$$

$$\delta_i = \frac{\Phi^*}{\sqrt{2 \cdot \sqrt{n}}} \quad \text{und}$$

wobei n die Dimension und Φ^* die zu erwartende Konvergenzrate angibt. Φ^* selbst ist abhängig von der Anzahl der Nachkommen und der Eltern und kann bei der vorliegenden Konstellation gleich Eins gewählt werden:

$$\Phi^* = 1.0 \ .$$

Der Startwert $\underline{x}^{(0)}$ für jede Teilpopulation wird zufällig aus dem Suchraum S

$$M \subseteq S := \{ \underline{x} \in R^n \,|\, \underline{a} \leq \underline{x} \leq \underline{b} \}$$

gewählt. Da die erste Population durch Realisationen eines normalverteilten Zufallsvektors erzeugt wird und für die Normalverteilung die Beziehung

$$P(|x - \mu| < 3\,\sigma) \approx 0.99$$

gilt, wird die anfängliche Schrittweite durch

$$s_i = \frac{b_i - a_i}{6}$$

festgesetzt, wodurch die erste Population weit im Suchraum verstreut wird.

Der Parameter `MIGRANTEN` legt die Anzahl der auszutauschenden Individuen je Nachbarpopulation fest: Bei einem bidirektionalen Ring werden also 2· `MIGRANTEN` Individuen verschickt. Dabei wird das „beste“ , „drittbeste“ etc. Individuum in die eine Richtung und das „zweitbeste“ , „viertbeste“ etc. in die andere Richtung verschickt. Entsprechend ersetzen die eintreffenden Individuen diese Migranten. Dieses Emmigrationsschema ist willkürlich - betrachtet man jedoch die Emmigrationsschemata ähnlich konzipierter Algorithmen, so bemerkt man, daß immer *Kopien* der besten Individuen (also *geklonte Individuen*) in alle Richtungen verschickt werden. Ein solches Schema kann aber in der Natur nicht beobachtet werden.

Kapitel 5

Test und Analyse

5.1 Vorbemerkungen

Da eine mathematische Theorie über die Konvergenzeigenschaften mehrgliedriger Evolutionsstrategien erst in den Anfängen steckt, können Aussagen bezüglich der Parameterwahl bisher nur durch empirische Tests gewonnen werden. Allerdings existiert bisher kein allgemein akzeptiertes Sortiment von Testproblemen für die globale Optimierung. Deshalb beschäftigt sich Abschnitt 5.2 mit der Auswahl von Testproblemen, bevor in Abschnitt 5.3 ein Testplan aufgestellt wird. In Abschnitt 5.4 schließlich werden die Testergebnisse zusammengestellt und graphisch aufbereitet.

5.2 Auswahl der Testfunktionen

Für das globale Optimierungsproblem existiert im Grunde genommen keine Sammlung typischer Testprobleme. Die Beispiele in Dixon und Szegö (1978) sind von geringer Dimension und für ein parallel arbeitendes Verfahren wie etwa die entwickelte Evolutionsstrategie viel zu einfach. Törn und Žilinskas (1989) entdeckten eine weitere Schwäche dieser Testprobleme, zu deren Darstellung einige Begriffe eingeführt werden müssen:

Definition 5.1:
Die Menge der Startpunkte $K_{ALG}(z^*) \subseteq M$, für die ein Optimieralgorithmus ALG zu einem lokalen Minimum $z^* := f(\underline{x}^*)$ konvergiert, heißt die Konvergenzmenge[1] des Verfahrens ALG für das lokale Minimum z^*.

□

Definition 5.2:
Die Konvergenzmenge des *Verfahrens des steilsten Abstiegs mit unendlich kleiner Schrittweite (SD^∞)*, das den Trajektorien der Differentialgleichungen

$$\dot{\underline{x}} = -\frac{\nabla f(\underline{x})}{1 + ||\nabla f(\underline{x})||}$$

[1] Vgl. Dixon u.a. (1975), S. 35.

folgt, heißt das Attraktionsgebiet[2] $A(z^*)$ des lokalen Minimums z^*:

$$K_{ALG}(z^*) \equiv A(z^*) \quad \Leftrightarrow \quad ALG = SD^{\infty}.$$

□

Angenommen, in M existieren k lokale Minima $z_1^*, \ldots, z_k^*$ mit

$$z_1^* < z_2^* < \ldots < z_k^*$$

und einem Lebesgue-Maß von

$$\mu\left[A(z_i^*)\right] > 0 \qquad \forall i = 1, \ldots, k \ .$$

Dann kann man

$$p_i = \frac{\mu\left[A(z_i^*)\right]}{\mu(M)}$$

als die Wahrscheinlichkeit dafür interpretieren, daß das Verfahren des steilsten Abstiegs zum lokalen Minimum z_i^* konvergiert, wenn der Startpunkt zufällig in M gewählt wird[3]. Für sämtliche Testprobleme in Dixon und Szegö (1978) berechneten Törn und Žilinskas (1989), daß das globale Minimum z_1^* das am leichtesten zu findende lokale Minimum ist:

$$p_1 = \max\{p_i | i = 1, \ldots, k\}.$$

Aufgrund dieser Tatsache ist es nicht verwunderlich, daß bereits die sequentielle Evolutionsstrategie häufig zum globalen Minimum dieser Testprobleme konvergiert. In den Tabellen 5.1 bis 5.3 werden die Resultate einer Testserie über 100 Generationen für drei Testfunktionen der sogenannten Shekel-Familie zusammengefaßt. Dabei bezeichnet ein *Lauf*, daß auf jedem der 24 Prozessoren eine sequentielle (10,100)-Evolutionsstrategie abgearbeitet wurde. In der ersten Zeile der Tabellen steht jeweils die globale Minimalstelle.

$\approx \underline{x}^*$	Lauf 1	Lauf 2	Lauf 3	Lauf 4	Lauf 5	Summe
(4.0, 4.0, 4.0, 4.0)'	11	16	12	17	11	67
(1.0, 1.0, 1.0, 1.0)'	2	1	1	0	1	5
(8.0, 8.0, 8.0, 8.0)'	0	0	0	0	1	1
(6.0, 6.0, 6.0, 6.0)'	8	4	8	7	8	35
(3.0, 7.0, 3.0, 7.0)'	3	3	3	0	3	12

Tabelle 5.1: Testergebnisse für die Testfunktion Shekel 5

Unter den gleichen Rahmenbedingungen wurde auch eine Testreihe für zwei Testprobleme nach Hartmann durchgeführt. Die Ergebnisse sind in der Tabelle 5.4 zusammengefaßt.

[2]Vgl. Dixon u.a. (1975), S. 36; Törn und Žilinskas (1989), S. 8.
[3]Vgl. Törn und Žilinskas (1989), S. 9.

$\approx \underline{x}^*$	Lauf 1	Lauf 2	Lauf 3	Lauf 4	Lauf 5	Summe
(4.0, 4.0, 4.0, 4.0)'	15	14	17	20	17	83
(1.0, 1.0, 1.0, 1.0)'	3	2	0	0	0	5
(8.0, 8.0, 8.0, 8.0)'	2	0	0	0	0	2
(6.0, 6.0, 6.0, 6.0)'	2	2	1	2	4	11
(3.0, 7.0, 3.0, 7.0)'	2	4	3	2	1	12
(2.0, 9.0, 2.0, 9.0)'	0	0	1	0	1	2
(5.0, 5.0, 3.0, 3.0)'	0	2	2	0	1	5

Tabelle 5.2: Testergebnisse für die Testfunktion Shekel 7

$\approx \underline{x}^*$	Lauf 1	Lauf 2	Lauf 3	Lauf 4	Lauf 5	Summe
(4.0, 4.0, 4.0, 4.0)'	16	16	10	14	14	70
(1.0, 1.0, 1.0, 1.0)'	0	1	1	1	0	3
(8.0, 8.0, 8.0, 8.0)'	0	0	1	1	0	2
(6.0, 6.0, 6.0, 6.0)'	3	0	4	1	3	11
(3.0, 7.0, 3.0, 7.0)'	2	3	2	0	5	12
(2.0, 9.0, 2.0, 9.0)'	0	0	0	0	0	0
(5.0, 5.0, 3.0, 3.0)'	1	0	1	1	1	4
(8.0, 1.0, 8.0, 1.0)'	0	0	0	0	0	0
(6.0, 2.0, 6.0, 2.0)'	1	2	2	2	1	8
(7.0, 3.6, 7.0, 3.6)'	1	2	3	4	0	10

Tabelle 5.3: Testergebnisse für die Testfunktion Shekel 10

In der Tabelle 5.5 sind neben den relativen Größen der Attraktionsgebiete der globalen Minima für die fünf Testprobleme auch die globalen Konvergenzhäufigkeiten der sequentiellen Evolutionsstrategie (ES) und die theoretischen Konvergenzhäufigkeiten des Verfahrens des steilsten Abstiegs (SD^{∞}) mit Zufallsstart eingetragen. Diese Ergebnisse zeigen deutlich die guten nicht-lokalen Eigenschaften der Evolutionsstrategie auf.

Wenn aber bereits die sequentielle Evolutionsstrategie mit zufälligen Startwerten für den Zufallszahlengenerator und für den Objektvariablenvektor bei diesen Testproblemen häufig zum globalen Minimum konvergiert, dann kann man erwarten, daß eine parallele Version mit 24 Teilpopulationen, die ihre Ergebnisse untereinander austauschen, sicher zum globalen Minimum konvergiert.

Die Testprobleme für die parallele Evolutionsstrategie sollten also einen höheren Schwierigkeitsgrad haben als die Probleme in Dixon und Szegö (1978). Gefordert werden sollte, daß für die Dimension des Problems mindestens $n \geq 10$ gilt und daß viele lokale Minima im Suchgebiet vorhanden sind.

Dazu eignet sich etwa eine Verallgemeinerung einer Testfunktion nach Rastrigin[4] und ein Testproblem nach Fletcher und Powell[5]. Neben diesen multimodalen

[4]Die ursprüngliche Version findet man bei Törn und Žilinskas (1989), S. 185.
[5]Vgl. dazu Schwefel (1977), S. 327f.

