Experimentelle Untersuchungen über den Einsatz von Genetischen Algorithmen in Selbstorganisierenden Datenstrukturen

Diplomarbeit von
Andreas Grimm
an der
Fachhochschule Köln, Abt. Gummersbach

14. Dezember 1991

Zusammenfassung

Inhalt dieser Diplomarbeit behandelt die Definition und die Durchführung von Experimenten zum Einsatz von Genetischen Algorithmen bei dem Einsatz im Bereich der Selbstorganisierenden Datenstrukturen. Hierbei unterteilt sich die Arbeit in zwei Teile. Der erste Teil ist eine theoretische Zusammenfassung und Ausarbeitung zum dem Thema, der zweite Teil beschreibt die Durchführung und die Ergebnisse der Experimente die sich aus der theoretischen Ausarbeitung ergeben haben.

Im Rahmen der theoretischen Ausarbeitung wird zuerst eine Zusammenfassung der Genetischen Algorithmen gegeben und dann werden die Definitionen vorgenommen, die für die Experimente notwendig sind. Hierbei müssen einige Definitionen der Literatur erweitert werden. Hierzu gehört u.a. die Definition der Mittleren Weglänge zur Beurteilung der Effizenz der Graphen.

Bei dem Teil der Experimente geht es insbesondere darum, qualitative Aussagen über das Verhalten des beschriebenen Systems zu treffen. Hierzu wird eine, im Anhang beschriebene Umgebung benutzt, in der die Versuche durchgeführt werden. Außerdem wird versucht, eine Interpretation der Ergebnisse zu geben. Besonders interessant sind die Ergebnisse zu der Problematik der Topologischen Optimierung.

Es stellte sich heraus, daß bei den Topologischen Optimierungen sehr gute Ergebnisse erzeugt werden. Man erreicht dort Fitnesswerte von mehr als neunzig Prozent. Auch die Reaktion auf eine Veränderung der Fitnessfunktion und die Anpassung auf eine veränderte Umwelt lassen auf eine gute Anwendbarkeit der Genetischen Algorithmen in diesem Bereich hoffen. Etwas schlechter ist das Verhalten bei der Optmierung der Ordnung im Graphen. Hier werden bei weitem nicht so gute Ergebnisse erreicht und auch die Optimierungsgeschwindigkeit ist nicht so gut wie bei den Topologischen Optimierungen. Sehr schlecht jedoch ist die Kombination der Fitnessfunktionen, es werden keine sinnvollen Strukturen erzeugt und es werden Fitnesswerte von kaum mehr als fünfzig Prozent erzeugt. Auch eine Veränderung der Umwelt wird sehr schlecht aufgenommen und verarbeitet.

Vorwort

Diese Arbeit stellt meine Diplomarbeit im Fach Allgemeine Informatik an der Fachhochschule Köln, Abteilung Gummersbach dar. Sie wurde als solche im Wintersemester 1991 angefertigt.

Sie stellt für mich ein grosse Herausforderung dar, da mir hier zu ersten Mal die Möglichkeit gegeben wurde, eine Arbeit von diesem Umfang anzufertigen. Auch die Tatsache, daß ich ein mir fremdes Thema in dieser Tiefe aufarbeiten konnte erfüllt mich mit großer Freude.

Ich versichere hiermit, die Arbeit selbstständig angefertigt zu haben. Alle Hilfsmittel sind im Literaturverzeichnis aufgeführt.

Sehr herzlich danke ich an dieser Stelle meinem Betreuer, Herrn Professor Dr. Jochum für seine großartige Unterstützung dieser Arbeit und seinen Anregungen, die diese Arbeit maßgeblich beeinflußt haben.

Leverkusen, im Dezember 1991

Andreas Grimm

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1

Einleitung

1.1 Motivation

Eines der mit am häufigsten bearbeiteten Themen in der Informatik ist die Frage nach einer schnellen Organisationsform und Zugriffsmethode auf gespeicherte Daten im System der Verarbeitungsanlage. Sehr oft ist die Effizenz eines Systems allein abhängig von eben dieser Frage. SCHLAGETER und STUCKY bezeichnen dieses Thema als *ein zentrales Gebiet der Informatik*[Sch83]. WIRTH bezeichnet die Fähigkeit der schnellen Datenspeicherung bei modernen Computern für so bedeutend, daß sie die Haupteigenschaft, die schnelle Berechnung in vielen Fällen belanglos erscheinen läßt[Wir83].

Bei DENERT und FRANCK[Den77] steht zu diesem Thema

> "Entwicklung und Analyse von Algorithmen stehen in der Informatik traditionell im Vordergrund - und zwar sowohl in der historischen Entwicklung wie auch in der bis heute üblichen Ausbildung, wo das Schwergewicht anfänglich meist auf Algorithmen gelegt wird.
>
> Demgegenüber sind komplexe Datenstrukturen erst spät in dem Blickpunkt des Interesses gerückt und Gegenstand wissenschaftlicher Untersuchungen geworden. Und dies, obwohl Algorithmen und Datenstrukturen einander gegenseitig bedingen: Jeder Algorithmus operiert auf gewissen Daten, und seine Formulierung wird wesentlich von deren Struktur bestimmt."

Das Teilgebiet der Informatik, welches sich mit der Anordnung von Daten im System und dem Auffinden von gespeicherten Daten beschäftigt, ist die *Datenorganisation.*

Die Datenorganisation ist eines in der Theorie am besten behandelten Themen. Man findet entsprechende Betrachtungen bei [Bau84, Den77, Knu73] und bei vielen anderen. Bei diesen Werken wird stets darauf geachtet, daß die Daten in möglichst kurzer Zeit dem Benutzer zu Verfügung stehen.

Eine minimale Zugriffszeit auf die Daten ist Ziel und Zweck der Bemühungen in diesem Zweig der Theorie.

In der Praxis steht und fällt eine Datenbank mit der Zeit, die sie benötigt um Daten dem Anwender zur Bearbeitung darzubieten. Der Benutzer reagiert sehr sensibel, wenn es um Wartezeiten bei dem Auffinden von Daten auch in einer großen Datenbank geht (Zitat:" *Wenn das so lange dauert, da kann ich die Daten ja schon selber suchen*"). Auch wenn objektiv die Daten sehr rasch zur Verfügung stehen, wird bei eingeführten Systemen die subjektive Wartezeit als lästig empfunden.

Derselbe Effekt tritt ein, wenn Daten in die Datenbank eingefügt werden oder aus der Datenbank entfernt werden sollen. Die Wartezeit auf dem Computer wird als lästig und störend empfunden. Wie wichtig diese Problemstellung ist, erkennt man, wenn man bedenkt, daß ein Viertel der Einsatzzeit eines Rechners für diese Aktionen verwendet wird[Knu73].

Versucht man nun, diese Wartezeit zu eliminieren, so stellt man fest, daß die gesunkene Suchzeit durch einen gesteigerten Organisationsaufwand kompensiert wird. Damit wird eine gesunkene Zugriffszeit bei der Suche nur über eine erhöhte Wartezeit beim Einfügen und Löschen erreicht. Deshalb wird bei der Datenorganisation darauf Wert gelegt, das Verfahren zu verwenden, das für die am häufigsten vorkommenden Operationen eine optimale Zugriffszeit verwendet. Der Systementwickler muß also bei der Entwicklung sehr genau darauf achten, welche der Operationen[1] die häufigste ist und entsprechend die Organisationsform im Rechner wählen.

Außerdem gibt es noch andere Kriterien, die bei der Datenbank Anforderungen an die eingesetzten Algorithmen stellen. In diesem Zusammenhang schreibt N.WIRTH[Wir83], daß "*die Wahl eines Algorithmus*[2]*... beim Sortieren*[3]*... stark von der Struktur der zu bearbeitenden Daten*" abhängt. Ändert sich also die Struktur der Daten, so muß sich auch der Verwaltungsalgorithmus ändern. Bei zu rascher Änderung, einer dynamischen Struktur der Daten, ist der Aufwand für eine kontinuierliche Umstellung des Verwaltungssystems dann größer als die Verwaltung der Daten selber.

Auch WEDEKIND[Wed76] schreibt, daß Zugriffszeiten unter anderem im wesentlichen von "*Speicherungsstrukturen mit den Parametern der dazugehörigen Speicherzuordnungsstrukuren*" abhängig sind.

Bisher werden die Daten in den Systemen in festen graphischen Strukturen abgelegt, die auf Grund mathematischer Überlegungen als besonders geeignet für die Organisation der Daten in der Rechenanlage gelten.

Diese Ablagetechnik ist besonders auf bestimmte Operationen, also Aktionen auf den Daten, abgestimmt. Im Prinzip existieren drei Operationen auf Datenstrukturen: [Maj76]:

---

[1] einfügen, lesen, löschen

[2] Anm.: für die Anordnung im Rechner

[3] Nach Wirth versteht man unter Sortieren "*allgemein den Prozess des Anordnens einer gegebenen Menge von Objekten in einer bestimmten Ordnung*"[Wir83]

1. die *Einfügeoperation* : Daten werden in die Datenbank eingefügt
2. die *Suchoperation* : Daten werden im Bestand gesucht
3. die *Löschoperation* : Daten werden aus der Datenbank entfernt

Wie im weiteren beschrieben wird, sind die Anforderungen an die Datenstruktur für die einzelnen Operationen sehr verschieden.

Ändert sich nun die Aufgabenstellung der Datenbank, so ist eine Umgestaltung des Systems, die nur mit größtem Aufwand zu erreichen ist, notwendig; es kostet Zeit und Geld, die neue, zweckmäßige Datenorganisation aufzubauen.

Aus diesem Grunde gibt es schon seit längerer Zeit Versuche mit *Selbstorganisierenden Datenstrukturen*, also mit einem System, bei dem sich die Datenorganisation den Aufgaben des Gesamtsystems anpasst. Die Grundlagen werden u.a. bei MEHLHORN [Meh88] dargestellt.

Die Überlegung bei dieser Arbeit hier liegt in einer Negierung der oben gemachten Präposition von DENERT und FRANCK. Im Gegensatz zu der Aussage, daß sich die Algorithmen der Datenorganisation anpassen, wird versucht, Algorithmen so zu definieren, daß sich die Daten den Algorithmen anpassen. Dieser Ansatz wird in der Standartliteratur nicht behandelt. Gesucht wird also ein System mit einer Anpassung von Strukturen an eine bestimmte Aufgabenstellung.

Genetische Algorithmen zeichnen sich jedoch gerade dadurch aus, daß sie sich auf Umwelteinflüße durch selektive Fortpflanzung der Individuen an die veränderte Situation anpassen. Damit stellt sich die Frage, ob nicht gerade *Genetische Algorithmen* ein Hilfsmittel sind, um dieses Ziel der sich kontinuierlich an den Anforderungen anpassenden Datenorganisation zu erfüllen.

Hierzu muß untersucht werden, in wie weit sich bei *Genetischen Algorithmen* die Daten in Abhängigkeit zu einem äußeren Zivilisationsdruck verhalten und in der Organisation anordnen.

Zu dieser Untersuchung gehört sowohl das generelle Verhalten der Daten, aber auch Untersuchungen über die Komplexität des Problems und damit über das Zeitverhalten bei der Durchführung der für die Organisation notwendigen Operationen.

Bei dieser Arbeit geht es nicht um die Frage logischer, sondern nur um die Frage der physikalischen Datenorganisation im Speicher eines Rechners. Das erste wird zwar im weiteren kurz angerissen, wird jedoch nicht abschließend behandelt.

1.2 Gliederung

Die vorliegende Arbeit wird entsprechend der Aufgabenstellung gegliedert.

Im zweiten Kapitel werden die Grundlagen der *Genetischen Algorithmen* erläutert. Dieser Teil ist eine Zusammenfassung der Theorie und soll dem

Verständnis der Grundlagen dienen. Dies ist notwendig, da *Genetische Algorithmen* ein neuerer, unbekannter Zweig der nicht-numerischen Optimierung darstellen. Unter dieser Form der Optimierung versteht man Verfahren, die nicht einem Verfahren der gezielten Suche entsprechen. Außerdem wird in diesem Kapitel eine Zusammenfassung der Literatur zum Thema *Datenorganisation* gegeben.

Das dritte Kapitel der Arbeit behandelt eine kurze graphentheoretische Einleitung und die theoretischen Grundlagen für die Beschreibung der Datenorganisation (in Form eines Graphen). Hierzu gehört auch die Beschreibung der Operationen der *Genetischen Algorithmen* und die Darstellung des Graphen in einer binären Form. Außerdem wird in diesem Kapitel eine Zusammenfassung der Literatur zum Thema *Datenorganisation* gegeben. Im dritten Kapitel werden die theoretischen Grundlagen und die Definition des Individuums besprochen. Dieses Individuum ist das zu optimierende Element bei den *Genetischen Algorithmen*, hier also der Graph.

Das vierte Kapitel beschreibt die Fitnessfunktion und deren theoretische Grundlagen. Hierbei handelt es sich um das zweite elementare Charakteristikum dieser Form der Algorithmen.

Der nächste Teil der Arbeit, das fünfte und sechste Kapitel, beschäftigt sich mit der Definition und Durchführung der Experimente sowie deren Auswertung. Hier gehen insbesondere die Erkenntnisse der vorigen Teile in die Experimentalumgebung ein.

Im letzten Kapitel der Arbeit werden die Ergebnisse beurteilt. Hierbei werden auch Komplexitäts- und Aufwandsbetrachtungen vorgenommen. Außerdem soll der Versuch gemacht werden, andere Anwendungsgebiete für die Aufgabenstellung zu finden.

Desweiteren werden im Anhang die *Experimentelle Umgebung* und verschiedene Ergebnisse beigelegt.

Teil I

Grundlagen

Kapitel 2

GA's und Datenstrukturen

Die folgende Einleitung in die *Genetischen Algorithmen* ist im wesentlichen identisch mit [Gri90] und wurde nachträglich mit Elementen aus [Dav91] erweitert. Sinn dieses Abschnittes ist es, eine elementare Einführung in die Theorie der *Genetischen Algorithmen* zu geben.

2.1 Einleitung in die Genetischen Algorithmen

Genetische Algorithmen stellen einen recht unbekannten, neuen Zweig in der Erforschung der Möglichkeiten in der Programmierung von komplexen Prozessen dar. Man kann sich schon bei der Namensgebung vorstellen, daß hier zwei Naturwissenschaften zusammengenommen werden :

- Die Biologie mit ihrem Wissen über *evolutionäre Prozesse* , und
- Die Informatik mit ihrem Wissen über die *Algorithmisierbarkeit* von Prozessen und natürlichen Vorgängen. Diese *Algorithmisierbarkeit* bezeichnet hierbei die Formulierung der Problemstellung in Anweisungen und Daten[Wir85].

Man nennt die Wissenschaft über die Vorgänge in der Evolution und über die Fortpflanzung und die Selektion der am besten angepassten Lebewesen *Genetik*. Man betrachtet also bei den *Genetischen Algorithmen* Vorgänge, die sich optimal an äußere Gegebenheiten anpassen[1]. Hierbei geht es im wesentlichen um eine Anpassung von einem Spezies an eine Umwelt, die sich i.a. als feindlich[2] dem Individuum zeigt. Ohne diese Anpassung an diese Umwelt wäre ein Aussterben des Spezies nicht zu vermeiden. Jedoch entwickelte die Natur im Laufe

[1] Hier sei erwähnt, das Darwin nicht von dem Überleben des "Besten" sprach, sondern vom "Survivel of the Fittest".

[2] oder lebensbedrohend

von Jahrmillionen Mechanismen, um Lebewesen auf ihren Lebensraum hin zu optimieren. Diese Mechanismen sind im Detail kompliziert, jedoch im Überblick mit einer entsprechenden Abstraktion so zu vereinfachen, daß man sie auch in der Informatik einsetzen kann.

J.Holland entwickelte diese Richtung der Forschung bei dem Versuch, natürliche Systeme mit Computerhilfe zu adaptieren. Insbesondere ging es ihm darum, Systeme zu entwickeln, die die Prozesse der Natur nachbilden[Gol89].

2.1.1 Einführendes Beispiel

A.K.Dewdney hat ein schönes, anschauliches Beispiel gegeben :

> "Stellen Sie sich einen abstrakten Ozean vor, in dem ebenso abstrakte Organismen hausen. Nennen wir sie endliche, lebende Klümpchen, kurz Eleks. Jedes Elek ist mit dem einfachsten denkbaren Entscheidungsapparat ausgestattet, also mit dem biologischen Äquivalent dessen, was Informatiker einen endlichen Automaten nennen. Daneben enthält es genau ein Chromosom, das den Automaten durch eine Folge von Symbolen beschreibt.
>
> Die Eleks bewohnen eine digitale Ursuppe, die sich auf regelmässige Weise verändert. Sollen sie überleben, müssen sie diese Veränderungen genau vorhersehen können.
>
> In der Ursuppe, die ich kürzlich in meinem Computer gebaut habe, starben all die Eleks aus, die schlechte Hellseher waren. Die besten Prognostiker dagegen bekamen Nachkommen, die manchmal besser abschnitten als ihre Erzeuger. So entstand in einem Evolutionsprozess schließlich eine perfekte Hellsehersippe" [Dew86]

In diesem Beispiel sind alle wichtigen Eigenschaften der genetischen Algorithmen aufgezählt:

1. Das *Individuum*, hier das Elek, das sich durch Generationen seiner Entwicklung weiter modifiziert und sich anzupassen sucht.
2. Der *Zivilisationsdruck* oder die *Umwelt*, d.i. der Faktor, der die Bewertung der Lebenseigenschaften ermöglicht, und an dem die Fähigkeit zum Überleben gemessen werden kann. Der Zivilisationsdruck in dem Beispiel besteht hier in der Fähigkeit der Vorhersage.

Wir wollen nun die Problematik näher betrachten. Besonders interessieren bei dieser Zusammenfassung die Anwendungsgebiete und die generalisierte Methode der Genetischen Algorithmen.

2.1.2 Anwendungsgebiete der Genetischen Algorithmen

Die Genetischen Algorithmen werden hauptsächlich im Bereich der "Optimierungsprobleme" eingesetzt. Hierbei ist es nicht wichtig, welcher Art die eigent-

liche Optimierung ist, es kann sich sowohl um eine numerische als auch um eine nicht-numerische Problemstellung handeln.

Viele numerische Probleme werden mit Hilfe von entsprechenden Programmteilen aus dem Bereich *Numerik* gelöst. Hier seien als Beispiel der Algorithmus der *regula falsi* und das *Newton*-Verfahren genannt[Eng84, Pre89].

Betrachten wir die Verfahren im Vergleich zu den Genetischen Algorithmen. Die numerischen Verfahren haben als spezialisierte Verfahren Vorteile in dem Zeitverhalten. Ihnen fehlt aber das, was Goldberg[Gol89] mit dem Begriff *Robustheit*[3] bezeichnet.

Unter Robustheit versteht Goldberg die Fähigkeit von Algorithmen, auch unter "widrigen" *Umweltbedingungen* korrekte Ergebnisse zu liefern. *Umweltbedingungen* nennt man die Problemstellung, auf die Algorithmen angewendet werden. Unter widrigen Umweltbedingungen versteht man dann Einflüße auf den Algorithmus, die entgegen der definierten Einschränkungen stehen. Hierzu gehören in der *Numerik* sehr häufig vorkommende Unstetigkeiten oder mehrere Extremalstellen.

Nimmt man als Algorithmus das oben benannte *Newton*-Verfahren und behandelt die Funktion $f(x) = -x^2$ als Umwelt.

Man kann diese Umgebung als gutartig bezeichnet, da sie genau ein Maximum hat und dieses schnell (ohne andere lokale Extreme) zu einem Ergebnis führt.

Es ist festzustellen, daß der Algorithmus für Kurven, die streng monoton ihrem Extremalen zustreben, eine zuverlässige und eindeutige Lösung hervorbringt.

Die läßt sich an der Iterationsformel darstellen, die allgemein die folgende Form hat

$$x^{(\nu+1)} = \rho(x^{(\nu)}) = x^{(\nu)} - \frac{f(x^{(\nu)})}{f'(x^{(\nu)})}, \nu = 0, 1, 2, \ldots$$

Setzt man nun dieses Verfahren auf eine "widrige" Umwelt an, so wird es

1. entweder nicht terminieren, oder
2. eine größere Menge von Lösungen nicht finden

Als Beispiel für eine solche Funktion kann hier die Sinusfunktion dienen. Diese Funktion hat eine unendlich große Anzahl von Maxima und führt mit Sicherheit zu einer Fehlersituation, da unendlich viele Maxima nicht erkannt werden.

Genetische Algorithmen zeichnen sich hier dadurch aus, daß die Wahrscheinlichkeit des Auftreffens mehrerer Maxima entsprechend der Anzahl der an der Suche beteiligten Individuen steigt.

Gegeben sei ein Gerät mit fünf Schaltern. Diese Schalter können die Werte 0 und 1 annehmen. Je nach Einstellung der Schalter gibt das Gerät eine bestimmte

[3] engl. robustness

Gewinnsumme aus. Gesucht wird die Schalterstellung mit dem maximalen Gewinn
Die normale Lösung des Problems würde so aussehen, daß alle Schalterstellung durchgetestet und die Ergebnisse bewertet werden.
Damit würden maximal $2^5 = 32$ Schalterstellungen ausgetestet.
Das entsprechende, auf Genetische Algorithmen arbeitende Programm wird mit deutlich weniger Schritten auskommen, da Informationen über das Verhalten des Gerätes wie Erbinformationen weiter bei der Suche nach der optimalen Lösung eingesetzt werden. Im weiteren wird ein *Genetischer Algorithmus* beispielhaft dargestellt. Zur generellen Charakterisierung der Genetischen Algorithmen[Gol89]:

1. Genetische Algorithmen arbeiten mit der Codierung einer Parametermenge, nicht mit einem Parameter.
2. Genetische Algorithmen suchen mit einer Menge, einer Population, von Punkten und nicht von einem Punkt aus.
3. Genetische Algorithmen bewerten die Ergebnisse der Operation und nicht von außen zugeführte Informationen.
4. Genetische Algorithmen berechnen ihre Weiterarbeit mit Hilfe von Regeln aus der Wahrscheinlichkeitsrechnung und nicht nach deterministischen Regeln.

2.1.3 Die Theorie der Genetischen Algorithmen

Terminologie

Zum besseren Verständnis der folgenden Abschnitte sollen an dieser Stelle die notwendigen Begriffe erläuternd definiert werden :

Individuum Ein Individuum $\mathcal{I}$ ist ein Element mit wenigstens einem Gen. Dieses Individuum wird durch das Gen definiert und erhält seine speziellen Ausprägungen durch das Gen.

Gen Zahlenkette, bestehend aus den Werten 0 und 1. Zur Veränderung des Gens gibt es drei, im weiteren Text definierte Operatoren: die **`Reproduktion`**, die **`Kreuzung`** und die **`Mutation`**. In der Literatur wird auch der Name **Chromosom** genutzt. Die Anzahl der Gene im Individuum wird durch die Variable *lchrom* bezeichnet.

Population Die Anzahl der existierenden Individuen. Ist diese Anzahl zu klein, so terminiert der Algorithmus nicht schnell genug bei einem hinreichend gutem Wert, ist dieser Wert jedoch zu groß, so wird die Rechenzeit für eine Generation zu groß. Die maximale Größe der Population wird durch die Variable *maxpop* dargestellt.

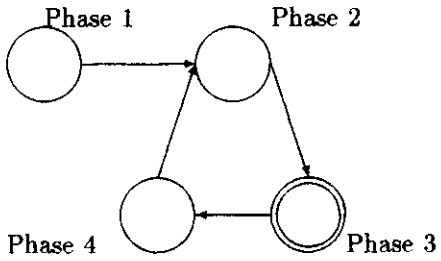
Fitness Das Beurteilungskriterium für die Fortpflanzung der Individuen. Die Aufstellung einer geeigneten Fitness-Funktion ist notwendig für die Effizienz eines Genetischen Algorithmus. Synonym verwende ich auch das Wort *Zivilisationsdruck* oder *Umwelt.*

Tabellarisch kann man eine Verbindung zwischen biologischen und Begriffen aus der Informatik aufbauen :

biologischer Begriff	Erklärung
Chromosom	Gen oder "String"
Gene	Eigenschaft, Eingenschaftsausprägung
Locus	Ort des Gen in der Kette
Genotyp	Struktur des Individuums
Phenotyp	Parametersatz, alternative Lösung
Epistasis	Nichtlinearität

Vorgehensweise

Das Vorgehen beim Einsatz von Genetischen Algorithmen läßt sich einfach als Regelkreis darstellen. Dieser Regelkreis hat die folgende Form:



Phase 1 : Startphase: Die Individuen werden durch den Zufallsgenerator erzeugt.

Phase 2 : Rechenschritt: Die Individuen passieren die Zielfunktion ("Umwelt").

Phase 3 : Beurteilungsphase: Die Anpassung der Individuen auf die Umwelt wird getestet und bewertet. Hier kann der Algorithmus terminieren.

Phase 4 : Fortpflanzungsphase: Neue Individuen werden erzeugt.

Die einzelnen Phasen werden noch weiter verfeinert :

Die Startphase

In der Startphase werden n Individuen $\mathcal{I}$ kreiert. Jedes Individuum x wird durch seine Gene repräsentiert, so das gilt :

$$x \in \{\mathcal{I} | x_k \in \{0,1\} für k = 1,2,\ldots,lchrom\}$$

mit $lchrom = ||x||$.
Ein Individuum $x \in \mathcal{I}$ wird also erzeugt durch das Anwenden des Zufallsoperators X auf jedes x_k. Sinnvollerweise definiert man eine Funktion $f(x) =$ `random()`, die den Zufallsoperator realisiert.

Wichtig hierbei ist, daß die Bandbreite[4] der Zufallsvariablen groß genug ist. Eine zu kleine Bandbreite, wie sie von einigen compilereigenen Generatoren erzeugt werden, vermitteln ein verkehrtes Bild von dem System.

Festgelegt werden muß dann noch die Mächtigkeit *lchrom* und die Anzahl der Individuen $popsize = n$. Diese Variablen müssen nun so angepasst werden, daß sie der Problemstellung entsprechen. Betrachtet man nun ein Problem, so kann man die Anzahl der möglichen Lösungen und die Größe der Definitionsmenge abschätzen. Die Definitionsmenge wird durch den Namen $\mathcal{D}$ gekennzeichnet. Damit ergibt sich die Anzahl der Gene *lchrom* durch die Berechnungsformel

$$lchrom = \log_2(||\mathcal{D}||)$$

Voraussetzung der Machbarkeit ist, daß die Anzahl der Eingaben abzählbar ist. Sollte dieses nicht der Fall sein, so muß der Definitionsraum diskretisiert werden. Die Größe der Population ist dagegen kein Fixum, sondern abhängig von dem Verhalten der Individuen in der Umwelt. Somit sollte die Größe der Population gelten : $1 \leq popsize \leq maxpop$.

Der Rechenschritt

Das Individuum passiert die Zielfunktion, d.h. es wird für ein bestimmtes Individuum ein Ergebnis erzielt. Wichtig ist, das die Ergebnisse nicht nach dem Durchlauf ausgewertet werden, sondern daß die Beurteilung erst nach Durchführung aller Durchläufe vorgenommen wird.

Die Zielfunktion sollte wie eine mathematische Funktion betrachtet werden, das Ergebnis ist durch die Eindeutigkeit einer Funktion eine Charakterisierung des Individuums.

Die Beurteilungsphase

Diese Phase folgt dem Rechenschritt. Sie könnte eigentlich mit der Fortpflanzungsphase zusammengefaßt werden. Allerdings ist es zur Betrachtung besser, die Phasen zu trennen.

[4] d.i. die Anzahl der Bits, die für Bildung der Zufallszahl benutzt werden

Die Beurteilungsphase vergleicht die Ergebnisse der einzelnen Individuen miteinander. Damit kann man eine Hierarchie der Individuen aufbauen. Das Ergebnis kann als direktes Maß der Fitness betrachtet werden. Hierzu werden die Ergebnisse addiert. Damit erreicht man einen Wert F für die Gesamtfitness. Die Berechnung erfolgt also nach der Formel

$$F = \sum_{k=1}^{popsize} f(x_k)$$

Mit diesem Wert kann man den prozentualen Wert für die Fitness berechnen. Dieser prozentuale Wert für die Fitness berechnet sich als Quotient [5]

$$f_p(x) = 100 \cdot \frac{f(x)}{F}$$

Als ein weiterer Wert kann die Abweichung des Wertes $f(x)$ von einem Erwartungswert $\bar{f}$ betrachtet werden. Dieser Erwartungswert läßt sich über das arithmetische Mittel, also über

$$\bar{f} = \frac{F}{popsize}$$

berechnen.
Damit berechnet sich die Abweichung von dem erwarteten Wert über die Formel

$$\sigma(x) = |f(x) - \bar{f}|$$

An dieser Stelle muß man sich Gedanken über das Abbruchkriterium machen. Dieses Abbruchkriterium kann entweder das Erreichen eines bestimmten Schwellenwertes oder einer bestimmten Generation sein. Dieses richtet sich nach dem Eintreten und der Berechenbarkeit[6] eines Ereignisses. Das zuerst erreichte Kriterium ist hinreichend für den Abbruch.

Die Fortpflanzungsphase

Die nächste folgende Phase ist die Fortpflanzungsphase. Diese macht die eigentliche Besonderheit der *Genetischen Algorithmen* aus. Es gibt drei Methoden der Weiterentwicklung von Individuen :

1. die Reproduktion[7], also die unveränderte Fortführung des Individuums,

[5] Goldberg rechnet mit einem etwas anderen Wert: $pselect_i = \frac{f(x)}{\sum_j}$

[6] oder der Bestimmbarkeit. Es gibt Fitnessfunktionen, deren Extrema nicht bestimmbar sind. In diesem Falle kann man nur über eine Abschätzung der Qualität der Fitness oder die Anzahl der Generationen als Abbruchkriterium gelten.

[7] engl. reproduction

2. die Kreuzung[8], der Austausch von Genen zwischen zwei Individuen und

3. die Mutation[9], die zufällige Veränderung der Erbinformationen.

Diese Mechanismen werden nach bestimmten Regeln angewendet.
Zuerst soll die Reproduktion betrachtet werden. Hierzu stelle man sich ein Roulette-Rad im herkömmlichen Sinn vor. Wie bei der Beurteilung festgestellt, kann man die Fitness über die Funktion $f_p(x)$ darstellen. Dieser Operator beschreibt die Fitness eines einzelnen Individuums im Vergleich zu der Population.

Würde man nun die Fitness das Individuums als Anteil auf diesem Rad darstellen, so kann man diesen exakten Anteil berechnen nach der Formel:

$$\mathcal{A}(x) = f_p(x) \cdot 3,6$$

Da das Drehen eines idealen Roulette- Rades ein Zufallsexperiment ist, ebenso wie das Werfen eines idealen Würfels, ergibt sich aus der Berechnung der Anteile des Rades nach der Fitness, daß die Wahrscheinlichkeit eines Individuums sich zu reproduzieren, abhängig von eben dieser Fitness ist. Mit anderen Worten bedeutet dieses, daß die Wahrscheinlichkeit der Weiterexistenz eines Individuums exakt proportional zu dem Anteil an der Gesamtfitness F ist. Der erste Schritt der Fortpflanzung ist nun, für jedes benötigte Individuum der nächsten Generation das Roulette- Rad genau einmal zu drehen.

Der nächste Schritt in der Fortpflanzung ist die Kreuzung. Dabei werden die erzeugten Individuen zufallsmäßig paarweise zusammengefaßt. Man könnte dieses Zusammenfassen mit dem Wort "Paarung" beschreiben. Diese Paare werden Teile ihrer Gene austauschen. Hierzu wird wiederum eine Stelle in dem Gen-String markiert und die nachfolgenden Gene werden ausgetauscht.
Bsp.:
Gegeben sind die beiden folgenden Zeichenketten :
$A_1 =$ `01101010`
$A_2 =$ `11011011`
Nachdem eine Kreuzungsstelle mit $p = 5$ mit Hilfe des Zufallsgenerators festgelegt wurde, ergibt sich das Bild :
$A_1 =$ `01101|010`
$A_2 =$ `11011|011`
und nach der Kreuzung :
$A_1' =$ `01101011`
$A_2' =$ `11011010`
Auf diese Art und Weise werden Erbinformationen ausgetauscht.

Der letzte Schritt der Fortpflanzung ist die Mutation. Hierbei nimmt man an, daß mit einer relativ geringen Wahrscheinlichkeit p sich die Gene innerhalb eines Generationswechsels ändern.

[8] engl. crossover
[9] engl. mutation

Formelmäßig läßt sich dieses wie folgt ausdrücken :

$$\forall x \in \mathcal{I} : x_i = \begin{cases} \bar{x}_i & : \quad X = 1 \\ x_i & : \quad X = 0 \end{cases}$$

mit sehr kleiner Eintrittswahrscheinlichkeit $P(X = 1)$. In der Literatur wird dieser Wert meistens mit $\ll 0,01$ angenommen. Dieses spiegelt die in der Natur sehr kleine Mutationswahrscheinlichkeit wieder. Die Mutation stellt eine Sammlung von unerklärbaren Änderungen in der Erbsubstanz dar, die auf diese Art nachempfunden werden.

Durch die Anwendung dieser Regeln wird eine Anzahl neuer Individuen erzeugt, die wieder in den Regelkreis eintreten können. Durch den Auswertungsschritt wird dabei sowohl die alte Erbinformation gespeichert, andererseits durch die Mischung von Erbgut eine wahrscheinlich bessere Fitness erzeugt. Die Wirkungsweise dieses Regelkreises soll im folgenden Beispiel verdeutlicht werden.

2.1.4 Darstellung eines Genetischen Algorithmus

Das folgende Beispiel ist in dieser Form dem Buch von Goldberg[Gol89] entnommen. Es behandelt die Durchführung eines Generationsschrittes für eine Generation. Wir betrachten die Funktion $f(x) = x^2$ für die Menge der natürlichen Zahlen im Bereich $[0, \ldots, 31]$. Die Größe der Definitionsmenge $\mathcal{D}$ beträgt also maximal 32 Werte. Mit der oben beschriebenen Formel $lchrom = \log_2(\|\mathcal{D}\|)$ ergibt sich eine Größe von 5 Genen, also $lchrom = 5$.
Der Einfachheit halber bezeichnet jedes Gen eine Dualstelle[Kla83], also $x_k \cdot 2^k$ und

$$f(x) = (\sum_{k=0}^{5} x_k \cdot 2^k)^2$$

Man erzeugt nun 4 Individuen mit den folgenden Werten :

Individuum	Genaufbau
1	01101
2	11000
3	01000
4	10011

Diese 4 Individuen kann man nun auf die oben beschriebene Funktion ansetzen, d.h. man führt die Rechenphase durch.

Nach Durchführung des Rechenschrittes ergibt sich folgendes Bild :

Individuum	Wert von x	Wert von x^2
1	13	169
2	24	576
3	8	64
4	19	361

Man berechnet zusätzlich die Werte von F, $f_p(x)$, $\bar{f}$ und den maximalen Wert $f_{max}(x)$. Es ergibt sich :

Art	Wert
F	1170
$\bar{f}$	293
$f_{max}(x)$	576

Die vollständige Tabelle hat das folgende Bild :

Individuum	Genaufbau	x-Wert	$f(x)$	$f_p(x)$	$\sigma(x)$
1	01101	13	169	14	0.58
2	11000	24	576	49	1.97
3	01000	8	64	6	0.22
4	10011	19	361	31	1.23

Mit diesen Werten kann man nun die Fortpflanzungsphase durchführen. Dazu tritt man zuerst in die Reproduktionsphase ein, und stellt sich also ein Roulette-Rad vor. Auf diesem Roulette- Rad erhalten die Individuen entsprechend ihrer Fitness Anteile. In diesem Beispiel werden vier neue Individuen erzeugt. Entsprechend der obigen Regel wurde Individuum 1 einmal, Individuum 2 zweimal, Individuum 3 nicht und Individuum 4 einmal gezogen.

Die neue Tabelle hat also das folgende Aussehen :

Individuum	Genaufbau
1'	01101
2'	11000
3'	11000
4'	10011

Der nächste Schritt umfaßt die Paarung der Individuen. Damit wird die obige Tabelle erweitert :

Individuum	Genaufbau	Ehepartner
1'	01101	2'
2'	11000	1'
3'	11000	4'
4'	10011	3'

Im nächsten Schritt wird ein Kreuzungspunkt für die entsprechenden Ehepartner berechnet. Als Kreuzungspunkt des Ehepaares 1'-2' wird die Stelle 4 ermit-

telt, für das Ehepaar 3'-4' ist es die Stelle 2.

Nach Durchführung der Kreuzung ergibt sich damit das Bild :

Individuum	Genaufbau	Ehepartner	Kreuzungspunkt	neues Individuum
1'	01101	2'	4	01100
2'	11000	1'	4	11001
3'	11000	4'	2	11011
4'	10011	3'	2	10000

Man kann die Werte für die neuen Individuen[10] berechnen:

Individuum	Genaufbau	x- Wert	x^2- Wert
1'	01100	12	144
2'	11001	25	625
3'	11011	27	729
4'	10000	16	256

Vergleicht man nun die Ergebnisse zwischen den Generationen :

	Generation 1	Generation 2	Verbesserung
Gesamtfitness F	1170	1754	49,91 %
Durchschnittsfitness $\bar{f}$	293	439	49,82 %
maximaler Wert x	576	729	26,56 %

so stellt man eine deutliche Verbesserung in der Fitness zwischen den Generationen fest, teilweise bis nahezu 50 %.

2.1.5 Schemata

Betrachtet man das vorige Beispiel, so stellt man fest, daß sich bestimmte optimale Ergebnisse dann ergeben, wenn die Gene gewissen Schemata entsprechen. Ein Schema[Hol68] ist ein bestimmtes Auftreten der Symbole 0 und 1. Um ein Schema zu beschreiben, führt man noch zusätzlich das Symbol $*$ ein.

Dieses zusätzliche Symbol wird von Goldberg als "*don't care*"-Symbol, also als "*man beachte es nicht*"-Symbol bezeichnet. Analog zu der Programmiersprache Prolog könnte man von einem "anonymen Symbol" sprechen.

Dieses Symbol übernimmt lediglich eine Platzhalterfunktion. Somit beschreibt die Zeichenkette $*010*$ die folgenden Individuen : 00100, 10100, 00101, 10101.

Allerdings ist das Symbol "$*$" nur ein Metasymbol, es wird niemals in einem Genetischen Algorithmus wirklich eingesetzt, sondern dient ausschließlich der Beschreibung von auftretenden Gleichheiten.

Man betrachtet die Anzahl der auftretenden Schemata für eine fixe Anzahl von Genen. Jede Stelle kann 3 Werte annehmen, also gibt es 3^{lchrom} verschie-

[10] Eine Mutation findet nicht statt :
Der Erwartungswert für eine Mutation beträgt $\mu = 5 \cdot 4 \cdot 0,001 = 0,02$

dener Schemata. In unserem Beispiel mit $lchrom = 5$ gibt es also $3^5 = 243$ verschiedene Schemata.

Diese Erweiterung der Betrachtung macht im Ablauf dadurch Sinn, daß sie den Aufwand bei großen Systemen verringert. Denn betrachtet man jedes Individuum einzeln, so hat man genau 4 einzelne Informationen zur Verfügung. Durch den Vergleich ergeben sich neue Informationen die helfen, eine Beurteilung der Individuen durchzuführen.

Betrachte man anderseits ein bestimmtes Individuum der Länge 5. Dieses Individuum ist Mitglied eines Schemas , weil jede Position mit einem bestimmten Symbol oder dem Anonymen Symbol besetzt ist. Normalerweise hat jedes Individuum 2^{lchrom} Schemata, und eine Population der Größe *popsize* hat zwischen 2^{lchrom} und $popsize \cdot 2^{lchrom}$ Schemata, abhängig von der Verteilung der Population.

Man kann nun feststellen, daß sich verschiedene Schemata recht unterschiedlich während einer Forpflanzungsphase verhalten. Man betrachte die beiden Schemata "1∗∗∗0" und "∗∗11∗". Bei der Kreuzung wird das erste Schema fast sicher zerstört. Das zweite Schema dagegen verhält sich sehr viel stabiler, insbesondere dadurch, daß die Kreuzung schon zwischen den beiden Einsen einsetzen muß. Ein Schema dieser Art, mit kurz hintereinanderliegenden Genen ist gegen eine Zerstörung durch Kreuzung sicher, es wird auch "Block"[11] genannt. Es vererbt sich von Generation zu Generation ohne besondere Verwaltung.

Die sich stellende Frage lautet: Wieviele Schemata sind zu erzeugen ? Nimmt man eine Population von n Individuen, so stellt sich heraus, daß die optimale Anzahl im Bereich von (n^3) liegt[12]. Dieses läßt sich vorzüglich mit der Anzahl der Funktionsauswertungen vergleichen. Diesen Effekt nennt Goldberg "implicit parallelism" oder "implizierter Parallelismus".

2.1.6 Zusammenfassung

Dieses Kapitel sollte anhand eines Beispieles und einiger Erklärungen ein grundlegendes Verständnis über die Art der *Genetischen Algorithmen* formen. Hier sollen die wesentlichen Eigenarten zusammengefasst werden :
Es gibt vier wichtige Unterschiede zwischen konventionellen und genetischen Algorithmen:

1. Genetische Algorithmen manipulieren direkt die Kodierung. Sie manipulieren Entscheidungs- und Kontrollvariablen, während andere Algorithmen über Funktionen rechnen. Weil Genetische Algorithmen im Kodierungsbereich arbeiten, ist es schwierig Fehler zu begehen, auch wenn die Funktionen dazu neigen.

[11] im Orginal: "building blocks"
[12] vgl. [Gol89], Kap.2

2. Genetische Algorithmen suchen aus einer Population heraus, nicht von einem einzelnen Punkt an. Es wird also nicht von einem fixen Punkt heraus gesucht, sondern von mehreren Anfangspunkten. Die Wahrscheinlichkeit, ein Nebenmaximum fehlerhafterweise als Maximum zu erkennen sinkt.

3. Es findet eine blinde Suche statt, keine informierte mit Vorwissen über alte Ergebnisse. Die einzige ausgewertete Information ist der Ergebniswert[13].

4. Es wird mit stochastischen Mitteln, d.h. mit Mitteln der Wahrscheinlichkeitsrechnung gesucht und nicht mit deterministischen Regeln. Die Auswertungsregeln (z.B. bei der Kreuzung und bei der Mutation) werden mit geeigneten Zufallsgeneratoren ermittelt und müssen nicht nur mit Auswertungen der Ergebnisse arbeiten.

2.2 Einführung in die Datenstrukturen

Unter einer Datenstruktur versteht man eine *Menge* von *Daten* und eine Menge von Beziehungen zwischen den Daten. In der Literatur werden die verschiedensten Definitionen gegeben.

Wie aus der Literatur zu entnehmen ist,

> "...gerichtete Graphen sind zunächst einmal geeignet, um strukturierte Daten zu beschreiben."[Maj76]

Aber zusätzlich zu der Darstellung der Daten im System, mit der sich die nächsten Kapitel beschäftigen, sind die Operationen in dem System von Interesse. Diese Operationen sind die dynamischen Komponenten.

2.2.1 Klassifizierung der möglichen Suchverfahren

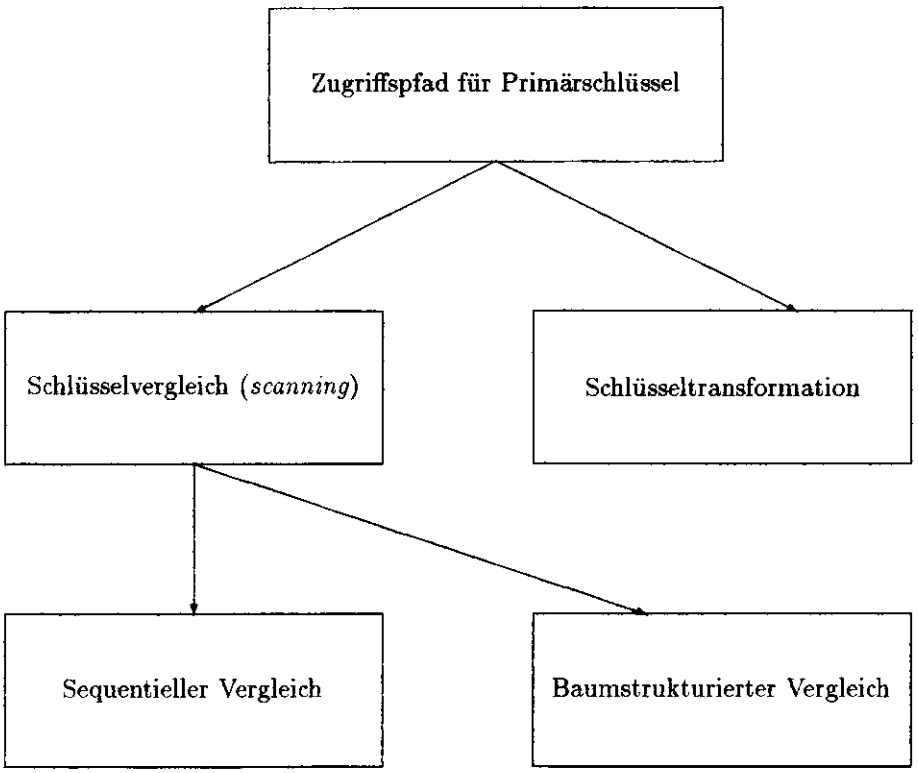
In diesem Abschnitt sollen die Grundlagen von Suchverfahren auf Daten besprochen werden, um so eine Grundlage sowohl für die Aufstellung der Fitnessfunktion als auch für die Bearbeitung der Daten.
Man kann die Organisation der Daten in zwei Kathegorien einteilen :

1. Hierarchische Suchverfahren, die mit Schlüsselvergleich in einer beliebigen Struktur von Daten nach dem gesuchten Element fahnden, und

2. Berechnende Suchverfahren, die über die Berechnung der Position aus dem Primärschlüssel den Ort des Elementes ermitteln.

Man kann die einzelnen Suchverfahren entsprechend ihrem Verhalten und der Art der Datenablage im Rechner klassifizieren. WEDEKIND[Wed76] führt hierzu das folgende Schema an:

[13] engl. payoff value



Zu den einzelnen Gliederungsstufen gehören :

Sequentieller Vergleich :

- Cartesische Speicherungsstruktur (physisch sequentiell)
- String- Struktur (logisch sequentiell)

Baumstrukturierter Vergleich :

- Digitalbäume
- Binärbäume
- Mehrwegbäume

Schlüsseltransformation : Gestreute Speicherungsstrukturen mit

- direkter Adressierung

- indirekter Adressierung

Diese verschiedenen Datenablagearten unterscheiden sich nicht nur durch ihren logischen Aufbau, sondern auch durch qualitative und quantitative Merkmale. Hierzu gehören :

1. Zugriffszeiten
2. Schreib- und Lesezugriffe
3. Organisationsaufwand
4. Plattenplatzbedarf, u.a.

Im weiteren wird die Betrachtung auf die Klasse der "Schlüsselvergleichenden Suchverfahren" beschränkt. Zu diesen Suchverfahren zählen sowohl die *sequentiellen Suchverfahren* als auch die *baumstrukturierten Suchverfahren*. Diese verschiedenen Organisationsformen sollen im weiteren näher betrachtet werden.

2.2.2 Die einzelnen Organisationsformen

Sequentielle Organisationsformen

Bei der *Sequentiellen Organisationsform* werden die einzelnen Datenelemente entsprechend eines *Ordnungssystems* in dem Medium sortiert abgelegt.

Unter einem Ordnungssystem versteht man hierbei ein reines Relationensystem mit einer zweistelligen Relation, die man Ordnung nennt[Mue79]. Hierbei benutzt man ausschließlich lineargeordnete[14] Mengen, d.h. Daten.

Eine Menge von Daten, die entsprechend eines Ordnungssystems organisiert ist, nennt man eine *lineare Liste*[Den77]. Im wesentlichen zeichnet sich eine solche *lineare Liste* durch die folgenden Definitionen aus :

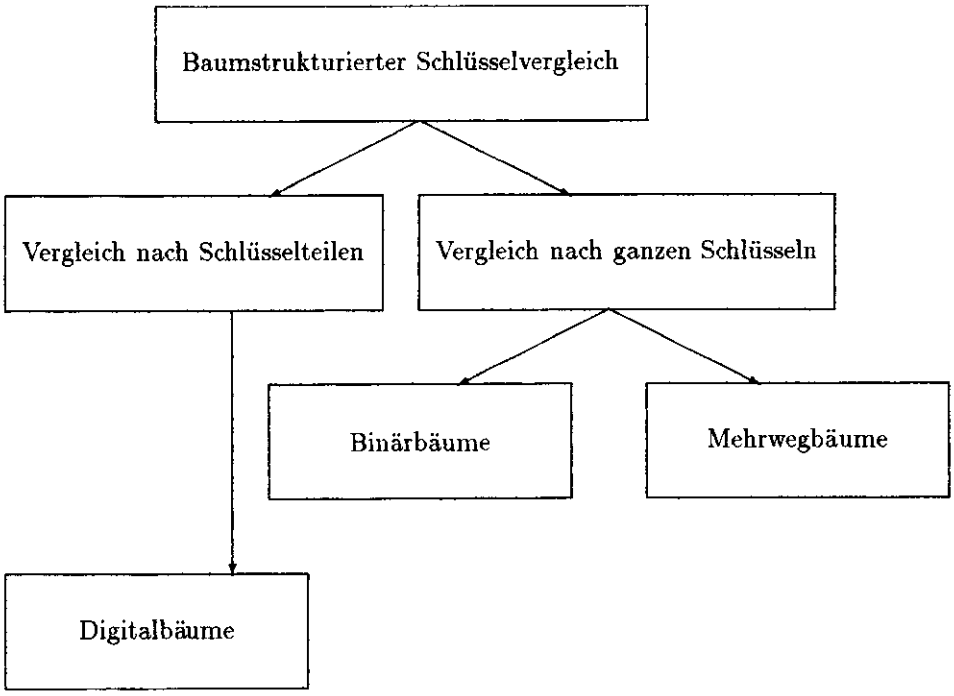
1. es gibt genau ein Listenelement, das keinen Vorgänger hat (Listenanfang)
2. es gibt genau ein Listenelement, das keinen Nachfolger hat (Listenende)
3. alle übrigen Listenelemente haben genau einen Vorgänger und einen Nachfolger

Es ist deutlich zu erkennen, daß die *Sequentielle Datenorganisation* eine entartete, spezielle Baumstruktur ist. Deshalb werden sie im weiteren nicht mehr gesondert betrachtet.

[14] in der Literatur auch *Kette* oder *totale Ordnung* genannt

Baumstrukturierte Organisationsform

Die Baumstrukturierten Organisationsformen lassen sich nach WEDEKIND[Wed76] weiter aufgliedern, so daß sie dem folgenden Schema entsprechen:



Es werden die Betrachtungen ohne Begrenzung der Allgemeingültigkeit auf ganzschlüssel- vergleichende Strukturen beschränkt. Mehrwegbäume sind Datenbäume , bei denen jeder Knoten Ursprung von n Söhnen sein darf, wobei n implementierungsabhängig ist. Beschränkt man n auf den Wert 2, so spricht man von einem Binärbaum. Allen Bäumen ist gemein, daß sie genau einen Startknoten haben.

2.2.3 Eigenschaften von Organisationsformen

Um Vor- und Nachteile der verschiedenen Datenorganisationen darstellen zu können, muß man die Maße definieren, mit denen man diese Beurteilung durchführen kann. Das Maß sollte hierbei einen Wert für die Datenorganisation darstellen, ohne aber spezifisch für die Datenorganisation zu sein.

Diese Maße sollten sich orientieren an den wichtigen Beurteilungspunkten jeder Datenorganisation. Diese sind :

1. Platzbedarf
2. Zeitaufwand
   - zum Suchen von Daten
   - zum Einfügen von Daten
   - zum Löschen von Daten
3. Organisationsaufwand

Eine ideale Datenstruktur ist eine Struktur, die bei minimalen Platzbedarf und Organisationsaufwand den geringsten Zeitaufwand hat.

Bei der Betrachtung dieser Aufgabenstellung stellt man fest, daß die eigentliche Problemstellung zweigeteilt ist. Diese zwei Teile der Problemstellung sind

1. die Topologische Optimierung von Zugriffsgraphen, also die physikalische Anordnung der Daten im Rechner, und
2. die Optimierung der Ordnung der Daten, also der Aufbau der Zugriffspfade im System. Interessant für die Aufgabenstellung ist hierbei die Zeit, die in einer Datenstruktur für das Auffinden von Informationen benötigt wird. Diese Zeit, durch die Variable t_{search} gekenntzeichnet, kann durch die Anzahl der Suchschritte[15] im System gekennzeichnet werden. Verständlich ist, daß zuviel vorgenommene Suchschritte hierbei Zeitverlust bedeuten, damit Ineffizienz und sollten vermieden werden.

Um nun eine Fitnessfunktion zu ermitteln, muß man also die beiden Teilaufgaben zuerst gesondert, dann in Kombination betrachten. Die Aufgabe besteht nun in der Umsetzung dieser Theorie in eine praktische Versuchsanordnung und die experimentelle Untersuchung der dabei auftretenden Phänomene.

[15] Ein Suchschritt ist hierbei das Aufsuchen eines Datenelementes und der Vergleich mit dem vorgegebenen Schlüssel

Kapitel 3

Das Individuum

Wie in dem einleitenden Kapitel über *Genetische Algorithmen* erklärt wurde, benötigt man für die Durchführung des Algorithmus sowohl eine Fitnessfunktion als auch ein Individuum, daß der Fitnessfunktion (dem sogenannten Zivilisationsdruck) unterworfen wird.

Dieses Individuum besteht in der Literatur[Gol89] aus einer *Begrenzten Kette* von 0- und 1-Zuständen, die sich entsprechend dem Verhalten in der "Natur" anpassen.

In diesem Kapitel geht es um das Problem, eine Datenstruktur, die aus verketteten Daten besteht, auf eine solche Binärstruktur abzubilden.

Diese Abbildung muß eineindeutig sein, also eine echte Darstellung der Struktur dieser Daten. In der Literatur, die im weiteren herangezogen wird, ist die Darstellung einer Datenstruktur als Graph gegeben. Bei näherer Betrachtung stellt man fest, daß diese Abbildung in Form eines Isomorphismus[1] geschieht. Es gibt Literaturstellen, die eine Datenstruktur direkt über einen Graphen definieren[2].

Diese spezielle Form der Abbildung ermöglicht es, im weiteren Kontext die Worte "*Datenstruktur*" und "*Graph*" synonym zu benutzen.

3.1 Graphentheoretische Grundlagen

Zur Definition der Abbildung einer Struktur müssen einige graphentheoretische Grundlagen eingeführt werden, die für den weiteren Ablauf notwendig sind.

Betrachtet man die Daten isoliert, also als einzelne Datenpunkte, so lassen sich diese auch als *Knoten* auffassen. Um jetzt die Daten in eine Beziehung zu

[1] Man spricht von einem Isomorphismus genau dann, wenn es zwischen zwei Strukturen A und A' eine bijektive Abbildung gibt, so daß man A in A' überführen kann und umgekehrt [Arb81, Lip81]

[2] vgl. [Mue79],Def. 15.1

bringen, müssen diese verbunden und in Bezug gebracht werden. Hierzu definiert man Kanten, also Verbindungen zwischen zwei Knoten. Mit Hilfe der Kanten und Knoten kann man nun einen Graphen definieren. Diese Definition stammt von DÖRFLER[Doe77], die in ihrer Substanz identisch ist mit der Definition von EVEN[Eve83] und KNUTH[Knu68]:

Definition 3.1 (Graph) *Gegeben sei die Menge aller Knoten $\mathcal{V}$ mit $\mathcal{V} = \mathcal{V}(X)$. Man definiert einen Graphen über eine Menge von Paaren $e = [x, y]$ mit $x, y \in \mathcal{V} \wedge x \neq y$. Sei nun eine Menge $\mathcal{E}$ als Menge aller e definiert, also als Menge aller Verbindungen oder Kanten.*

Aus diesen Voraussetzungen definiert man nun einen Graphen X als Paar

$$X = (\mathcal{V}, \mathcal{E})$$

Alternativ gibt es noch andere Definitionen, die sich aber in den Grundsätzen gleichen. Zum Vergleich soll an dieser Stelle die Definition von MÜLLER [Mue79] dargestellt werden.

Definition 3.2 (Alternative Definition des Graphen) *Ein Graph ist ein Tripel $\mathcal{G} = (P, K, v)$, bestehend aus einer Menge P von Punkten, (Ecken, Knoten), einer Menge K von Kanten und einer Funktion v, die*

1. *jeder Kante aus K zwei (nicht unbedingt verschiedene) Punkte aus P zugeordnet (ungerichteter Graph) bzw.*
2. *jeder Kante aus K ein geordnetes Paar von Punkten aus P zuordnet (gerichteter Graph). Kurz:*
   (a) *$\mathcal{G} = (P, K, v)$ ist ungerichteter Graph $\leftrightarrow$ P, K sind Mengen $\wedge v|K \rightarrow \{\{a, b\}|a, b \in P\}$*
   (b) *$\mathcal{G} = (P, K, v)$ ist gerichteter Graph $\leftrightarrow$ P, K sind Mengen $\wedge v|K \rightarrow P \times P$*

Gilt $v(k) = \{a, b\}$, so sagt man ($k \in K; a, b \in P$): a und b sind Endpunkte von k. Gilt $v(k) = (a, b)$, so heißt $a = v_A(k)$ Anfangspunkt von k und $b = v_E(k)$ Endpunkt von k. In beiden Fällen sagt man: k verbindet a mit b.

a (und auch b) indiziert mit k.

Hierbei gilt, das x und y bei einem ungerichteten Graphen keiner Ordnung unterliegen. Bei einem gerichteten Graphen impliziert sich eine Ordnung wie an anderer Stelle beschrieben. Ist eine Struktur aus Knoten und Kanten wie oben genannt gegeben, so spricht man von einem *schlichten Graphen.* Die Bedingung $x \neq y$ verbietet hierbei *Schlingen*[3]. Sollen in den weiteren Anwendungen Schlingen zugelassen werden, so wird dieses explizit angeführt.

[3] das sind Kanten der Form $[x, x]$

Im weiteren soll die folgende Sprechweise genutzt werden, die von DÖRFLER eingeführt wurde:

Eine Kante $[x, y]$ verbindet ihre Endpunkte x und y. Sind x,y durch eine Kante verbunden, so heißen x,y adjazent und x Nachbar von y. Ein Knoten x und eine Kante $e = [x, y]$ heißen inzident und zwei Kanten heißen adjazent, wenn sie einen gemeinsamen Endpunkt besitzen.

Als Erweiterung der Begriffes 'Graph' soll nun die folgende Definition dienen:

Definition 3.3 (Gerichteter und Ungerichteter Graph) *Für einen beliebigen Graphen X mit $X = (\mathcal{V}, \mathcal{E})$ sei $e_1 = [x, y]$ und $e_2 = [y, x]$. Ein Graph heißt "Gerichteter Graph" genau dann, wenn gilt*

$$\forall e_1, e_2 \in \mathcal{E} : e_1 \neq e_2$$

Ein Graph heißt "Ungerichteter Graph" genau dann, wenn gilt

$$\forall e_1, e_2 \in \mathcal{E} : e_1 = e_2$$

Diese obrigen Definitionen reichen hier nicht aus, so daß sie im folgenden durch die Definitionen der Begriffe *Wald* und *Baum* ergänzt werden müssen. Diese Spezialformen der Graphen sind für die weiteren Betrachtungen von großer Bedeutung.

Wälder und Bäume

Eine bestimmte Form der Anordnung von Kanten sind *Kreise* oder *Zyklen*. Diese Zyklen haben gerade bei bestimmten Anwendungen von Graphen große Bedeutung, zum Beispiel beim "*Traveling Salesman Problem*", bei dem in einem Graphen der größte Zyklus (die beste Rundreise) gesucht wird. Zyklen lassen sich wie folgt definieren:

Definition 3.4 (Zyklus) *Gegeben sei ein beliebiger Graph X. Kann man nun von einem Knoten x einen Weg über Kanten im Graph so machen, daß man wieder am Punkt x endet, ohne daß eine Kante mehr als einmal begangen wurde, so spricht man von einem Zyklus oder von einem Kreis.*

Außerdem wird die folgende Definition benötigt:

Definition 3.5 (Schlichter Graph) *Gegeben sei ein Graph $\mathcal{X}$. Hat $\mathcal{X}$ keine Zyklen und keine Mehrfachkanten, so spricht man von einem "schlichten" Graphen.*

Mit Hilfe dieser Definition ist es nun recht einfach, den Begriff eines *Waldes* zu definieren:

Definition 3.6 (Wald) *Gegeben sei ein beliebiger, gerichteter, schlichter Graph X. Kann man in diesem Graphen keinen Zyklus finden, d.h. der Graph ist zyklenfrei, so nennt man diesen Graphen einen "Wald".*

Eine weitere Einschränkung der Datenstrukturen ist bei der Definition des Baumes gegeben.

Betrachten wir zuerst eine bestimmte Form von Graphen, die unter den Begriff der *zusammenhängenden Graphen* fallen. Diese *zusammenhängenden Graphen* werden wie folgt definiert:

Definition 3.7 (Zusammenhängende Graphen) *Ein beliebiger Graph X heißt genau dann zusammenhängend, wenn es von jedem Knoten x mindestens einen Weg zu einem beliebigen Knoten y des Graphen X gibt.*

Verknüpft man diese Eigenschaft mit der des Waldes, so erhält man die Definition des *Baumes*. Diese Art von Graphen unterteilt sich wieder in verschiedene Unterarten. Die wichtigste Unterart soll in der nächsten Definition erwähnt werden.

Definition 3.8 (Baum) *Ein Baum ist ein zusammenhängender Wald. Zusätzlich kann man die folgende Festlegung treffen: Existiert ein ausgezeichnetes Element w, das als Wurzel*[4] *bezeichnet wird mit der Eigenschaft, daß dieses Element keinen Vorgänger hat und von dem aus jedes Element a erreichbar ist, so spricht man von einem "Wurzelbaum".*

In der weiteren Betrachtung wird mit dem Begriff *Baum* impliziet der *Wurzelbaum* gemeint.

In einem anderen Zusammenhang ist der Begriff des *Gerüsts* definiert. Wichtig im Zusammenhang mit den graphentheoretischen Grundlagen ist die folgende Definition:

Definition 3.9 (Gerüst) *In jedem zusammenhängenden Graphen $\mathcal{G}$ gibt es einen Teilgraphen $\mathcal{G}'$, so daß alle Knoten Teil des Teilgraphen sind. Dieser Teilgraph heißt "spannender Graph" $\mathcal{G}'$ zu $\mathcal{G}$.*
Und außerdem:
Ist $\mathcal{G}'$ ein Baum, so heißt $\mathcal{G}'$ spannender Baum oder Gerüst auf $\mathcal{G}$.

Mit diesen Hilfsmitteln kann man nun die graphentheoretischen Eigenschaften von Datenstrukturen beschreiben[5].

3.2 Definition des Individuums

Wie schon oben beschrieben, interessiert hauptsächlich die Möglichkeit der Darstellung des Individuums. Da die Daten als solche gegeben sind, läßt sich das Individuum als Kantenmenge $\mathcal{E}$ verstehen. Insbesondere ist wichtig, daß sich jede Kante als Produkt einer Relation $e = [x, y]$ darstellen läßt. Hierfür eignet sich insbesondere die sogenannte "Adjazenzmatrix":

[4] engl. Root
[5] vgl. Seite 21

Definition 3.10 (Darstellung von ungerichteten Graphen durch Matrizen) *Seien Daten als Knoten und die Verbindungen zwischen den Daten als Kanten eines Graphen definiert. Die Anzahl der Knoten in dem Graphen sei definiert über* $n = |\mathcal{G}|$

Es sei $\mathcal{G} = (\{p_1, \ldots, p_n\}, \{k_1, \ldots, k_n\})$ *ein endlicher, ungerichteter Graph.*

1. *Die Inzidenzmatrix von* $\mathcal{G}$ *ist eine* $n \times m$*- Matrix* $B_{\mathcal{G}} = (b_{i,j} | i = 1, \ldots, n \wedge j = 1, \ldots, m)$ *über* $\{0, 1\}$ *mit*

$$b_{i,j} = \begin{cases} 1 & : \textit{falls } p_i \textit{ ein Endpunkt von } k_j \textit{ ist} \\ 0 & : \textit{sonst} \end{cases}$$

2. *Die Adjazenzmatrix von* $\mathcal{G}$ *ist eine* $n \times n$*- Matrix* $A_{\mathcal{G}} = (a_{i,j} | i, j = 1, \ldots, n)$ *mit*

$$a_{i,j} = \begin{cases} 1 & : \textit{falls } p_i \textit{ und } p_j \textit{ in } \mathcal{G} \textit{ durch eine Kante verbunden sind.} \\ 0 & : \textit{sonst} \end{cases}$$

Analog werden diese Matrizen für gerichtete Graphen definiert.

Definition 3.11 (Darstellung von gerichteten Graphen durch Matrizen) *Sei* $G = (P, K)$ *ein endlicher gerichteter Graph ohne Schlingen mit* $P = \{p_1, \ldots, p_n\}, K = \{k_1, \ldots, k_m\}$.

1. *Die Matrix* $B_{\mathcal{G}} = (b_{i,j} | i = 1, \ldots, n \wedge j = 1, \ldots, m)$ *mit*

$$b_{i,j} = \begin{cases} 1 & : \textit{falls } p_i \textit{ Anfangspunkt von } k_j \textit{ ist} \\ -1 & : \textit{falls } p_i \textit{ Endpunkt von } k_j \textit{ ist} \\ 0 & : \textit{sonst} \end{cases}$$

heißt (gerichtete) Inzidenzmatrix von $\mathcal{G}$.

2. *Die Matrix* $A_{\mathcal{G}} = (a_{i,j} | i, j = 1, \ldots, n)$ *mit*

$$a_{i,j} = \begin{cases} 1 & : \textit{falls } p_i \textit{ und } p_j \textit{ in } \mathcal{G} \textit{ durch eine Kante verbunden sind.} \\ 0 & : \textit{sonst} \end{cases}$$

heißt (gerichtete) Adjazenzmatrix von $\mathcal{G}$.

Das Ergebnis ist auf jeden Fall eine binäre Matrix, die eine 1 : 1- Abbildung des Graphen[6] im Rechner darstellt. Diese Verfahren werden u.a. von EVEN[Eve83] beschrieben.
Sucht man nun eine möglichst einfache Darstellung für einen Graphen, so bietet sich eine der obigen Darstellungen an. Die sicherlich einfachere Darstellung ist die Adjazenzmatrix, die im weiteren betrachtet werden soll.
Bei der Betrachtung dieser Adjazenzmatrix stellt man Eigenheiten bei der Erstellung der Matrix fest:

[6] und damit der Datenstruktur

Satz 3.1 (Form der Adjazenzmatrix für Gerichtete Graphen) *Sei $\mathcal{A}(X)$ eine zu einem Graphen X gehörende Adjazenzmatrix. Ist der Graph ein "Gerichteter Graph", so ist die Matrix eine quadratische $m \times m$- Matrix.*

Analog gilt für einen *Ungerichteten Graphen*:

Satz 3.2 (Form der Adjazenzmatrix für Ungerichtete Graphen) *Sei $\mathcal{A}(X)$ eine zu einem Graphen X gehörende Adjazenzmatrix. Ist der Graph ein "Gerichteter Graph", so ist die Matrix eine obere rechte $m \times m$- Dreiecksmatrix oder eine symetrische $m \times m$- Matrix.*

Diese Speicherungsform hat in der Praxis mehrere Vorteile :

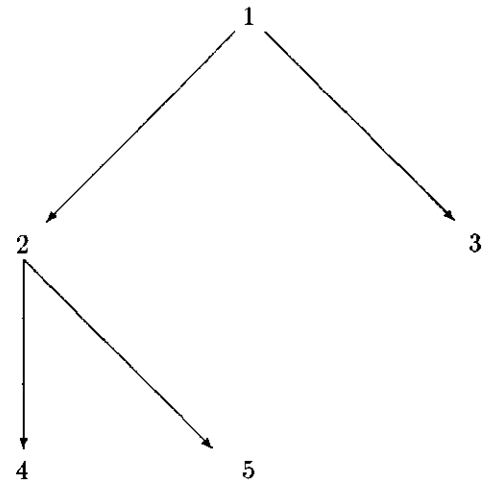
1. Diese Speicherungsform ist extrem speicherfreundlich, da sie je Verbindung pro Datenelement jeweils ein Bit verbraucht.
2. Man kann bei dieser Speicherungsform auf eine geschlossene mathematische Theorie (z.B. bei [Eve83] und [Doe77]) zurückgreifen.

Wichtig für die folgende Betrachtung ist die Feststellung, daß ein *ungerichteter Graph* eine elementare Einschränkung in der Art der Graphen beinhaltet, die allgemeinere Form ist die des gerichteten Graphen. Dies wird einsichtig, wenn man die obigen Sätze über die Form der Adjazenzmatrizen zur Hilfe nimmt. Ein *ungerichteter Graph* stellt eine zur Hauptdiagonalen symetrische Matrix dar, so daß gilt: $a_{ij} = a_{ji}$.

Dies ist eine wichtige Einschränkung, gerade im Aspekt zu den im weiteren definierten Operationen auf den Individuen. Im weiteren wird also von gerichteten Graphen ausgegangen und es werden ausschließlich diese Strukturen betrachtet.

Beispiel für eine Adjazenzmatrix

Als Beispiel für eine *Adjazenzmatrix* sei der folgende binäre Baum gegeben:



Man kann diesen Baum durch die folgende Adjazenzmatrix darstellen:

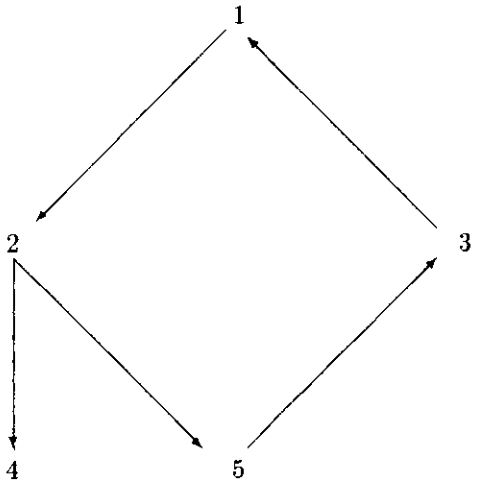
```
0 1 1 0 0
0 0 0 1 1
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0
```

Wie deutlich zu erkennen ist, liegt die Adjazenzmatrix als dünnbesetzte binäre 5×5- Matrix vor.

Die Adjazenzmatrix nimmt für verschiedene Formen des Graphen bestimmte, charakteristische Formen an, von denen zwei im weiteren erläutert werden sollen.

Graphen mit Zyklen

Gegeben sei ein Graph der folgenden Form



Die entsprechende Adjazenzmatrix zeichnet sich dadurch aus, daß keine Spalte existiert, die unbesetzt ist. Dies stellt sich in der entsprechenden Matrix so dar:

```
0 1 0 0 0
0 0 0 1 1
1 0 0 0 0
0 0 0 0 0
0 0 1 0 0
```

Die extremste Form der Zyklenbildung ist der *Kurzzyklus*, also ein Teilgraph der nur aus einem Knoten besteht. Dieser *Kurzzyklus* wird durch eine **1** auf der Hauptdiagonalen repräsentiert.

Graphen mit isolierten Knoten

Ein isolierter Knoten ist ein Teilgraph aus einem Knoten, der keinen Vorgänger und keinen Nachfolger hat. In der Adjazenzmatrix stellt sich diese Eigenschaft durch eine vollständig unbesetzte Zeile und Spalte dar.

Abschließend ist also festzustellen, daß die *Adjazenzmatrix* $\mathcal{A}$ eine geeignete Darstellung eines Graphen in binärer Form ist. Die verbleibende Problematik besteht in der Zweidimensionalität des Individuums, da GOLDBERG [Gol89] von einer eindimensionalen Struktur ausgeht.

Man kann nun diese Matrix auch in eine eindimensionale Form überführen, in dem man die Zeilen (oder die Spalten) in Reihenfolge aneinander hängt. Im weiteren Text wird jedoch versucht, die Operation auf den Individuen so zu definieren, daß sie weiterhin anwendbar sind.

3.3 Definition von Operationen

Auch wenn die Verknüpfung zwischen *Graph* und *Datenstruktur* isomopher Natur ist, müssen die Operationen auf den einzelnen Strukturen definiert werden. Die unterschiedlichen Operationen beruhen auf den verschiedenen semantischen Hintergründen bei Graphen und Datenstrukturen.

Während die Datenstrukturen primär die Aufgabe von *Speicherung, Auffinden* und *Löschen* haben, werden Graphen in der Form eines Kalküls betrachtet. Aber sicherlich sind die Operationen auf die andere Struktur zu übertragen.

3.3.1 Operationen auf Datenstrukturen

Bevor die Operationen auf den Individuen beschrieben werden können, müssen die Operationen auf den Datenstrukturen definiert und betrachtet werden.