Name	Minima	Lauf 1	Lauf 2	Lauf 3	Lauf 4	Lauf 5	Summe
Hartmann 3	$\underline{x}_G^*$	24	24	23	24	24	119
	$\underline{x}_L^*$	0	0	1	0	0	1
Hartmann 6	$\underline{x}_G^*$	15	15	19	18	15	82
	$\underline{x}_L^*$	9	9	5	6	9	38

Tabelle 5.4: Testergebnisse für die Testfunktionen Hartmann 3 und 6

Testfunktion	$\frac{\mu(A(z_1^*))}{\mu(M)}$	SD^∞	ES
Shekel 5	0.30	36	67
Shekel 7	0.30	36	83
Shekel 10	0.20	24	70
Hartmann 3	0.70	84	119
Hartmann 6	0.65	78	82

Tabelle 5.5: Relative Größe der Attraktraktionsgebiete des globalen Minimums

Testfunktionen wird zu Vergleichszwecken auch ein unimodales Problem nach Schwefel (1977), S. 319, in den Testkatalog aufgenommen. Nachfolgend werden die Testprobleme vorgestellt.

Problem 1 : (Kugelmodell)

$$f(\underline{x}) = \sum_{i=1}^{n} x_i^2 \quad \text{mit } n = 30$$

Minimum : $\underline{x}^* = \underline{0}$, $f(\underline{x}^*) = 0$.

Hier handelt es sich um ein unimodales (streng konvexes) Problem: Die einzige lokale Minimalstelle ist zugleich die globale Minimalstelle. Abbildung 5.1 zeigt den mit einem Höhenlinienbild unterlegten Funktionsgraph von Testfunktion 1 für $n = 2$.

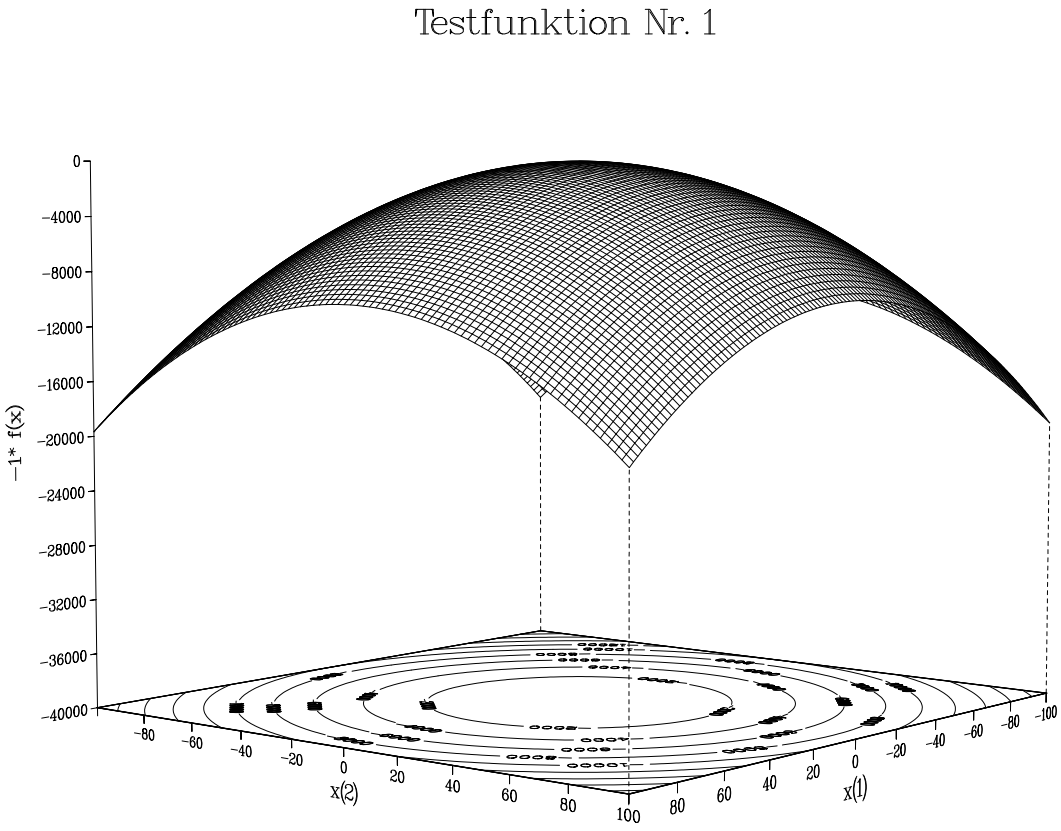

Abbildung 5.1: Funktionsgraph von Testproblem 1

Problem 2 (3, 4) : (nach Rastrigin)

$$f(\underline{x}) = \sum_{i=1}^{n} \left\{ x_i^2 + A\,[1 - \cos(\omega\, x_i)] \right\} \quad \text{mit } n = 10,\ 20,\ 30$$

globales Minimum: $\underline{x}^* = \underline{0}$, $f(\underline{x}^*) = 0$.

Bei diesem multimodalen Problem wird Problem 1 mit einer Störung überlagert. Das Ausmaß der Störung kann durch geeignete Wahl der Parameter A und ω manipuliert werden: Durch den Parameter A kann die Amplitude der Schwingung, die Problem 1 überlagert, eingestellt werden. Mit dem Parameter ω kann man die Frequenz der Schwingung festlegen. Wählt man $A = 0$, so geht dieses Problem in Problem 1 über. Abbildung 5.2 zeigt den Funktionsgraph von Problem 2 (3, 4) für $n = 2$ mit den Parametern $A = 50$ und $\omega = 2\,\pi$. Durch diese Parameterwahl wird gewährleistet, daß die lokalen Minimalstellen nichtnegative ganzzahlige Koordinatenkomponenten besitzen: $x_i^* \in N$. Für die Anzahl der lokalen Minimalstellen gilt ($a = 1000$):

$$M := \left\{ \underline{x} \in R^n : |x_i| \leq a + \frac{1}{2} \ , a \in N \right\} \ \Rightarrow \ (2\,a + 1)^n \text{ lokale Minimalstellen.}$$

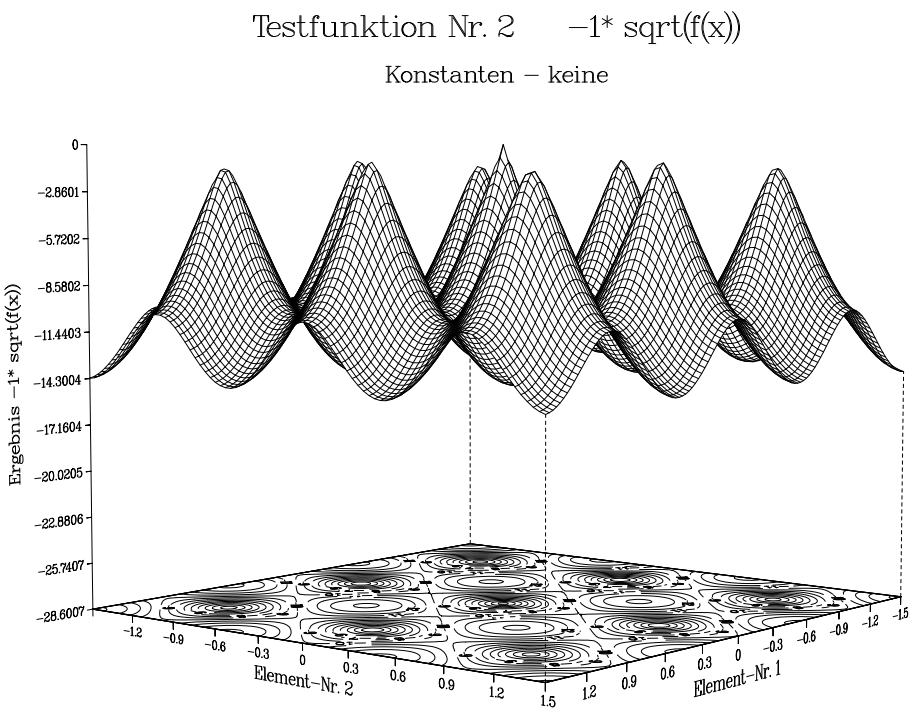

Abbildung 5.2: Funktionsgraph von Testproblem 2, 3 und 4

Problem 5 : (nach Fletcher/Powell)

$$f(\underline{x}) = \sum_{i=1}^{n} [A_i(\underline{x}) - B_i(\underline{x})]^2 \quad \text{mit } n = 10 \text{ und}$$

$$\begin{aligned} A_i(\underline{x}) &= \sum_{j=1}^{n} [a_{ij} \sin(\alpha_j) + b_{ij} \cos(\alpha_j)] \\ B_i(\underline{x}) &= \sum_{j=1}^{n} [a_{ij} \sin(x_j) + b_{ij} \cos(x_j)] \end{aligned}$$

wobei a_{ij} und b_{ij} ganzzahlig gleichverteilte Zufallszahlen aus dem Intervall [-100; 100] $\subset N$ sind. Für $\underline{\alpha}$ wurde $\alpha_i = 0 = const.$ für alle i gewählt.

globales Minimum: $\underline{x}^* = \underline{0}$, $f(\underline{x}^*) = 0$.

Im Bereich $x_i \in [-\pi,\ \pi]$ existieren bis zu 2^n lokale Minimalstellen. Abbildung 5.3 zeigt den Funktionsgraph für das Testproblem 5.

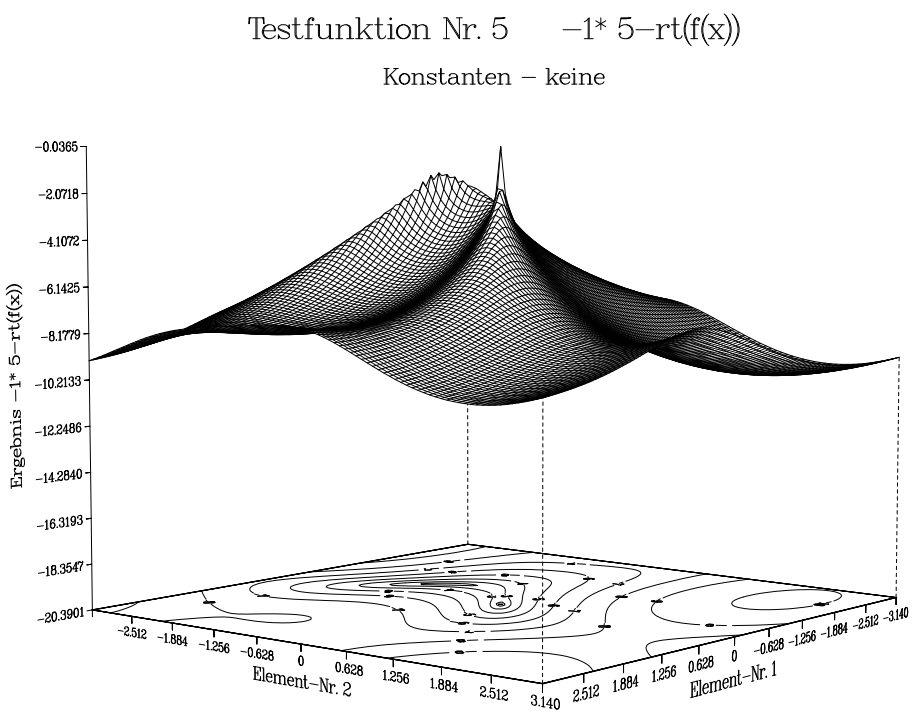

Abbildung 5.3: Funktionsgraph von Testproblem 5

5.3 Testplan

5.3.1 Konvergenzgeschwindigkeit

Zunächst soll getestet werden, wie sich die verschiedenen Varianten der parallelen Evolutionsstrategie bezüglich ihrer Konvergenzgeschwindigkeit verhalten. Dazu werden alle Varianten auf das unimodale Problem 1 angesetzt. Um den Beobachtungszeitraum möglichst groß zu halten, wird für die Startwerte $x_i^{(0)} \approx \pm 10^{50}$ gewählt. Das Performanzmaß $Q(g)$ wird wie folgt festgelegt:

$$Q(g) = \min\{z_p^*(g) | p = 1, \ldots, 24\} \quad ,$$

wobei $z_p^*(g)$ den besten Zielfunktionswert der Population p zur Generation g bezeichnet.