Eine graphische Struktur, sei mit $\mathcal{G}$ gekennzeichnet. Jede Operation o ist dann definiert[7] als

Definition 3.12 (Operationen auf Daten) *Eine Operation o auf eine Datenstruktur $\mathcal{G}$ ist definiert durch*

$$o : \mathcal{G} \rightarrow \mathcal{G}'$$

Ausgehend hiervon kann man nun die elementaren Funktionen definieren:

Definition 3.13 (Suchen in der Datenstruktur) *Eine Operation $o : \mathcal{G} \rightarrow \mathcal{G}$ heißt Suchoperation in einer Datenstruktur, wenn gilt: $\forall x \in \mathcal{G}$ wird ein Weg von Wurzel zum Knoten gefunden, falls ein solcher Weg existiert.*

Eine Suchoperation ist besser für eine Datenstruktur geeignet, je schneller das Datum gefunden wird. Hierbei spielen verschiedenen Faktoren eine Rolle, die im nächsten Kapitel beschrieben werden sollen.

Desweiteren werden die folgenden zwei Operationen definiert:

Definition 3.14 (Einfügen in die Datenstruktur) *Eine Operation $o : \mathcal{G} \rightarrow \mathcal{G}'$ heißt Einfügeoperation in einer Datenstruktur, wenn gilt:*

$$|\mathcal{G}| + 1 = |\mathcal{G}'|$$

Ein Einfügen in eine Datenstruktur ist also eine Erweiterung des Graphen um einen Knoten. Die komplementäre Operation ist wie folgt definiert:

Definition 3.15 (Löschen aus der Datenstruktur) *Eine Operation $o : \mathcal{G} \rightarrow \mathcal{G}'$ heißt Löschoperation in einer Datenstruktur, wenn gilt:*

$$|\mathcal{G}| = |\mathcal{G}'| + 1$$

[7] nach [Maj76]

Das Löschen verringert also die Datenstruktur um ein Element. Es ist bekannt, daß eine Datenstruktur, die einer Suchoperation gegenüber gutmütig ist, einer Einfügeoperation aber gegenüber sehr rechenintensiv sein kann[Maj76].

Diese Operationen dienen aber letztlich nur dazu, eine bestimmte Ordnung in der Datenstruktur zu erzeugen. Es ist aber relativ irrelevant, eine bestimmte Ordnung zu untersuchen, da für jede Anwendung eine andere Ordnung von Interesse sein kann. Es interessiert also mehr, ob sich eine graphische Struktur überhaupt einer Ordnung fügt und wie schnell diese Ordnung eingehalten wird. Hierbei muß im weiteren die Einhaltung der Ordnung untersucht werden. Dieses wird ein Teil der folgenden Experimente sein.

3.3.2 Operationen auf Graphen

Parallel auf den üblichen Operationen in Datenstrukturen werden noch Operationen auf Graphen definiert, die zwar auf die Datenstrukturen übertragbar sind, aber im Allgemeinen nicht eingesetzt werden.

Diese Operationen werden im folgenden definiert. Hierzu seien zwei Graphen $\mathcal{G}_1$ und $\mathcal{G}_2$ mit $\mathcal{G}_1 = (P_1, K_1)$ und $\mathcal{G}_2 = (P_2, K_2)$ gegeben. Voraussetzung für die folgenden Definitionen ist, daß gemeinsame Kanten in beiden Graphen gleiche Endpunkte haben.

Dann gibt es die folgenden Operationen auf Graphen:

Definition 3.16 (Operationen auf Graphen) *Die folgenden Operationen sind auf Graphen definiert:*

1. *Vereinigung:* $\mathcal{G}_1 \cup \mathcal{G}_2 : (P_1 \cup P_2, K_1 \cup K_2)$
2. *Durchschnitt:* $\mathcal{G}_1 \cap \mathcal{G}_2 : (P_1 \cap P_2, K_1 \cap K_2)$
3. *Differenz:* $\mathcal{G}_1 \setminus \mathcal{G}_2$ *: der durch* $P_1 \setminus P_2$ *aufgespannte Teilgraph von* $\mathcal{G}_1$.
4. *Herausnehmen einer Kante:* $\mathcal{G}_1 - k : (P_1, K_1 \setminus \{k\})$
5. *Herausnehmen eines Punktes:* $\mathcal{G}_1 - p : \mathcal{G}_1 \setminus (\{p\}, 0)$
6. *Hinzufügen einer Kante:* $\mathcal{G}_1 + k : (P_1, K_1 \cup \{k\})$
7. *Hinzufügen eines Punktes:* $\mathcal{G}_1 + p : \mathcal{G}_1 \cup (\{p\}, 0)$

Die letzten Definitionen können auch für die Einfüge- und Löschoperation bei den Datenstrukturen benutzt werden.

Das Hinzufügen eines Datums geschieht durch das Einfügen eines Punktes und wenigstens einer Kante. Das Löschen eines Datums geschieht analog durch Entfernen des Knotens und aller Kanten zum Knoten.

3.4 Operationen auf Individuen

Interessant für die Anwendung in den *Genetischen Algorithmen* sind die Operationen auf den Algorithmen,

1. die Reproduktion des Individuums
2. die Kreuzung
3. die Mutation

Diese Operationen sind bis jetzt nur für eindimensionale Strings definiert. Ich arbeite allerdings im folgenden mit zweidimensionalen Strukturen, so daß sich die Frage nach der Übertragbarkeit dieser Operationen stellt und diese entsprechend untersucht werden müssen.

3.4.1 Die Reproduktion

Die erste Phase der Fortpflanzung ist die Reproduktion. Mit ihr wird der Grundstock für die neue, nachfolgende Generation gelegt. Die Idee bei der Fortpflanzung liegt bei einer *Selektiven Auswahl* der am besten angepaßten Individuen. Dabei soll aber nicht ein bestimmter Phänotyp vollständig übernommen werden, sondern es sollen nur seine Chancen im Vergleich zu den "schwächeren" Individuen gesteigert werden.

Zu diesem Zweck muß eine Rangfolge mit Hilfe der Fitnessfunktion erzeugt werden. Die Aufgabe der Fitnessfunktion liegt also in der Bewertung der Individuen.

In der Natur wird dieses durch vielfältige Maßnahmen geregelt. Ein Individuum der Natur kann seine bessere Fitness durch zum Beispiel eine bessere Nahrungsaufnahme beweisen und so sowohl eine höhere Lebenserwartung als auch damit verbunden eine erweiterte Paarungshäufigkeit erreichen.

Bei den Genetischen Algorithmen muß nun dieser Effekt nachempfunden werden. Dies geschieht normalerweise über die Rouletteradselektion[Gol89]. Zu diesem Zweck sei ein Rouletterad mit vielen einzelnen Feldern definiert. Jedes Roulette- Radfeld wird mit einem Individuum besetzt. Je besser die Fitness ist, desto mehr Felder werden dem Individuum zugeordnet.

Hiernach werden entsprechend der Populationsgröße eine neue Anzahl von Individuen gezogen, die die Basis für die weiteren Operationen darstellen. Damit haben Individuen mit einer besseren Fitness eine bessere Chance, ihr Erbgut in die nächste Generation einzubringen.

Betrachtet man die Reproduktion der Individuen, so stellt man fest, daß diese Operation nur mittelbar etwas mit der Struktur der Individuen zu tuen hat. Der Zusammenhang besteht nur über die Fitnessfunktion, da diese die Struktur auswertet und entsprechende Werte für die Reproduktion ausgibt. Der Vorgang der Reproduktion selber wird bei Goldberg[Gol89] beschrieben.

Hierzu wird der prozentuale Anteil der Gesamtfitness berechnet. Dieser prozentuale Anteil wird bei der Auswahl der zu reproduzierenden Individuen genutzt. Somit kann man die, in der Literatur beschriebene Operation zur Reproduktion ohne Veränderungen weiter benutzen.

3.4.2 Die Kreuzung

Die in der Literatur beschriebene Operation der Kreuzung ist auf eine eindimensionale Struktur beschränkt, also auf eine Kette von 0- und 1-Werten. Es werden Individuen paarweise zusammengefaßt und dann wird eine Stelle durch Zufall bestimmt, der auf diese Stelle folgende Teil wird zwischen den beiden Ketten ausgetauscht.

Wie bereits oben beschrieben wurde, hat allerdings das hiergenutzte Individuum eine zweidimensionale Struktur. Somit muß diese Operation so neu definiert werden, daß ihr Sinn erhalten bleibt, aber die Funktionalität auf zwei Dimensionen ausgedehnt wird.

Man muß bei der Kreuzung zwischen verschiedenen Strategien unterscheiden. Diese unterscheiden sich sowohl in ihrem theoretischen Ansatz als auch in der Durchführung der Kreuzung.

Die erste Strategie kann unter den Stichworten "*Teilen und Vertauschen*" betrachtet werden. Sie wird bei GOLDBERG[Gol89] unter anderem bei den *Simple Genetic Algorithms* (SGA) genutzt. Die zweite Strategie ist eine "*Verschmelzungsstrategie*", die auf der Dominanz bestimmter Eigenschaften beruht.

Zum Zwecke des Vergleichs wird die Vorgehensweise aus dem einleitenden Kapitel über Genetische Algorithmen wiederholt, um dann unter Betrachtung der zweidimensionalen Struktur des Individuums diese zu erweitern.

Kreuzung haploider Individuen

Das Standartverfahren der Kreuzung ist relativ einfach beschrieben, auch wenn dieses Verfahren dann auf verschiedenste Weise modifiziert wurde.

Im wesentlichen werden die beiden Erbträger paarweise und zufällig nebeneinander gelegt und es wird eine beliebige Stelle zufällig auf diesem Erbträger ausgewählt. An dieser Stelle wird der String aufgeschnitten und die Erbinformation getrennt. Dann wird diese Information vertauscht, so daß am Ende zwei neue Strings vorliegen.

Beispiel:
Die Strings

```
10111|010
01100|111
```

werden dann zu den folgenden Individuen gekreuzt:

```
10111|111
01100|010
```

Dieses Grundverfahren wird nun auf die verschiedensten Arten variiert. Die erste Variation besteht in der Veränderung der Anzahl der Kreuzungspunkte. Bei einem String mit k Genen gibt es $(k-1)$ einzelne mögliche Kreuzungspunkte. Damit kann auch die Anzahl der möglichen Trennstellen zwischen 1 und $(k-1)$ variieren. Auf Grund der speziellen Aufgabenstellung der Kreuzung muß diese Anzahl zusätzlich ungerade sein. Trotzdem kann man also die Anzahl der Trennungen variieren und muß nicht nur mit einer Trennstelle arbeiten.

Eine zweite Variation liegt im Schützen von bestimmten Bereichen vor der Kreuzung. Auf diese Art und Weise kann festgelegte Information als Block transferiert werden.

Bei der Übertragung dieser Algorithmen auf zweidimensionale Individuen muß nun ein Weg gefunden werden, der neben den beschriebenen Mechanismen auch die besondere Struktur der Wesen berücksichtigt.

Die erste Überlegung bestand darin, die Matrix in eine lineare Form zu übertragen und dann die Systematik für eindimensionale Wesen anzusetzen. Dieser Algorithmus hat zwar den Vorteil, daß die eigentliche Kreuzung literaturkonform ist, aber die Zweidimensionalität bleibt hierbei nahezu unberücksichtigt.

Projeziert man jedoch den Mechanismus der Kreuzung auf die neue Situation, so bieten sich die folgenden beiden Methoden an:

1. Wiederum faßt man die Individuum paarweise zusammen. Man wählt eine Stelle i zwischen zwei Zeilen und eine Stelle j zwischen zwei Spalten aus. Damit unterteilt man das Individuum in vier Sektoren oder *Quadranten*. Man vertauscht nun den ersten und den dritten Quadranten der Matrizen und erhält somit die beiden neuen Individuen für die weitere Bearbeitung.

2. Auch bei der nächsten Methode werden die Individuen zusammengefaßt. Per Zufall wird nun eine Parallele zur Hauptdiagonale ausgewählt. Nun wird die entstehende obere Dreiecksmatrix ausgetauscht und so ein neues Individuum erzeugt.

Da sich bei der Überlegung für oder wider einer speziellen Methode keine hinreichenden Gründe zugunsten einer Alternative zeigen, sollen bei der Durchführung der Experimente beide Versionen getestet werden.

Anders sieht es bei ungerichteten Graphen aus. Hier greift keine der dargestellten Methoden, sondern hier muß ein anderes Verfahren genommen werden. Dieses liegt insbesondere daran, daß jede der obrigen Methoden die besondere Eigenschaft der Adjazenzmatrix für ungerichtete Graphen, nämlich die Symetrie an der Hauptdiagonalen, zerstört.

Allerdings würde die Beschränkung auf einen ungerichteten Graphen eine nicht wünschenswerte Einschränkung in die Art des Systems darstellen, ist es nicht notwendig eine Betrachtung für diesen Fall vorzunehmen.

Betrachtung von Dominanz und Diploidität

Auch bei den Genetischen Algorithmen werden die, aus der Genetik stammenden Begriffe Dominanz und Diploidität diskutiert, um zum Ersten eine Optimierung auch schwieriger Problemstellungen zu ermöglichen und zum Zweiten diese in der Natur vorkommenden Phänomene auch bei diesen künstlichen Systemen einzusetzen. In dem nächsten Abschnitt wird versucht, die Einsatzmöglichkeiten von Dominanzen und Diploidität bei der Aufgabenstellung zu untersuchen.

Unter Dominanz versteht man ein Phänomen, daß als eines der ersten bei der Erforschung der modernen Genetik aufgefallen ist. Die Entdeckung dieses, ursprünglich als Störung betrachteten Mechanismus wird dem Urvater der Genetik, MENDEL[Kno83] zugeschrieben.

Der auftretende Effekt kann wie folgt beschrieben werden:
Es existieren offensichtlich Gene, die bei einer Kombination mit anderen ihre eigenen Eigenschaften durchsetzen und die anderen Gene in ihrer Ausprägung unterdrücken, also in ihrem Verhalten dominant sind.

Dieser Mechanismus ist in der Natur notwendig, um die zum Teil gegensätzlichen Informationen bei der Kreuzung zu eliminieren. Diese gegensätzlichen Informationen werden möglich, da bei den Mechanismen der Kreuzung mit einer Doppel- Helix Informationen doppelt in der Erbsubstanz vorliegen. In der Literatur wird die Existenz einer Doppel- Helix unter dem Begriff *Diploidität* geführt.

Beispiel:
Es seien zwei Chromosomenketten gegeben, die in diesem Beispiel[8] die Form haben:

```
AbCDe
aBCde
```

Hierbei seien Großbuchstaben dominant gegenüber Kleinbuchstaben. Bei der Paarung geschieht nun folgendes:
Aus obigen Chromosomen wird das folgende Individuum erzeugt:

```
ABCDe
```

Beim Auftreten eines dominanten und eines nicht- dominanten[9] Genes wird das dominante Gen die Merkmalsausprägung bestimmen.

Bei der Übertragung in ein haploides System, so wie es bei dieser Arbeit vorliegt, läßt sich ein alternativer Kreuzungsmechanismus beschreiben: Beide

[8] aus [Gol89]
[9] oder auch rezessiv genannten

Individuen werden zeilen- und spaltenweise miteinander verglichen. Treten nun gleiche Gene auf, so werden diese ohne Änderung übertragen. Sind die Gene verschieden, so wird das dominante Gen (wahlweise eine 0 oder eine 1) übernommen. Im Beispiel sieht dieses wie folgt aus:

Beispiel:
Dominant bei diesem Beispiel ist die 1. Gegeben seien die beiden Individuen

01100	01000
00100	01101
01110	10010
10110	10000
01010	10101

Nach der Kreuzung ergibt sich der folgende Prototyp, der verdoppelt wird, um die Größe der Population zu erhalten

```
01100
01101
11110
10110
11111
```

Im Gegensatz zu den anderen Paarungsmethoden wird hier keine Zufallsvariable einbezogen. Sollte dies jedoch notwendig sein, so kann man die Auswahl des dominanten Gens auf diese Weise betreiben.

3.4.3 Die Mutation

Die dritte und letzte Phase der Fortpflanzung wird als Mutation bezeichnet. Die Biologie versteht unter Mutation eine *Änderung im Erbbestand*[Kno83]. Individuen die von einer Mutation betroffen wurden, werden mit dem Wort *Mutanten* bezeichnet. Es werden in der Genetik verschiedene Formen von Mutationen unterschieden:

1. Erbliche Änderungen eines einzelnen Gens, die sogenannte *Genmutation*,
2. eine Änderung in der Struktur der Chromosomen (*Chromosomenmutation*), und
3. sogar eine Änderung des gesamten Chromosomenbestandes[10], die *Genomenmutation*.

[10] des Genoms

Desweiteren wird zwischen Mutationen in der Keimbahn, die in die Erbfolge eingehen und Mutationen in Sekundärzellen (*Somazellen*) unterscheiden. Die letztgenannte Form der Mutation ist nur auf das existierende Individuum beschränkt. Diese Art der Mutation wird unter anderem für die Krankheit Krebs verantwortlich gemacht. Im Bereich der Genetischen Algorithmen beschränkt man sich auf die Implementierung der Genmutation in der Keimbahn.

Anders als bei der Kreuzung ist die Mutation nicht von der Struktur des Individuums abhängig. Diese Operation ist zwar für eine eindimensionale Form vorgesehen, allerdings kann mit einer einfachen Definitionsänderung auch ein Individuum bearbeitet werden, das in einer Matrix definiert ist.

Hierzu wird die vorne definierte Formel in die folgende geändert und so die Mutation definiert :

Definition 3.17 (Die Formel für die Mutation) *Für die Mutation wird jede Zeile und jede Spalte der Matrix gegen den Zufallsoperator abgetestet. Hierzu sei ein Zufallsoperator X mit $P(X = 1)$ definiert und hieraus folgt die formelmäßige Darstellung:*

$$\forall x \in \mathcal{I} : x_{ij} = \begin{cases} \bar{x_{ij}} & : \quad X = 1 \\ x_{ij} & : \quad X = 0 \end{cases}$$

Wichtig bei der Mutation ist die Festlegung der Mutationswahrscheinlichkeit p_{mut}, also der Wahrscheinlichkeit, daß ein Gen durch Mutation verändert wird. Diese Wahrscheinlichkeit beträgt in der Natur zwischen $1 : 10^4$ und $1 : 10^9$ (nach [Kno83]). Bei der Festlegung der Mutationswahrscheinlichkeit für das hier verwendete Individuum kann folgendes beachtet werden:
Im Gegensatz zu den eindimensionalen Strings verfügt die Matrix über deutlich mehr Gene. Aus diesem Grunde muß die Mutationswahrscheinlichkeit wesentlich kleiner als bei den Strings sein. Die exakte benutzte Mutationswahrscheinlichkeit muß experimentell ermittelt werden.

3.4.4 Einsatz von Constraints

Im Laufe der Experimente soll auch der Einsatz von Constraints getestet werden. Unter Constraints versteht man Einschränkungen über die Art der Individuen im laufenden System. Diese Einschränkungen in dem laufenden System können sein:

1. Das Individuum muß nach der Fortpflanzung der Definition eines Baumes genügen.
2. Das Individuum muß nach der Fortpflanzung der Definition eines Waldes genügen.

Wenn ein Individuum bei der Kontrolle nach den Anforderungen versagt, so kann das System auf drei Arten reagieren:

1. Das Individuum erhält eine Fitness von Null und scheidet damit aus dem Fortpflanzungskreislauf aus.

2. Der Fitnesswert wird herabgesetzt, die Wahrscheinlichkeit der Fortpflanzung wird damit stark reduziert.

3. Ein spezieller Reparaturalgorithmus erzeugt ein, den Bedingungen der Constraints angepasstes Individuum, das weiterhin im Zyklus des Systems verbleibt. Das entsprechende Vorgehen wird im Abschnitt 4.3.2 behandelt.

Die verschiedenen Arten der Reaktion haben entsprechende Vor- und Nachteile. Ganz zu verwerfen ist die letzte Reaktion, da hier ein Eingriff in das laufende System vorgenommen wird, durch den neue Individuen im Ablauf "künstlich" entstehen. Es wäre zu diskutieren, in wieweit ein solcher Eingriff in bestimmten Situationen sinnvoll oder notwendig ist, allerdings sei die Frage gestellt, ob in diesem speziellen Fall ein Eingriff dieser Art die Ergebnisse der Untersuchung verfälscht.

Auch die Herabsetzung der Fitness ist abzulehnen, da hier defektes Erbgut eine zwar geringe, aber doch vorhandene Wahrscheinlichkeit der Reproduktion besitzt.

Im Verlauf der Untersuchung soll jedoch getestet werden, in welchem Maße eine Förderung erwünschter Strukturen Vorteile für die Entwicklung des Systems bringt. Hierzu erhält ein "gutes" Individuum einen bestimmten Wert als Zuschlag zum Fitnesswert.

3.5 Betrachtung zweidimensionaler Schemata

Abschließend soll in diesem Abschnitt eine Betrachtung der Schemata durchgeführt werden. Es interessiert hierbei, ob durch die Definition von zwei- und mehrdimensionalen Individuen eine Änderung bei dem Theorem der Schemata notwendig wird.

Um dieses festzustellen, müssen zuerst die Erkenntnisse für eindimensionale Individuen zusammengefaßt werden[Gol89].

Prinzipiell ist ein Individuum eine binäre Zeichenkette[11] A mit k einzelnen Elementen a_1 bis a_k. Die Reihenfolge dieser einzelnen Elemente ist hierbei nicht von Belang. Die Durchführung der Operationen erfordert nun eine Population von Individuen, die mit $\mathbf{A}(t)$ gekennzeichnet wird, wobei t die laufende Generation zwischen der ersten und der letzten Generation bezeichnet.

Zuerst wird die Wahrscheinlichkeit[Moo74] der Zerstörung bei der Fortpflanzung betrachtet. Hierzu wird ein String A mit k Genen und ein Teilstring A' mit k' Genen untersucht. Das System hat dann $(k-1)$ Möglichkeiten, einen Trennpunkt zu finden. Damit der Teilstring erhalten bleibt, darf dieser Trennpunkt nicht innerhalb eines Teilstrings sein. Die Wahrscheinlichkeit, daß der

[11] oder ein String

Trennpunkt außerhalb liegt, läßt sich über die Formel

$$p_u = \frac{(k-1)-(k'-1)}{(k-1)} + c = \frac{k-k'-2}{k-1} + c$$

berechnen.
Hinzu kommt die Wahrscheinlichkeit p_{mut}, die die Quote ausdrückt, das ein Element des Strings mutiert. Diese ist in der obigen Formel mit dem Faktor c gekennzeichnet. Genauer ausgedrückt hat dieser Faktor die Form

$$c = (k - k') \cdot p_{mut}$$

Ergänzt man nun das eingesetzte Alphabet um das Symbol '$*$', so erhält man ein Gebilde, das den Namen Schema hat. Als Definition kann man dies so schreiben:

Definition 3.18 (Schema) *Sei A ein String und A' ein Teilstring $\subseteq A$. Ein Schema ist ein Ausdruck über dem Alphabet $V = (1, 0, *)$, der den Teilstring A' repräsentiert. Hierbei haben die Zeichen $0, 1$ des Alphabets die gleiche Bedeutung wie im String und das Zeichen "$*$" kann sowohl den Wert 0 als auch den Wert 1 annehmen. Dieser Ausdruck wird mit H bezeichnet.*

Beispiel:
Die Zeichenkette `0110110` sei gegeben. Dann repräsentiert das Schema `*1**1**` den String, genauso wie das Schema `01*****`, nicht aber `1***110`.

Bei GOLDBERG[Gol89] werden zwei Maße für die Schemata definiert, die Differenz δ und die Präzision[12] $\tilde{o}$.

Die Differenz δ gibt ein Maß für die Kompaktheit des Schemas an, je kleiner δ ist, desto enger liegen die "festen" Werte im Schema zusammen.

Beispiel :
$\delta(0**1**) = 4$
$\delta(01****) = 2$

Ein Sonderfall liegt vor, wenn das Schema nur einen festen Wert hat. In diesem Fall wird für δ der Wert 0 genommen. Die Präzision $\tilde{o}$ gibt an, wieviele der Werte im Schema "fest", also unveränderlich sind.

Beispiel :
$\tilde{o}(0**1**) = 2$
$\tilde{o}(0110**) = 4$

Auf diese Weise definiert GOLDBERG[Gol89] eine Funktion zur Prognose über das Verhalten des Systems. Diese Prognose ist nur abhängig von den obigen Funktionen und einigen Systemvariablen.

[12] bei Goldberg heißt dieser Wert "order of a schema"

Sei m die Anzahl der Individuen, die zum Schema H passen. Zu einem Zeitpunkt t wird $m = m(H,t)$ mit der folgenden Formel berechnet[Gol89]:

$$m(H,t+1) = m(H,t)\frac{f(H)}{\bar{f}}$$

wobei $f(H)$ die mögliche Fitness des Schemas H und $\bar{f}$ die Gesamtfitness ist. Diese Grundformel wird erweitert zu

$$m(H,t+1) \geq m(H,t) \cdot \frac{f(H)}{\bar{f}} \left[1 - p_c \cdot \frac{\delta(H)}{l-1} - \tilde{o}(H)p_m\right]$$

wobei l die Länge des Strings, p_c die Wahrscheinlichkeit des Stattfindens der Kreuzung[13] und p_m die Mutationswahrscheinlichkeit ist.

Bei der Übertragung dieser Formel, die als *Schema Theorem* oder *Fundamentales Theorem der Genetischen Algorithmen* bekannt ist, stellt man folgendes fest:

Es ist anzunehmen, das nach einer Übertragung der Funktionen δ und $\tilde{o}$ das Theorem auch für mehrdimensionale Individuen anwendbar ist. Eine solche Übertragung kann die folgende Form haben:
Der erste Schritt liegt in der Berechnung der Zerstörbarkeit des Schemas. Hier sei nun $\mathcal{A}$ eine binäre $k \times k$ Matrix, also eine Matrix mit k^2 Elementen, und $\mathcal{A}'$ eine $n \times m$ Untermatrix von $\mathcal{A}$ mit $1 \leq m, n \leq k$. Außerdem kommt wieder das erweiterte Alphabet mit $(0, 1, *)$ zum Einsatz.

Es gibt zwei Möglichkeiten, diese Untermatrix durch Teilung bei der Kreuzung zu zerstören: Entweder durch einen Schnitt in der ersten Dimension (auch x genannt) oder durch einen Schnitt in der zweiten Dimension (auch y genannt). Diese beiden Schnitte sind vollkommen unabhängig voneinander. Für jede Dimension gilt nun

$$p_u = \frac{k-n-2}{k-1} + c$$

mit

$$c = (k-n) \cdot p_{mut}$$

Nun ist die Auswahl der Schnittpunkte voneinander unabhängig, so daß nach [Bos86] gilt, daß die Wahrscheinlichkeiten für jede Dimension zu einer Gesamtwahrscheinlichkeit multiplikativ verknüpft werden. Es gilt also

$$p_G = \left(\frac{k-n-2}{k-1} + (k-n) \cdot p_{mut}\right) \cdot \left(\frac{k-m-2}{k-1} + (k-m) \cdot p_{mut}\right)$$

Je nach Wahl der Größe des Schemas, also von n und m ist dieses stabiler oder anfälliger gegen Teilung als ein entsprechendes eindimensionales Schema.

[13] hier: $p_c = 1$

Eine solche Ergänzung läßt sich auch für die Funktionen δ und $\tilde{o}$ machen. Zuerst wird die Funktion um einen Orientierungsindex erweitert, so daß für die x-Richtung die Funktionen δ_x und $\tilde{o}_x$ existieren.

Ohne die Definitionen für eindimensionale Wesen zu verletzen, kann man nun $\delta = \max(\delta_x, \delta_y)$ und $\tilde{o} = \max(\tilde{o}_x, \tilde{o}_y)$ festlegen. Da die Kernformel

$$m(H, t+1) = m(H, t)\frac{f(H)}{\bar{f}}$$

keinerlei Informationen über die Struktur des Schemas enthält, und auch die übrigen Variablen unverändert bleiben, kann davon ausgegangen werden, daß das *Schema Theorem* auch für zweidimensionale Individuen seine Gültigkeit behält.

Kapitel 4

Die Fitnessfunktion

In diesem Abschnitt sollen die Grundlagen für die Optimierung gelegt werden. Wie schon vorne beschrieben, besteht die Optimierung aus zwei Teilen:

1. Optimierung der Topologie, und
2. Optimierung der Ordnung

Diese Optimierung wird also zuerst in zwei Teilen vorgenommen, hiernach wird eine Optimierung der Gesamtstruktur vorgenommen.

Wichtig bei der Durchführung einer Optimierung ist die Betrachtung der Ergebnisse der bisherigen Theorie und die Betrachtung möglichst aller verwandten Gebiete, bei dieser Optimierung ist insbesondere die mathematische Graphentheorie und die Theorie der Algebraischen Strukturen von Bedeutung.

4.1 Topologische Optimierungen

Wie schon erwähnt liegen Daten sehr häufig in Form von Bäumen vor, diese Organisationsform ist die am besten dokumentierte. Ein solcher Baum wird in dem folgenden Beispiel dargestellt. In der Literatur wird als Maß für die *Topologische Optimalität* die *Mittlere Weglänge* angeführt. Diese *Mittlere Weglänge* ist durch KNUTH[Knu68] als auch von DENERT und FRANCK[Den77] wie folgt eingeführt:

4.1.1 Die Definition der *mittleren Weglänge*

Zuerst sei die Gesamtweglänge in der folgenden Form definiert:

Definition 4.1 (Die Gesamtweglänge in Bäumen) *Gegeben sei ein Baum $\mathcal{B}$. Dann sei die Gesamtweglänge definiert als*

$$\mathrm{gwl}_B(\mathcal{B}) = \sum_{i=0}^{h} i \cdot n_i$$

wobei n_i die Anzahl der Knoten auf dem Niveau i und h die Höhe[1] *des Baumes darstellt.*

Hieraus lässt sich die *mittlere Weglänge* mit der Formel

$$\mathrm{mwl}(\mathcal{B}) = \frac{1}{n} \cdot \sum_{i=0}^{h} i \cdot n_i$$

berechnen, wobei hier n die Anzahl der Knoten im Baum ist.
Damit kann man die Suchzeit für ein Element berechnen als ein Wert

$$1 \leq t_{search} \leq t_{\mathrm{gwl}_B(\mathcal{B})}$$

und damit

$$t_{search} = t_{\mathrm{mwl}(\mathcal{B})}$$

Also kann als Vergleichswert für die Effizenz die *mittlere Weglänge* angenommen werden.

Diese Definitionen beziehen sich ausschließlich auf die graphische Struktur eines Baumes. Eine Erweiterung für Wälder und Graphen wird im weiteren gegeben.

Es ist festzustellen, daß dieser Wert sich unabhängig von irgendwelchen Suchstrategien berechnet, also *strategieneutral* und *ordnungsneutral* ist. Mit seiner Hilfe lassen sich die verschiedenen Baumstrukturen einordnen.

Allerdings kann man für die verschiedenen Strukturen Aussagen für das Zeitverhalten treffen. Dieses Zeitverhalten kann als Maßstab für den Aufbau der Datenstruktur und des Suchalgorithmus stehen.

Spezielle Werte für die *Mittlere Weglänge*

Die *Mittlere Weglänge* kann für verschiedene Topologien näherungsweise über die folgenden Formeln berechnet werden. Diese Berechnungen wurden von KNUTH[Knu68] vorgenommen und helfen bei der Klassifizierung der Datenstruktur. Bekannt sind die folgenden Näherungen :

- in vollständigen Bäumen: $\mathrm{mwl}_{min} \sim \log_2(n) - 2$
- im Durchschnitt über alle Bäume mit n Knoten: $\mathrm{mwl}_{mittel} \sim \sqrt{n}$

[1] d.i. die Länge seines längsten Weges

- bei entarteten Bäumen[2] : $\text{mwl}_{max} \sim n$

Der Nachteil der *Mittleren Weglänge* liegt in der fehlenden Definition der Funktion für *Graphen* und *Wälder*. Aus diesem Grund muß sie erweitert werden, so daß sie auch auf *Wälder* und *Graphen* anwendbar ist.

4.1.2 Eine Erweiterung der Definition der *Gesamtweglänge*

Die erste Erweiterung betrifft die Anwendbarkeit auf Wälder. Diese Erweiterung basiert auf der *Mittleren Weglänge* für Bäume.

Definition 4.2 (Gesamtweglänge für Wälder) *Gegeben sei ein Wald $w \in \mathcal{W}$, bestehend aus m Bäumen $b_1, b_2, \ldots, b_m \in \mathcal{B}$. Die Gesamtweglänge für einen $w \in \mathcal{W}$ ist dann definiert als*

$$\text{gwl}_W(w) = \sum_{x=1}^{m} \text{gwl}_B(b_x) = \sum_{x=1}^{m} \sum_{i=0}^{h} i \cdot n_{i,x}$$

Hierbei ist folgendes zu beachten:
Knoten, die von mehreren Wurzeln erreicht werden können, werden entsprechend ihrer Erreichbarkeit mehrfach gezählt.

Entsprechend der vorigen Festlegung definiert man die *Mittlere Weglänge* für *Wälder*

Definition 4.3 (Mittlere Weglänge für Wälder) *Gegeben sei ein Wald $w \in \mathcal{W}$, bestehend aus m Bäumen $b_1, b_2, \ldots, b_m \in \mathcal{B}$. Die Mittlere Weglänge ist dann definiert als*

$$\text{mwl}_W(w) = \sum_{x=1}^{m} \text{mwl}_B(b_x) = \sum_{x=1}^{m} (\frac{1}{n_x} \sum_{i=0}^{h_x} i \cdot n_{i,x})$$

Wie schon vorne beschrieben, bezeichnet n_x die Anzahl der Knoten in den jeweiligen Bäumen. Wichtig bei dieser Definition ist die Tatsache, daß bei $m = 1$ die Definition der *Mittleren Weglänge* für Bäume der entsprechenden Definition für Wälder entspricht.

Eine weitere Ergänzung dieser Definition ist notwendig, um die *Gesamtweglänge* und die *Mittlere Weglänge* als Funktion auf einen Graphen anzuwenden. Im Unterschied zu Bäumen und Wäldern verfügt ein allgemeiner Graph über keine Wurzel[3]. Hieraus folgt, daß auf einen Graphen $\mathcal{G} \notin \{\mathcal{B}, \mathcal{W}\}$ die Funktion der Gesamtweglänge nicht angewendet werden kann. Der Graph muß also betrachtet werden, daß er über einen Wald oder über einen Baum beurteilt werden kann. Zu diesem Zweck definiert man

[2] z.B. lineare Listen

[3] oder ein anderes ausgezeichnetes Element, vgl. Seite 25

Definition 4.4 (Gerüst eines Graphen) *Ein Gerüst des zusammenhängenden Graphen $x \in \mathcal{G}$ ist ein Baum $x' \in \mathcal{B}$, der Teilgraph von x ist und alle Punkte von x enthält.*

und es gilt

Satz 4.1 (Vollständigkeitssatz für Gerüste) *Jeder zusammenhängende Graph besitzt ein Gerüst.*

Den entsprechenden Beweis entnehme man bei MÜLLER[4]. Hiermit kann man nun die *Gesamtweglänge* auch über Graphen definieren:

Definition 4.5 (Gesamtweglänge in zusammenhängenden Graphen) *Gegeben sei ein zusammenhängender Graph $x \in \mathcal{G}$ mit einem zufällig, aber fest vorgegebenen Knoten a und einem zugehörigen Gerüst $x' \in \mathcal{B}$, der von a aus entwickelt wurde. Die Gesamtweglänge ist dann definiert als*

$$\mathrm{gwl}_G(x') = \sum_{i=0}^{h'_x} i \cdot n_i$$

mit der Wurzel in a.

Entsprechend definiert sich die *Mittlere Weglänge in zusammenhängenden Graphen.*

Definiert man nun noch eine Zusammenhangskomponente, so kann man über jeden, auch nicht zusammenhängenden Graphen die Funktion der *Gesamtweglänge* definieren.

Definition 4.6 (Zusammenhangskomponenten) *Sei $x \in \mathcal{G}$ ein ungerichteter Graph und a ein beliebiger, aber fester Knoten im Graphen. Die Zusammenhangskomponente von a in x ist der Teilgraph x' von x, der durch die Menge aller mit a durch Kanten verbindbaren Knoten aufgespannt wird.*

Es gilt, daß sich jeder Graph $x \in \mathcal{G}$ mit k Zusammenhangskomponenten in genau k zusammenhängende Teilgraphen zerlegen läßt. Damit lassen sich in jedem Graphen genau k Gerüste bilden, die nach der Definition über die Berechnung der *Gesamtweglänge* in Bäumen zur Bestimmung der *Gesamtweglänge* in nicht zusammenhängenden Graphen dienen können. Damit erhält man die folgende Erweiterung der Definition über die *Gesamtweglänge in zusammenhängenden Graphen* zu der entgültigen Definition der *Gesamtweglänge in Graphen.*

Definition 4.7 (Gesamtweglänge in Graphen) *Gegeben sei ein Graph $x \in \mathcal{G}$ mit k Zusammenhangskomponenten und mit k gewählten Knoten a_k und zugehörigen Gerüsten $x'_k \in \mathcal{B}$. Dann ist die Gesamtweglänge definiert als*

$$\mathrm{gwl}_G(x) = \sum_{y=1}^{k} \mathrm{gwl}_B(x'_y)$$

[4] vgl.[Mue79], S.55

Die *Mittlere Weglänge* berechnet sich entsprechend, so daß gilt

Definition 4.8 (Die *Mittlere Weglänge* in Graphen) *Gegeben sei ein beliebiger Graph $x \in \mathcal{G}$ mit k zusammenhängenden Teilgraphen $\tilde{x_1}, \tilde{x_2}, \ldots, \tilde{x_k}$ und entsprechenden k Gerüsten $x'_1, x'_2, \ldots, x'_k \in \mathcal{G}$. Die "Mittlere Weglänge" berechnet sich damit nach der Formel*

$$\mathrm{mwl}_G(x) = \sum_{y=1}^{k} \mathrm{mwl}_B(x'_y)$$

Diese Funktion der *Mittleren Weglänge* erfüllt alle notwendigen Anforderungen, um auch in Graphen beliebiger Art zum Einsatz kommen zu können.

Eine Kostenfunktion als Bewertungsmaß für Graphen

Alternativ zu der *Mittleren Weglänge* kann man auch eine Kostenfunktion als Bewertungsfunktion zum Einsatz kommen lassen.

Diese Art der Kostenfunktion wird insbesondere im Bereich des *Operations Research* bei der sog. Knappsack- Methode angewendet, um z.B. Packungs-Probleme zu berechnen. Entsprechende Methoden werden u.a. bei MÜLLER-MERBACH [Mue73] beschrieben.

In der direkten Anwendung wird einer entsprechenden Position im Graphen ein Wert zugeordnet, der als Kosten für das Besetzen der entsprechenden Stellung betrachtet werden kann.

Diese Kosten werden höher, je näher die Position an der Wurzel der Datenstruktur ist, am höchsten jedoch direkt an der Wurzelposition.

Geht man von einer Datenstruktur mit k Knoten aus, so kann man die Kosten der Wurzel mit dem Wert $k + 1$ angeben. Für jeden tieferliegenden Knoten werden Kosten nach der Formel $k - Tiefe + 1$ berechnet.

Auch bei dieser Funktion lassen sich optimaler und schlechtester Wert exakt berechnen. Der optimale Wert wird für die lineare Liste, der schlechteste Wert wird für einen Graphen mit einer Anordnung mit den k Knoten des Graphen als Wurzeln von k Teilgraphen erreicht.

- optimaler Wert: $\sum_{x=1}^{k} x + 1$
- schlechtester Wert: $k \cdot (k + 1)$

4.2 Die Optimierung der Ordnung

Die zweite Problematik bei den *Datenstrukturen* ist das Problem des Auffindens eines Datums in der Struktur. Diese Problematik ist unabhängig von der Organisation der Daten.
Das Auffinden läßt sich aber auch als Funktion definieren:

Definition 4.9 (Das Finden von Werten) *Das Auffinden von Werten in der Datenstruktur hat die folgenden Werte:*

$$\text{find}(x) = \begin{cases} 1 & : \text{ das Datum wird gefunden} \\ 0 & : \text{ das Datum wird nicht gefunden} \end{cases}$$

Eine optimale Datenstruktur fördert aber nicht nur das Auffinden von Daten[5], sondern auch ein möglichst schnelles Auffinden von Daten in der Struktur.

Um hier ein entsprechendes Verhalten zu beschreiben, muß man die Suche in der Datenstruktur[6] betrachten.

4.2.1 Formen der Suche

Geordnete Suche

Unter der *Geordneten Suche* bezeichnet man die Suche in einem *Geordneten Graphen*, wobei sich der Begriff des Graphen in diesem Zusammenhang in der Literatur auf Bäume reduziert.

Bei diesem Verfahren der Suche wird aus dem Wert des erreichten Knoten auf den zu wählenden Nachfolger geschlossen. Diese, in der Literatur bei KNUTH [Knu68] und bei WIRTH[Wir83] beschriebenen Verfahren gehen von einer Totalen Ordnung[Dor78, Lip81, Mue79] aus, in der die Elemente in dem Graphen angeordnet sind.

Eine wesentliche, für die hier untersuchte Anwendung bedeutende Eigenschaft liegt in der Reduktion der Nachfolger eines Knoten auf eine maximale Anzahl.

Damit reduziert sich die Anzahl der brauchbaren Bäume um all jene, die einem Knoten beliebig viele Nachfolger zuordnen. Die Literatur[Den77] spricht aus diesem Grunde von t-ären Bäumen. Diese werden durch das Tupel $(\mathcal{T}, <)$ beschrieben. Hierbei bezeichnet $\mathcal{T}$ einen t-ären Baum und $<$ eine Ordnungsstruktur. Während die Definition der Bäume an anderer Stelle vorgenommen wird, muß die Ordnung noch eingeführt werden.

Nach MÜLLER[Mue78] versteht man unter einer Ordnungsstruktur ein reines Relationensystem mit einer zweistelligen Relation, genannt *Ordnung*.

Man unterscheidet zwischen einer *Halbordnung* und einer *Totalen Ordnung*. Diese sind wie folgt definiert:

Definition 4.10 (Die Halbordnung) *Eine halbgeordnete Menge ist eine Ordnungsstruktur $(\mathcal{A}, \sqsubseteq)$, bestehend aus einer Grundmenge $\mathcal{A}$ und einer Halbordnung $\sqsubseteq$ (gelesen "vor") auf $\mathcal{A}$. $\sqsubseteq$ ist also eine Relation auf $\mathcal{A}$ mit den folgenden Eigenschaften:*

1. *Reflexivität:* $\forall x(x \in \mathcal{A} \rightarrow x \sqsubseteq x)$

[5] dieses ist eine zwingende Vorraussetzung in einer Datenbank

[6] die im weiteren als Graph dargestellt wird

2. *Transitivität:* $\forall x, y, z(x \sqsubseteq y \wedge y \sqsubseteq z \rightarrow x \sqsubseteq z)$

3. *Antisymetrie:* $\forall x, y(x \sqsubseteq y \wedge y \sqsubseteq x \rightarrow x = y)$

Definition 4.11 (Die Totale Ordnung) *Eine totalgeordnete Menge ist eine Ordnungsstruktur* $(\mathcal{A}, \sqsubseteq)$*, wobei* $\sqsubseteq$ *eine totale Ordnung auf* $\mathcal{A}$ *ist, d.h. zusätzlich zu den Regeln der Halbordnung gilt:*

4. *Vergleichbarkeit:* $\forall x, y(x \in \mathcal{A} \wedge y \in \mathcal{A} \rightarrow x \sqsubseteq y \vee y \sqsubseteq x)$

Im Gegensatz zu einigen Literaturstellen muß die Ordnungsstruktur bei Bäumen bzw. Datenstrukturen eine *Totale Ordnung* und darf keine *Halbordnung* sein.

Sehr häufig wird in der Literatur die Relation ">" genutzt, so z.B. bei WIRTH [Wir83].

Bei der Suche in einem *geordneten Baum* wird nun, beginnend bei der Wurzel, durch den Baum gewandert. Hierbei wird der Nachfolgeknoten durch Vergleich mit dem aktuellen Knoten bestimmt. Die Knoten werden auf diese Art durchsucht, bis das richtige Datum erreicht wird.

Der Vorteil dieser Suche liegt in dem schnellen Auffinden mit wenigen Suchschritten in der Datenstruktur. Allerdings hat dieses Suchverfahren die Nachteile, daß die Ordnung strikt sein muß, d.h. alle Elemente müssen sich der Ordnung vollständig unterwerfen, da sonst ein Datum nicht gefunden wird.

Suchstrategien

Aus dem Bereich der Künstlichen Intelligenz kommt der Begriff der *Suchstrategien*, der in dem nächsten Abschnitt kurz angerissen werden soll. Unter diesem Begriff versteht man das Vorgehen bei der Suche in *Nichtgeordneten Graphen*. Zuerst wurde die *Geordnete Suche*, also die Suche in dem Graph, über den Informationen über den Aufbau[7] vorliegen untersucht, die Maßnahmen bei der *Nichtgeordneten Suche* sind grundsätzlich von den bei der *Geordneten Suche* verschieden. Hier kommt die Strategie bei der Suche zu einer ausschlaggebenden Bedeutung.

Man unterscheidet zwei verschiedene Strategien[Joc88] bei der Suche,

1. die *Tiefensuche*[8] und

2. die *Breitensuche*

Diese Strategien unterscheiden sich vor allem in ihrer Vorgehensweise:
Die *Tiefensuche* durchsucht zuerst einen Ast bis zum Blatt und sucht von dort

[7] also Sortierung oder der andere Informationen
[8] engl. deep-first oder back-tracking

aus mit sukzessiven Rückwärtsschritten und Abarbeiten in den Nebenästen weiter nach dem Ziel. Die *Breitensuche* dagegen sucht und vergleicht zuerst zwischen den Knoten auf einer Stufe und wählt zur Fortführung der Suche den Knoten mit dem besten Wert aus.

Beide Verfahren haben Vor- und Nachteile. Diese müssen bei der Auswahl der Strategie bekannt sein und auch berücksichtigt werden.

Dieses trifft vor allem auf die Faktoren "Effektivität" und "Effizienz" zu. Das Verfahren der "Tiefensuche" findet auf jeden Fall das Datum, hat aber bei tiefen Graphen ein unproportionales Suchverhalten; das Verfahren der Breitensuche findet das Datum mit weniger Suchschritten, bei allerdings deutlich höheren Speicher- und Rechenbedarf.

4.2.2 Maße für die Suche

Bewertung mit der Geordneten Suche

Zur Bewertung einer Datenstruktur bei der Geordneten Suche wird in der Standartliteratur stets die *Mittlere Weglänge*[Knu68] als Maß herangezogen. Hierbei wird prinzipiell davon ausgegangen, daß die entsprechende *Totale Ordnung* strikt eingehalten wird.

Die *Mittlere Weglänge* jedoch bewertet nur die topologischen Eigenschaften der Datenstruktur. Die Einhaltung der Ordnung wird nicht beurteilt. Es muß also nach einer alternativen Beurteilung der Ordnung gesucht werden. Eine solche Beurteilung stellt die folgende Funktion dar:

Definition 4.12 (Beurteilungsfunktion für die Ordnung) *Gegeben sei eine beliebige Datenstruktur und ein zugeordneter Graph $\mathcal{X}$ mit einer Menge $x_1, x_2, \ldots, x_n$ von n Knoten und $k_1, k_2, \ldots, k_m$ von m Kanten. Dann gibt es einen Wert y, der definiert ist als*

$$y = \sum^{m} y_m$$

mit

$$y_m = \begin{cases} 1 & : \; falls \; \exists k_m = (x_a, x_b) \wedge x_a \leq x_b \\ 0 & : \; falls \; \neg\exists k_m = (x_a, x_b) \\ -1 & : \; falls \; \exists k_m = (x_a, x_b) \wedge x_a > x_b \end{cases}$$

Diese Beurteilungsfunktion ist berechenbar mit einem festen Maximum

$$y_{max} = \sum_{a=1}^{n} a$$

und einem festen Minimum

$$y_{min} = -y_{max}$$

Bewertung mit der Nichtgeordneten Suche

Um Suchstrategien zu bewerten, ist die *mittlere Weglänge* ungeeignet, ebenso wie bei der *Geordneten Suche*

Zu dem Zweck der Beurteilung einer Strategie muß man also einen anderen Weg beschreiten :

Analog zu der topologischen Optimierung wird nun versucht, einen entsprechenden Wert zu der *Mittleren Weglänge* als Beurteilung zu definieren. Wieder sei ein beliebig strukturierter Graph $\mathcal{B}$ gegeben. Dieser Graph besteht aus $n_{\mathcal{B}}$ Knoten und Blättern, also aus $n_{\mathcal{B}}$ verschiedenen Daten. Die Struktur der Daten ist wiederum ohne Bedeutung.

Sei $t_{search}(n)$ die Anzahl der Schritte des Suchalgorithmus bis zum Auffinden des gesuchten Datums n. Hierbei ist die verwendete Suchstrategie nur bei der Durchführung von Bedeutung. Hiermit wird nun versucht, einen Wert zu berechnen, mit dem man eine Datenstruktur bewerten kann.

Dazu definiert man den Wert t_{search} über die Suche nach allen Werten. Man kann diesen Wert als *Gesamtsuchzeit* bezeichnen. Diese sei definiert als

$$t_{search} = \sum t_{search}(n)$$

also als Summe über alle Laufzeiten.

Dieser Wert ist grundsätzlich verschieden von der *Gesamtweglänge* und damit auch von der *mittleren Weglänge*, denn hier geht der Suchalgorithmus als wertendes Element mit ein.

Es wird also Wissen über den Aufbau der Daten in der Datenstruktur verarbeitet und geht in die Bewertung ein.

Im Gegensatz dazu ist das Ziel der *mittleren Weglänge* eine Beurteilung der Topologie ohne Rücksicht auf den semantischen Gehalt des Graphes.

Betrachtet man die *Gesamtsuchzeit* als Wertungskriterium für Graphen, so stellt man fest, daß sich der Suchaufwand auch über die *mittlere Suchzeit* beschreiben läßt. Diese *mittlere Suchzeit* berechnet sich über die Formel

$$t_{mittel} = \frac{\sum t_{search}(n)}{n_{\mathcal{B}}}$$

die als Maßstab für die Qualität des Graphens unter dem spezifischen Suchalgorithmus dienen kann.

Auch hier ist wieder festzustellen, daß allgemein gilt : $t_{mittel} \neq \mathrm{mwl}_G(\mathcal{B})$, damit sollte sich ein weiterer Wert zur Beurteilung eines Graphens bieten.

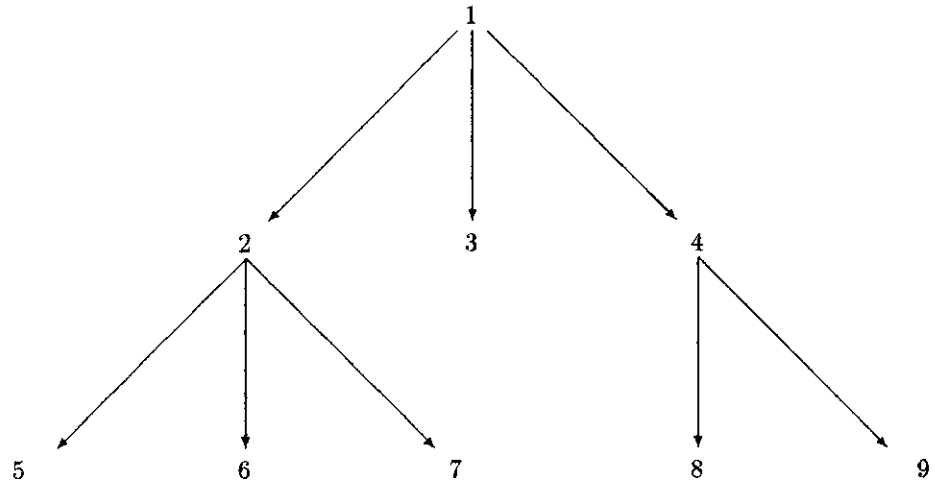
Bei nähere Betrachtung stellt sich jedoch heraus, daß diese Fitnessfunktion nicht als solche geeignet ist. Interessant hierbei ist, daß sich bei dem Ablauf keine Unterschiede in den Fitnesswerten feststellen lassen.

Man kann den Vorgang der Suche[9] auch durch eine Fädelung der Daten darstellen. Diese Fädelung findet man bei KNUTH[Knu68] und bei DENERT und

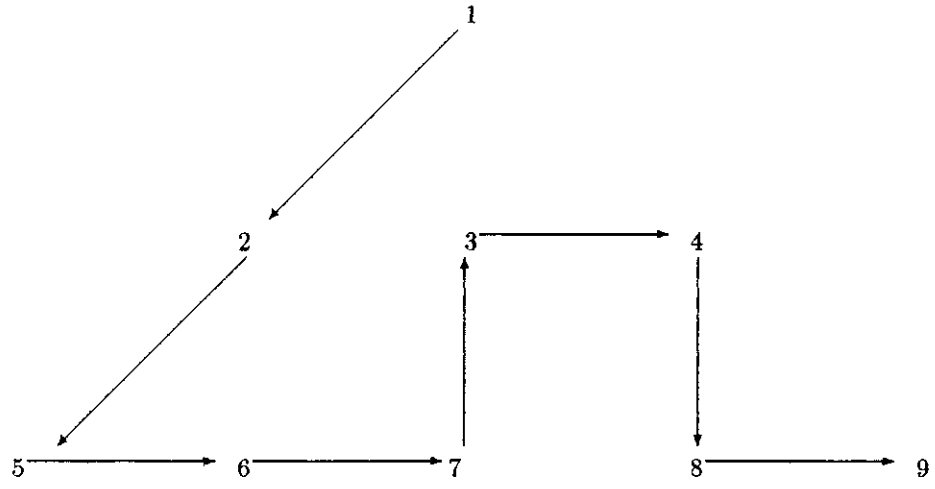
[9] sowohl bei der Tiefen- als auch bei der Breitensuche

FRANCK [Den77]. Diese Fädelung läßt sich wie folgt darstellen:

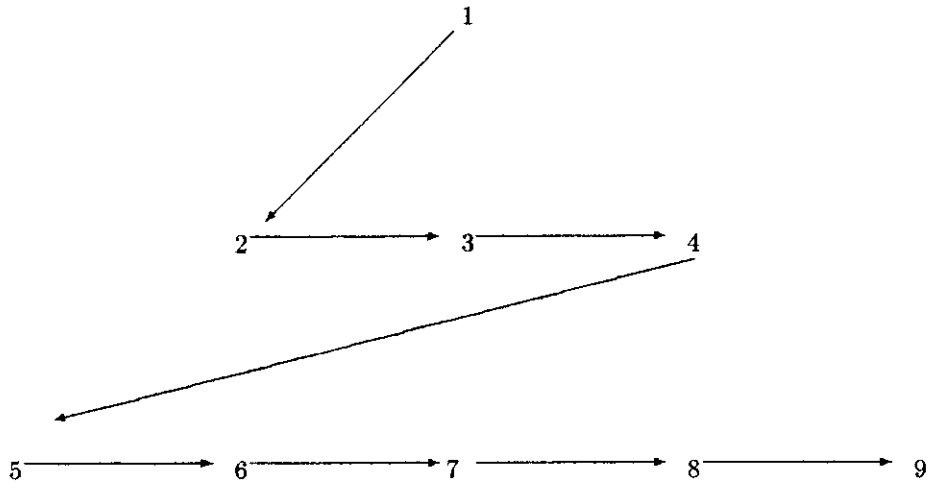
Gegeben sei der folgende Baum:



Anhand dieses Baumes werden nun die Daten nach der Reihenfolge ihres Auffindens in eine lineare Liste eingetragen. Dieses sieht für die Tiefensuche so:



und für die Breitensuche wie folgt aus:



Werden jetzt nicht nur einzelne Daten, sondern alle Daten in einer beliebigen Reihenfolge gesucht, so wird diese Liste jede Länge zwischen 1 und der Anzahl der Knoten genau einmal annehmen. Hiermit ergibt sich eine konstante Fitness für einen beliebigen Graphen mit n Knoten, welche sich nach der folgenden Formel berechnet:

$$Fitness = \frac{1}{n} \cdot \sum_{x=1}^{n} x$$

Dieser, zunächst nicht beachtete Nebeneffekt ist hinreichend, um die *Mittlere Suchzeit* als Fitnessfunktion abzulehnen.

Allerdings kann man unter dem folgenden Aspekt dennoch die Suchzeiten betrachten.

Man kann mit einigem Recht annehmen, daß nicht alle Daten in einem Datenbanksystem gleichhäufig gelesen oder manipuliert werden. Trägt man die Häufigkeit der speziellen Operation, z.B. der Suchvorgänge in eine Häufigkeitsliste ein, so erhält man die Möglichkeit, auch hier eine Relation $\sqsubseteq$ zu definieren, die der Definition einer Ordnung genügt.

Definition 4.13 (Suchzeiten bei der Nichtgeordneten Suche)
Sei $\mathcal{D}$ eine Datenstruktur, $x_1, x_2, \ldots, x_n \in \mathcal{D}$ Elemente und o eine Suchoperation auf $\mathcal{D}$. Dann gibt es eine Tabelle $H_{\mathcal{D}}$ der Häufigkeiten der Suchaufrufe mit n Elementen.

Dann gilt $x_A \sqsubseteq x_B$ genau dann, wenn x_A häufiger gesucht wird als x_B.

Diese Definition ist eine *Totale Ordnung* auf $\mathcal{D}$. Mit dieser Definition läßt sich sowohl für die *Geordnete* als auch für die *Nichtgeordnete Suche* feststellen, daß das Verhalten der Datenstruktur, damit auch des zugeordneten Graphen, insbesondere unter einer *Totalen Ordnung* interessant ist.

Hierbei ist nicht von Bedeutung, ob die Ordnung über den Schlüsselvergleich [10] oder über die Häufigkeitsverteilung erzeugt wird.

Als zweiter Teil der Fitnessfunktion wird also die vorne beschriebene Ordnung betrachtet, ohne jedoch auf die Einschränkung zu achten, daß die Datenstruktur in Form eines t-ären Baumes vorliegen muß. Diese Bedingung ist, z.B. bei der Ausnutzung der Suchzeiten nicht notwendigerweise verlangt.

4.3 Kombination der Fitnessfunktion

Wie schon oben beschrieben, setzt sich die Fitnessfunkton aus zwei Teilfunktionen zusammen. Diese wurden in den vorigen Abschnitten erläutert. Zur besseren Übersicht sollen sie an dieser Stelle noch einmal dargestellt werden.

Für die *Topologische Optimierung* ist die Fitnessfunktion mit Hilfe der *Mittleren Weglänge* definiert. Allerdings ist die *Mittlere Weglänge* negativ orientiert, d.h. f_1 ist je besser, je kleiner f_1 ist. Diese Funktion sollte zur Beurteilung jedoch einen positiven Wert haben. Zu diesem Zweck definiert man mit der *Maximalen Mittleren Weglänge* mwl_{max}. Diese berechnet sich nach KNUTH[Knu68] über die mittlere Weglänge der linearen Liste. Somit hat dieser Teil der Fitnessfunktion das folgende Aussehen :

$$f_1 = \text{mwl}_{max} - \text{mwl}_B(x'_y)$$

Für die *Optimierung der Ordnung* wird die im obigen Text definierte Bewertungsfunktion genutzt, die das folgende Aussehen hat:

$$f_2 = \sum^{m} y_m$$

mit

$$y_m = \begin{cases} 1 & : \; falls \; \exists k_m = (x_a, x_b) \wedge x_a \leq x_b \\ 0 & : \; falls \; \neg\exists k_m = (x_a, x_b) \\ -1 & : \; falls \; \exists k_m = (x_a, x_b) \wedge x_a > x_b \end{cases}$$

Um nun eine Gesamtfitnessfunktion zu definieren, muß eine Kombination aus diesen beiden einzelnen Fitnessfunktionen hergestellt werden. Diese Kombination geschieht über eine gewichtete Addition der Einzelfitnessfunktionen.

Hierfür wird ein Gewichtungsparameter r eingeführt, der die Gewichte auf die Einzelfunktionen verteilt.

Damit ergibt sich die Gesamtfitness nach der folgenden Formel[11]

$$Gesamtfitness = r \cdot f_1 + (1 - r) \cdot f_2$$

[10] wie bei der *Geordneten Suche*

[11] nach Goldberg[Gol89]

Wichtig hierbei der Wichtungsfaktor r, der die Verteilung der beiden Fitnessfunktionen auf die Gesamtfitness darstellt. Dieser Wichtungsfaktor kann einen beliebigen Wert zwischen 0 und 1 annehmen.

Bei der Einstellung 0 wird ausschließlich eine Wertung mit der Teilfitnessfunktion f_2 vorgenommen, die Einstellung 1 erzeugt eine Wertung über die Teilfitnessfunktion f_1, also mit der *Mittleren Weglänge.*

Alternativ kann man sich auch eine multiplikative Verknüpfung der beiden Einzelfitnessfunktionen vorstellen. Diese Verknüpfung sähe dann wie folgt aus:

$$Gesamtfitness = r \cdot f_1 \cdot f_2$$

wobei r ein Dehnungsfaktor zur Skalierung darstellt.

Interessant sind außer der Definition auch die Betrachtung der Randwerte und der möglichen Werteverläufe.

4.3.1 Betrachtung der Gesamtfitnessfunktion

Für die Betrachtung der Gesamtfitnessfunktion sei eine Datenstruktur mit einer beliebigen aber festen vorgegebenen Knotenanzahl k gegeben. Hiervon ausgehend kann man Grenzwerte der Einzelfunktionen und auch der Gesamtfitnessfunktion theoretisch ermitteln.

Zuerst soll die Funktion f_1 betrachtet werden. Hierzu wird bezug auf die schon vorne definierten Grenzwerte genommen und auf die veränderte Definition der *Mittleren Weglänge* für Graphen abgeändert. Diese Abänderung muß durch die folgende Überlegung gelenkt sein:

4.3.2 Maßnahmen zur Fitnessverbesserung

Um den Fitnesswert zu verbessern, werden drei Methoden vorgestellt, die auf verschiedene Art und Weise das Ergebnis in eine bestimmte vorgegebene Richtung lenken sollen. Es sollen nicht bestimmte Ergebnisse erzeugt werden, sondern es interessiert das Verhalten des Systems auf äußere steuernde Einflüße. Wenn das System stabil ist, so sollte sich dieser Einfluß in deutlichen Grenzen halten. Die erste der vorgestellten Methoden ist aus der Literatur[Gol89] entnommen, die anderen beiden sind theoretische Ansätze, von denen einer auch implementiert werden soll.

Die Methode des Generation Gapping

Die Methode des *Generation Gapping* geht von der Überlegung aus, daß es möglich sein muß, die Fitness durch Übernahme von Gen- Informationen zu steigern. Diese Idee, die von DE JONG[Gol89] stammt, formuliert ein sogenanntes *Generation Gap*[12].

[12] dieser Ausdruck kann mit Generationsüberlappung übersetzt werden

Dieser Mechanismus wird über einen Parameter G gesteuert, der mit Werten zwischen 0 und 1 besetzt wird. Der Wert ist wie folgt definiert:

$G = 1$: Keine überlappenden Generationen

$0 < G < 1$: überlappende Generationen

Unter einer Überlappenden Generation versteht man die unveränderte Übernahme von Individuen von einer Generation in die nächste. Die Anzahl der überlappenden Generationen wird über den Parameter festgelegt und nach der Formel $n \cdot G$ berechnet.

Hierbei bezeichnet n die Größe der Population in dem System. Es werden bei dieser Vorgehensweise also die besten Individuen einer Generation übertragen und damit wird ein Abfall der Fitness gebremst und die jeweils besten Individuen werden weiterhin in der Population vorkommen, denn oft wird eine gute oder sehr gute Gesamtfitness durch die Mechanismen der *Genetischen Algorithmen* zerstört und ein Verlust an Laufzeit ist die Folge. Man kann dieses Verhalten als eine Ventilfunktion betrachten.

Die Methode der Baumförderung

Bei der **Methode der Baumförderung** oder besser *Förderung der besten Ergebnisse* werden bestimmte Ergebnisse durch eine Verbesserung des Fitnesswertes gefördert. Als mathematische Funktion ausgedrückt, läßt sich der Zusammenhang wie folgt ausdrücken:

$$\text{Baumförderung}(x) = \begin{cases} 200 & : \text{ das Individuum entspricht den Bedingungen} \\ 0 & : \text{ das Individuum entspricht nicht den Bedingungen} \end{cases}$$

Ziel dieser Förderung ist eine Besserstellung bestimmter, besonders ausgezeichneter Individuen. Damit verlieren unerwünschte Ergebnisse ihren prozentualen Anteil an der Gesamtfitness und werden bei der Reproduktionsphase nicht mehr berücksichtigt.

Die Methode der Korrekturphase

Ein weiterer Effekt bei dem hier beschriebenen System sind die Individuen, die nach der konventionellen Methode nach Reproduktion, Kreuzung und Mutation den Anforderungen der Fitnessfunktion nicht gerecht werden. Diese Individuen kann man mit dem Begriff *Totgeburt* bezeichnen.

Definition 4.14 (Die Totgeburt) *Erfüllt ein Individuum nach Durchführung eines Zyklus nicht mehr die Anforderungen der Fitnessfunktion, also der Umwelt, so wird es im nächsten Zyklus automatisch eliminiert und die Charakteristika dieser Wesen scheiden aus dem Fortpflanzungskreislauf aus. Diese Individuen heißen "Totgeburten".*

Mit den *Totgeburten* vernichtet man aber auch ein möglicherweise nützliches Erbgut, deren Vernichtung nicht im Sinne einer gesunden Population sein kann.

Aus diesem Grunde erweitert man den Begriff der *Totgeburt* um den Begriff des *Handicapped Individuums.*

Definition 4.15 (Das Handikapped Individuum) *Ein "Handikapped Individuum" ist eine Totgeburt, die durch einen vorher definierten Algorithmus, der "Korrektur*[13]*" wieder zu einem Individuum gewandelt wird, das von der Umwelt akzeptiert wird.*

Diese Definitionen helfen dabei, eine Spezifikation der Korrekturfunktion durchzuführen. Um diese Korrekturfunktion zu definieren, benötigt man im Vorfeld eine Abstandsfunktion:

Definition 4.16 (Der Abstand zweier Individuen) *Der Abstand zweier Individuen ist definiert als*

$$x, y \in \mathcal{I} : x \ominus y : \sum_{i=1}^{lchrom} |x_i - y_i|$$

Hiermit kann man die Korrekturfunktion wie folgt definieren:

Definition 4.17 (Die Korrekturfunktion) *Eine Korrekturfunktion k ist definiert als*

$$k(\mathcal{I}) : \forall x : x \notin \mathcal{I} \rightarrow x' \in \mathcal{I}$$

mit der Bedingung, das gilt:

$$x' \ominus x \Longrightarrow min$$

Zusätzlich definiert man einen Wert δ_x, der den Defekt an dem Individuum $x \in \mathcal{I}$ beschreibt. Dieser Wert beschreibt die Anzahl der Änderungen, die notwendig sind, um eine Überführung von einer *Totgeburt* zu einem *Handikapped Individuum* durchführen zu können. Hierbei gilt, daß eine Änderung ein Wechsel eines Chromosom von 0 nach 1 oder von 1 nach 0 ist.

Ergänzend muß bei der Korrekturfunktion ein Schwellenwert angegeben werden, bei dessen Unterschreitung eine Korrektur nicht mehr erfolgen kann. Dieser Schwellenwert ξ kann als Parameter in das System eingehen und so das Verhalten der Individuen beeinflussen.

Unterschiede zu den Constraints

Die Theorie beschreibt die Möglichkeit der Bestrafung von Irregularien mit Hilfe einer Bestrafungsfunktion. Diese Bestrafungsfunktion, auch Einschränkung[14]

[13] ähnlich eines chirugischen Eingriffs
[14] engl. *Constraint*, vgl. Goldberg[Gol89], S.85 f.

genannt, verringert den Wert der Fitnessfunktion vor der Ausführung der Bewertung und Fortpflanzung.

Hierbei werden die Fitnesswerte herabgesetzt, wenn die Individuen den Anforderungen der Bewertungsfunktion nicht erfüllen. Damit erhalten diese Individuen eine Behinderung bei der Reproduktion durch die eingesetzten Algorithmen.

Man kann also den Einsatz der *Constraints* als einen destruktiven Eingriff in das laufende System bezeichnen.

Im Gegensatz hierzu ist der korrigierende Eingriff konstruktiv, d.h. Erbinformation wird erhalten. Desweiteren wird dieser Algorithmus nach der Ausführung der Fortpflanzungsphase angestossen, so daß die Individuen hiernach wieder die Beurteilungsphase durchlaufen.

Nicht optimale Erbeigenschaften werden dann durch die Beurteilung aus dem System eliminiert.

Sinn der Korrekturfunktion

Das Ziel der Korrekturfunktion ist das Erhalten von Erbinformationen, die für das Gesamtsystem wichtig und notwendig sind.

Damit gehen Informationen, die in fehlerhaften Individuen vorhanden sind nicht mehr verloren, sondern werden in den Kreislauf zurückgeführt. Sollte die Information trotz der Korrektur den Anforderungen nicht genügen, so wird sie durch die Beurteilungsphase aus dem System entfernt.

Durch den Einsatz der Korrekturphase wird eine deutliche Verbesserung des Ergebnisses[15] erhofft. Allerdings wird auf den Einsatz der Korrekturphase in diesem System verzichtet, da die Intention der Versuche mit dem Einsatz dieser Maßnahme nicht zu vereinbaren ist.

Bei Versuchen mit einem ähnlichen Verfahren sind allerdings von [Bra91] sehr gute Ergebnisse erzielt worden, so daß hier noch eine Möglichkeit in der Erzeugung bestimmter, gewollter Strukturen besteht.

[15] des Optimums, bzw. die Geschwindigkeit bis zum Erreichen des optimalen Wertes

Teil II

Experimente

Kapitel 5

Topologische Optimierung

5.1 Vorbemerkung

Die im folgenden beschriebenen Experimente basieren auf den Ergebnissen der theoretischen Überlegungen in den vorigen Kapiteln. Zusätzlich hierzu müssen jedoch einige Bedingungen erfüllt werden, damit die Ergebnisse auch entsprechenden Ansprüchen genügen.

Die erste Bedingung liegt in der festen Beschreibung der Experimentalumgebung. Diese Experimentalumgebung wird im Anhang ab Seite 111 dargestellt. Wichtig an dieser Stelle ist die Festlegung der modifizierbaren Parameter. Diese Parameter sind in der folgenden Tabelle dargestellt, zusätzlich zu den Parametern werden auch noch die möglichen Einstellungen angegeben:

Parameter	Wertebereich	Bedeutung des Parameters
`MaxGeneration`	0 - 1000	Anzahl der Generationen
`MaxKnoten`	0 - 100	Anzahl der Knoten
`MaxWesen`	0 - 100	Anzahl der Individuen
p_{mut}	0 - 1	Mutationswahrscheinlichkeit
`GenerationGap`	ja / nein	Generation Gap
`Baumförderung`	0 - 500	Baumförderung
`Constraints`	0, 1	0 → keine Graphen 1 → keine Bäume

Diese Parameter werden im Rahmen der Experimente modifiziert und die Veränderung werden protokolliert.

Die zweite Bedingung betrifft die Modifikation der Parameter. Diese Modifikation darf nur in wohldefinierten Schritten mit maximal einem Parameter gleichzeitig vorgenommen werden. Nur auf diese Weise kann gewährleistet werden, daß die Modifikation und die beobachtete Verhaltensänderung im System zusammengehörig sind.

Die dritte Bedingung bei der Durchführung der Experimente bezieht sich auf eine mehrmalige Durchführung der Einzelversuche. Um eine gewisse Sicherheit zu erreichen, daß die aufgetretenen Ergebnisse nicht einmalige Zufälle sind und deshalb nicht reproduzierbar, wurden die Versuche mehrmals, mindestens jedoch fünfmal durchgeführt. Diese Wiederholungen wurden ausgedehnt, wenn

- entweder das Ergebnis sehr uneinheitlich war, oder
- es nicht mit den Erwartungen übereinstimmte.

Einzelne Experimente[1] wurden bei der Durchführung bis zu 20-mal wiederholt. Aus diesem Grunde stellen die im folgenden dargestellten Kurvenverläufe keine exakten Einzelergebnisse dar, sondern es sind qualitative Kurven, bei denen besondere Verhaltensmuster verdeutlicht werden. Einige besonders deutliche Ergebnisse sind ab Seite 105 dargestellt.

Die letzte Bedingung für eine ordnungsgemäße Durchführung der Versuche ist die Vollständigkeit dieser Versuche. Es sollen alle Ergebnisse, die während der Versuche angefallen sind, auch dargestellt werden. Dieser Punkt soll verhindern, daß im weiteren Schlüsse gezogen werden, die nur auf Vermutungen oder Annahmen beruhen, außerdem sollen diese Schlüsse nachvollziehbar sein.

Kurvendarstellungen

Im weiteren werden die Verläufe in Form von Kurven dargestellt. Ein Diagramm besteht hierbei aus drei verschiedenen Kurven, die zusammen das Gesamtbild der Ergebnisse des Experimentes ergeben. Diese drei Kurven sind

1. die Kurve der optimalen Werte: diese Kurve gibt Auskunft über den Entwicklungsverlauf der optimalen Werte, also der Fitnesswerte des besten Individuums der jeweiligen Generation
2. die Kurve des berechneten Durchschnitts: die Kurve des Fitnesswertes, der sich als arithmetrisches Mittel über alle Individuen der Generation ergibt
3. die Kurve des schlechtesten Individuums: die Kurve des jeweils schlechtesten Individuums der laufenden Generation.

Alle weiteren Auswertungen beziehen sich auf diese Kurven, es sei denn, sie beziehen sich direkt auf die erzeugten Graphen.