Zu untersuchen wären nun folgende Fragen:

- Welche Rekombinationsart (diskret/intermediär) führt zu schnellerer Konvergenz ?
- Wie wirkt sich die periodische Migration auf die Konvergenzgeschwindigkeit aus ?
- Stehen Rekombinationsart und Migration in einem Zusammenhang ?

5.3.2 Konvergenzsicherheit

Beim Test auf die globale Konvergenzsicherheit, werden alle Varianten auf die Probleme 2 bis 5 angesetzt. Dabei sind die Probleme 2 bis 4 zwar strukturell gleich, jedoch wird die Dimensionalität des Problems von 10 über 20 auf 30 gesteigert.

Bei dieser Testreihe sollen folgende Fragen untersucht werden:

- Wie „gut“ ist bereits die sequentielle Evolutionsstrategie ?
- Welche Rekombinationsart ist bzgl. globaler Konvergenz geeigneter ?
- Was nutzt die periodische Migration ?
- Gibt es einen Zusammenhang zwischen der Migrationsperiode und der Anzahl der migrierenden Individuen ?
- Stehen Rekombinationsart und Migration in einem Zusammenhang ?

5.3.3 Varianten der parallelen Evolutionsstrategie

Die verschiedenen Varianten der parallelen Evolutionsstrategie unterscheiden sich durch folgende Parameter:

- Migrationsperiode (MP) : 0, 1, 10 Generation(en)
- Anzahl der Migranten (Mig) : 1, 5 Migrant(en) je Nachbarpopulation
- Rekombinationsart: Intermediär (I) oder diskret (D)

Dabei sind die Kombinationen mit MP = 0 und Mig = 1 bzw. 5 nicht sinnvoll, so daß sich insgesamt 10 Varianten ergeben. Tabelle 5.6 gibt einen Überblick:

MP	Mig	Rek.	Variante
0	0	I	0
		D	1
1	1	I	2
		D	3
	5	I	4
		D	5
10	1	I	6
		D	7
	5	I	8
		D	9

Tabelle 5.6: Varianten der parallelen Evolutionsstrategie

Da jede Variante fünfmal auf die fünf Probleme angesetzt wird, müssen 250 Läufe gefahren werden. Um die Fülle der anfallenden Zwischenergebnisse begrenzen zu können, wird das Performanzmaß dahingehend modifiziert, daß $Q(g)$ über die 5 Läufe gemittelt wird:

$$\bar{Q}(g) = \frac{1}{L}\sum_{i=1}^{L} Q_i(g) \quad \text{mit } L = 5.$$

5.4 Testergebnisse

5.4.1 Zur Konvergenzgeschwindigkeit

In der Abbildung 5.4 ist der Konvergenzverlauf aller Varianten für das unimodale Problem 1 logarithmisch über 2000 Generationen aufgetragen.

Hier zeigt sich, daß die Varianten mit intermediärer Rekombination eine schnellere und gleichmäßigere Konvergenz aufweisen als die Varianten mit diskreter Rekombination. Dieses Phänomen kann man gut nachvollziehen, wenn man bedenkt, daß durch die intermediäre Rekombination alle Vektorkomponenten

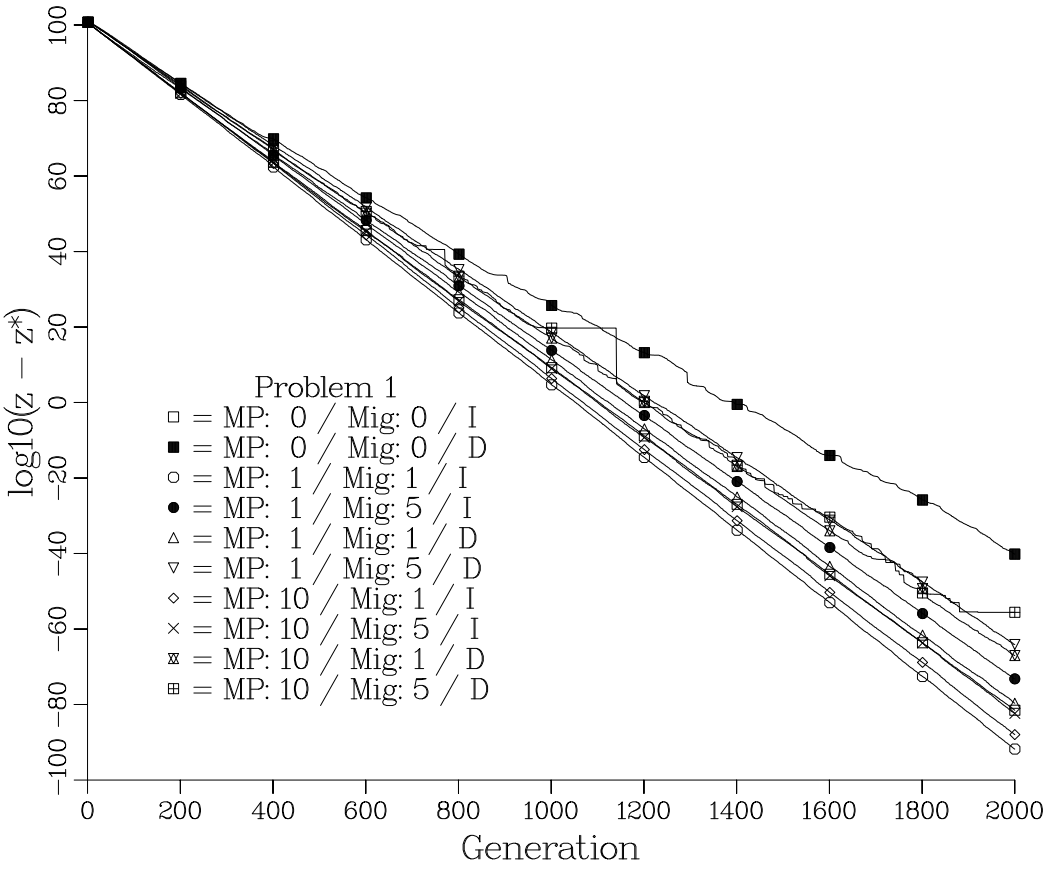

Abbildung 5.4: Konvergenzverlauf aller Varianten für Problem 1

allmählich in die Gradientenrichtung eingemittelt werden. Abbildung 5.5 zeigt neben der Zielannäherung auch die gemittelte beste aktuelle Schrittweite der Variante 2 (MP: 1 / Mig: 1 / I). Daraus läßt sich nun die Konvergenzgeschwindigkeit bestimmen:

Definition 5.3:
Sei (ε_k) eine Folge nichtnegativer Zahlen mit $\varepsilon_k \to 0$ für $k \to \infty$. Dann besitzt die Folge (ε_k) lineare R-Konvergenz oder geometrische Konvergenz, wenn ein Index k_0, eine Zahl $C > 0$ und eine Zahl $r \in [0, 1)$ existieren, so daß gilt[6]:

$$\varepsilon_k \leq C \cdot r^k \quad \forall k > k_0$$

□

[6]Vgl. Göpfert u.a. (1986), S. 108.

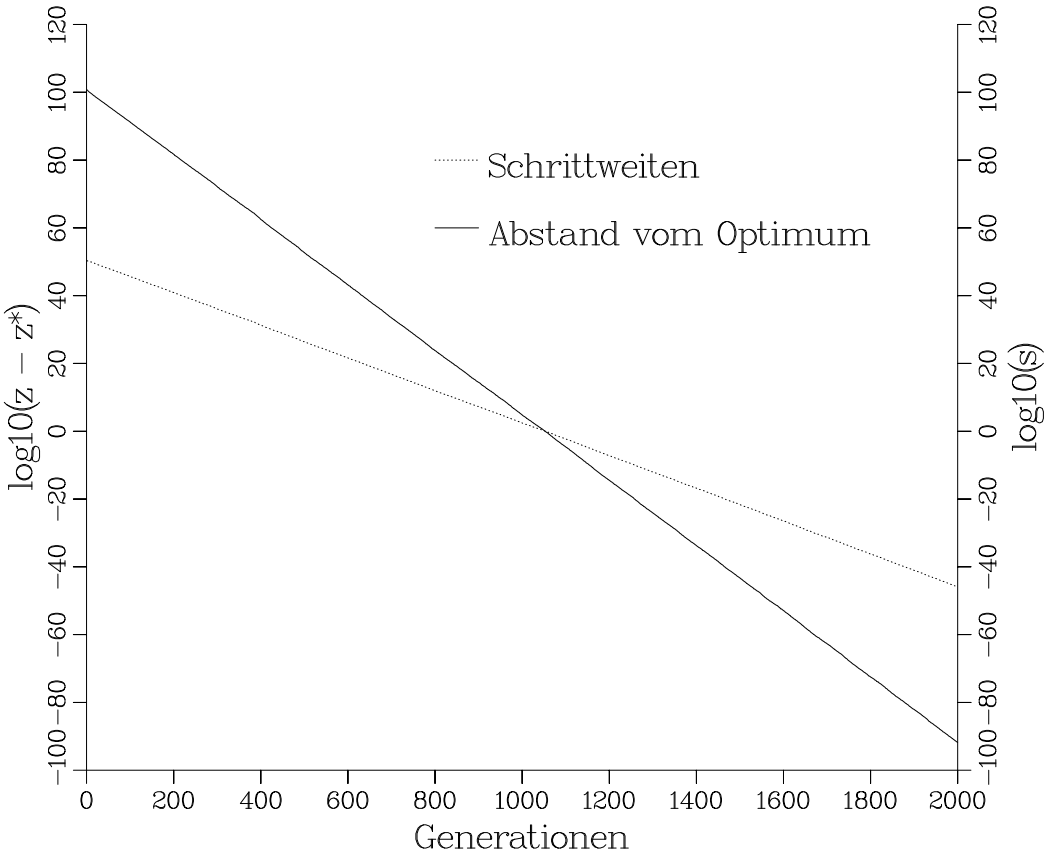

Abbildung 5.5: Schrittweiten von Variante 2 bei Testproblem 1

Für den Zielfunktionswert läßt sich die Beziehung

$$z^*(k) = 10^{-\frac{19}{200}k+100} \tag{5.1}$$

ableiten, wobei k die Generation bezeichnet. Zieht man aus (5.1) die Wurzel, so gibt dieser Wert gerade den Abstand von der globalen Minimalstelle durch die euklidische Norm an:

$$\varepsilon_k := || \underline{x}_k - \underline{x}^* || = 10^{-\frac{19}{400}k+50} \tag{5.2}$$

Wählt man nun $C = 10^{50}$, dann ergibt sich für $k_0 = 1$ und für

$$r = 10^{-\frac{19}{400}} \approx 0.8964 \in [\,0,\,1) \quad .$$

Die parallele Evolutionsstrategie mit Migration und intermediärer Rekombination zeigt also eine *geometrische Konvergenz*.

Ohne Migration zeigt die Evolutionsstrategie mit intermediärer Rekombination (MP: 0 / Mig: 0 / I) hier vergleichbare Werte, während ihr diskreter Pendant am schlechtesten abschneidet.

Insgesamt läßt sich also sagen, daß die Konvergenzgeschwindigkeit weniger von der Migration als von der Rekombinationsart abhängig ist: Die intermediäre Rekombination sorgt für eine schnellere Konvergenz.

5.4.2 Zur Konvergenzsicherheit

Zur globalen Konvergenzsicherheit sollen zuerst die Probleme 2 bis 4 untersucht werden. Diese Probleme sind strukturell gleich, jedoch wird die Dimension von 10 über 20 auf 30 erhöht. Die Abbildungen 5.6, 5.7 und 5.8 zeigen, daß der Konvergenzverlauf aller Varianten für diese Probleme tendenziell gleich ist.

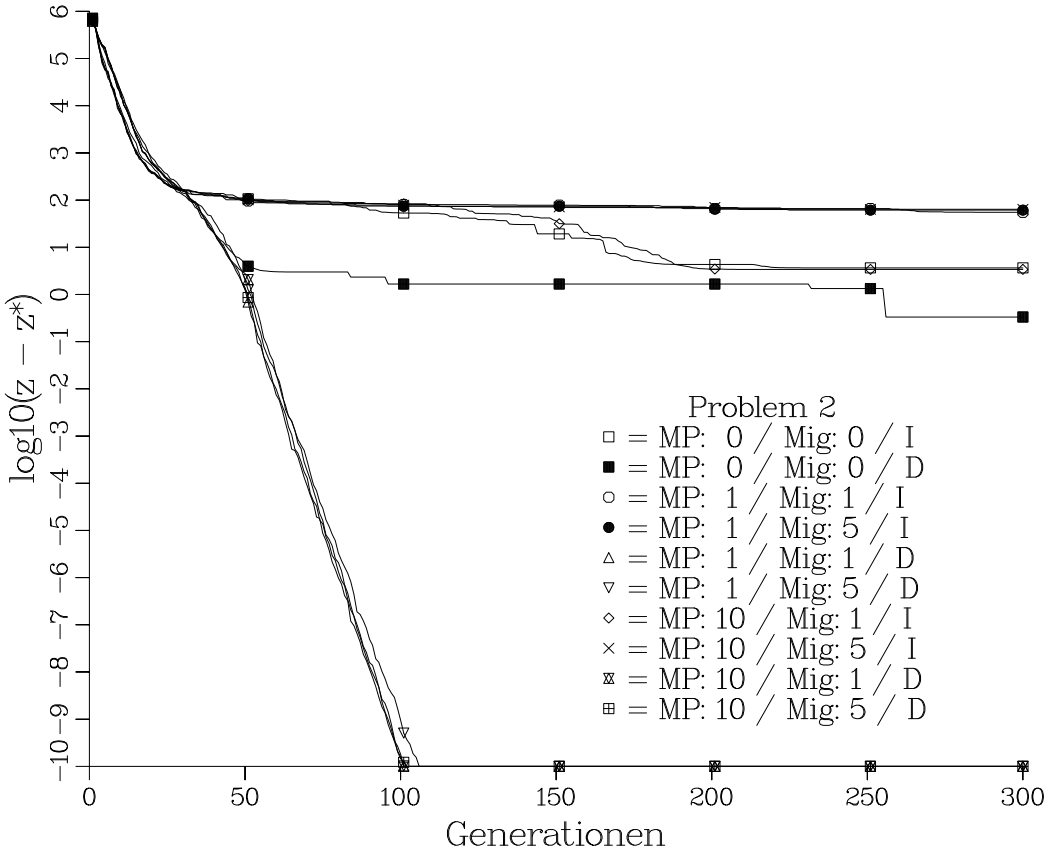

Abbildung 5.6: Konvergenzverlauf aller Varianten für Problem 2

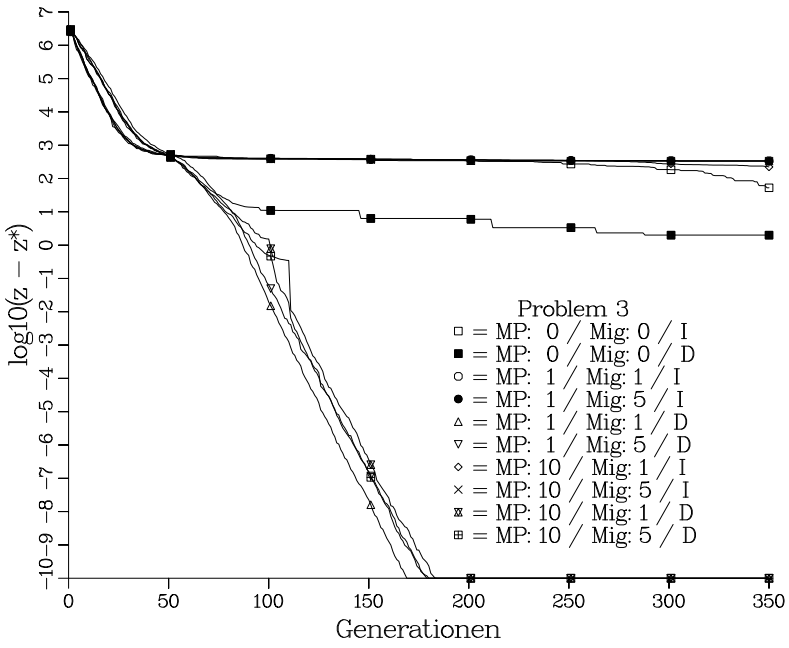

Abbildung 5.7: Konvergenzverlauf aller Varianten für Problem 3

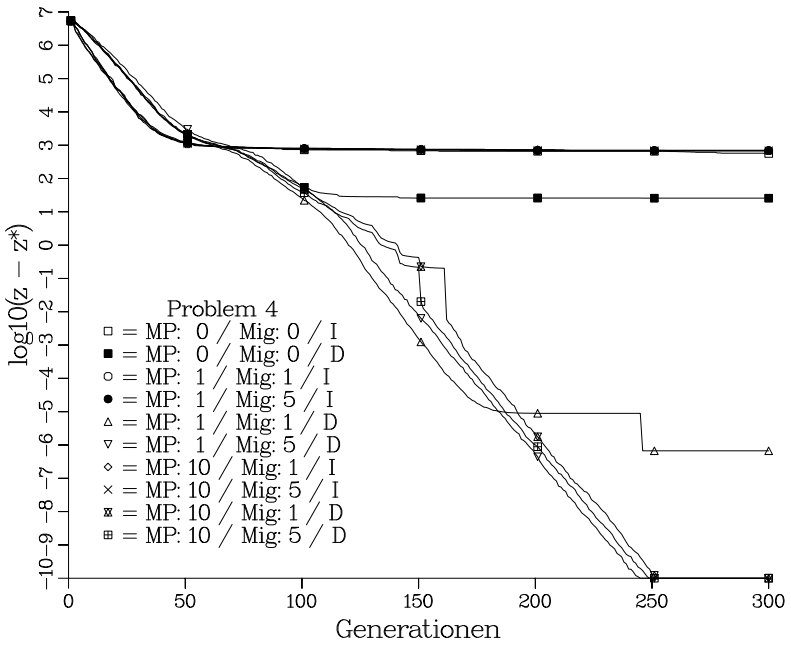

Abbildung 5.8: Konvergenzverlauf aller Varianten für Problem 4

Man sieht, daß alle Varianten mit intermediärer Rekombination vorzeitig zu einem der vielen lokalen Minima konvergieren, während die diskreten Varianten mit Migration das *globale Minimum* finden können. Dagegen ist die diskrete Variante 1 (MP: 0 / Mig: 0 / D), bei der die Teilpopulationen keine Individuen austauschen, nicht in der Lage, das globale Minimum zu bestimmen.

Betrachtet man für Problem 2 die Schrittweiten der Variante 6 (MP: 10 / Mig: 1 / I), dann erkennt man an der schnellen Verkleinerung der Schrittweiten ab der 170. Generation, daß hier das lokale Minimum approximiert wird (Abb. 5.9). Auch die eintreffenden Migranten aus anderen Populationen mit zum Teil größeren Schrittweiten können jetzt keine Verbesserung mehr erbringen.

Bei der diskrete Variante 8 (MP: 10 / Mig: 1 / D) bringt das alle 10 Generationen eintreffende Individuum deutliche Verbesserungen für den gesamten Konvergenzverlauf: Man kann davon ausgehen, daß ab der 50. Generation zumindest eine Teilpopulation im Attraktionsgebiet des globalen Minimums liegt, welches während der nächsten 50 Generationen nun immer genauer approximiert wird. Abbildung 5.10 zeigt den Konvergenzverlauf und die mittlere Schrittweite des besten Individuums der Variante 8 bei Problem 2.