5.2 Festlegung der verschiedenen Größen

Die erste Untersuchungsserie beschäftigte sich mit dem Versuch, sowohl eine repräsentative Knotenzahl als auch eine entsprechende Population zu finden.

[1] z.B. bei den Constraints oder der alternativen Fitnessfunktion

Diese Informationen sind für die weiteren Untersuchungen insoweit wichtig, als daß eine zu kleine Knotenzahl Ergebnisse hervorbringt, die nicht das Verhalten größerer Individuen beschreibt, eine zu große Knotenzahl wird unnötige Rechenzeit verbrauchen. Zum anderen ist es bei den Versuchen notwendig, das Zeitverhalten bei unterschiedlichen Knotenzahlen zu schätzen, um eine Aussage über das Zeitverhalten auch größerer Strukturen treffen zu können.

Die zweite Fragestellung bezieht sich auf die Größe der Population. Hierbei gilt: eine zu kleine Population wird sich nach einer gewissen Anzahl von Generationen an ein Individuum anpassen. Anpassung heißt in diesem Zusammenhang die Reduktion der gesamten Population auf das Erbmaterial eines Individuums. Damit ändert sich nach einer gewissen Zeit nichts mehr an dem Erbgut und nur noch die Mutation bringt noch Änderungen in das System. Dieser Effekt wird in der Biologie unter dem Begriff *Inzucht*[Kno83] geführt.

Eine zu große Population ist hinderlich bei der Kristallisation des Gesamtbildes. Denn sowohl bei dem ersten wie bei dem zweiten Parameter interessiert ein kleiner Wert, der dennoch den notwendigen Kriterien entspricht.

Bei den folgenden Versuchen wird deshalb wie folgt vorgegangen:
Alle Parameter in dem System werden konstant gehalten; bis auf jene, die untersucht werden. Die beiden Variablen werden ansonsten getrennt untersucht. Die erste Untersuchung betrifft die Populationsgröße , die zweite die repräsentative Knotenzahl.

5.2.1 Untersuchung der Populationsgröße

Zur Bestimmung der optimalen Populationsgröße wurde die Knotenanzahl so hoch angesetzt, daß sie nicht zu klein sein konnte. Hier wurde eine Knotenanzahl von 100 Knoten gewählt. Bei den weiteren Parametern wurde der Fall der größtmöglichen Einschränkung[2] gewählt.

Es wurde also eine Mutationswahrscheinlichkeit $p_{mut} = 0$, die größte Einschränkung bei den Constraints, kein Generation Gapping und keine Baumförderung gewählt. Hiernach wurde die Anzahl der Individuen zwischen 0 und 100 variiert. Die Schrittweite wurde dabei auf 5 Individuen eingestellt. Ab einer bestimmten Größe wurde die Schrittgröße zuerst auf 2, dann auf 1 reduziert. Der Anfangswert bei der Untersuchung liegt bei einer Population von 100. Die Anzahl der laufenden Generationen beschränkte sich dabei auf 1000.

Bei der Durchführung der Versuche stellte sich nach kürzester Zeit heraus, daß sich die gesamte Population einem Individuum angleicht. Im Durchschnitt herrscht eine Mortabilitätsquote von 70 bis 80 Prozent vor. Dieses negative Ergebnis konnte nach Analyse nur zwei Ursachen haben:

1. Die Population ist zu gering und das Erbmaterial ist deshalb nicht gut genug.

[2] der *worst case*

2. Die Constraints sind zu streng, d.h. es werden zu viele Individuen erzeugt, die den Anforderungen nicht genügen,

Zur näheren Erforschung des Effektes wurde dann das Programm so modifiziert, daß auch eine größere Anzahl von Individuen bearbeitet werden konnten.

Eine Ausdehnung auf 1000 Individuen brachte bei der Durchführung keine Veränderung im Verhalten. Es läßt sich also feststellen, daß die Geschwindigkeit des Aussterbens linear abhängig von der Anzahl der Individuen ist. Es ist also anzunehmen, daß der beobachtete Effekt nicht auf die Größe der Population zurückzuführen ist.

Deshalb wurde die Versuchsreihe mit einer veränderten Constraint- Einstellung wiederholt. Bei dieser Veränderung wurden alle Constraints aus dem System entfernt. In der Wiederholung stellte sich nun heraus, daß schon geringe Populationen eine recht hohe Lebenserwartung haben. So kann schon eine Population von 10 Individuen ohne Mutation im Durchschnitt etwa 600 Generationen, eine Population von 15 Individuen sogar 1200 Generationen überleben.

Selbstverständlich sind diese Zahlen abhängig von dem Kreuzungsalgorithmus, bei einer Änderung muß diese Untersuchung wiederholt werden. Bei dieser ersten Untersuchung wurde mit dem ersten Kreuzungsverfahren, der Kreuzung mit Austausch des ersten und dritten Quadranten gearbeitet. In diesem, speziellen Fall scheint es so, daß ein Erweitern der Population um 5 Individuen nahezu eine Verdoppelung der Überlebensdauer zu Folge hat. Um diese Aussage zu verifizieren, wurden Versuche mit den Populationsgrößen 5, 10, 15, 20 und 25 gemacht, die gegen diese Vermutung nicht verstießen.

Laufzeitverhalten bei Zunahme der Population

Zur Messung des Laufzeitverhaltens wurden alle interaktiven und graphischen Elemente aus dem System genommen. Die Laufzeitmessung erfolgte mit Hilfe eines Programms des Betriebssystems. Dieses wurde notwendig, da das eingesetzte Betriebssystem als Mehrplatzsystem keine sinnvolle Handmessung zuläßt. Man stellt fest, daß sich die Laufzeit bei Hinzunahme eines Individuums um etwa 15 CPU-Sekunden vergrößert. Die exakten Zeiten können der folgenden Tabelle entnommen werden:

Anzahl der Individuen	CPU - Sekunden
4	1:14
6	1:32
8	2:05
10	2:29
20	5:24
30	7:18
40	9:12

Der ungleich geringere Wert für die erste Messung ergibt sich aus dem Über-

gewicht des Grundprogrammes, bei dem die Größe der Population keine Rolle spielt.

Bei der Hinzunahme der Graphischen Oberfläche und der Interaktiven Komponenten vergrößern sich die Zeiten kaum. Allerdings entsprechen die 5, 5 CPU-Minuten der Messung dann etwa 45 Minuten Laufzeit, so daß die Größe von 20 Individuen durchaus die obere Grenze des Sinnvollen darstellt.

5.2.2 Festlegung der Anzahl der Knoten

Für die Durchführung dieser Versuche galten im weiteren die obigen Einstellungen. Lediglich die Anzahl der Individuen wurde konstant mit 20 angenommen, und die Anzahl der Knoten variierte im weiteren. Beginnend bei 5 Knoten wurde die Größe des Graphen bis auf 100 Knoten vergrößert. Diese Vergrößerung ist in Abständen zu je 5 Knoten vorgenommen worden. Bei diesem Vorgang sind die folgenden Ergebnisse aufgetreten:
Das Verhalten des Graphen ändert sich kaum mit der Zahl der verwendeten Knoten. Es ist lediglich zu bemerken, daß der Verlauf der Ergebniskurve bei steigender Knotenanzahl an der Generationenachse gedehnt wird.

Diese Dehnung ist nach einer ersten Einschätzung linear von der Anzahl der Knoten abhängig. Die Verifikation dieser Vermutung erwies sich als sehr schwer durchführbar. Denn es fehlte insbesondere ein Maß zur Bestimmung des Verlaufes der Kurve, so daß für die Verifikation dieser Vermutung ausschließlich eine Abschätzung zur Verfügung stand. Für diese Abschätzung ist das folgende Vorgehen gewählt worden:
Gezählt wurden die Generationen bis zum Erreichen einer bestimmten, hohen Fitness mit der Mehrzahl der Individuen. Hierzu wurde der Verlauf des Durchschnittsteils der Ergebnisskurve betrachtet.

Bei der Untersuchung wurde die Generation gemessen, bei der dieser Durchschnittsteil die Grenze von 90 Prozent der möglichen Fitness erreichte.
Es ergaben sich die folgenden Werte:

Anzahl der Knoten	Generationen bis zum Erreichen der 90 Prozent
10	4
14	7
20	9
26	13
30	22

Mit Hilfe dieser Messung stellt man nun fest, das bei Hinzunahme eines Knotens mit einer Verschlechterung des Ergebnisverhaltens, d.h. mit einer Dehnung an der Generationenachse um 0, 24 Generationen zu rechnen ist.

Unter diesen Voraussetzungen erschien eine Knotenzahl von 20 als sinnvoll, da das Verhalten schon deutlich zu erkennen und außerdem auch ein zügiger Ablauf der Optimierung gewährleistet ist.

Zeitverhalten bei Hinzunahme eines Knotens

Anders als das Verhalten des Ergebnisses bei Hinzunahme eines Knoten verhält sich das System bei dem Verbrauch effektiver Rechenzeit.
Durch die Struktur der Individuen als Matrix wird die Hinzunahme eines Knotens durch Erweiterung der Matrix um eine Zeile und eine Spalte realisiert. Dies heißt, daß bei einer Erweiterung eines Graphen mit n Knoten um einen Knoten die Matrix um $2n + 1$ Gene wachsen wird.

Da dieser Faktor ein Polynom darstellt, wird sich auch die Laufzeit in dem System polynominell vergrößern. Zur Kontrolle dieser These wurde ebenso wie bei der Untersuchung zur Anzahl der Individuen eine Messreihe durchgeführt. Die Ergebnisse sind der folgenden Tabelle zu entnehmen:

Anzahl der Knoten	CPU - Sekunden
3	1:06
4	1:15
5	1:21
10	1:56
15	3:15
20	5:05
25	6:26
30	8:47

Das Ergebnis dieser Messung kann man als eine Bestätigung der obigen Vermutung werten, daß das System durchaus nicht $\mathcal{NP}$-vollständig ist, da feststellbar ist, daß hier ein Wachstum vorliegt, daß mit einem Polynom 2. Grades beschrieben werden kann[3].

5.3 Versuche zur Optimierung

Im Gegensatz zu den ersten Versuchen interessiert bei der folgenden Serie das Verhalten des Systems bei der Optimierung. Ziel dieser Untersuchungsserie ist die Bestimmung der Parameter zum Erreichen eines optimalen Fitnessgrades und damit zu dem besten Optimierungsverhalten des Individuums. Hierbei wurden zuerst die Standarteinstellung, danach die einzelnen Parameter variiert und die Veränderungen ausgewertet.

5.3.1 Auswertung des Durchlaufes mit Standartparametern

Wie anhand der folgenden Kurve zu erkennen ist, liegt eine ausgeprägte Konvergenz zum Optimum vor. Schon bei dem Einsatz der Standartparameter ist

[3] Bei einem Wachstum mit einem Polynom 3.Grades würde eine größere Steigerung erreicht.

eine schnelle und stetige Optimierung zu erkennen. Innerhalb von 20 Generationen wird ein Individuum mit einer Fitness von mehr als 95 Prozent erzeugt. Da dieses Ergebnis so außergewöhnlich war, wurde der Versuch mehrmals wiederholt. Bei dieser Serie von Wiederholungen ergab sich, daß 18 von 20 Versuchen dieses Verhalten zeigen. Ein durchschnittlicher Wert von mehr als 90 Prozent Optimalität wurde in 17 von 20 Versuchen nach spätestens 50 Generationen erreicht. Nur bei einem Versuch wurde eine Fitness unter 91 Prozent erreicht. Bei allen übrigen Versuchen erreichte die Fitness einen Wert von im Durchschnitt 93 Prozent bei den besten Individuen. Dabei war das generelle Verhalten bei allen Versuchen so, wie es in dem Graphen dargestellt wurde.

Bei genauer Betrachtung des Graphen läßt er sich in drei Teilgraphen oder Abschnitte zerlegen. Die Abschnitte werden im weiteren durch I, II und III gekennzeichnet. Wie im weiteren zu sehen sein wird, liegt die Vermutung nahe, daß die Art der Abschnitte abhängig von der Parametereinstellung ist.

Abschnitt I beginnt mit der ersten Generation und endet mit dem Erreichen eines ersten lokalen Maximums, das bei dem meisten Versuchen sogar ein globales Maximum ist. Dieser erste Abschnitt hat bei den Versuchen mit den Standartparametern eine durchschnittliche Dauer von 18 bis 20 Generationen. Anschließend an diesem ersten Abschnitt folgt eine Phase mit abfallenden Fitnesswerten, die Individuen verlieren im Schnitt etwa 10 Prozent ihrer Fitness. Diese Phase dauert circa ebenso lange wie Abschnitt I und wird mit der Abschnittnummer II gekennzeichnet. Neben der Länge der Phase interessiert bei diesem Abschnitt auch das Maß des Fitnessverlustes, wobei dieser nach Möglichkeit klein zu halten ist.

Die dritte und letzte Phase zeigt eine Kurve ähnlich eines physikalischen Einschwingvorgangs. Dieser Abschnitt ist bei dem Einsatz der Standartparametern gekennzeichnet durch regelmäßige Schwingungen mit kleiner Amplitude, welche jedoch im Verlauf kontinuierlich abnimmt und schließlich in einer Linie endet. Die Dauer dieser Phase ist innerhalb der Zeit nicht beschränkt.

Entsprechend des Verlaufes der Phasen, ihres Eintrittzeitpunktes und ihrer Ausprägung kann man nun den Verlauf der Optimierung bei der Veränderung der Parameter beschreiben.

Interessant ist aber auch, daß dieser Verlauf der Kurven schon bei anderen Versuchen aufgetreten ist. So beschreibt LUCANSIUS[Luc91] ebenfalls als Ergebnis eine Kurve, die der Kurve in dieser Arbeit entspricht.

Es kann nun ein Ziel für die Veränderung der Parameter definiert werden:

1. Der Abschnitt I sollte verkleinert werden und die Steigung sollte vergrößert werden,

2. der Abschnitt II sollte ebenfalls verkürzt werden oder der Abfall der Amplitude verkleinert werden, und

3. es sollten auch im Abschnitt III Verbesserungen in der Fitness stattfinden.

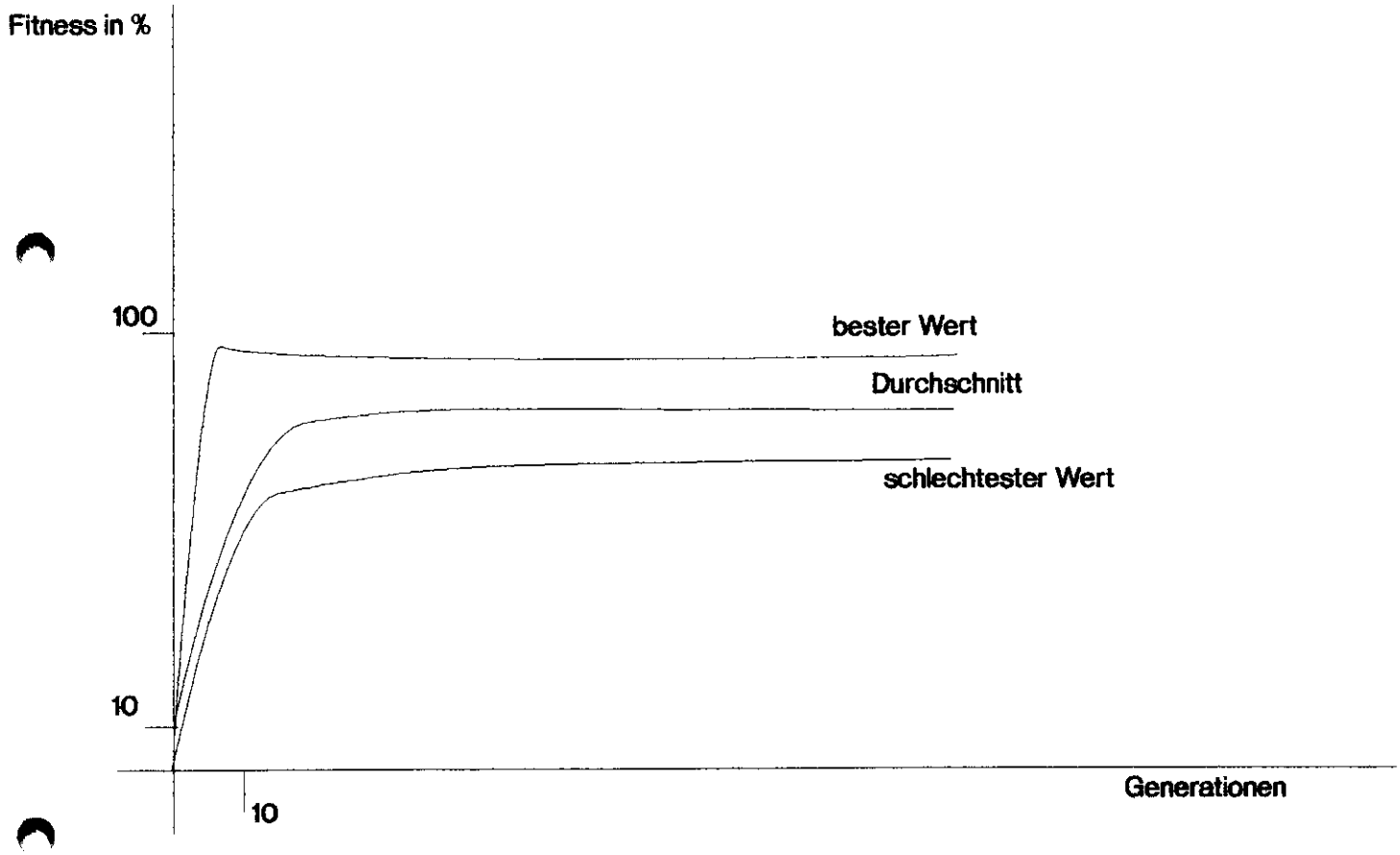


Abbildung 5.1: Kurvenverlauf mit Standartparametern

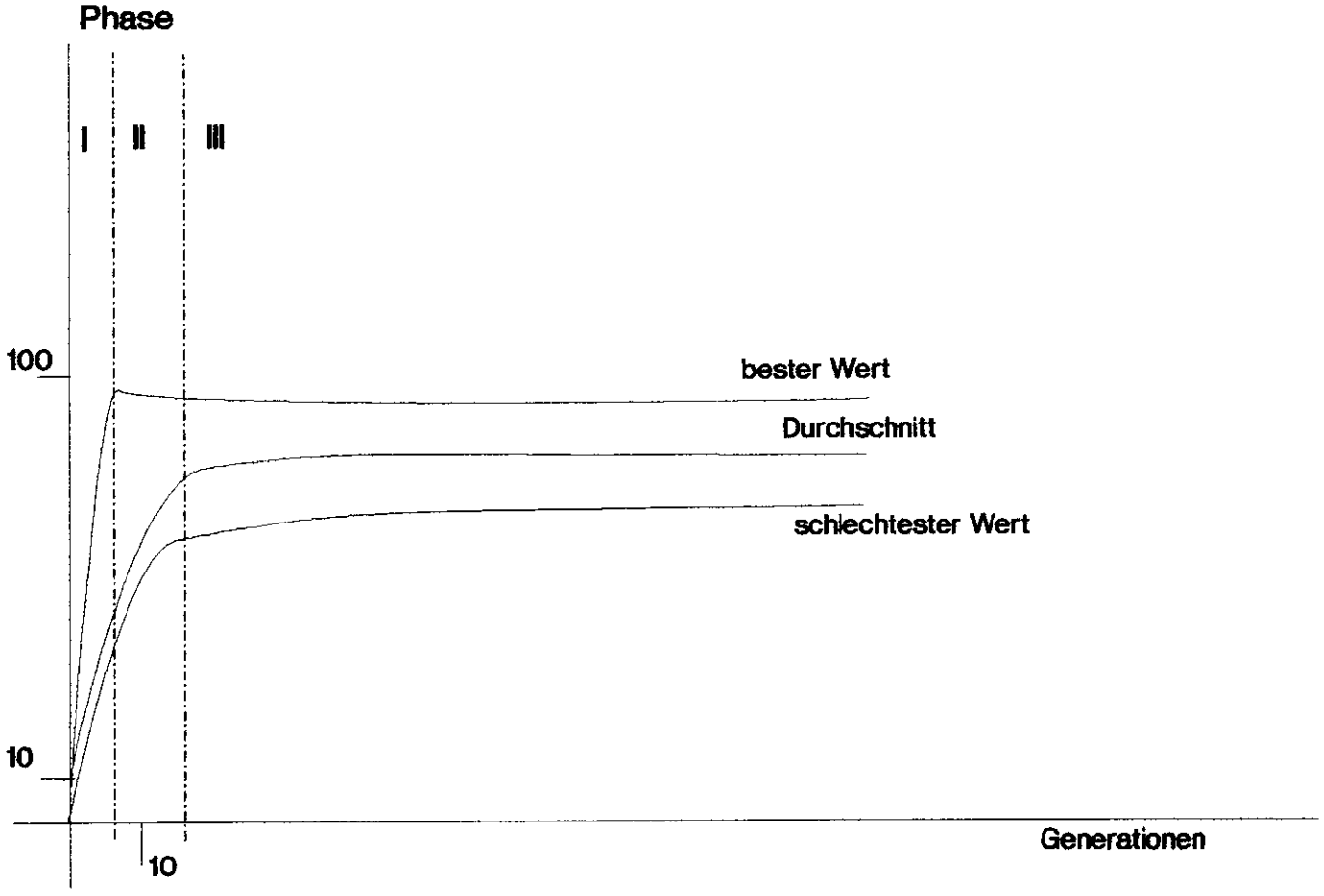


Abbildung 5.2: Darstellung des Phasenschemas

Diese Verbesserung sollten entweder erzeugt werden können oder alternativ ausgeschlossen werden.

Um die Ziele zu erreichen, sollen die verschiedenen Methoden der GENETISCHEN ALGORITHMEN eingesetzt werden.

5.3.2 Der Einsatz von Constraints

Mit dem Einsatz von Constraints soll insbesondere einer Steigerung des Optimierungsverhaltens erreicht werden. Dies geschieht insbesondere dadurch, daß Individuen, die nicht den Anforderungen entsprechen, eliminiert wurden.

Diese Anforderungen sind Beschränkungen in der Form des Graphen. Im Ablauf wird nun zuerst die Existenz der *Wurzellosen Graphen*, danach auch die der *Wälder* verboten. Unter diesen Voraussetzungen werden nun die Experimente durchgeführt.

Verbot von wurzellosen Graphen

Mit dem Verbot der wurzellosen Graphen sollen *Wälder* und *Bäume* in der Population gefördert werden. Hierdurch wird eine Verbesserung der Durchschnitts- und damit auch der besten Fitness erreicht werden.

Bei der Durchführung der Experimente zeigten sich die folgenden Ergebnisse:
Es ergab sich bei keinem Versuch eine Verbesserung der Fitnesswerte, es wurde in keinem Versuch eine Fitness erreicht die über 92 Prozent der möglichen Fitness lag. Auch bei der durchschnittlichen Fitness wurden die Ergebnisse der Versuche ohne Constraints bei keinem Versuch erreicht, sie lagen gewöhnlicherweise um zwei Prozentpunkte niedriger.

Im Gegensatz dazu gleichen sich die Kurvenverläufe. Unterschiede bestehen in Form und Ausprägung der ersten und der dritten Phase, während Phase II der ersten Versuchsserie gleicht. Diese Unterschiede sind bei der ersten Phase am deutlichsten ausgeprägt. Es ergibt sich eine sichtbare Verlängerung dieser Phase um 50 Prozent. Bei der dritten Phase sind die Veränderungen nicht ganz so gut zu beobachten, aber es scheint, daß die Amplitude im Schnitt nicht die Höhe der ersten Versuche erreicht und die Schwingung früher zu einem konstanten Wert führt. Während sich dieser Effekt nur durch subjektive Beobachtung beschreiben läßt, ist der erste durchaus meßbar. Durch eine Wiederholung des Versuches stellte sich heraus, daß die Verlängerung der Phase in der oben beschriebenen Form bei 14 von 15 Versuchen eintritt.

Verbot von wurzellosen Graphen und Wäldern

Im Gegensatz zu dem Verbot von wurzellosen Graphen ergibt sich bei dieser neuen Einstellung ein ganz anderes Ergebnisbild. Wie schon auf Seite 62 beschrieben, herrscht bei dieser Konstellation eine Mortabilität von 80 Prozent

vor. Aus dieser Eigenschaft resultieren sicherlich auch die folgenden Ergebnisse: Das Phasenmodell ist auf diesen Ablauf nicht anwendbar. Es findet aber auch keine eigentliche Entwicklung statt. Der Verlauf des Ergebnisgraphen unterscheidet sich vollständig von den vorherigen. Wie deutlich zu erkennen ist, wird bei dieser Einstellung lediglich das beste, in der Population vorkommende Individuum selektiert und es konzentriert sich das gesamte System auf dieses beste Individuum. Die Geschwindigkeit der Selektion scheint direkt abhängig von der Größe der Population zu sein. Dieser Effekt wurde nicht weiter untersucht.

Betrachtung der Ergebnisse

Anhand der durchgeführten Untersuchungen ist das folgende Fazit zu ziehen: Da jeder der beiden Constraints eine deutliche Verschlechterung der Ergebnisse mit sich bringt, ist der Einsatz von Constraints bei den laufenden Untersuchungen abzulehnen. Auch sind bestimmte graphische Strukturen nicht mit Hilfe der Constraints zu erreichen. Insbesondere kann man auf diese Weise keine Bäume erzwingen. Deshalb werden bei den weiteren Untersuchungen keine Constraints mehr eingesetzt.

5.3.3 Einsatz der Baumförderung

Im Gegensatz zu den Constraints geht die Baumförderung einen anderen Weg. Diese Unterschiede sind in den vorderen Kapiteln beschrieben worden. Doch bei der Betrachtung der Ergebnisse stellt man fest, daß der Einsatz der Baumförderung keinen Gewinn für die Aufgabenstellung mit sich bringt. Auch bei dem Einsatz der Baumförderung ist der Ablauf in drei Phasen zu unterteilen, die sich jedoch bei den ersten beiden Phasen grundlegend von den zuerst beschriebenen unterscheiden. In der ersten Phase sinkt der beste Wert von einem recht hohen Anfangswert auf einem durchschnittlichen ab. Dieser Endpunkt stellt ein lokales, oft sogar ein globales Minimum dar. Hiernach tritt das System in die Phase II, bei der die Werte wieder um einen geringen Teil steigen, um dann schließlich in der Phase III wieder in einem Einschwingprozess zu enden.

Das System erreicht jedoch nicht mehr die Anfangswerte, sondern nur noch einen Wert von circa 60 Prozent der Anfangswerte. Diese Anfangswerte selber liegen etwa bei 75 Prozent der optimal erreichbaren Werte. Auch bei mehrfacher Wiederholung ist keine Veränderung in dem Verhalten des Systems festzustellen. Dieses Verhalten disqualifiziert die *Baumförderung*, da die recht guten Anfangswerte nicht auf den Einsatz der GENETISCHEN ALGORITHMEN, sondern nur auf dem Algorithmus zur Erzeugung der Individuen zurückzuführen ist. Deshalb wird im weiteren keine Baumförderung mehr eingesetzt.

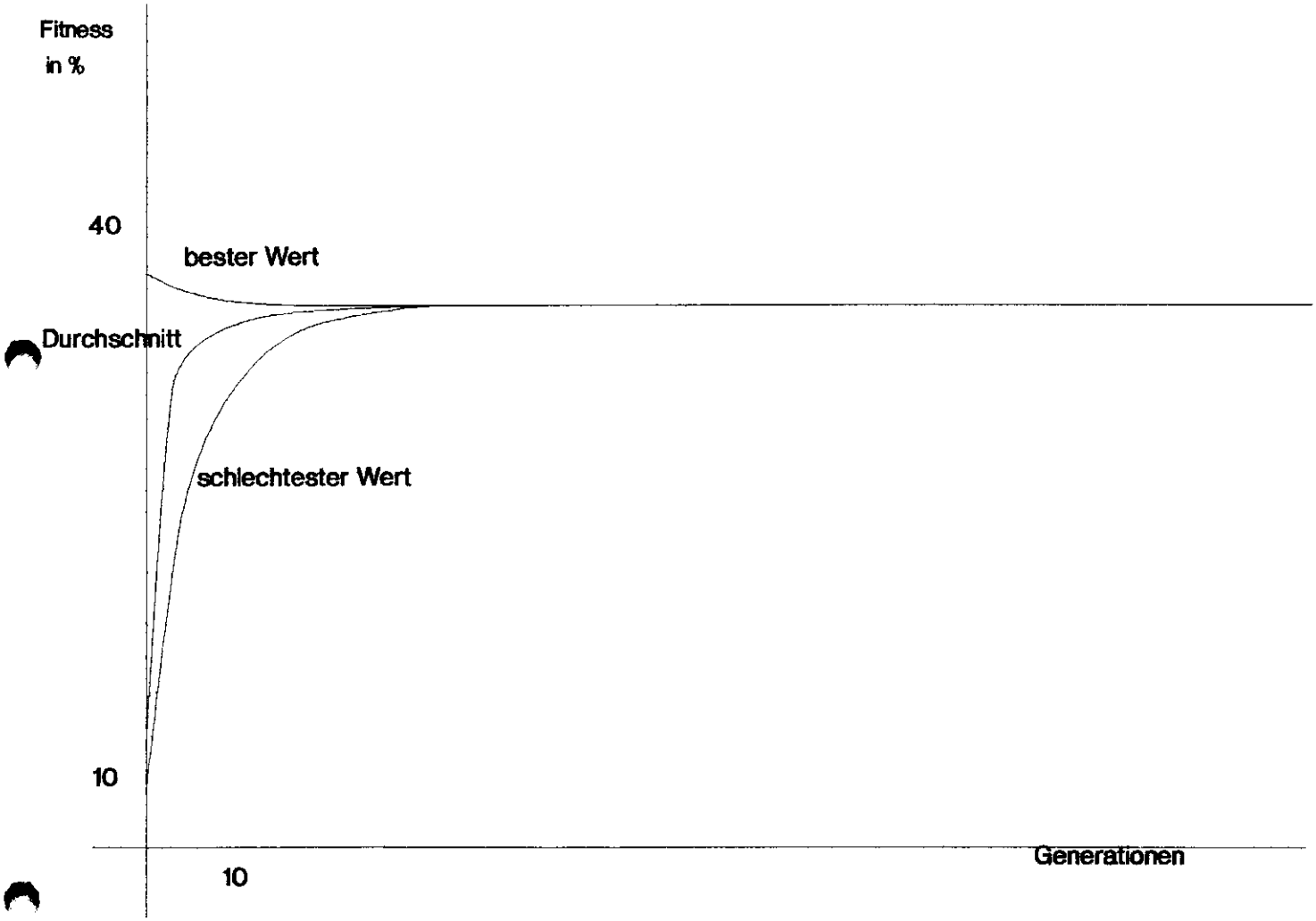


Abbildung 5.3: Kurvenverlauf mit maximalen Constraints

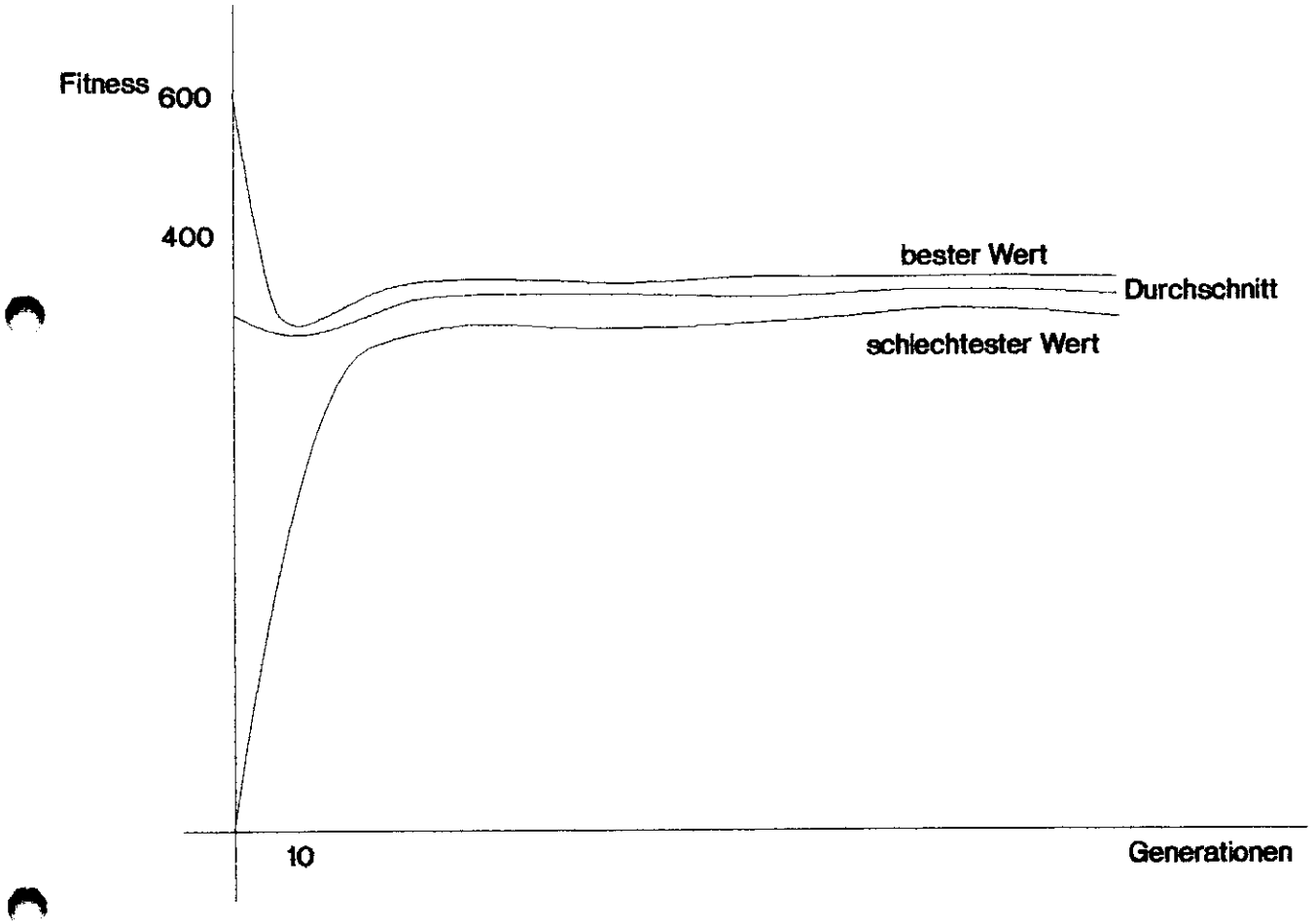


Abbildung 5.4: Kurvenverlauf bei Einsatz der Baumförderung

5.3.4 Einsatz des Generation Gapping

Mit dem Einsatz des *Generation Gapping* soll eine Ventilfunktion eingeführt werden, die ein schnelles Absinken der optimalen Werte verhindern soll. Hierbei wird, wie auf Seite 55 beschrieben, jeweils das beste Individuum der laufenden in die nächste Generation übernommen.

Der Einsatz dieser Funktion in dem System hatte die folgenden Effekte für die Ergebnisse der Durchläufe:
Das *Generation Gapping* hat keinen Effekt auf das Verhalten der schlechtesten Werte. Auch bei den durchschnittlichen Werten ist keine Verhaltensänderung festzustellen. Deutliche Änderungen ergeben sich jedoch bei der Kurve der optimalen Werte. Während bei den Untersuchungen auf Seite 66 eine Ergebnisfunktion in Form einer Schwingung vorliegt, hat bei dem Einsatz des *Generation Gapping* diese Kurve eine vollständig andere Form.

Der Kurvenverlauf ist nicht mehr schwingend, sondern gleicht eher einer Treppenfunktion, die in ihrer Art monoton steigend ist. Trotzdem ist eine Steigerung der Fitness nicht erreicht worden. Es gab bei allen Versuchen keinen einzelnen mit einer Fitness, die höher war als bei den Versuchen ohne *Generation Gapping.*

5.3.5 Veränderungen der Mutationswahrscheinlichkeit

Sinn der Mutation ist eine Auffrischung eventuell zu homogener genetischer Strukturen. Um die Wirkung der Mutation zu untersuchen, wird die folgende Untersuchungsserie durchgeführt. Hierzu wird bei einer Beibehaltung der Standartparameter die Mutationswahrscheinlichkeit von $p_{mut} = 0,00005$ bis auf eine Mutationswahrscheinlichkeit von $p_{mut} = 0,5$ erhöht. Diese Erhöhung findet in Dezimalpotenzen statt, so daß 5 Untersuchungen durchgeführt werden.

Dabei ist die letzte Untersuchung mit einer Mutationswahrscheinlichkeit von $2:1$ nicht mehr praktikabel, ihre Durchführung dient ausschließlich der Kontrolle und dem prinzipiellen Interesse. Denn diese Mutationswahrscheinlichkeit verändert in dem System jedes zweite Gen.

Bei der Auswertung dieser Versuche stellt man jedoch fest, daß eine Veränderung dieses Parameters keinerlei Einfluß auf das Verhalten der optimalen Werte hat. Die auftretenden Unterschiede können weder als typisch für die eingestellte Mutationswahrscheinlichkeit gelten, noch können sie in irgend einer Form als signifikant bezeichnet werden. Anders ist das Verhalten bei dem Kurvenverlauf der schlechtesten Werte. Bei diesem Kurvenverlauf stellt man bei zunehmender Mutationswahrscheinlichkeit auch eine zunehmende Unstetigkeit dieser Kurve fest. Das Extremum dieser Veränderung wird bei einer Mutationswahrscheinlichkeit von $p_{mut} = 0,5$ erreicht, bei der keine Stetigkeit mehr festzustellen ist. Hierbei erscheint dann auch die Stetigkeit der Durchschnittskurve nicht mehr vorhanden.

Die vergleichsweise besten Werte wurden mit einer Mutationswahrscheinlich-

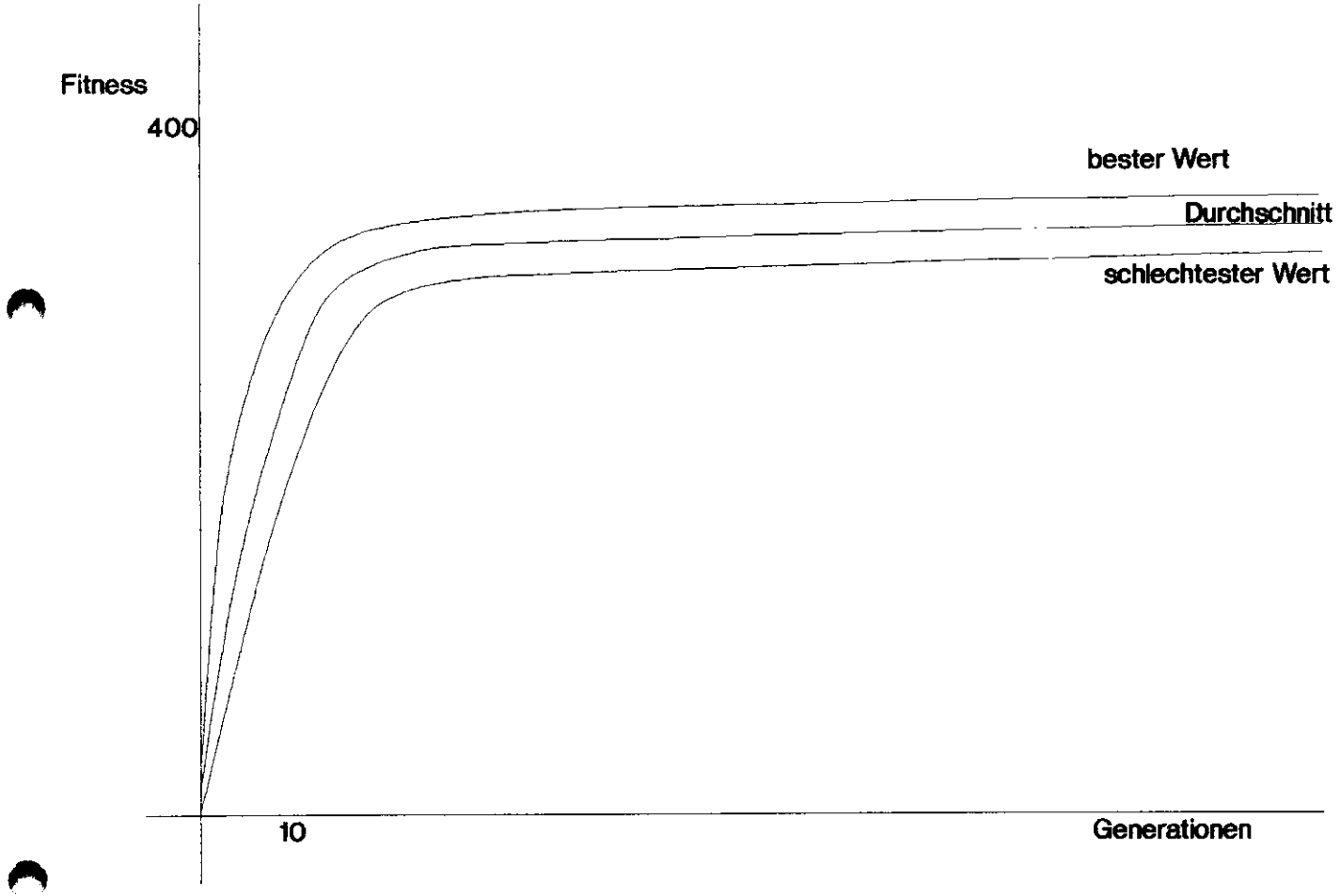


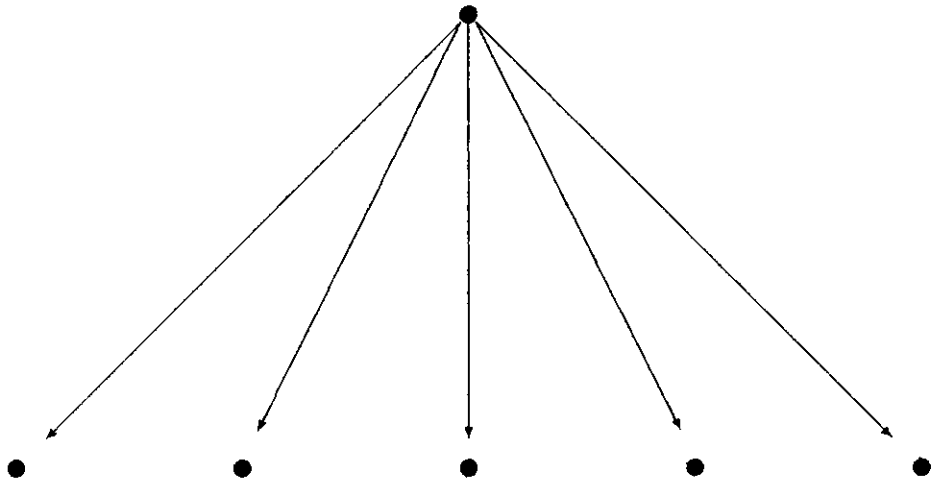
Abbildung 5.5: Kurvenverlauf bei Einsatz des Generation Gapping

keit von $p_{mut} = 0,0005$ erreicht, obwohl auch hier nicht von signifikanten Ergebnissen geredet werden kann, da nur eine Verbesserung um 1,2 Fitnesspunkten im Vergleich zu den Durchläufen ohne Mutation erreicht wurde. Außerdem ist bei dieser Mutationswahrscheinlichkeit auch noch mit einer stetigen Durchschnittskurve zu rechnen. Von den fünf durchgeführten Versuchen war diese Kurve bei vier Versuchen an nicht mehr als zwei Stellen unstetig. Diese Unstetigkeit war bei 80 Prozent auf eine sprungartige Verbesserung der optimalen Werte und damit auch auf eine Unstetigkeit in der Kurve der optimalen Werte zurückzuführen.

Abschließend zu den Versuchen mit der Mutationswahrscheinlichkeit kann das folgende festgestellt werden:
Der Einsatz der Mutation wirkt sich in diesem System mehr als störend aus, man kann in diesem Zusammenhang von einer destruktiven Funktion sprechen. Aufkommende feste Strukturen werden durch diese Funktion schon vor der Stabilisierung zerstört und vernichtet. Deshalb sollte auf dem Einsatz der Mutation bei diesem System verzichtet werden.

5.3.6 Darstellung der erzeugten Graphen

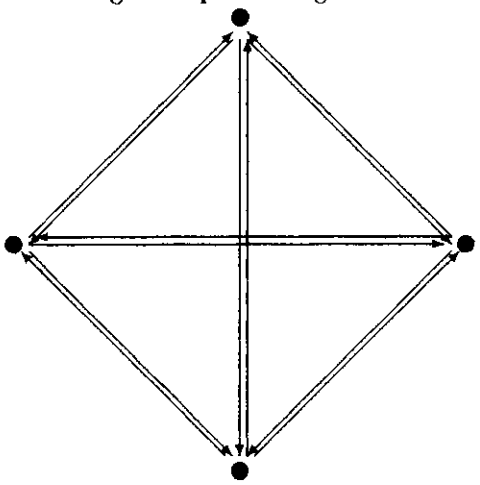
Bei den bisherigen Ergebnissen wurden nur die Ergebnisse der Fitnessfunktion betrachtet, nicht aber die erzeugten Graphen. Ausgehend von den Anfangsgraphen bildet sich stets eine spezielle Form des Graphen heraus. Diese spezielle Form hat das folgende Aussehen:



(Graph mit 6 Knoten)

Wie deutlich zu erkennen ist, handelt es sich um einen Wurzelbaum mit einer Wurzel und $(n-1)$ Blättern auf der ersten Ebene. Einzelne Exemplare weisen auch Querverbindungen innerhalb der ersten Ebene auf, ohne daß dieses Auswirkungen auf den Fitnesswert hätte. Dieses Bild wird genau dann erzeugt,

wenn die Wurzel konstant gehalten wird. Wird die Wurzel bei jedem Durchlauf neu bestimmt, so hat der erzeugte Graph das folgende Aussehen:



(Graph mit 4 Knoten)

Beiden Graphen ist gemein, daß sie eine minimale Weglänge aufweisen und beide werden von der Fitnessfunktion als optimal anerkannt.

Die Verringerung der Fitness wird durch Blätter auf der zweiten oder gar dritten Stufe erzeugt. Die Grundform bleibt jedoch im ganzen erhalten. Entsprechende Darstellungen finden sich bei den Bildschirmprotokollen im Anhang.

5.4 Versuche mit der alternativen Fitnessfunktion

Zur Kontrolle des Verhaltens der GENETISCHEN ALGORITHMEN und um die Bedeutung der Fitnessfunktion nachzuweisen, wird eine Kontrolluntersuchung mit einer veränderten Fitnessfunktion durchgeführt.

Sollte sich bei der Untersuchung keinerlei Veränderung im Verhalten zeigen, so ist die Wirksamkeit des Systems überhaupt in Frage gestellt. Deshalb muß bei dem Einsatz einer anderen Fitnessfunktion ein deutlich anderes Ergebnis als bei der vorigen auftreten. Unter Beibehaltung der Experimentalumgebung wird statt der Fitnessfunktion *Mittlere Weglänge* in der nächsten Serie die auf Seite 47 beschriebene Kostenfunktion als Fitnessfunktion benutzt.

Nach mehrmaligem Durchlauf ergibt sich das folgende Bild:
Anders als bei den vorigen Versuchen zeigt die Kostenfunktion ein sehr uneinheitliches Bild mit vielen, unvorhersehbaren Sprüngen. Es ist weder eine Monotonie noch eine Stetigkeit in der Kurve der optimalen Werte festzustellen. Die Kurve des berechneten Durchschnitts zeichnet sich ebenfalls durch einen absolut

uneinheitlichen Verlauf aus. Es läßt sich, im Gegensatz zu den ersten Versuchen, weder eine Prognose noch gar ein Phasenmodell erstellen.

Auch nach mehrfachen zusätzlichen Versuchen ist keinerlei Verlaufschema festzustellen, weder signifikante Maxima noch Minima.

Bei oberflächlichen Versuchen mit den Constraints und dem Generation Gap veränderte dieses Verhalten sich nicht, allerdings wurden dieselben Probleme mit den Constraints aus den ersten Versuchen wieder aktuell.

Ganz anders verhält sich das Ergebnis, wenn man sich den erzeugten Graphen betrachtet. Der erzeugte Graph entspricht in seiner Ausprägung genau den Erwartungen vor den Untersuchungen:
Die Knoten ordnen sich in einer linearen Liste an, auch hier wird nach kurzer Zeit der optimale Graph annähernd erreicht. Das Ergebnis hat hierbei die folgende Form:

(Graph mit 5 Knoten)

Dieses Ergebnis wird bei 19 von 22 Versuchen erreicht. Allerdings wurde es in allen Fällen in der nächsten Generation zerstört.

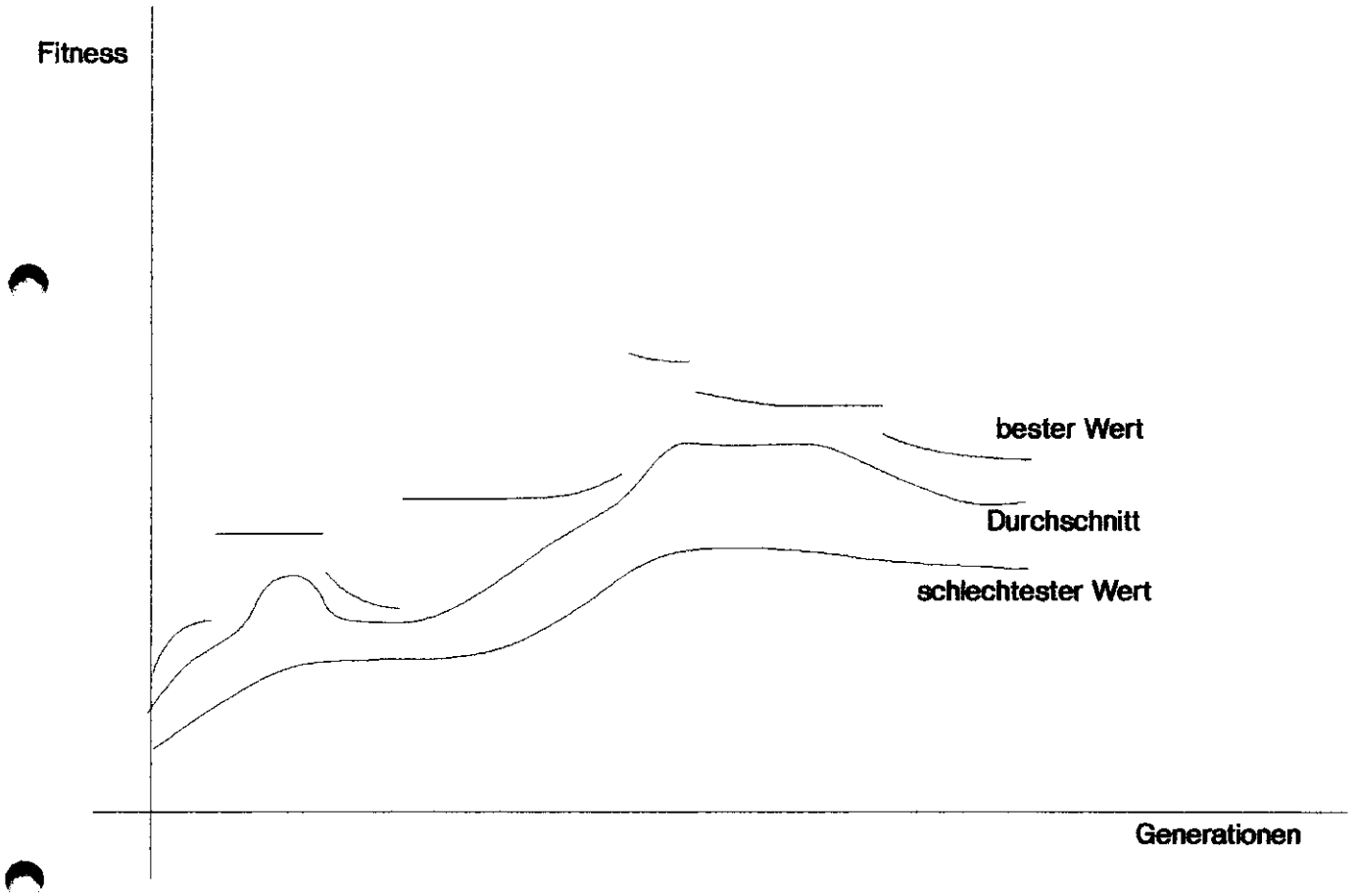


Abbildung 5.6: Kurvenverlauf bei Einsatz der Kostenfunktion

Kapitel 6

Optimierungen der Gesamtfunktion

Außer der Topologischen Optimierung muß die Optimierung der Ordnung betrachtet werden. Diese Untersuchung basiert auf denselben Grundlagen wie die Untersuchungen über die Topologische Optimierung, die einstellbaren Parameter entsprechen denen im vorigen Kapitel.

Auch die Experimente sollen, soweit notwendig, in Art und Reihenfolge denen der Topologischen Optimierung gleichen, ergänzt um jene, die aus der besonderen Natur der veränderten Fitnessfunktion stammen.

Im Anschluß an diese Untersuchungsserie wird die Kombination der Fitnessfunktionen zu der Gesamtfitnessfunktion durchgeführt und diese wird experimentell betrachtet. Das Ergebnis ist dann ein untersuchtes Gesamtsystem, welches in seiner Verhaltensweise bekannt und untersucht ist.

6.1 Untersuchungen der Ordnungsstrukturen

Bei der Untersuchung des Verhaltens der Ordnungsstrukturen bei der Optimierung durch GENETISCHE ALGORITHMEN muß, wie zuvor bei den Topologischen Optimierungen geprüft werden, ob die ausgesuchte Population und die angewendete Knotenzahl repräsentativ für das ganze System sind. Aus diesem Grund wird die Untersuchungsserie von Seite 62 im weiteren wiederholt.

Diese Untersuchungsserie beginnt mit der Bestimmung einer repräsentativen Knotenzahl. Basierend auf den gemachten Erfahrungen in den vorigen Kapiteln wird von einer Population von 20 Individuen ausgegangen. Man stellt im Gegensatz zu den Topologischen Optimierungen fest, daß schon bei sechs Individuen ein repräsentatives Verhalten eintritt. Gezählt werden die Generationen bis zum Eintritt eines Fitnesswertes von mehr als 75 Prozent. Folgende Ergebnisse wurden ermittelt:

Anzahl der Knoten	Generationen
10	77
20	150
30	250
40	340
50	430
60	530
100	1000

Parallel zu den ersten Versuchen und zur besseren Vergleichbarkeit wird auch hier bei den weiteren Versuchen mit einer Größe von 20 Knoten ausgegangen.

Ebenso interessiert die notwendige Population zum Überleben des Spezies. Hier ist, im Gegensatz zu den letzten Versuchen eine Parallelität zu den Versuchen bei der Topologischen Optimierung. Die Population stirbt erst bei einer Größe kleiner 6 Individuen aus. Diese Parallelität erklärt sich aus der Tatsache, daß die eingesetzten Systeme bis auf die Fitnessfunktion gleich sind.

Im weiteren wird also mit der gleichen Knotenanzahl und der selben Population wie bei den Topologischen Optimierungen gearbeitet. Somit kann man die Ergebnisse der Versuche direkt mit den ersten Ergebnissen vergleichen, da alle Voraussetzungen identisch sind.

6.1.1 Ergebnisverlauf bei dem Einsatz von Standartparametern

Der Verlauf der Ergebnisse hat die Form eines stetigen, monoton steigenden geradenartigen Graphen, der sich von einem Punkt nahe dem Koordinatenursprung dem kalkulatorischen Maximum nähert, um sich dann bei einem Fitnesswert von etwa 80 bis 85 Prozent unterhalb von diesem Maximum einzupendeln. Dabei behalten die einzelnen Kurven einen festen Abstand, das Verhalten der Kurve der besten Werte und der Kurven des kalkulatorischen Durchschnitts und der schlechtesten Werte ist identisch, aber zeitversetzt. Der angegebende Fitnesswert wurde in allen Versuchen erreicht, bei acht von zehn Versuchen wird dieser Wert innerhalb von 600 Generationen erreicht.

Auch bei dieser Kurve kann man ein Mehrphasenschema entwickeln, das den Kurvenverlauf charakterisiert. Es können hier zwei Phasen ausgemacht werden. Diese beiden Phasen werden im weiteren durch A und B gekennzeichnet. Die erste Phase A beginnt am Ursprung des Ergebnisgraphen und endet mit dem Übergang von der Steigung in den Einschwingprozess. Dieser Übergang wird als Beginn der Phase B gekennzeichnet. Diese Phase B endet mit dem Erreichen der maximalen Generation.

Bei näherer Betrachtung des Kurvenverlaufes stellt man eine gewisse Übereinstimmung zwischen diesem Phasenschema und dem bei den Topologischen Optimierungen fest. So läßt sich Phase A sicherlich mit der Phase I vergleichen.

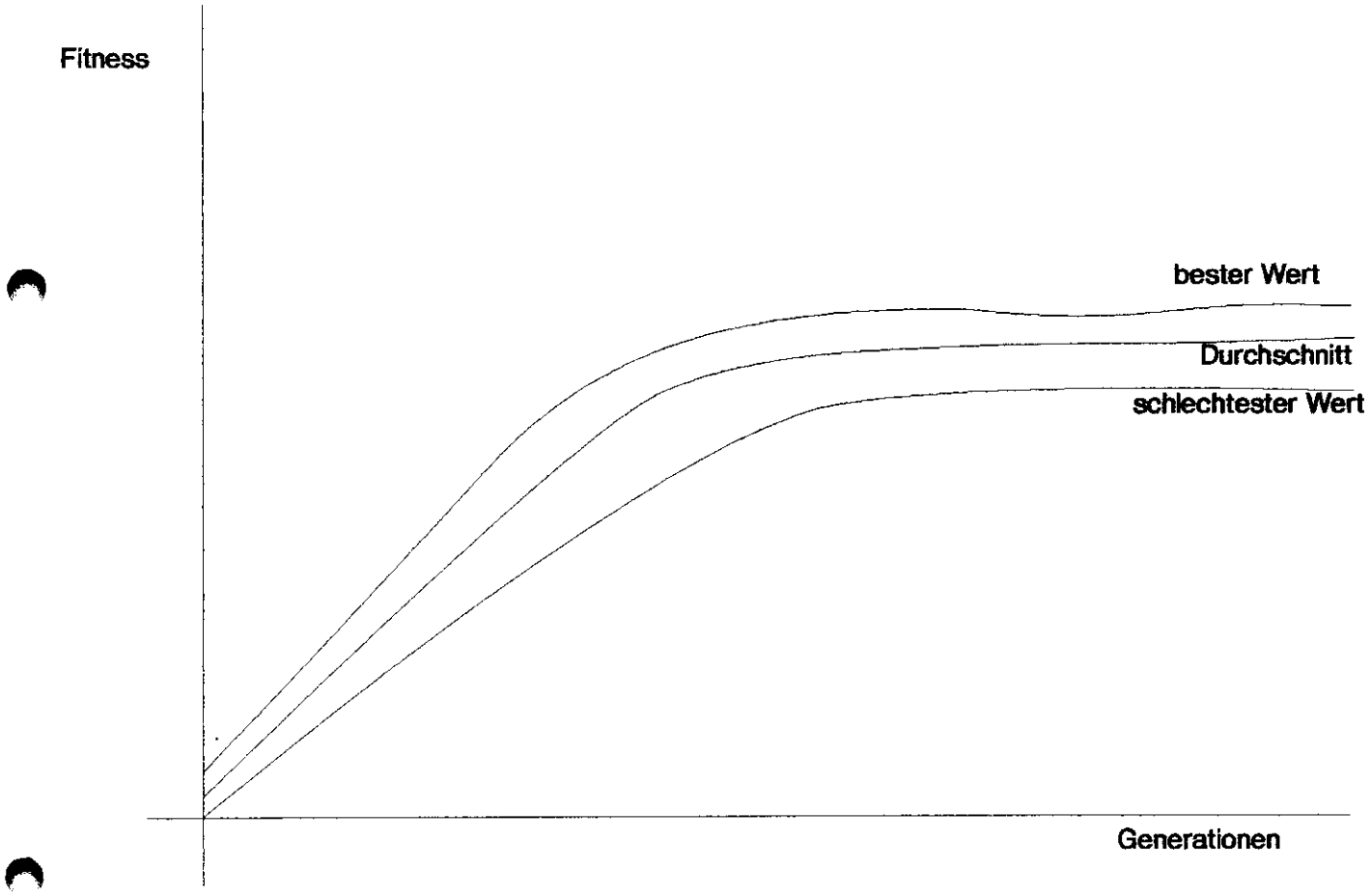


Abbildung 6.1: Kurvenverlauf mit Standartparametern

Sie ist aber deutlich in Richtung der Generationenachse gedehnt. Ebenso läßt sich Phase B mit der Phase III in Verbindung setzen. Hier ist besonders interessant, daß auch in der Phase B ein deutlicher Einschwingprozess zu erkennen ist. Jedoch, und dieses unterscheidet die beiden Phasen, erreicht die Ergebniskurve bei den Optimierungen der Ordnung in der Phase B bessere Werte als in der Phase A. Dieses unterscheidet die beiden Kurvenverläufe, da wie auf Seite 66 beschrieben, nach dem Eintritt der Phase II keinerlei Verbesserung mehr stattfindet.

Vermutlich liegt dieses Verhalten in dem Fehlen einer, der Phase II vergleichbaren Phase. Es hat für den Ablauf der Versuche unterschiedliche Bedeutungen: Zum ersten wird es unmöglich, das Ende der ersten Phase als Abbruchzeitpunkt festzulegen. Das System muß also laufen, bis

- entweder der maximale Wert erreicht wird
- oder eine sehr große Anzahl von Generationen abgelaufen ist.

Sicherlich würde nach dem Ablauf von einer großen Anzahl von Generationen auch ein Maximum erreicht, das nahe dem kalkulatorischen Maximum liegt. Dieses gilt nicht für ersten Versuche im vorigen Kapitel.

6.1.2 Versuche mit verschiedenen Systemparametern

Bei diesen Versuchen werden die, für die Optimierung der Ordnung relevanten Systemparameter getestet. Diese Systemparameter sind

1. die Mutationswahrscheinlichkeit
2. das Generation Gapping

Alle anderen Systemparameter beziehen sich ausschließlich auf die Topologische Optimierung. Die Bedingungen der Tests stimmen mit denen des vorigen Kapitels überein.

Versuche mit der Mutationswahrscheinlichkeit

Entsprechend den Versuchen im vorigen Kapitel mit der Mutationswahrscheinlichkeit werden diese auch hier durchgeführt. Es ergibt sich hierbei das folgende Bild:
Es wurde die gleiche Parametereinstellung wie bei den Versuchen mit der Topologischen Optimierung gewählt, also konstante Knotenanzahl und Population mit 20 Knoten und 20 Individuen. Außerdem wurde auch der Versuchsablauf dem der Topologischen Optimierung nachempfunden. Es wurde mit einer Mutationswahrscheinlichkeit von $p_{mut} = 0,00005$ begonnen und dann in Schritten einer Dezimalpotenz bis zu einer Mutationswahrscheinlichkeit von $p_{mut} = 0,5$ gesteigert. Selbstverständlich gelten alle bekannten Einschränkungen.

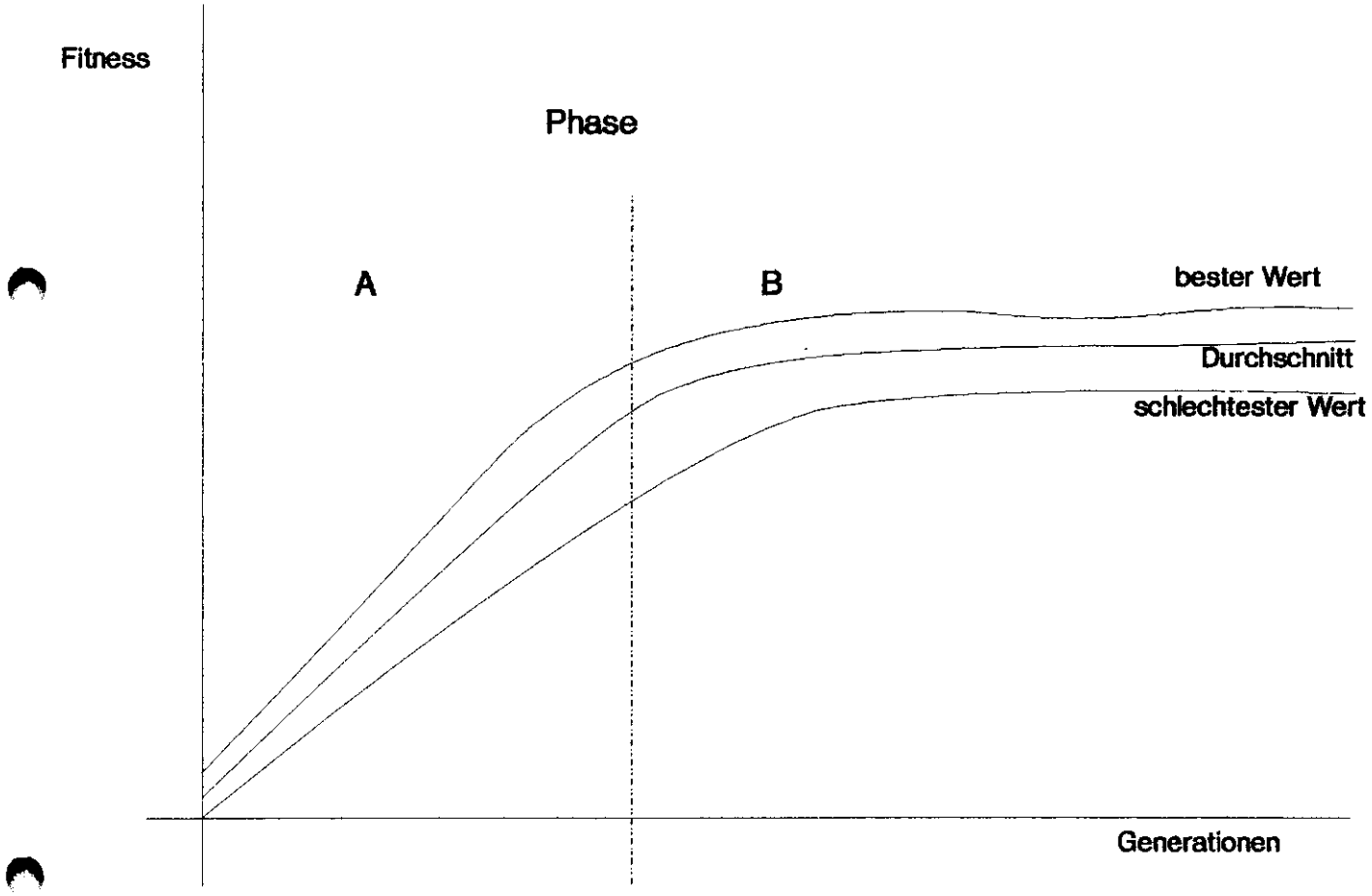


Abbildung 6.2: Kurvenverlauf mit den verschiedenen Phasen

Die verschiedenen Einstellungen erzeugen unterschiedliche Ergebnisse, die sich aber durchaus in zwei Effekte einteilen lassen. Diese Effekte, die im folgenden beschrieben werden, prägen sich mit steigender Mutationswahrscheinlichkeit immer deutlicher aus. Es ist festzustellen, daß sich die Effekte ausschließlich auf die Phase A beschränken. Die Phase B zeigt mit steigender Mutationswahrscheinlichkeit nur eine steigende Auflösung des Werteablaufes. Diese Auflösung betrifft sowohl die Kurve der Optimalen Werte und die Kurve der schlechtesten Werte. Bei dem Einsatz der Mutation streut diese Kurve über dreißig Fitnesswerte. Besonders deutlich wird diese Streuung bei einer Mutationswahrscheinlichkeit von $p_{mut} = 0,5$.

Die beobachtete Streuung gleicht in keiner Weise der Streuung, die auf Seite 73 beschrieben wurde. Dort streut bei einer Mutationswahrscheinlichkeit von $p_{mut} = 0,5$ nur die Kurve der schlechtesten Werte, diese aber über den gesamten, von der Kurve eingeschlossenen Bereich. Interessanter sind jedoch die Veränderungen der Kurven in der Phase A.

Hier ist festzustellen, daß mit steigender Mutationswahrscheinlichkeit die Steigung in der Kurve in dieser Phase A deutlich zunimmt. Dies bedeutet, daß sich die Kurve mit größerer Mutationswahrscheinlichkeit schneller einem Maximum nähert und in ihrer Form der Phase I immer ähnlicher wird. Die Erhöhung der Steigerung wird in der nächsten Graphik gezeigt.

Parallel mit dieser Steigerung der Optimierungsgeschwindigkeit geht aber der zweite Effekte in den Graphen ein. Dieser zweite Effekt ist ein deutlicher Verlust an erreichter Fitness. Nach den gemachten Erfahrungen sind die Steigerung der Kurve und der Verlust der Fitness proportional voneinander abhängig.

Diese Effekte treten bei den Mutationswahrscheinlichkeiten von $p_{mut} = 0,00005$ bis $p_{mut} = 0,05$ auf, bei einer Wahrscheinlichkeit von $p_{mut} = 0,5$ wird die Streuung so groß, daß keinerlei Strukturen mehr erkennbar sind. Hier erscheint nur noch eine gleichförmige, gestreute Menge von Werten.

Bei der hohen Wahrscheinlichkeit kommt zusätzlich hinzu, daß hier willkürlich Kanten erzeugt werden. Auf diese Weise werden, ohne daß es beabsichtigt wurde, neue Topologische Strukturen erzeugt, die mit der einfachen Optimierung nichts mehr zu tun haben.

Auf Grund dieser Erfahrungen ist es sinnvoll, die Mutationswahrscheinlichkeit so gering wie möglich anzusetzen. Also wäre es nur ratsam, mit einer Mutationswahrscheinlichkeit von $p_{mut} = 0,00005$ bzw. $p_{mut} = 0,000005$ zu arbeiten.

Einsatz des Generation Gapping

Beim Einsatz des Generation Gapping verändern sich einige Teile der Ergebniskurven. Dabei gilt, daß wie bei den Versuchen auf Seite 73 das Generation Gapping keinen Einfluß auf die Kurven der durchschnittlichen Werte und der schlechtesten Werte hat. Anders ist die Reaktion der Kurve der besten Werte. Hier findet eine deutliche Veränderung des Kurvenverlaufes statt. Diese Veränderung findet in der Phase B des Ablaufes statt. In der Phase A zeigt

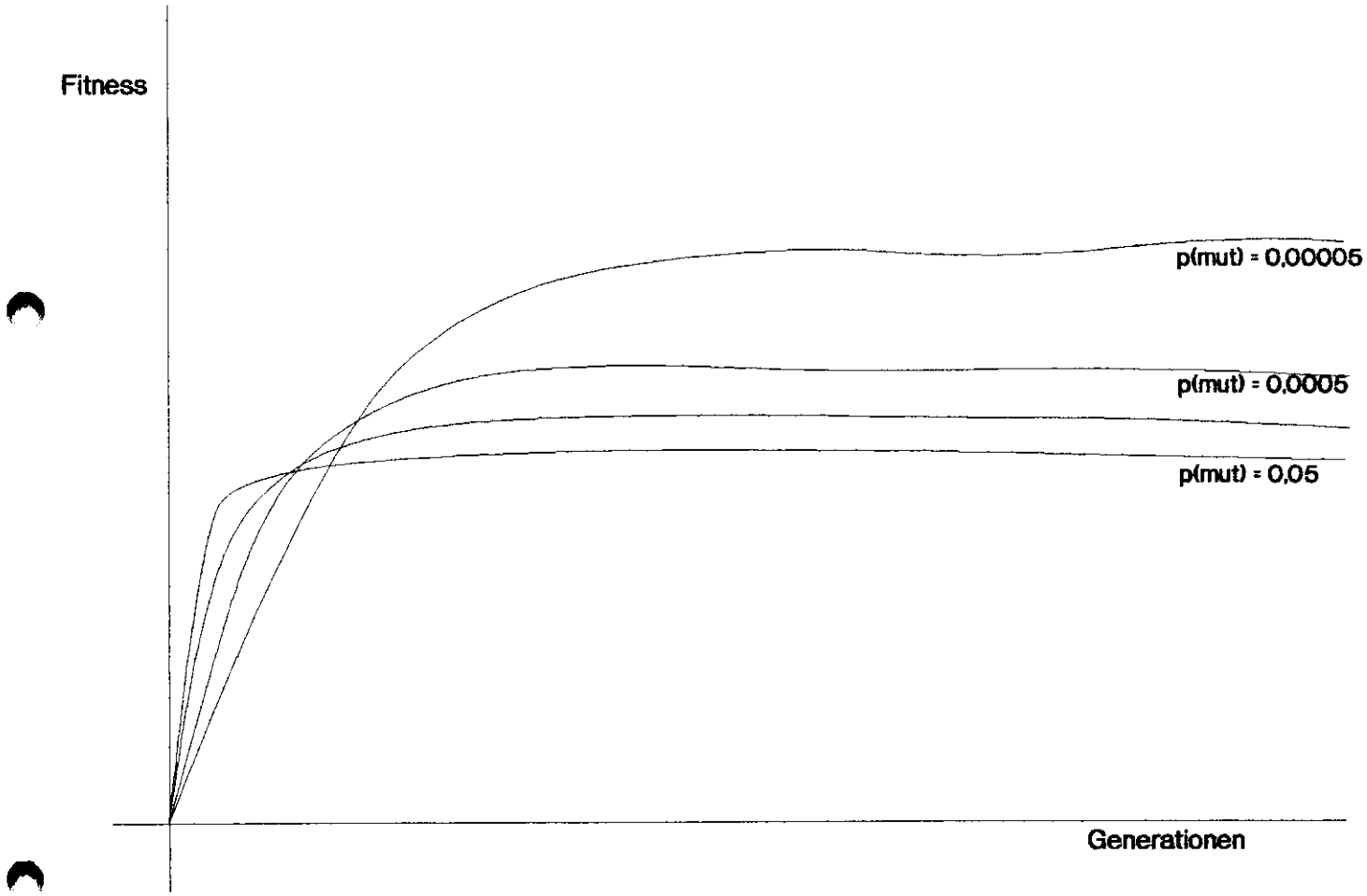


Abbildung 6.3: Kurvenverlauf bei unterschiedlicher Mutationswahrscheinlichkeit

das Generation Gapping keinerlei Wirkung, da hier ein monoton steigender Ergebnisverlauf bei jeder Generation ein besseres Individuum hervorbringt. Dadurch ändert sich das Gap in jeder Generation. In der Phase B jedoch zeigt sich ein anderes Bild. Der schwingungsartige Kurvenverlauf ist auch hier nicht mehr feststellbar. Stattdessen ist eine monoton steigende Treppenfunktion mit deutlichen Tendenzen zum Optimum festzustellen. Bei den durchgeführten Versuchen stellte sich keine Beschleunigung des Optimierungsverhaltens heraus.