Der Erfolg der diskreten Varianten könnte durch die spezielle Topologie der Zielfunktion bei Problem 2 (3, 4) erklärt werden: Die Parameter A und ω sind so eingestellt worden, daß sämtliche Ortsvektoren der lokalen Minimalstellen ganzzahlige Werte annehmen. Angenommen, eine Teilpopulation habe sich bereits

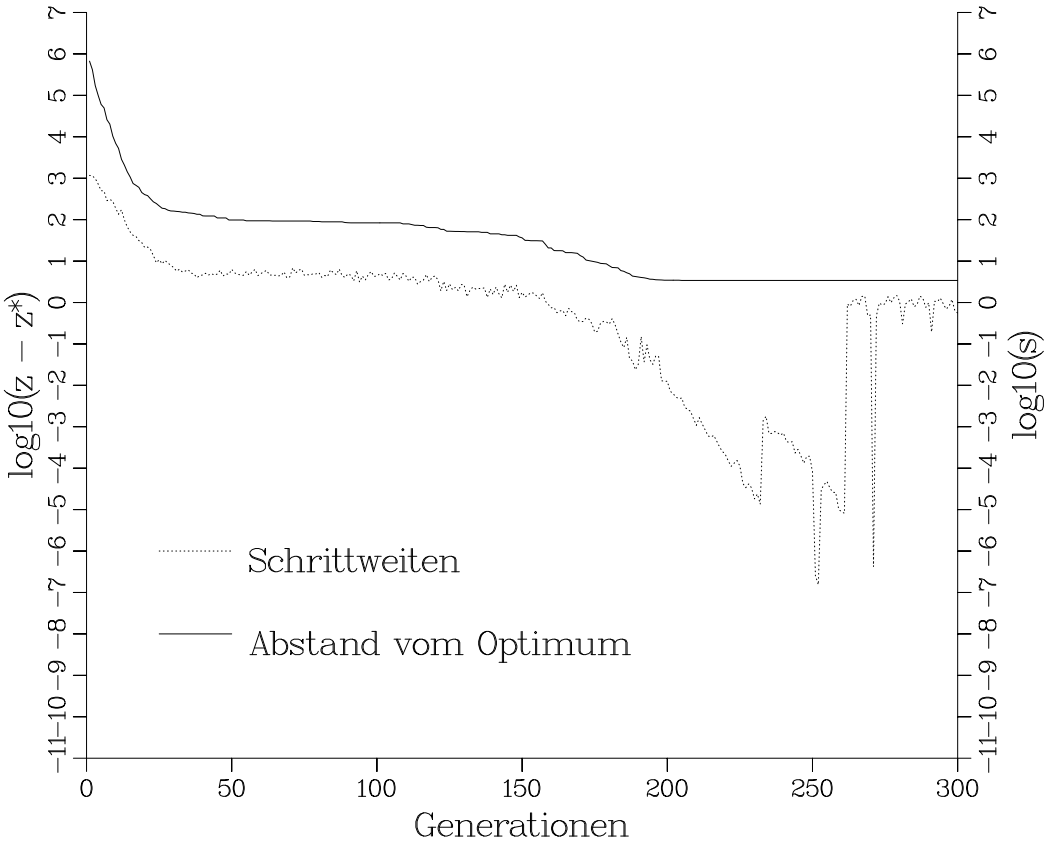

Abbildung 5.9: Schrittweiten der Variante 6 bei Problem 2

der lokalen Minimalstelle (n = 5)

```
( 0 , 1 , 0 , 1 , 1 )'
```

und eine andere Teilpopulation der lokalen Minimalstelle

```
( 1 , 0 , 0 , 1 , 0 )'
```

gut angenähert. Durch Migration treffen nun zwei solche Individuen aufeinander und erzeugen durch diskrete Rekombination etwa einen Nachkommen der Form

```
( 0 , 0 , 0 , 1 , 0 )'  .
```

Dieser Nachkomme ist dem globalen Optimum nun wesentlich näher als beide Elternteile. Nach weiteren Migrationen ist es nun leicht möglich, daß durch Rekombination mit einem Individuum aus einer anderen Population die globale Minimalstelle aufgefunden wird:

```
( 0 , 0 , 0 , 0 , 0 )'  .
```

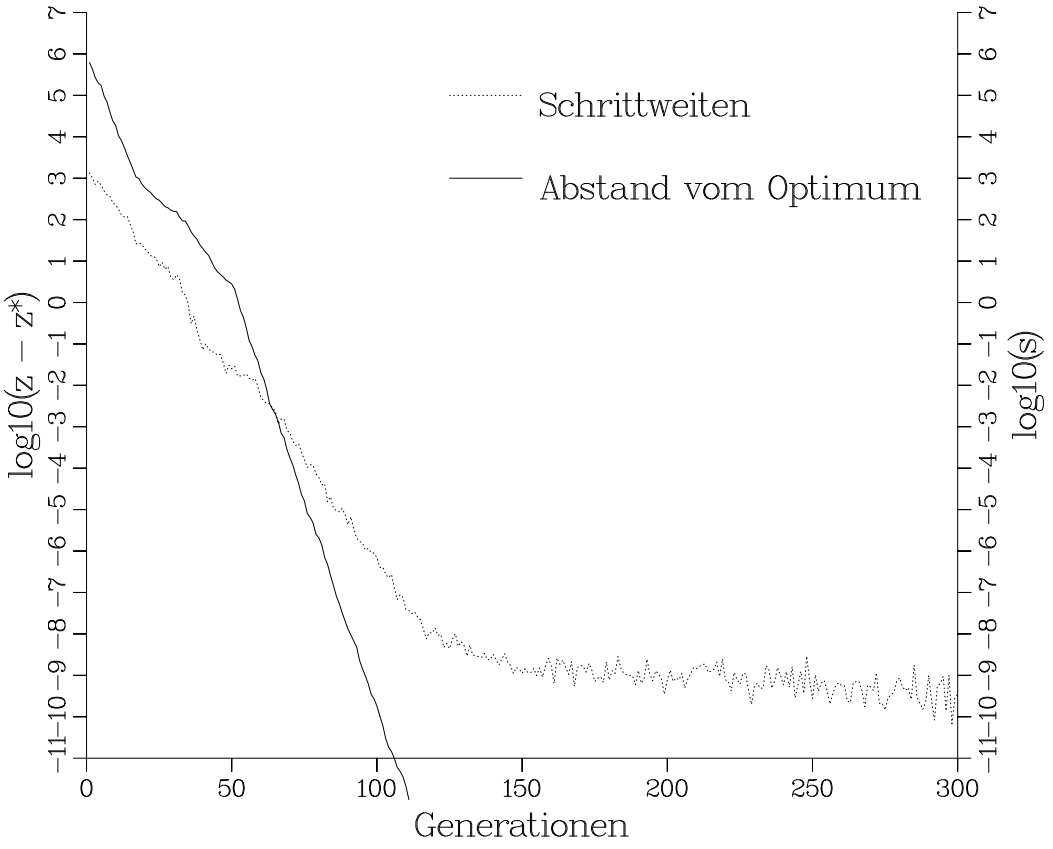

Abbildung 5.10: Schrittweiten der Variante 8 bei Problem 2

Diese These wird dadurch erhärtet, daß - wie bereits gesagt - die diskrete Variante 1 ohne Migration das globale Minimum nicht lokalisieren kann.

Die Wahrscheinlichkeit, daß für alle Genorte bzw. Vektorkomponenten genau das „richtige“ Gen gewählt wird, läßt sich durch

$$P\{\text{ „optimale Genwahl“ }\} = \left(\frac{1}{2}\right)^{n-m} \tag{5.3}$$

angeben, wobei n die Anzahl der Genorte insgesamt[7] und m die Anzahl solcher Genorte angibt, die bei beiden Elternteilen mit den gleichen Werten belegt sind. Im obigen Beispiel ergibt sich gemäß (5.3) eine Wahrscheinlichkeit von

$$P\{\text{ „optimale Genwahl“ }\} = \left(\frac{1}{2}\right)^{5-2} = 0.125 \ .$$

Bei höheren Dimensionen sinkt diese Wahrscheinlichkeit rapide, was auch ein Grund dafür sein wird, daß bei den Problemen 3 und 4 mehr Generationen benötigt werden.

[7]Das ist gleichbedeutend mit der Dimension des Problems.

Betreibt man das gleiche Gedankenspiel bei der intermediären Rekombination, dann zeigt sich, daß hier durch die Migration nichts gewonnen werden kann: Werden wie zuvor die Gene der beiden Eltern intermediär rekombiniert, dann wird das Genom des Nachkommen wie folgt aussehen:

```
( 0.5 , 0.5 , 0 , 1 , 0.5 )'  .
```

Der Wert von 0.5 in einer Komponente ist aber gerade ein besonders schlechter Wert, so daß die Effekte der Migration sofort durch die Selektion eliminiert werden.

Zusammenfassend läßt sich also sagen, daß bei dieser Problemklasse die Migration alleine bzw. die diskrete Rekombination alleine keinen Vorteil bezüglich der globalen Konvergenzsicherheit bringen. Erst durch ihre gemeinsame Verwendung kann das globale Optimum gefunden werden.

Abschließend soll noch der Konvergenzverlauf für das Testproblem 5 untersucht werden, der durch die Abbildung 5.11 wiedergegeben wird. Bei diesem Problem schneiden wieder die intermediären Varianten gegenüber den diskreten Varianten deutlich besser ab.

Am besten schneiden die Varianten mit intermediärer Rekombination und der Migration des besten Individuums ab (Variante 2 und 6). Dazu gesellt sich erstaunlicherweise auch die Variante 0 (ohne Migration). Dies könnte ein Hinweis darauf sein, daß dieses Problem doch nicht so schwer ist, wie zunächst angenommen wurde. Schließlich kann man nach Abb. 5.11 davon ausgehen, daß alle Varianten zum globalen Optimum konvergieren.

Die Schrittweiten der Varianten 2 und 6 sind mitunter großen Schwankungen unterworfen. Diese Schwankungen sind sicherlich auf die Effekte der Migration zurückzuführen. Die Abbildungen 5.12 und 5.13 mögen der Veranschaulichung dienen.