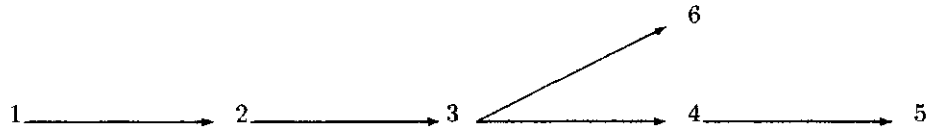
Der subjektive Eindruck jedoch ist der einer Verbesserung des Ablaufes der Kurve der Optimalen Werte. Deshalb wird das Generation Gapping im weiteren für den Einsatz bei der Kombination der Fitnessfunktionen für Topologische Optimierung und der Optimierung der Ordnung vorgesehen.

6.1.3 Darstellung der erzeugten Graphen

Auch im Anschluß zu diesen Untersuchungen sollen die erzeugten Graphen dargestellt werden. Die Topologie dieser Graphen ist hierbei ohne Bedeutung und soll bei der Betrachtung des Ergebnisses nicht berücksichtigt werden.

Im weiteren werden zwei typische Ausprägungen der Graphen dargestellt. Alle Graphen sind optimale Ordnungen und werden als solche von der Fitnessfunktion erkannt. Beide Graphen sind bei den Versuchen aufgetreten.

Der erste erzeugte Graph ist eine der linearen Liste ähnliche Struktur mit dem folgenden Aussehen:



(Graph mit 6 Knoten)

Der zweite Graph ähnelt in der Form eher einem Baum mit entsprechenden Querverbindungen. Dieser Graph wird im folgenden dargestellt:

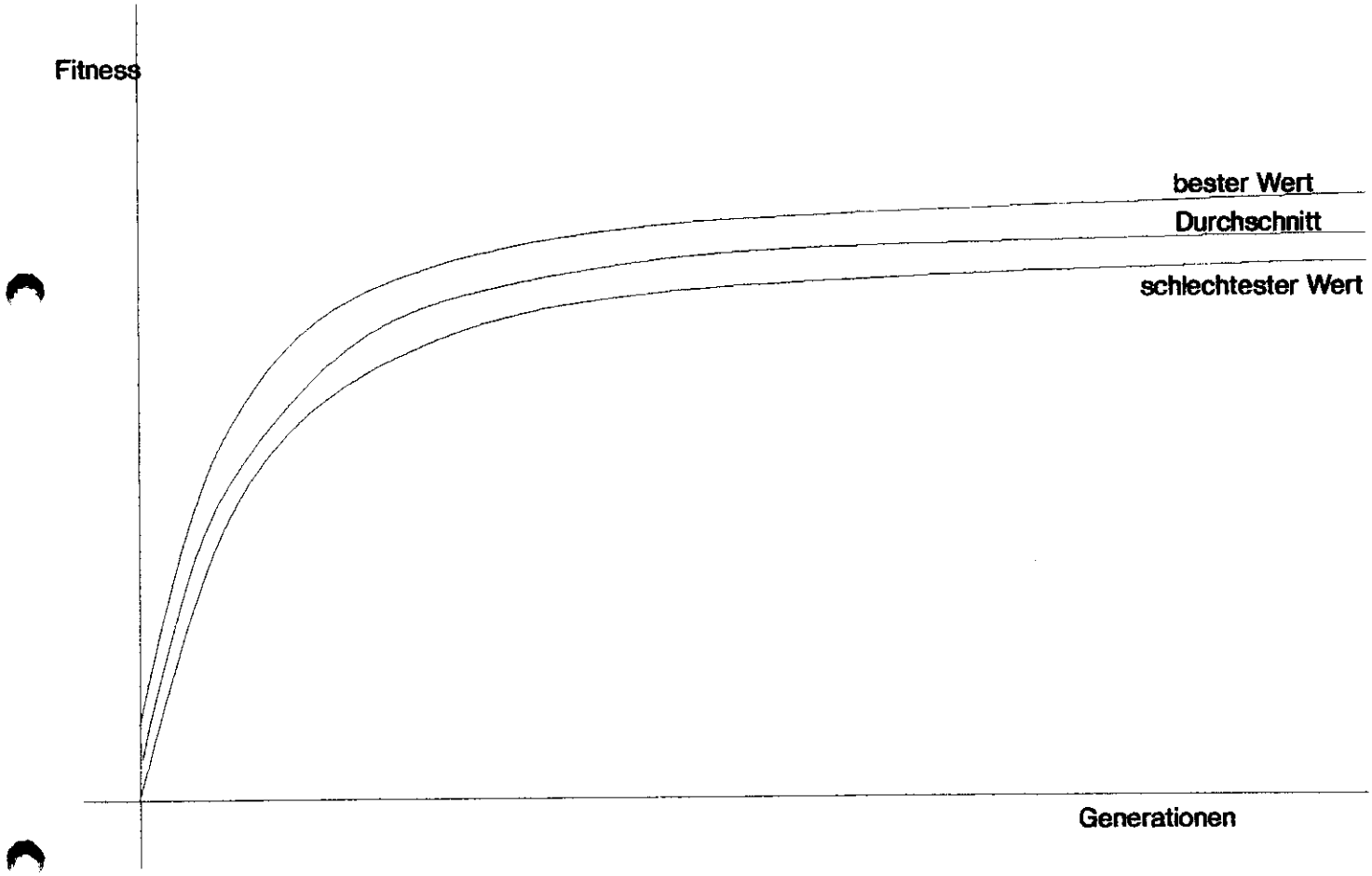
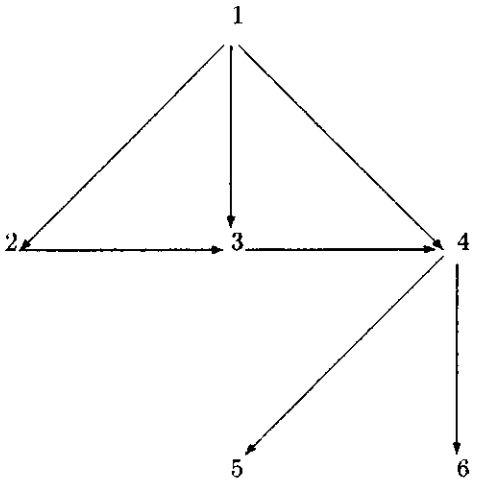


Abbildung 6.4: Das Generation Gapping



(Graph mit 6 Knoten)

Es war deutlich zu erkennen, daß signifikant unterschiedliche Graphen erzeugt wurden, deren Ausprägung durch anfänglich vorhandene Graphen beeinflußt wurden. Allerdings mußte festgestellt werden, daß die Topologie in keinem Fall erhalten wird, sogar eine vollständige Deformierung ist möglich, allerdings trat nur in einem von zwanzig Versuchen dieser Effekt ein.

Damit entfällt die Möglichkeit, diese Funktion im Anschluß an die Topologische Optimierung zu nutzen, um auf diese Weise eine Selbstorganisierende Datenstruktur zu erzeugen.
So wird im weiteren eine andere Form der Kopplung der Fitnessfunktion untersucht.

6.2 Kopplung der Fitnessfunktion

Bei dieser Kopplung der Fitnessfunktionen gehen beide Teilfunktionen in die Gesamtfitnessfunktion ein. Entsprechendes wurde auf Seite 54 beschrieben.

Zur Beurteilung des Verhaltens werden hierbei die Experimente zu den einzelnen Fitnessfunktionen wiederholt und die Ergebnisse werden verglichen. Auf Grund der Tatsache, das Populationsgröße und Knotenanzahl bei beiden Teilfunktionen gleichgehalten werden konnten, wird auch mit den beiden bekannten Werten im weiteren gearbeitet.

Allerdings werden die Ergebnisse der Teilfunktionen skaliert, bevor sie addiert werden können. Damit wird die Wichtung der Teilfunktionen nur noch von dem entsprechenden Parameter abhängig gemacht. Dieser Wichtungsparameter, der auf Seite 54 beschrieben wurde, soll zusätzlich zu den anderen Experimenten untersucht werden. Hierzu werden die Wiederholungen der Experimente bei

den verschiedenen Einstellungen des Wichtungsparameters durchgeführt. Die Einstellungen des Parameters für die Experimente sind das Verhältnis der Fitnessfunktionen untereinander. Für die Experimente werden die folgenden Wichtungen gewählt: $1:9$, $2:8$, ... bis $9:1$. Entsprechend wird der Wichtungsparameter mit den Werten $0,1$; $0,2$; $0,3$; ... bis $0,9$ besetzt. Somit sollten nicht nur die Kurvenverläufe mit den Einzelversuchen verglichen werden, sondern auch innerhalb der verschiedenen Wichtungen.

Damit kann das Verhalten aller Komponenten im Ablauf dargestellt und beurteilt werden. In diesem Zusammenhang kann man aber auch die durchgeführten Versuche als eine besondere Einstellung mit dem Wichtungsparameter betrachten. Die vorne durchgeführte Topologische Optimierung stellt sich als eine Einstellung im Verhältnis $1:0$, die Optimierung der Ordnung als eine Einstellung im Verhältnis $0:1$, also eine hundertprozentige Wichtung für jeweils eine der Seiten dar.

Die Vermutung bei diesen Versuchen liegt in der Hoffnung, bei einem bestimmten, experimentell zu ermittelnden Wert eine paritätische Nutzung beider Fitnessfunktionen zu erreichen, so daß ein Graph erzeugt wird, der in beiderlei Hinsicht optimal erscheint.

6.2.1 Variation der Wichtungen

In der ersten Versuchsserie werden in dem System lediglich die Wichtungen verändert. Wie erwartet verändern sich hierbei die Kurvenverläufe. Gesucht werden hier Ähnlichkeiten oder signifikante Abweichungen im Verlauf der Kurven. Diese Abweichungen sollten untersucht und beurteilt werden.

Da die größten Unterschiede bei den Kurven der Optimalen Werte auftreten, werden diese besonders betrachtet. Wie deutlich zu erkennen ist, liegt ein gleitender Übergang vom Graphen der Optimerierung der Ordnung bis zum Graphen der Topologischen Optimierung vor. Dieser gleitende Übergang führt von einem Graphen mit deutlichem Steigungsverhalten und erkennbaren Phasenschema der Optimierung der Ordnung über zwitterartige Graphen bis zu Graphen mit dem dreistufigen Phasenverlauf der Optimierung der Topologie.

Man kann den Kurvenverlauf in drei Abschnitte unterteilen. Der erste Abschnitt beginnt bei einer Verteilung von $1:9$ und endet mit der Verteilung $3:7$. Dieser Abschnitt wird gekennzeichnet durch das deutliche Erscheinen der Charakteristik der Topologischen Optimierung. Der zweite Abschnitt, der mit einer Verteilung von $4:6$ beginnt und mit einer Verteilung von $6:4$ endet, kann mit dem Namen *Mischungsgraphen* bezeichnet werden, da hier weder die eine noch die andere Charakteristik deutlich erkennbar ist. Der dritte Abschnitt, beginnend bei einer Verteilung von $7:3$ und bis zur Verteilung $9:1$ laufend, ist der Abschnitt der Optimierung der Ordnung.

Die weiteren Untersuchungen betreffen die Kurvenabläufe und die erreichten Werte. Zuerst werden die Charakteristika der Kurven in den einzelnen Abschnitten untersucht. Im Abschluß wird dann der Werteverlauf im einzelnen

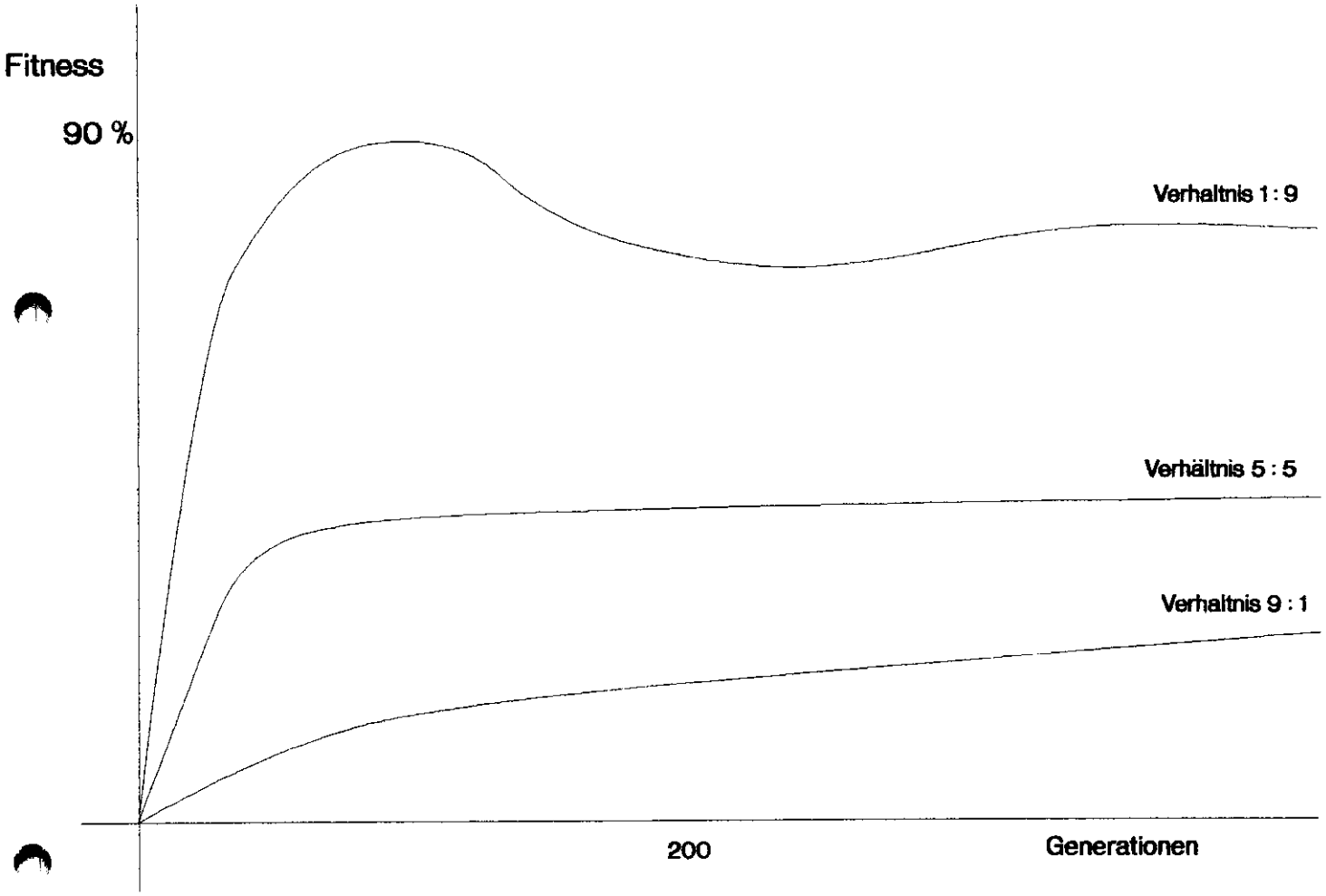


Abbildung 6.5: Darstellung des Verhaltens der optimalen Werte

untersucht.

6.2.2 Darstellung der Kurvencharakteristika

Besonders interessant sind hierbei die Graphen der gemischten Fitnessfunktionen. Diese Graphen treten bei einem Mischungsverhältnis von 4 : 6 bis 6 : 4 auf. Bei den übrigen Graphen überwiegt eine der Fitnessfunktionen, was sich deutlich in dem Ergebnisgraphen niederschlägt.

Betrachtung der Mischungsgraphen

An dieser Stelle soll nun das Verhalten dieser besonderen Mischungsgraphen dargestellt werden. Erstaunlich bei diesen Graphen ist die ausgeprägte Ineffizenz, sowohl in dem Erreichen eines hohen Fitnessniveaus als auch im Steigungsverhalten im Ablauf des Systems. Diese spezielle Ineffizenz äußert sich insbesondere darin, daß entweder ein schnelles Aussterben der gesamten Population oder aber häufiger ein Einpendeln auf einen relativ niedrigen Niveau stattfindet. Dieses niedrige Niveau liegt hierbei zwischen 40 und 60 Prozent erreichbarer Gesamtfitness und wird innerhalb von 20 bis 40 Generationen erreicht. Dabei beträgt der gesamte Fitnessgewinn maximal 30 bis 60 Punkte.

Eine Abänderung bei diesem sehr konstanten Verhalten ist nur selten bei dem Einhalten der erreichten Obergrenze möglich, hier kann es durch die Mutation durchaus auch zu einzelnen punktuellen Verbesserungen kommen, die aber nie mehr als zwei Fitnesspunkte betrugen.

Bei der Analyse dieses Verhaltens lag der Verdacht nahe, daß die Ursache dieses Verhaltens in einer zu kleinen Population lag. Um dieses zu verifizieren, wurden Versuche mit erhöhten Populationsgrößen durchgeführt. Diese Versuche begannen bei einer Population von 20 Individuen und endeten bei einer Population von 50 Individuen. Hierbei wurde die Größe der Population in Schritten zu je 6 Individuen vergrößert.

Bei der Durchführung der Versuche stellt sich heraus, daß die Anzahl der Individuen auf den prinzipiellen Ablauf keinerlei Einfluß zu haben scheint. Es wurde lediglich die Anzahl der Generationen bis zum entgültigen Erreichen des konstanten Zustandes verlängert. Innerhalb dieser Verlängerung stellt sich der Graph als eine wellenförmige Kurve dar. Allerdings wird innerhalb dieser Kurve der später erreichte maximale Wert nicht überschritten.

Es ist also für den Ablauf nicht von Relevanz, wieviele Individuen in der Population vorhanden sind. Damit ist zu vermuten, daß die Anzahl der Generationen nicht für das widrige Verhalten des Graphen verantwortlich sind.

Das Verhalten des Systems gleicht in seiner Art dem Verhalten bei dem Einsatz der Constraints.

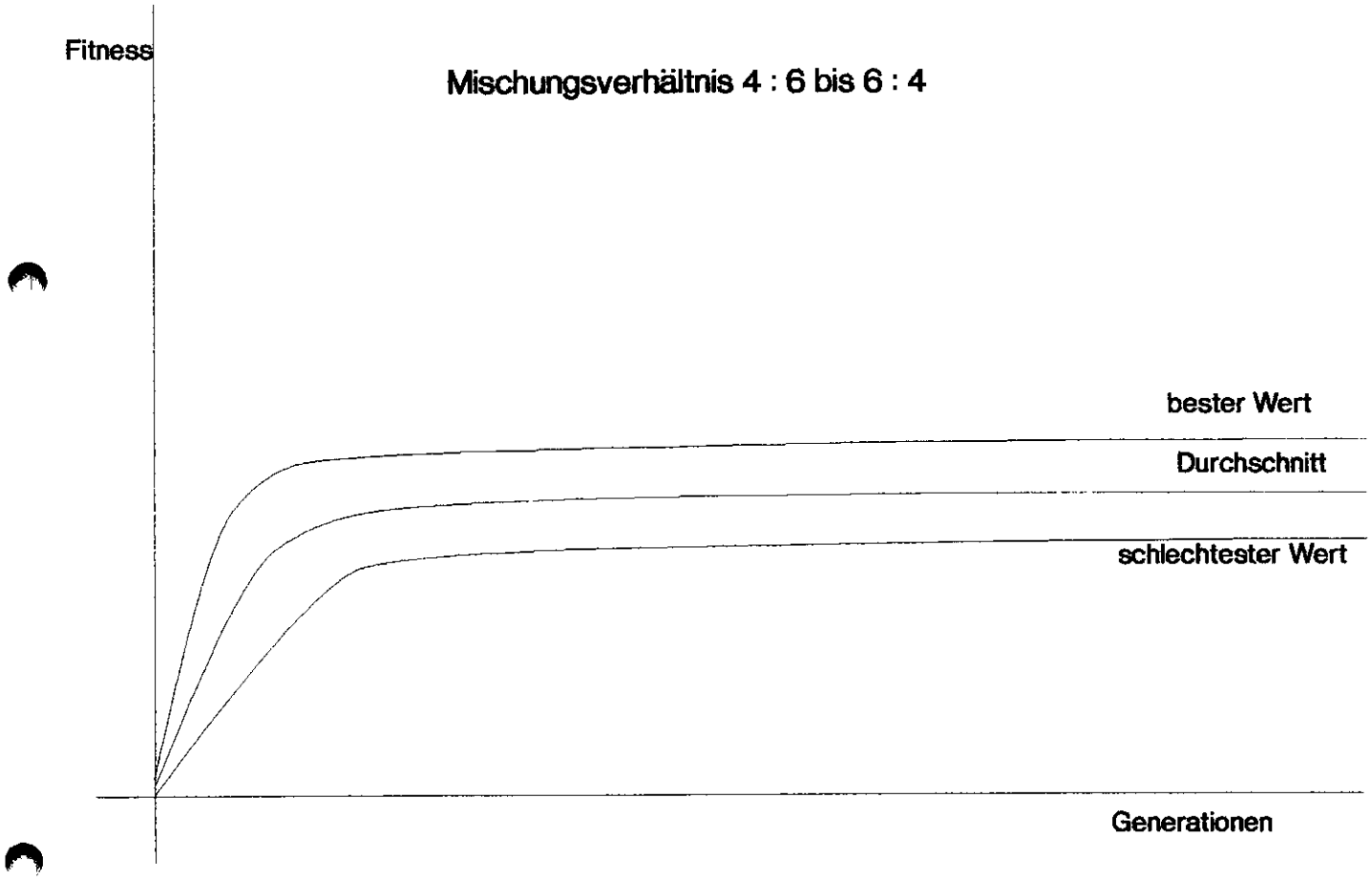


Abbildung 6.6: Verhalten des Mischgraphen in der zweiten Phase

Weitere Versuche zur Verbesserung der Fitness

Ein weiterer Versuch, dieses gleichförmige Verhalten zu verändern liegt in der Modifikation der Mutationswahrscheinlichkeit. Als weitere Versuchsserie wird also dieser Parameter wie in den vorigen Serien modifiziert, um so dieses negative Verhalten zu ändern.

Es wurde also mit einer festen Verteilung der Fitness von 4 : 6 eine Versuchsserie durchgeführt. Jedoch zeigte sich bei der Durchführung der Versuchsserie deutlich, daß die Mutationswahrscheinlichkeit auf die Kurve der optimalen Werte keinen nennenswerten Einfluß hat. Es gibt im Vergleich zu der obrigen Versuchsserie keine erhöhte Häufigkeit von Verbesserungen in dieser Kurve. Einen Einfluß hat diese erhöhte Mutationswahrscheinlichkeit auf die Kurve der schlechtesten Werte. Hier tritt der, schon auf Seite 73 und auf Seite 84 beschriebene Effekt der Auflösung der Kurve auf.

Diese Auflösung ist deutlich stärker, je höher die Mutationswahrscheinlichkeit ist. Dieses bedeutet, daß die Mutation als Förderungsmittel in diesem mittleren Bereich der Mischung der Fitnessfunktionen nicht oder nur sehr mangelhaft geeignet ist.

Es ist sehr wahrscheinlich, daß die gesuchte Einstellung zwischen den beiden Fitnessfunktionen nicht in dem mittleren Bereich liegt. Es muß also versucht werden, die optimale Einstellung im ersten oder im letzten Teil zu finden.

Verhalten der Kurve in der ersten Phase

Bei der Untersuchung dieser Phase kann man deutliche Parallelen zu den Ergebnissen bei den Untersuchungen der Topologischen Optimierung finden. Auch hier kann man die drei Phasen im Kurvenverlauf bestimmen und darstellen. Unterschiede bestehen jedoch in der Phase III des Ablaufs. Bei der Untersuchung der Kurven der Gesamtfunktion stellt man fest, daß diese Phase III nicht mehr wie vorne beschrieben werden kann. Während bei der Topologischen Optimierung die Kurve in der Phase III einen waagerechten Verlauf hatte, verläuft sie hier nun mit einer feststellbaren Steigung. Somit werden hier auch in der Phase III bessere Werte als in der Phase I erzeugt, vergleichbar mit der Optimierung der Ordnung. Insgesamt drängt sich der Eindruck auf, daß hier eine Vermengung der einzelnen Optimierungsmethoden stattfand, genau wie es bei dieser Kopplung beabsichtigt war.

Diese Phasen I und II stammen ihrer Form nach aus dem Bereich der Topologischen Optimierung, die Phase III jedoch entstammt in ihrer Form der Optimierung der Ordnung und wird dort als Phase B bezeichnet.

Dieses Verhalten und die Tatsache, daß die Population auch über eine große Anzahl von Generationen stabil erscheint, prädistiniert eine Einstellung aus diesem Bereich als mögliches Ziel der Untersuchungen.

Jedoch treten bei dem Versuch, gute Werte zu erreichen, Probleme auf, wie im weiteren beschrieben wird.

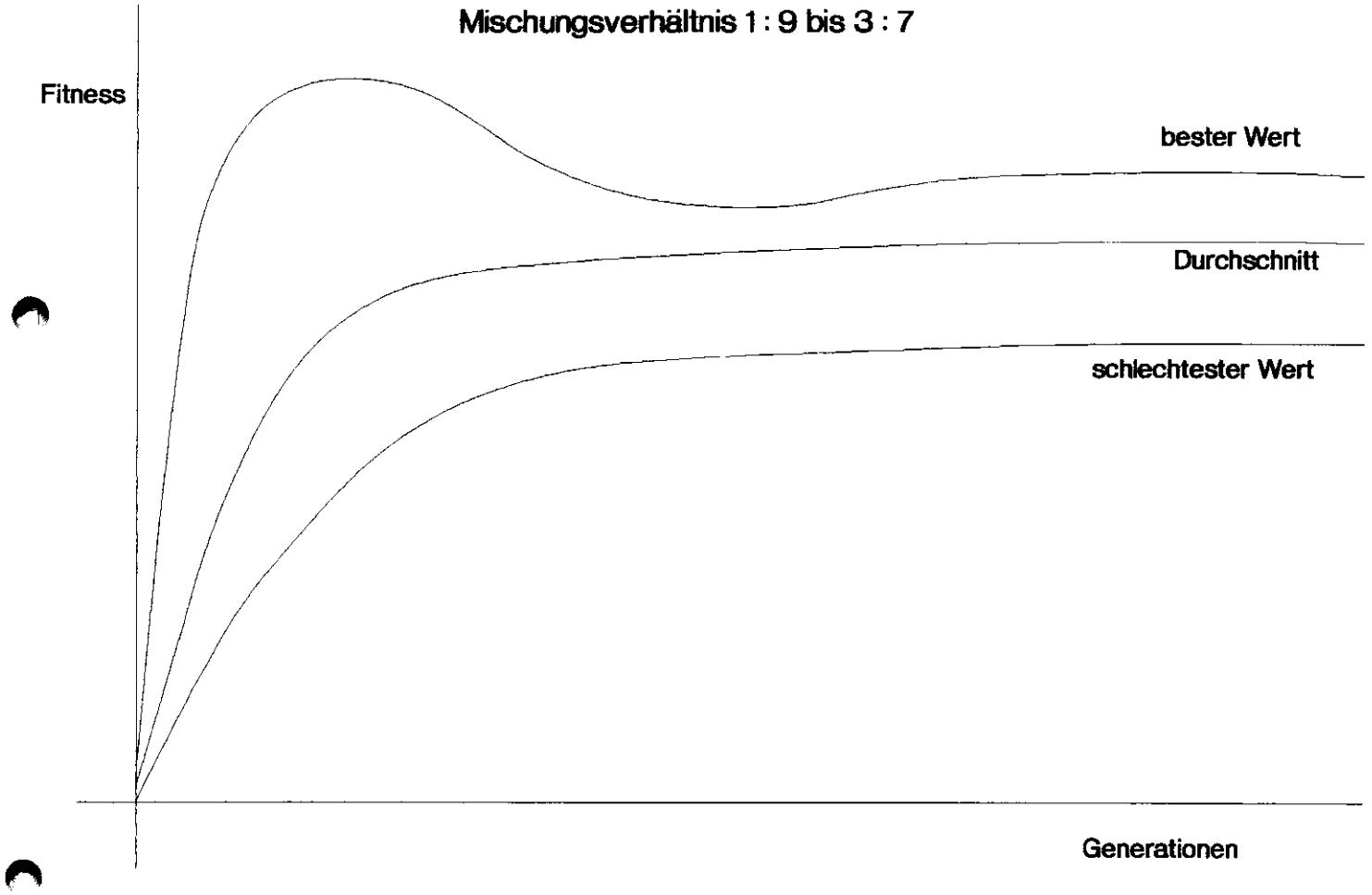


Abbildung 6.7: Kurvenverlauf in der ersten Phase

Verhalten der Kurve in der dritten Phase

In dieser dritten Phase stellt sich die Kurve wieder anders da. Sie gleicht in ihrer Form dem Ergebnis nicht der Optimierung der Ordnung, denn es gibt bestimmte, deutlich erkennbare Abweichungen. Diese Abweichungen wirken sich aber nicht auf das Gesamtergebnis aus.

Die erste Abweichung betrifft den Ablauf der Phase A'. In dieser Phase hat man im Vergleich zu der Optimierung der Ordnung einen deutlich höheren Anstieg festzustellen, außerdem verändert sich die Form des Graphen zu einem leichten, rechtsgerichteten Bogen. Die Phase A' erreicht jedoch nicht den Fitnesswert der Optimierung der Ordnung, sondern erreicht höchstens einen Wert um die 40 Prozent Fitness. In ihrer Form ähnelt diese neue Phase A' einer Mischung aus der Phase A und der Phase I.

Ein weiterer Unterschied liegt in der Form der Phase B'. Hier ist, im Gegensatz zu der vorigen Phase B eine leichte Steigung festzustellen. Es scheint, daß die Phase B' mehr der Phase A als der Phase B ähnelt.

Insgesamt ist aber der Kurvenverlauf von einer sehr kleinen, oft nur minimal erkennbaren Steigung geprägt, so daß von einem Optimierungsverhalten im engeren Sinne nicht die Rede sein kann. Es liegt die Vermutung nahe, daß die Optimierung nur durch den Einsatz des Generation Gapping erzeugt wurde. Dieser Mechanismus ist seit den Versuchen zu der Optimierung der Ordnung ständig aktiv.

6.2.3 Beschreibung des Werteverlaufes

Bei dem Verlauf der Werte stellt man fest, daß beginnend bei der ersten Phase die erreichte Fitness von einem Wert um die achzig Prozent kontinuierlich abnimmt und schließlich einen Wert um die vierzig Prozent erreicht. Dieser Abfall der Fitness wird in der folgenden Graphik dargestellt:

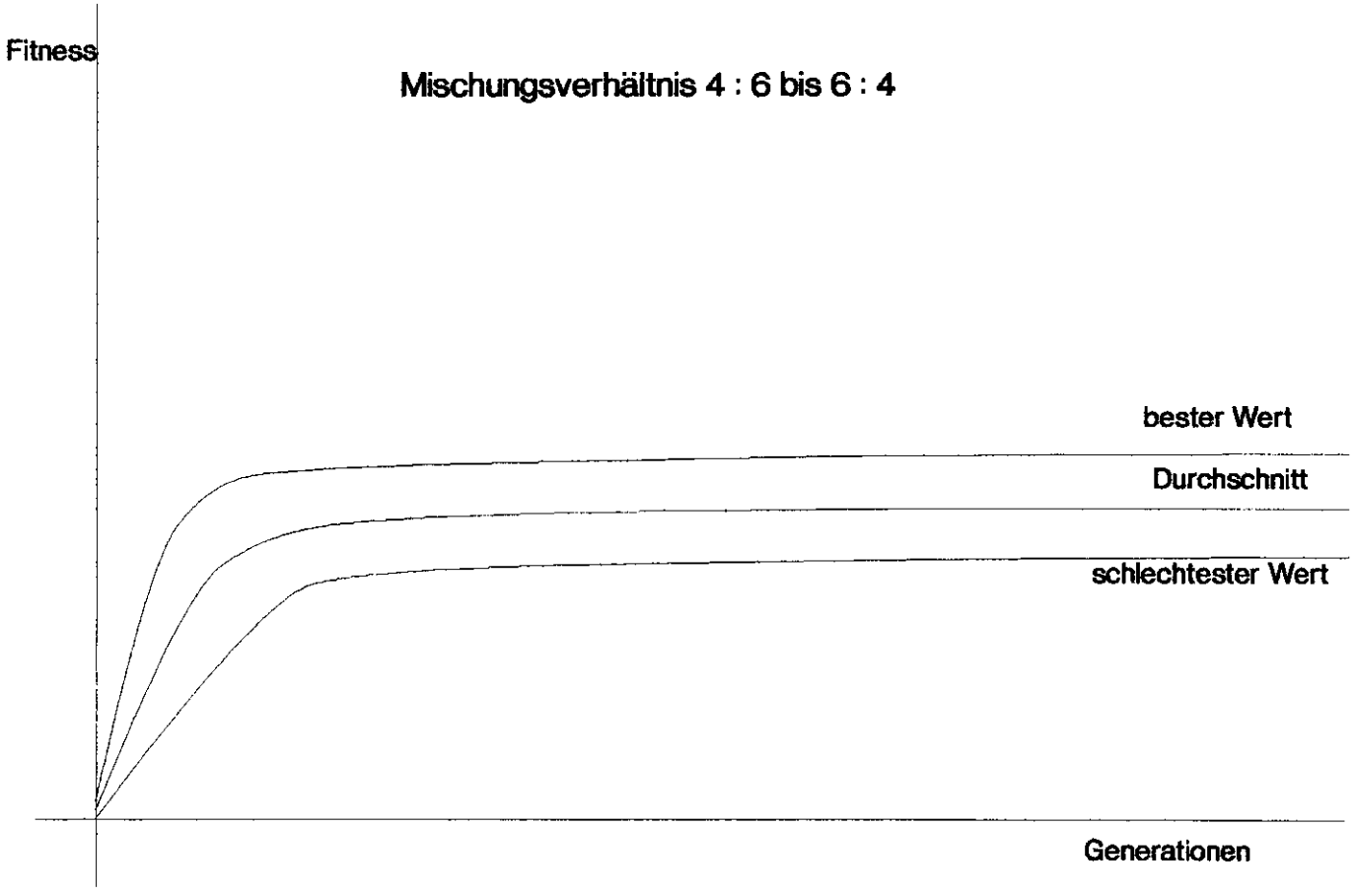
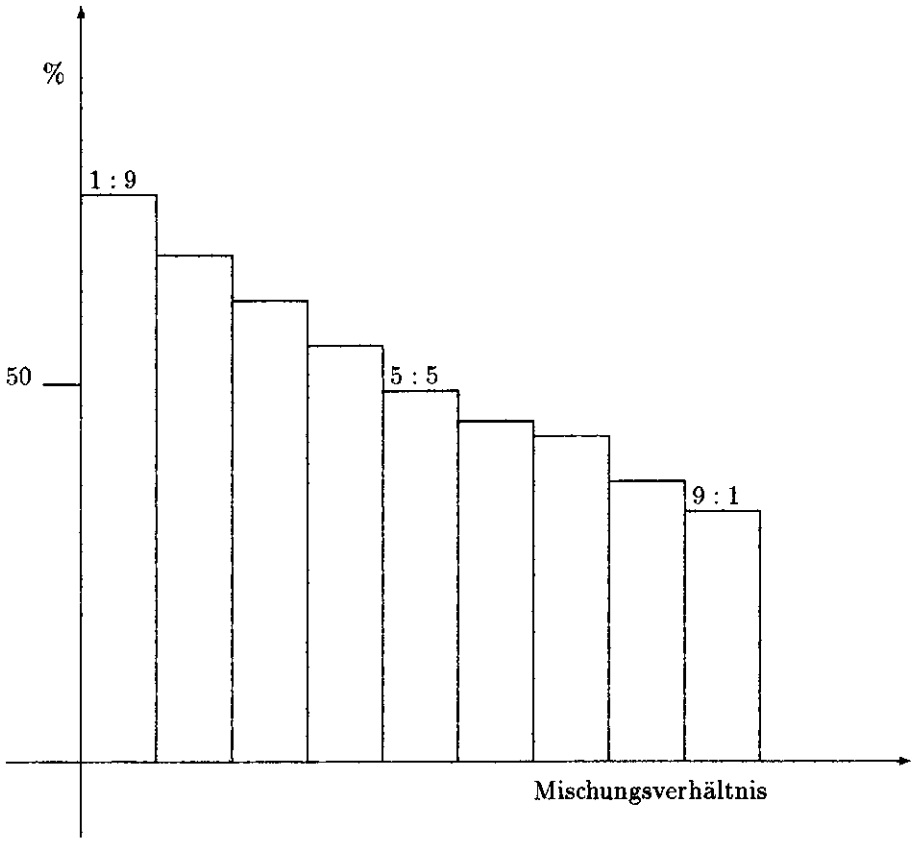


Abbildung 6.8: Kurvenverlauf in der dritten Phase



(Stufendiagramm)

Es scheint, daß der Verlust an Fitness in einem Zusammenhang mit der Parametereinstellung steht. Es wurde versucht, diesen Effekt durch Modifikation anderer Systemeinstellungen abzufangen. Bei einer näheren Betrachtung der vorhandenen Systemeinstellungen stellt man jedoch fest, daß die vorhandenen Mittel beschränkt sind:

- Der Einsatz von Constraints und die Korrekturfunktion erwiesen sich schon bei der Topologischen Optimierung als ungeeignet, bzw. wurden aus methodischen Gründen abgelehnt.
- Ebenso hat der Einsatz der Baumförderung nicht den erwünschten Erfolg gebracht.
- Die Größe der Population hat, wie schon bei speziellen Untersuchungen erkennbar, nichts mit der erreichten Fitness zu tun.