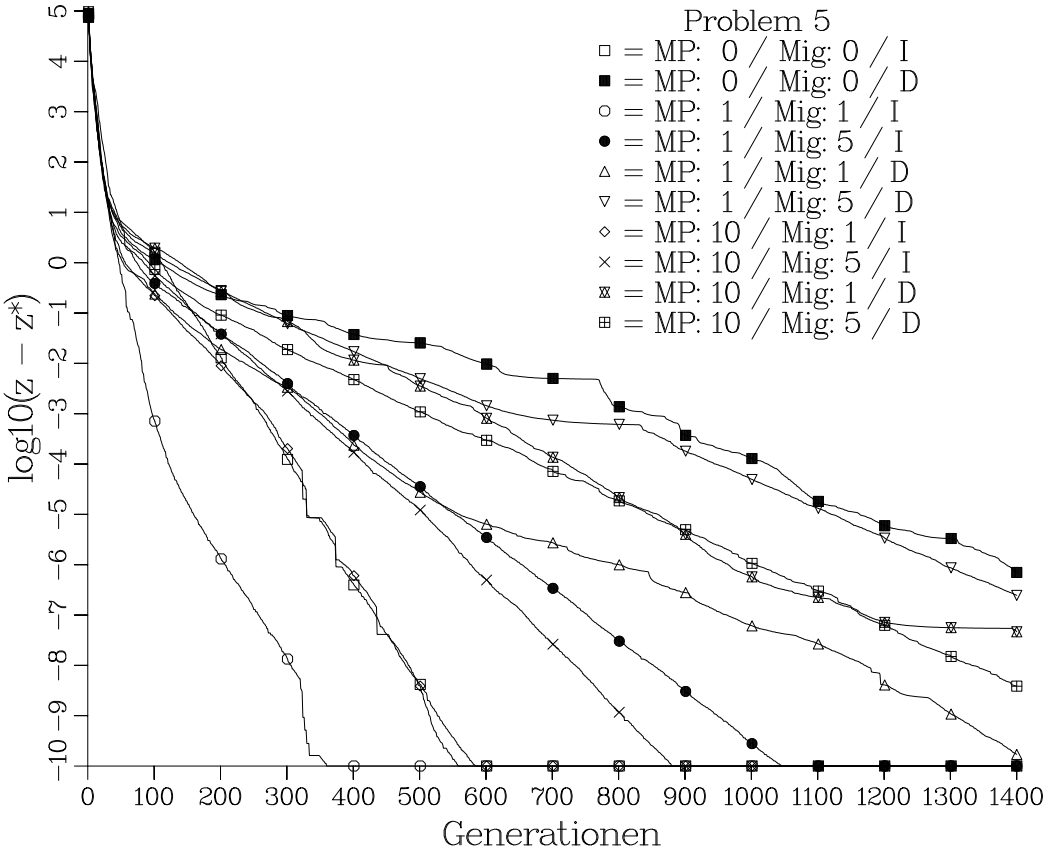

Abbildung 5.11: Konvergenzverlauf aller Varianten für Problem 5

Werden jeweils 5 Individuen mit jeder Nachbarpopulation ausgetauscht, so wird die Konvergenzgeschwindigkeit zwar verringert, die Konvergenz als solche jedoch gleichmäßiger. Das zeigt sich z.B. auch bei Variante 3 (Abb. 5.14):

Bei den diskreten Varianten ist im Grunde nur eine langsamere Konvergenz zu beobachten. Allerdings zeigt sich auch hier, daß der Austausch von je 5 Individuen zu einer gleichmäßigeren Konvergenz führt. Bei der Variante 4 mit dem Austausch des besten Individuums kann man aus der Abbildung 5.15 erkennen, wie die Variante aus lokalen Minima-Bereichen entkommt.

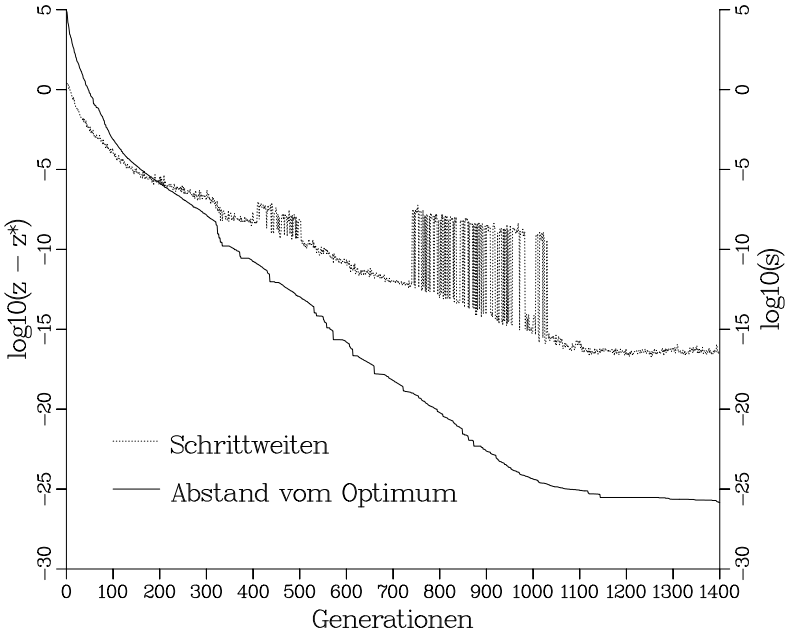

Abbildung 5.12: Schrittweiten der Variante 2 bei Problem 5

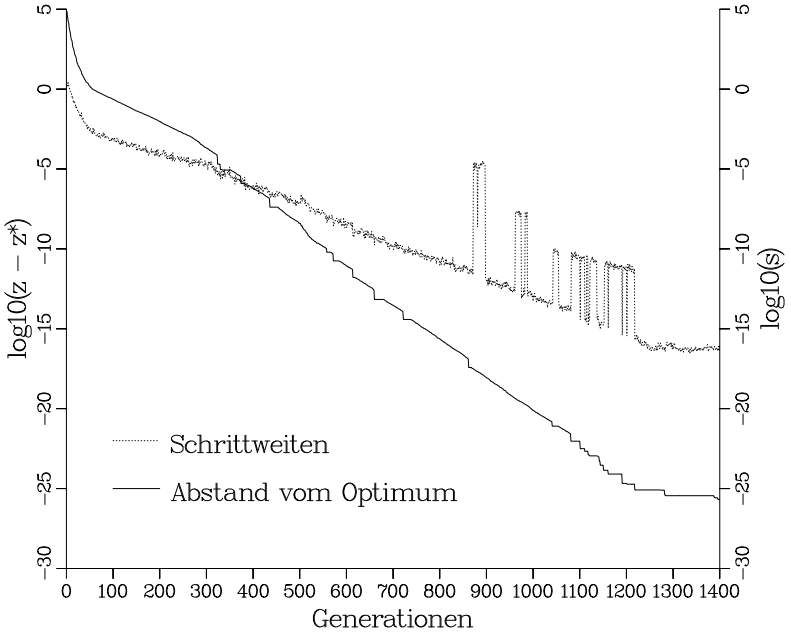

Abbildung 5.13: Schrittweiten der Variante 6 bei Problem 5

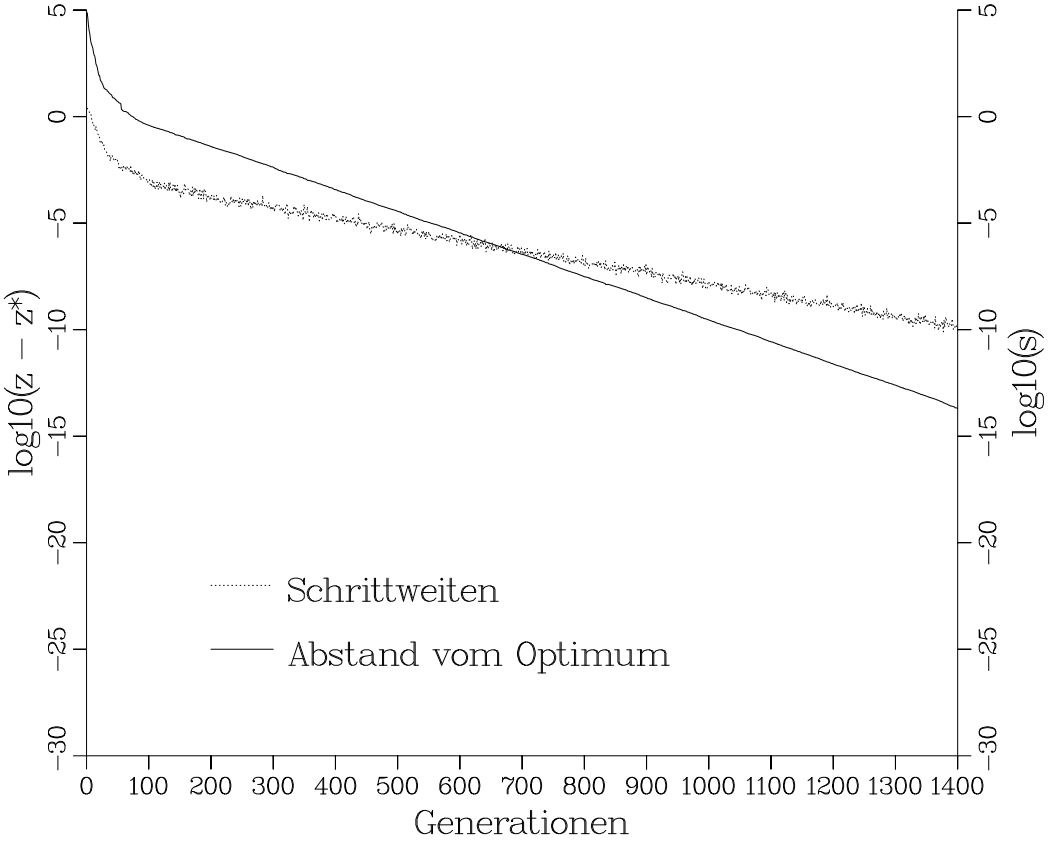

Abbildung 5.14: Schrittweiten der Variante 3 bei Problem 5

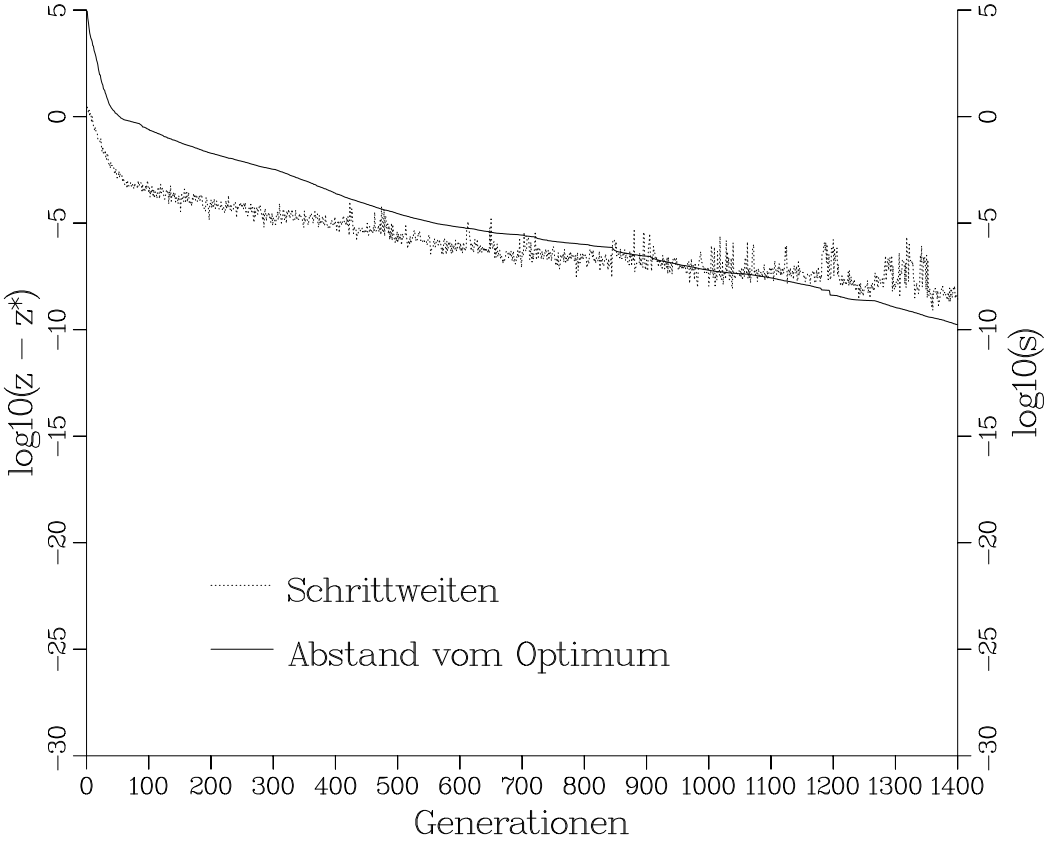

Abbildung 5.15: Schrittweiten der Variante 4 bei Problem 5

Kapitel 6

Zusammenfassung und Ausblick

In der vorliegenden Arbeit wurde eine parallele bzw. verteilte Evolutionsstrategie für ein Multi-Prozessorsystem mit einer (logischen) Ring-Topologie zur globalen Optimierung entwickelt. Dabei wurden bei einem grobkörnigen Ansatz die Prinzipien *Separation* und *periodische Migration* der biologischen Evolution nachgeahmt.

Bezüglich der Konvergenzgeschwindigkeit erwies sich die *intermediäre Rekombination* der *diskreten Rekombination* weit überlegen: Den intermediären Varianten konnte eine *geometrische Konvergenz* attestiert werden. Als besonders vorteilhaft erwies sich der häufige Austausch der besten Individuen: Diese Varianten wiesen die größte Konvergenzgeschwindigkeit auf.

Bei den Tests zur globalen Konvergenz zeigte sich, daß der Rekombinationsart größere Bedeutung zukommt: Die *diskrete Rekombination* scheint in solchen Fällen günstig zu sein, wo die Zielfunktion in der „Nähe“ des globalen Optimums starken Störungen ausgesetzt ist (Probleme 2, 3, 4). Jedoch genügt die diskrete Rekombination allein noch nicht. Erst in Verbindung mit *periodischer Migration* von Individuen zwischen den *separierten* Teilpopulationen kann globale Konvergenz erzielt werden. Bei der *intermediären Rekombination* kann auch durch Migration anscheinend nichts gewonnen werden: Sämtliche Varianten konvergieren vorzeitig zu einem lokalen Minimum.

Die Testergebnisse bzgl. des Problems 5 deuten darauf hin, daß dieses Testproblem für das entwickelte Verfahren keine Schwierigkeit bedeutet: Alle Varianten konvergieren zu dem globalen Minimum. Insbesondere die *intermediären* Varianten weisen eine hohe Konvergenzgeschwindigkeit auf. Dieses Problem scheint im Vergleich zu den Problemen 2, 3 und 4 relativ große Attraktionsgebiete zu haben - diese These wird durch die Tatsache erhärtet, daß auch die intermediäre Variante ohne Migration schnell zum globalen Minimum konvergiert. Die *diskreten* Varianten konvergieren wesentlich langsamer. Bei ihnen zeigt sich jedoch deutlich, daß der Austausch von mehreren Individuen zu einer gleichmäßigeren Konvergenz führt - das gilt mit Abstrichen auch für die intermediären Varianten.

Aus diesen wenigen Tests lassen sich noch keine Empfehlungen zur Parameterwahl bei der parallelen Evolutionsstrategie ableiten. Bevor jedoch weitere Tests gefahren werden, sollte zunächst ein Sortiment unterschiedlichster Testprobleme zusammengestellt werden. Schon bei den in dieser Arbeit verwendeten Testproblemen zeigte sich ein recht unterschiedliches Verhalten der verschiedenen Varianten. Interessant wäre in diesem Zusammenhang, wie sich die gesamte Strategie verhält, wenn auf jedem Prozessor eine andere Variante läuft. Das Zusammenspiel von Varianten, die langsam konvergieren, und solchen, die schnell ein lokales Minimum approximieren, könnte sich als ausgesprochen fruchtbar erweisen.

Für zukünftige Tests müßte auch die Wahl des Performanzmaßes überdacht werden. Ein Vergleich über Generationen wird dann nicht mehr möglich sein, wenn die Auswertung der Zielfunktion unterschiedlich lange dauert oder wenn die Nebenbedingungen an der globalen Minimalstelle aktiv werden, so daß sich viele Letalmutationen, also unzulässige Lösungen ergeben.

Die Auswertung der Testergebnisse wird sich bei größeren Testreihen als sehr aufwendig erweisen: Hier bieten sich rechnergestützte statistische Testverfahren an. Diese konnten in dieser Arbeit nicht verwendet werden, weil die statistische Masse der Testergebnisse zu gering war.

Zusammenfassend läßt sich sagen, daß der Ansatz mit den Prinzipien *Separation* und *periodischer Migration* vielversprechende Perspektiven bietet: Nur so konnten die Testprobleme 2, 3 und 4 gelöst werden. Hier sind auch durch eine Änderung der logischen Topologie noch viele Tests möglich bzw. erforderlich.

Anhang A

Literaturverzeichnis

Archetti und Frontini (1978)
Archetti, F. und F. Frontini : *The application of a global optimization method to some technological problems*, S. 179-188 in: Dixon und Szegö (1978).

Baram (1989)
Baram, Y. : *Associative Memory in Fractal Neural Networks*, IEEE Trans. on Systems, Men and Cybernetics, Vol. SMC-19, No. 5, S. 1133-1141, Sept./Oct. 1989.

Bazaraa und Shetty (1979)
Bazaraa, M.S. und C.M. Shetty : *Nonlinear Programming - Theory and Algorithms*, New York: Wiley 1979.

Becker und Lago (1970)
Becker, R.W. und G.V. Lago : *A global optimization algorithm*, S. 3-12 in: Proc. of the 8th Allerton Conf. on Circuits and System theory, Monticello/Illinois 1970.

Bertsekas und Tsitsiklis (1989)
Bertsekas, D.P. und J.N. Tsitsiklis : *Parallel and Distributed Computations*, Englewood Cliffs: Prentice Hall 1989.

Beyer (1989)
Beyer, H.-G. : *Ein Evolutionsverfahren zur mathematischen Modellierung stationärer Zustände in dynamischen Systemen*, Dissertation (A), Universität Weimar 1989.

Bormann (1989)
Bormann, A. : *Parallelisierungsmöglichkeiten für direkte Optimierverfahren auf Transputersystemen*, Diplomarbeit, Universität Dortmund 1989.

Born (1978)
Born, J. : *Evolutionsstrategien zur numerischen Lösung von Adaptationsaufgaben*, Dissertation (A), Math.-Naturwiss. Fakultät, Humboldt-Universität Berlin 1978.

Box (1965)
Box, M.J. : *A new method of constrained optimization and a comparison with other methods*, Comp. J. 8, S. 42-52, 1965.

Branin (1972)
Branin, F.H. : *Widely convergent methods for finding multiple solutions of simultaneous nonlinear equations*, IBM Journal of Research Developements, S. 504-522, 1972.

Bremermann (1970)
Bremermann, H.A. : *A method of unconstrained global optimization*, Mathematical Biosciences 9, S. 1-15, 1970.

Brooks (1958)
Brooks, S.H. : *A Discussion of Random Methods for Seeking Maxima*, Oper. Res. 6, S. 244-251, 1958.

Cohoon u.a. (1987)
Cohoon, J.P.; S.U. Hegde, W.N. Martin und D. Richards : *Punctuated Equilibria: A Parallel Genetic Algorithm*, S. 148-154 in: Grefenstette (1987).

Dantzig (1966)
Dantzig, G.B. : *Lineare Programmierung und Erweiterungen*, Berlin: Springer 1966.

Dixon u.a. (1975)
Dixon, L.C.W.; Gomulka, J. und G.P. Szegö : *Towards Global Optimization*, S. 29-54 in: Dixon und Szegö (1975).

Dixon und Szegö (1975)
Dixon, L.C.W. und G.P. Szegö (eds.) : *Towards Global Oprimization*, Amsterdam: North Holland 1975.

Dixon und Szegö (1978)
Dixon, L.C.W. und G.P. Szegö (eds.) : *Towards Global Oprimization 2*, Amsterdam: North Holland 1978.

Dixon und Szegö (1978a)
Dixon, L.C.W. und G.P. Szegö : *The Global Optimization Problem: An Introduction*, S. 1-15 in: Dixon und Szegö (1978).

Eiselt u.a. (1987)
Eiselt, H.A.; Pederzoli, G. und C.-L. Sandbloom : *Continuous Optimization Models*, Berlin und New York: de Gruyter 1987.

Evtushenko (1971)
Evtushenko, Y.G. : *Numerical methods for finding global extrema (Case of a non-uniform mesh)*, USSR Comp. Math. and Math. Phys. 11, S. 1390-1403, 1971.

Faddejew und Faddejewa (1973)
Faddejew, D.K. und W.N. Faddejewa : *Numerische Methoden der linearen Algebra*, 3. Aufl., München und Wien: Oldenbourg 1973.

Flynn (1966)
Flynn, M.J. : *Very high-speed computing systems*, Proc. IEEE 54, S. 1901-1909, 1966.

Futuyma (1990)
Futuyma, D.J. : *Evolutionsbiologie*, Basel: Birkhäuser 1990.

Gill u.a. (1989)
Gill, P.E.; Murray, W.; Saunders, M.A. und M.H. Wright: *Constrained Nonlinear Programming*, S. 171-220 in: Nemhauser u.a. (1989).

Goldberg (1989)
Goldberg, D.E. : *Genetic Algorithms in Search, Optimization, and Machine Learning*, Reading/Mass.: Addison Wesley 1989.

Göpfert u.a. (1986)
Göpfert, A. (Hrsg.) : *Lexikon der Optimierung*, Berlin: Akademie-Verlag 1986.

Gorges-Schleuter (1989)
Gorges-Schleuter, M. : *ASPARAGOS: An Asynchronous Parallel Genetic Optimization Strategy*, S. 422-427 in: Schaffer (1989).

Grefenstette (1987)
Grefenstette, J.J. (ed.) : *Genetic algorithms and their applications*, Proc. of the Second Int. Conf. on Genetic Algorithms, July 28-31, 1987, Hillsdale: Lawrence Earlbaum 1987.

Griewank (1981)
Griewank, A.O. : *Generalized decent for global optimization*, JOTA 34, S. 11-39, 1981.

Hermes und Hoffmeister (1990)
Hermes, U. und F. Hoffmeister : *Der Betrieb eines MultiCluster-Transputersystems*, Interner Bericht, FB Informatik, Lehrstuhl für Systemanalyse, Universität Dortmund 1990.

Hill (1969)
Hill, J.D.: *A Search Technique for Multimodal Surfaces*, IEEE Trans. on systems, science and cybernetics, Vol. SSC-5, No. 1, S. 2-8, Jan. 1969.

Hoare (1978)
Hoare, C.A.R. : *Communicating Sequential Processes*, Communications of the ACM 21:8, S. 666-677, 1978.

Hoffmeister und Schwefel (1988)
Hoffmeister, F. und H.-P. Schwefel : *Anwendung der Evolutionsstrategie auf Parallel-Rechnern: Ein Konzept*, Interner Bericht, FB Informatik, Lehrstuhl für Systemanalyse, Universität Dortmund 1988.

Holland (1975)
Holland, J.H. : *Adaptation in natural and artificial systems*, Ann Arbor: The University of Michigan Press 1975.

Hooke und Jeeves (1961)
Hooke, R. und T.A. Jeeves : *Direct search solution of numerical and statistical problems*, Journal of the ACM 8, S. 212-229, 1961.

Hoßfeld (1981)
Hoßfeld, F. : *Parallele Algorithmen*, Technical Report Jül-Spez-125, Kernforschungsanlage Jülich, Zentralinstitut für Angewandte Mathematik, Jülich 1981.

INMOS (1988)
INMOS Ltd. : *Transputer Reference Manual*, Prentice Hall 1988.

Jordan-Engeln und Reutter (1976)
Jordan-Engeln, G. und F. Reutter : *Formelsammlung zur Numerischen Mathematik mit Fortran IV - Programmen*, B.I. - Hochschultaschenbücher Bd. 106, 2. Aufl., Mannheim: Bibliographisches Institut 1976.

Kirkpatrick u.a. (1983)
Kirkpatrick, S.; Gelatt Jr., C.D. und M.P Vecchi : *Optimization by Simulated Annealing*, Science 220, S. 671-680, 1983.

Kursawe (1990)
Kursawe, F. : *Evolutionsstrategien für die Vektoroptimierung*, Diplomarbeit, Universität Dortmund 1990.

Leven u.a. (1989)
Leven, R.W.; B.-P. Koch und B. Pompe : *Chaos in dissipativen Systemen*, Braunschweig und Wiesbaden: Vieweg 1989.

Levy und Montalvo (1985)
Levy, A.V. und A. Montalvo : *The tunneling algorithm for the global minimization of functions*, SIAM J. Sci. Stat. Comp. 6, S. 15-29, 1985.

Luenberger (1973)
Luenberger, D.G. : *Introduction to linear and nonlinear programming*, Reading/Mass.: Addison Wesley 1973.

Manderick und Spiessens (1989)
Manderick, B. und P. Spiessens : *Fine-Grained Parallel Genetic Algorithms*, S. 428-433 in: Schaffer (1989).

Marti (1986)
Marti, K. : *Controlled Random Search Procedures for Global Optimization*, S. 457-474 in: V.I. Arkin, A. Shiraev und R. Wets (eds.) : Stochastic Optimization, Lecture Notes in Control and Information Science, Berlin: Springer 1986.

Mattern (1989)
Mattern, F. : *Verteilte Basisalgorithmen*, Informatik Fachberichte 226, Berlin: Springer 1989.

McMurty und Fu (1966)
McMurty, G.J. und K.S. Fu: *A Variable Structure Automaton Used as a Multimodal Searching Technique*, IEEE Trans. on automatic control, Vol. AC-11, No. 3, S. 379-387, Juli 1966.

Mockus (1975)
Mockus, J. : *On Bayesian Methods of Optimization*, S. 166-181 in: Dixon und Szegö (1975).

Mück (1989)
Mück, A. : *Einfluß verschiedener Wahrscheinlichkeitsverteilungen auf das Konvergenzverhalten von Evolutionsstrategien*, Diplomarbeit, Universität Dortmund 1989.

Nehmer (1985)
Nehmer, J. : *Softwaretechnik für verteilte Systeme*, Berlin: Springer 1985.

Nelder und Mead (1965)
Nelder, J.A. und R. Mead : *A Simplex method for function minimization*, Comp. J. 7, S. 308-313, 1965.

Nemhauser u.a. (1989)
Nemhauser, G.L.; Rinnooy Kan, A.H.G. und M.J. Todd (eds.) : *Optimization*, Handbooks in Operations Research and Management Science Vol. 1, Amsterdam: North Holland 1989.

Perihelion (1989)
Perihelion Software Ltd. : *The Helios operating system*, Prentice Hall 1989.

Pettey u.a. (1987)
Pettey, C.B.; M.R. Leuze und J.J. Grefenstette: *A Parallel Genetic Algorithm*, S. 155-161 in: Grefenstette (1987).

Rappl (1984)
Rappl, G. : *Konvergenzraten von Random Search Verfahren zur globalen Optimierung*, Dissertation, HSBw München 1984.

Rastrigin (1963)
Rastrigin, L.A. : *The convergence of the random search method in the*

extremal control of a many-parameter system, Automation and Remote Control ARC 24, S. 1337-1342, 1963.

Rechenberg (1973)
Rechenberg, I. : *Evolutionsstrategie*, Stuutgart/Bad Cannstatt: Frommann 1973.

Rinnooy Kan und Timmer (1989)
Rinnooy Kan, A.H.G. und G.T. Timmer : *Global Optimization*, S. 631-662 in: Nemhauser u.a. (1989).

Rosenbrock (1960)
Rosenbrock, H.H. : *An automatic method for finding the greatest or least value of a function*, Comp. J. 3, S. 175-184, 1960.

Schaffer (1989)
Schaffer, J.D. (ed.) : *Genetic algorithms*, Proc. of the 3rd Int. Conf. on Genetic algorithms, San Mateo: Morgan Kaufman 1989.

Schneider (1986)
Schneider, P. : *Algorithmen zur Parameteradaptation in der mehrgliedrigen Evolutionsstrategie bei der diskreten Optimierung*, Diplomarbeit, Universität Dortmund 1986.

Schumer und Steiglitz (1968)
Schumer, M.A. und K. Steiglitz : *Adaptive step size random search*, IEEE Trans. on Automatic Control AC-13, S. 270-276, 1968.

Schwefel (1977)
Schwefel, H.-P. : *Numerische Optimierung von Computer-Modellen mittels der Evolutionsstrategie*, Basel und Stuttgart: Birkhäuser 1977.

Schwefel (1980)
Schwefel, H.-P. : *Unterprogramme EVOL, GRUP, KORR - Programme und Benutzeranleitungen*, Interner Bericht KFA-STE-IB-3/80, Kernforschungsanlage Jülich 1980.

Schwefel (1981)
Schwefel, H.-P. : *Numerical Optimization of Computer Modells*, Chichester: Wiley 1981.

Schwefel (1987)
Schwefel, H.-P. : *Collective Phenomena in Evolutionary Systems*, S. 1025-1033 in: 31st Ann. Meeting of the Internat. Soc. for General System Research, Budapest, Vol. II, June 1987.

Schwefel (1989)
Schwefel, H.-P. : *Optimum Seeking by Imitating Natural Intelligence*, S. 179-182 in: Papers of the Workshop 'Adaptive Learning', Schloß Reisensburg (Günzberg), 16. - 21. Juli 1989, FAW-B-89020, FAW Ulm 1989.

Sedlag und Weinert (1987)
Sedlag, U. und E. Weinert : *Biogeographie, Artbildung, Evolution*, Wörterbuch der Biologie, Jena: VEB Fischer 1987.

Späth (1983)
Späth, H. : *Cluster-Formation und -Analyse: Theorie, FORTRAN-Programme und Beispiele*, München und Wien: Oldenbourg 1983.

Tanese (1987)
Tanese, R. : *Parallel Genetic Algorithms for a Hypercube*, S. 177-183 in: Grefenstette (1987).

Tanese (1989)
Tanese, R. : *Distributed Genetic Algorithms*, S. 434-439 in: Schaffer (1989).

Törn und Žilinskas (1989)
Törn, A. und A. Žilinskas : *Global Optimization*, Lecture Notes in Computer Science Vol. 350, Berlin: Springer 1989.

Treccani (1975)
Treccani, G.: *On the Convergence of Branin's Method: A Counter Example*, S. 107-116 in: Dixon und Szegö (1975).

Vanderbilt und Louie (1984)
Vanderbilt, D. und S.G. Louie : *A Monte Carlo Simulated Annealing Approach to Optimization over Continuous Variables*, Journal of Computational Physics 56, S. 259-271, 1984.

Veer (1990)
Veer, B. : *The CDL Guide*, Bristol: Distributed Software Ltd. 1990.

Vilkow u.a. (1975)
Vilkow, A.V.; N.P. Zhidow und B.M. Shchedrin : *A method of search for the global minimum of a function of one variable*, J. of Comput. Math. and Math. Phys. 15, S. 1040-1042, 1975.

Yao (1989)
Yao, Y.: *Dynamic Tunneling Algorithm for Global Optimization*, IEEE Trans. on systems, man and cybernetics, Vol. SMC-19, No. 5, S. 1222-1230, 1989.

Žilinskas (1978)
Žilinskas , A. : *On Statistical Models for Multimodal Optimization*, Math. Operationsforsch. Statist., Ser. Statistics, Vol. 9, No. 2, S.255-266, 1978.

Žilinskas (1980)
Žilinskas , A. : *On the Use of Statistical Models for the Construction of the Optimization Algorithms*, S. 138-147 in: A.V. Balakrishnan und M. Thoma (eds.) : Optimization Techniques, Part 2, Lecture Notes in Control and Information Science, Vol. 23, Berlin: Springer 1980.