Auch die obligatorischen Versuche mit der Mutationswahrscheinlichkeit erbrachten keinerlei Veränderung in dem Verhalten.

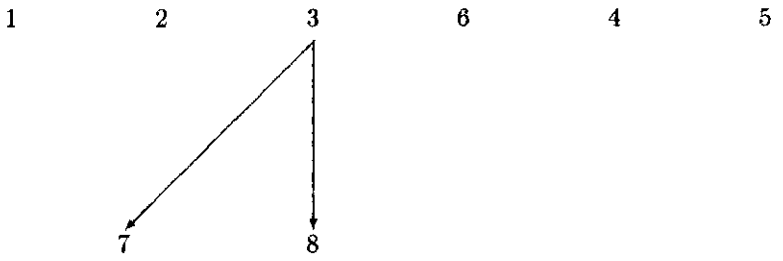
Dieses Verhalten, insbesondere der Abfall in der erreichten Fitness bei der Modifikation des Mischungsparameters zu einer Stärkung der Optimierung der Ordnung und der weiteren Versuche sind Indikatoren für die Richtigkeit der Vermutung, daß die gesuchte Einstellung der Parameter im ersten Bereich der Mischungen zu finden ist.

6.2.4 Darstellung der erzeugten Graphen

In diesem letzten Teil der Darstellung der Experimente sollen die, durch die Gesamtfunktion erzeugten Graphen exemplarisch vorgestellt werden. Entsprechende Darstellungen aus der Experimentalumgebung sind im Anhang enthalten.

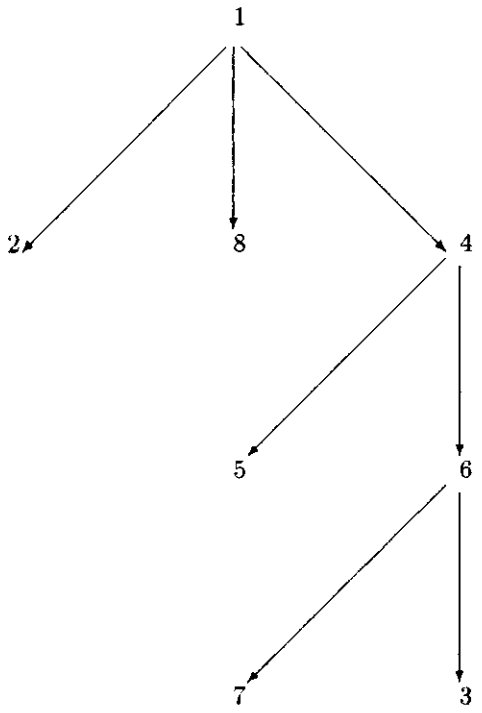
Besonders auffällig zeigt sich das Verhalten der Graphen im Bereich mit besonders guten Fitnesswerten. Hier stellt man bei einer genauen Kontrolle fest, daß das Erreichen von besonders guten Fitnesswerten einhergeht mit einem deutlichen Verlust an Kanten in den Graphen. Die besten Werte erreichten hierbei Graphen mit überdurchschnittlich vielen Wurzeln und wenigen Blättern.

Die erzeugten Graphen haben prinzipell das folgende Aussehen:



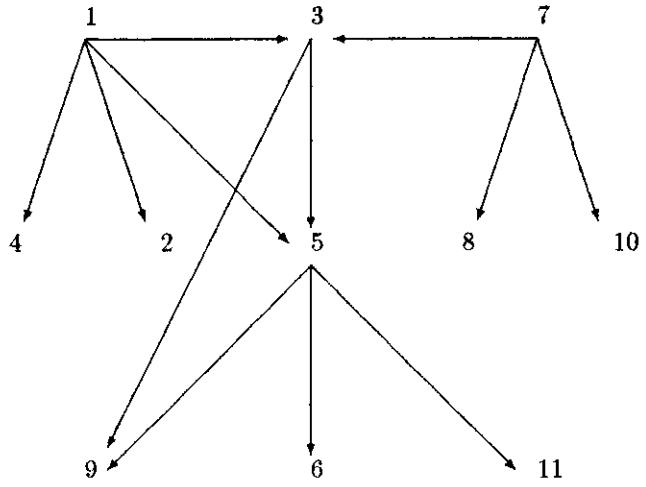
(Graph mit 8 Knoten)

Durch die spezielle Definition der Fitnessfunktion erreicht dieser Graph jedoch nicht eine Fitness über 90 Prozent. Ein anderer Graph wird bei der Mischung der Fitnessfunktionen erzeugt. Diese Graphen haben ein uneinheitliches Aussehen. Exemplarisch sei der folgende Graph dargestellt:



(Graph mit 8 Knoten)

Als letztes sei noch ein Graph mit schlechter Fitness dargestellt, wie er aus der dritten Phase der Mischung hervorgeht. Diese Darstellung ist lediglich als Information über das Verhalten bei dieser Parametereinstellung gedacht, mit dem deutlichen Hinweis, daß der folgende Graph eine Fitness von maximal 45 Prozent erreicht hat:



(Graph mit 11 Knoten)

Kapitel 7

Zusammenfassung

Wenn man die Ergebnisse der Experimente würdigen und diskutieren will, so muß man das, was man erreicht hat mit dem vergleichen, was erreicht werden oder was Ziel der Untersuchung sein sollte.

Die hier zu behandelnde Fragestellung unterteilte sich in drei Einzelkomplexe,

1. Optimierung der Ordnung,
2. Optimierung der Topologie, und
3. die Mischung der einzelnen Probleme zu einem gesamten Komplex.

Aus Gründen der Übersichtlichkeit werden diese drei Komplexe gesondert betrachtet und bewertet. Bei diesen Einzelbetrachtungen interessieren die folgenden Fragestellungen besonders:

- Warum läuft die Optimierung so ab, wie sie hier beobachtet wurde ?
- Wie kann man das beobachtete Ergebnis interpretieren ?
- Was kann man mit diesem Ergebnis anfangen ?

Bei der ersten und bei der letzten Frage begibt man sich leider sehr schnell in das Reich der Spekulation. Gerade bei der Fragestellung nach dem Warum ist die Gefahr besonders hoch, den Boden der Tatsachen zu verlassen.

Diese Tatsachen sind die Ergebnisse der Experimente, die zu diesem Zweck durchgeführt wurden. Abschließend ist noch ein Vergleich mit anderen Ergebnissen aus der Literatur von Interesse. Hier werden die Titel von GOLDBERG[Gol89] und DAVIS[Dav91] ausschlaggebend sein.

Die folgende Betrachtung wird in chronologischer Reihenfolge durchgeführt.

7.1 Betrachtung der Topologischen Optimierung

Bei der Topologischen Optimierung liegt ein deutlicher Funktionsablauf vor. Dieser Ablauf ist typisch und taucht bei allen Variationen der Systemparametereinstellung auf. Er ist in Form und Ausprägung in Kapitel 5 beschrieben.

Wichtig für die Interpretation sind die verschiedenen Phasen. Phase I ist eine Phase mit überdurchschnittlich gutem Optimierungsverhalten. Danach pendelt sich das System bei einem sehr guten Wert über 90 Prozent ein. Dieser Einschwingprozess ist als durchaus natürlicher Prozess zu verstehen, ebenso wie die Tatsache, daß nie eine 100 prozentige Fitness erreicht wurde.

Dieses Ergebnis, wie auch das bei der Optimierung der Ordnung entspricht in seiner Form und Ausprägung den, in der Literatur veröffentlichten Ergebnissen. Auf Seite 66 wurde darauf hingewiesen, daß ein vergleichbarer Kurvenverlauf auch an anderer Stelle erzeugt wurde.

Speziell der stark steigende Kurvenverlauf in der Phase I ist sehr positiv. Die Tatsache, daß ein hundert prozentiges Optimum nicht erreicht wird, liegt in der Natur dieser Optimierungsprobleme und wird auch bei anderen Versuchen bemerkt. Insbesondere bei DAVIS[Dav91] wird dieser Effekt beschrieben.

Auch der Kurvenverlauf in der Phase II und III läßt sich erklären und deuten. Nachdem in der Phase I ein sehr gutes lokales Maximum aufgetreten ist, bringt jede Veränderung und jede weitere Generation eine Verschlechterung mit sich, da die guten Individuen zerstört werden. Diese Zerstörung der Fitness findet dabei solange statt, bis sie durch den generellen Drang zur Optimierung abgestoppt und wieder in eine steigende Tendenz umgewandelt wird. Dieser Prozess mit den ausgeprägten Schwingungen im ersten Teil und dem Einschwingprozess in der Phase III erscheint typisch und verständlich, wenn man versucht eine Parallele zu der Genetik zu finden. Auch hier wird das Optimum nicht erreicht, sondern nur sehr gute Individuen. Der Sinn liegt in einer möglichst hohen Flexibilität gegenüber Umweltveränderungen. Hinzu kommt, daß ein Einschwingprozess in der Natur, insbesondere in dem Bereich der Physik sehr häufig vorkommt, so daß der Verdacht nahe liegt, daß das System durchaus natürliche Vorgänge nachempfindet.

Wie schon bei [Gri91] festgestellt wurde, gibt dieses Ergebnis Hoffnung, daß der Einsatz der GENETISCHEN ALGORITHMEN insbesondere bei der Aufgabenstellung im Operations Research oder bei anderen Problemstellungen bei der Optimierung von Graphen eingesetzt werden kann. Diese Aufgabenstellung wurden u.a. von TARASZOW[Tar91] untersucht. Wichtig ist, daß es sich hier nicht um eine Optimierung im Graphen handelt. Die Probleme des Optimalen Weges[1] o.ä. können mit diesem System nicht gelöst werden.

Die erzeugten Graphen entsprechen zwar der Vorstellung von einem optimalen Aufbau, jedoch sind sie für den Einsatz in einer Datenstruktur nicht

[1] z.B. Traveling Salesman

geeignet.

7.2 Betrachtung der Optimierung der Ordnung

Auch für diese Betrachtung wird die Darstellung der Kurvenverläufe in den vorigen Kapiteln zu Hilfe genommen. Und auch bei dieser Optimierung stimmt das Ergebnis mit den Ergebnissen ähnlicher Aufgabenstellungen aus der Literatur überein. Auch dieses wird bei [Dav91] beschrieben.

Hier liegt in der Phase A ein deutliches monotones Wachstum vor, welches auf eine stetige Optimierung schließen läßt. Diese stetige Optimierung führt mit einer relativ großen Sicherheit zu dem angepeilten Ziel. Allerdings ist diese Optimierung nicht so schnell wie das entsprechende Verfahren bei der Topologischen Optimierung. Trotzdem kann man bei dieser Phase von einer stabilen Optimierung sprechen.

In der Phase B ist wieder eine Einpendelung zu beobachten, die der in den Phasen II und III entspricht. Auch hier wird ein positiver Zustand nahe des Optimums von den Mechanismen der GENETISCHEN ALGORITHMEN zerstört.

Unter Einbeziehung der einschlägigen Literatur ist für diese Optimierung festzustellen, daß sie in ihrer Form durchaus typische Abläufe erzeugt[2], und damit genau wie die Topologische Optimierung mit GENETISCHEN ALGORITHMEN ein sicheres Verfahren darstellt.

Der erzeugte Graph zeigt zwar eine recht gute, grobe Ordnung, jedoch muß man folgendes bedenken: Eine Ordnung ist nur dann akzeptabel, wenn alle Elemente dieser Ordnung genügen. Die meisten erzeugten Graphen müßten also nachbehandelt, d.h. mit anderen Algorithmen bearbeitet werden. Trotzdem erscheint es möglich, GENETISCHE ALGORITHMEN als Vorphase eines Sortieralgorithmuses einzusetzen.

7.3 Mischung zu einer Gesamtfitness

Ganz anders stellt sich die Gesamtfitness dar. Durch die Mischung der an sich gutmütigen Einzelfitnessfunktionen entsteht eine Gesamtfitness, die in ihrer Fitness einen Wert über 75 Prozent nur sehr selten überschreitet.

Es muß bei der Mischung folgendes beachtet werden: Die einzelnen Fitnessfunktionen fragen verschiedene Eigenschaften im Graphen ab, es ist möglich, daß die Eigenschaft der Ordnung der topologisch optimalen Struktur schadet. Insbesondere der vollständige Graph[3] stellt für die Topologie ein Optimum dar, während er für eine Ordnung eine sehr schlechte Struktur darstellt.

[2] vgl.[Dav91], S.69
[3] vgl. S. 76

So ist es nicht verwunderlich, daß bei einer, auch nur annähernden Gleichgewichtung der Teilfunktionen ein schlechtes Ergebnis entsteht.

GOLDBERG[Gol89] schreibt, daß es ausgesprochen schwer ist, zwei Fitnessfunktionen miteinander zu mischen. Dieses gilt auch in Fällen, bei denen die Fitnessfunktionen nicht eine ähnliche Eigenschaft optimieren, also sie nur peripher beeinflussen, so wie bei diesen Versuchen.

Denn die topologische Eigenschaft der minimalen Weglänge im Graphen und die Eigenschaft einer Ordnung scheinen miteinander keine Gemeinsamkeiten zu haben. Trotzdem erscheint es, daß eine Mischung der Einzelfitnessfunktionen kein hinreichend gutes Ergebnis bringt, sich diese Teilfitnessfunktionen also ausschließen.

Um eine Verbesserung des Gesamtergebnisses zu erreichen, also die für den Verlauf nachteilige Mischung zu verhindern, muß man eine Fitnessfunktion aufstellen, die nur aus einer Optimierung besteht.

Ein entsprechender Ansatz kann ein Kapitel von MEHLHORN[Meh88] liefern. Dort schreibt der Autor von einer Selbstorganisierenden Datenstruktur, die sich auf Grund der Zugriffshäufigkeit neu aufbaut. Es handelt sich hierbei um einen gewichteten Baum[4], der sich je nach Zugriffshäufigkeit bei jedem Durchlauf neu bildet. Hierbei liegt die Topologie des Baumes fest vor, diese ist entweder ein *Binärbaum* oder eine *Lineare Liste*. Es werden lediglich die Elemente nach der Zugriffshäufigkeit hin neu in dieser Struktur verteilt. MEHLHORN spricht in diesem Zusammenhang von einer sich amortisierenden Datenstruktur.

Bei genauerem Betrachten dieses Ansatzes stellt man jedoch fest, daß hier lediglich eine feste Struktur einer bestimmten Ordnung unterworfen wird. Dieser Ansatz ist nicht mit der hier praktizierten Optimierung der Ordnung zu vergleichen, da hier die Topologische Struktur variabel gehalten wird[5], sondern es handelt sich um ein Anordnungsproblem. Diese Anordnungsprobleme sind an anderer Stelle untersucht worden und hier brachten die GENETISCHEN ALGORITHMEN gute Ergebnisse[6].

---

[4] vgl. [Meh88] S.147

[5] obwohl sie bei der Feststellung der Ordnung nicht berücksichtigt wird

[6] vgl. [Gue91]

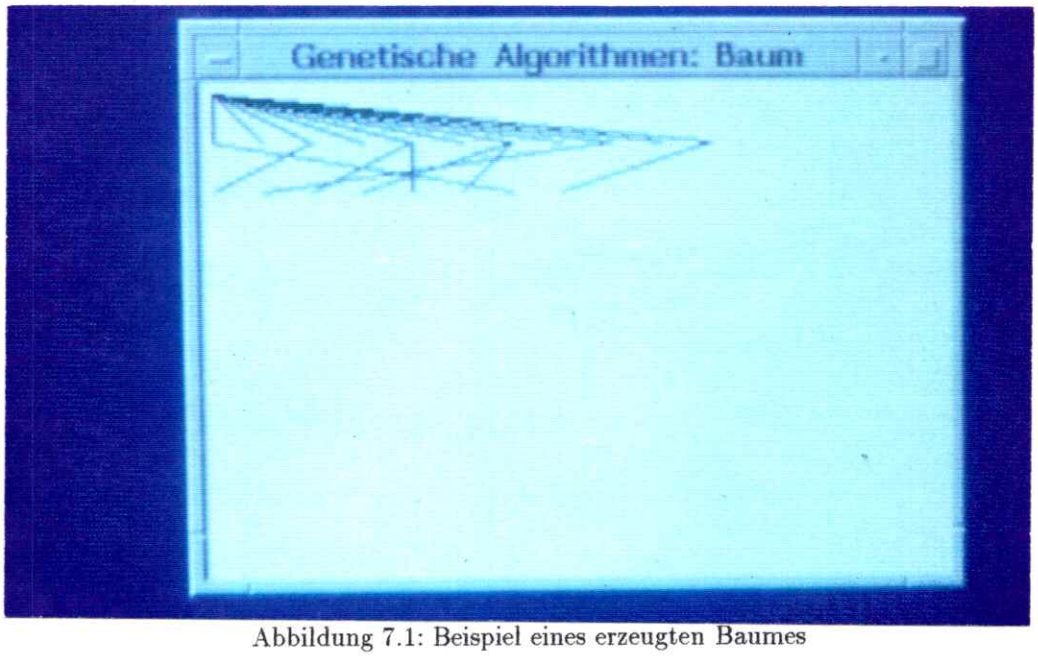


Abbildung 7.1: Beispiel eines erzeugten Baumes

7.4 Einige besondere Ergebnisse

In diesem Abschnitt werden einige, besonders auffällige Ergebnisse der Untersuchungen dokumentiert. Es handelt sich hierbei um Aufnahmen, die während den Versuchen von dem Monitor genommen wurden.

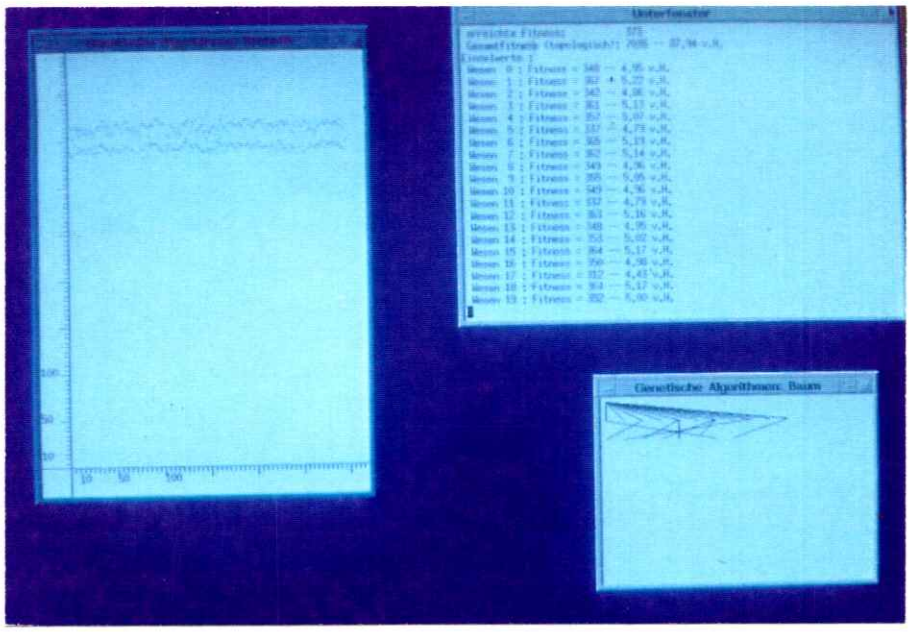


Abbildung 7.2: Versuchablauf bei der Topologischen Optimierung

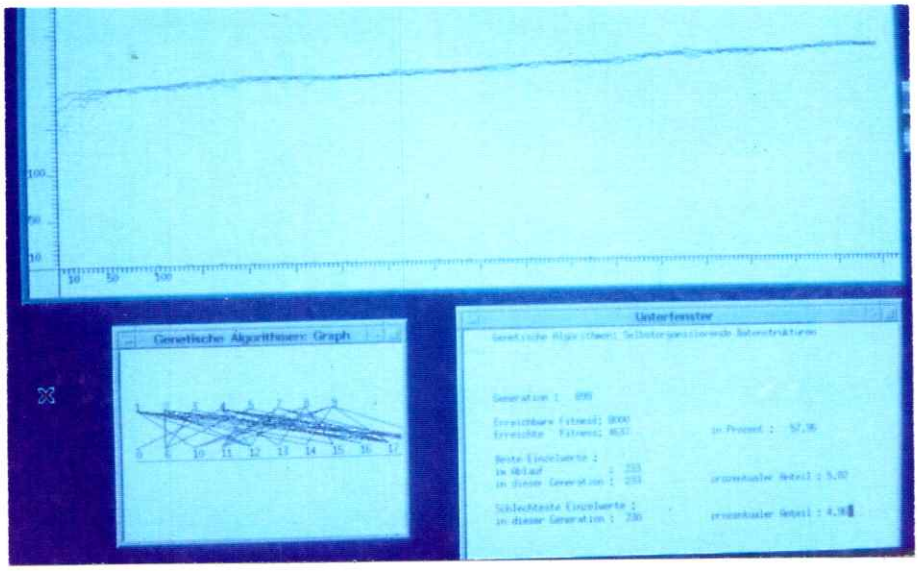


Abbildung 7.3: Darstellung einer optimierten Ordnung

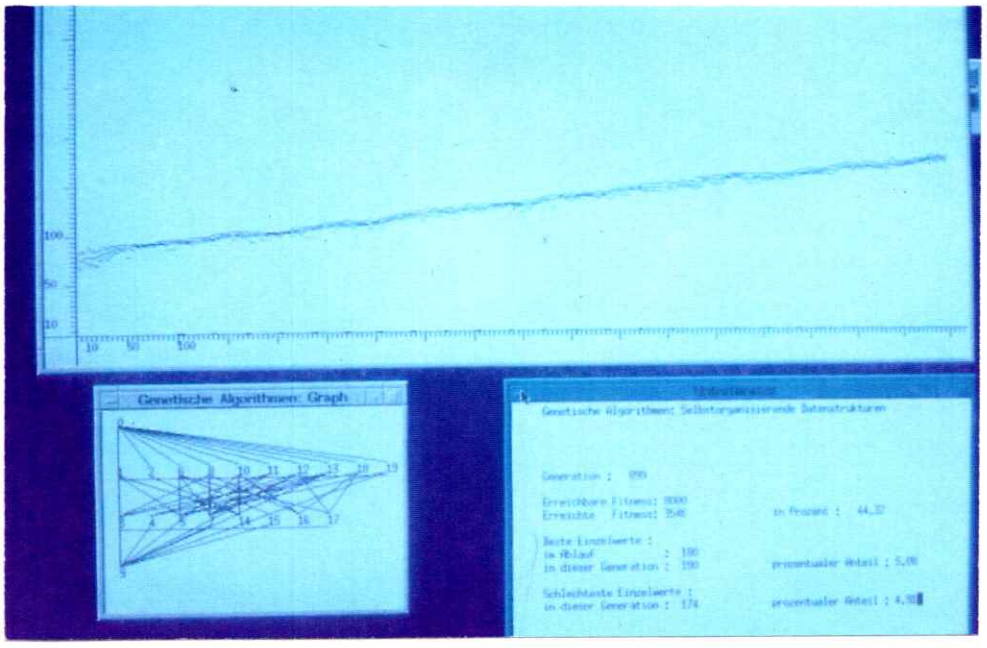


Abbildung 7.4: Versuchablauf bei der Optimierung der Ordnung

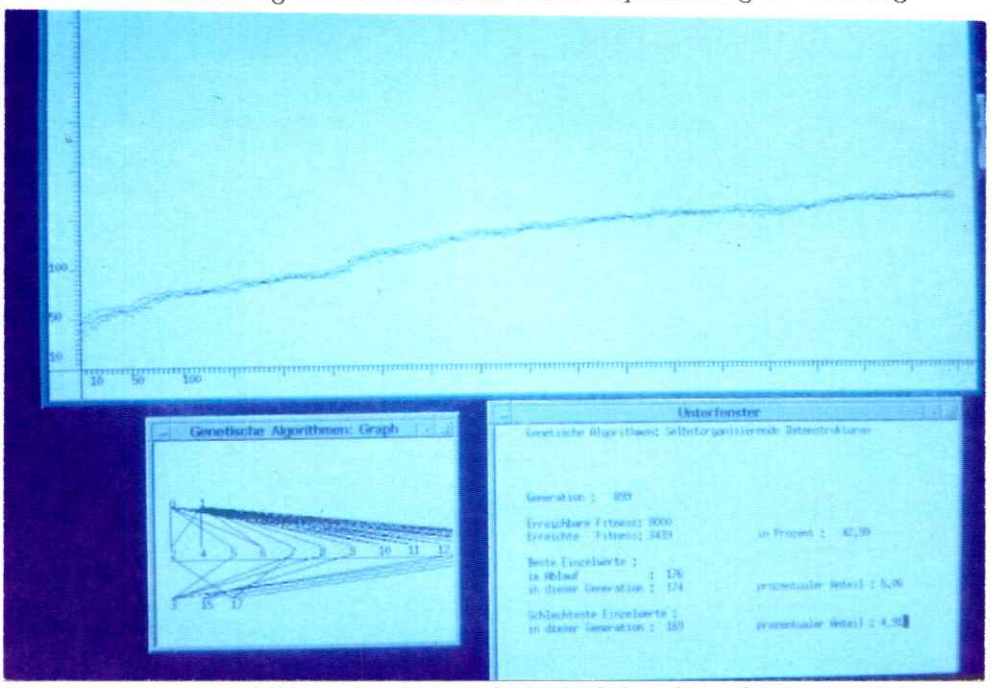


Abbildung 7.5: Gesamtfitnessfunktion, Verteilung 1 : 9

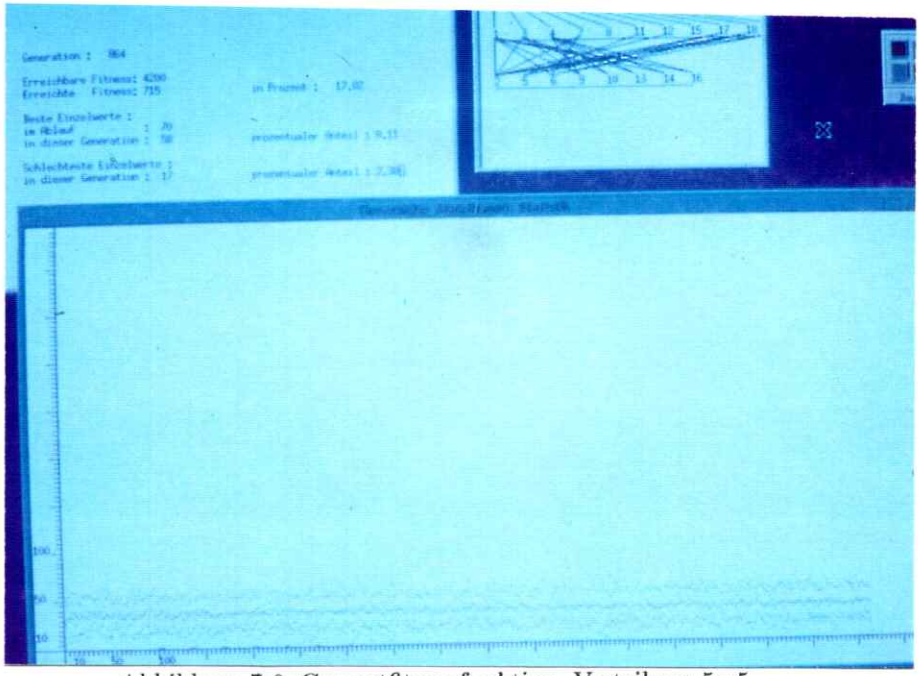


Abbildung 7.6: Gesamtfitnessfunktion, Verteilung 5 : 5

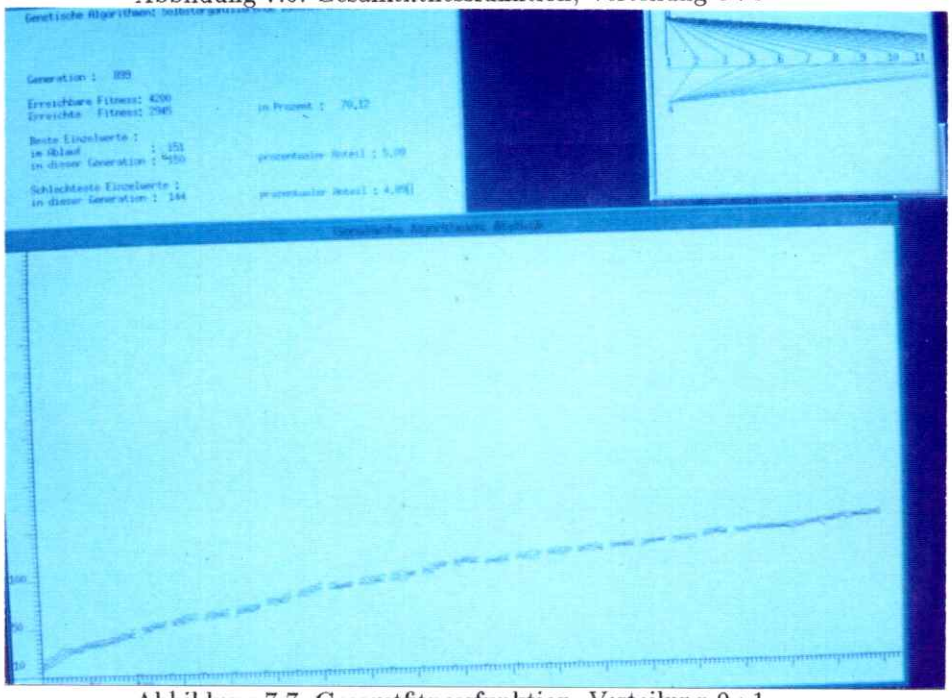


Abbildung 7.7: Gesamtfitnessfunktion, Verteilung 9 : 1

7.5 Abschließende Bemerkung

Nach Abschluß der Versuche und Auswertung der Ergebnisse läßt sich folgendes feststellen:

Das Ziel, durch Optimierung der Ordnung und Topologischer Optimierung eine sich selber organisierende Datenstruktur zu erzeugen, bzw. eine Möglichkeit zur Entwicklung einer solchen aufzuzeigen, wurde nicht erreicht.

Erreicht wurde der Aufbau eines Systems zur Topologischen Optimierung und zur Optimierung der Ordnung. Allerdings kann man mit diesen Ergebnissen sehr viel anfangen. So lassen sich die Erkenntnisse der Topologischen Optimierung z.B. im Bereich Fertigungstechnik bei der Optimierung der Produktionswege nutzen.

Man kann sicherlich Selbstorganisierende Datenstrukturen erzeugen, indem man dem Ansatz von MEHLHORN[Meh88] folgt, wie oben beschrieben wurde.

Jedoch stellt sich die Frage, ob der Effekt nicht auch durch andere Algorithmen erzeugt werden kann.

Denn gerade bei der langsamen Optimierung ist ein entsprechend programmierter konventioneller Algorithmus den GENETISCHEN ALGORITHMEN überlegen.

Auch scheint es kaum sinnvoll, GENETISCHE ALGORITHMEN zum Sortieren von Daten einzusetzen, da die konventionell programmierten Sortieralgorithmen mehr als deutlich diesem System überlegen sind. So läßt sich sagen, daß GENETISCHE ALGORITHMEN für den Einsatz bei Selbstorganisierenden Datenstrukturen nicht sonderlich geeignet sind. Mit den hier erzielten Ergebnissen ist es höchst zweifelhaft, ob es möglich sein wird, mit Hilfe von GENETISCHEN ALGORITHMEN eine Selbstorganisierende Datenstruktur zu erzeugen.

Teil III

Anhang

Anhang A

Die Experimentalumgebung

In diesem Teil des Anhangs wird die Experimentalumgebung dargestellt. Diese Experimentalumgebung besteht hauptsächlich aus einem Programm und der hinzugehörigen Hardwareumgebung. Diese beiden Komponenten sollen nun dargestellt werden.

A.1 Hardwareumgebung

Das Programm sollte unabhängig von einer speziellen Hardwareumgebung lauffähig sein. Um jedoch eine problemlose Implementierung auf einem anderen Rechner zu gewährleisten, muß die Umgebung die folgenden Regeln erfüllen.

Das angewandte Betriebssystem muß ein Unix System V sein, das den Normen der X/Open Gruppe, insbesondere der Bände 1 - 4 und 6 - 7 erfüllen[XOp89]. Als graphische Oberfläche wurde für das System X11 gewählt und benutzt.

Die Experimente wurden auf einem Rechner mit einem 80386SX Prozessor mit 16 MHz Taktfrequenz durchgeführt.

A.2 Die Software

Die Software besteht aus 13 Moduln. Diese Moduln werden im allgemeinen über eine festgelegte Schnittstelle angesprochen. Die Software wurde in der Programmiersprache `C` entwickelt und hält sich ohne Ausnahme an die Sprachdefinition von KERNIGHAM und RITCHIE[Ker88]. Diese Programmiersprache bietet sich bei der gewählten Umgebung an, da hier die meisten Resourcen zur Verfügung stehen.

Das Makefile

Das Makefile zeigt die Abhängigkeiten zwischen den einzelnen Moduln. Es wird von dem Utility "**make**" gelesen und ausgewertet. Sollte ein Modul nach der letzten Übersetzung verändert worden sein, so werden alle Moduln, die von diesem Modul abhängig sind, neu übersetzt.

Durch die Benutzung dieser Datei wird ein Programm mit dem Namen "**genetics**" erzeugt, das unter X11 gestartet werden kann.

```
LIB=-lXm -lXt -lX11 -lPW -lnsl_s -lnet -lc -lbsd -lcurses -lm
OBJ=Hauptprogram.o Zeige_Baum.o Zufall.o Initial.o Kontrolle.o Bewert.o \
    Fpflanz.o Basicwin.o Benutz.o Ordnung.o domenu.o
genetic: ${OBJ}
        cc -O ${OBJ} -o genetics ${LIB}
Hauptprogram.o: Hauptprogram.c Zeige_Baum.c Benutz.c genetics.h
Benutz.o:       Benutz.c  domenu.c domenu.h genetics.h
Zeige_Baum.o:   Zeige_Baum.c genetics.h
Initial.o:      Initial.c Zeige_Baum.c Zufall.c genetics.h
Kontrolle.o:    Kontrolle.c genetics.h
Bewert.o:       Bewert.c genetics.h
Fpflanz.o:      Fpflanz.c Zufall.c genetics.h
Basicwin.o:     Basicwin.c genetics.h
Zufall.o:       Zufall.c genetics.h
domenu.o:       domenu.c domenu.h
Ordnung.o:      Ordnung.c genetics.h
```

Globale Variablen

Dieser Abschnitt definiert Makros und globale Variablen für das gesamte Programm. Die Definitionen sind nach Anwendungsgebieten sortiert.

```
/*-------------------------------------------------------------------*/
/*                                                                   */
/* Modul: genetics.h                          Version 1.0            */
/*                                                                   */
/* Globale Variablendefinitionen fuer GENETICS                       */
/*                                                                   */
/* Datum : 31.August 1991                                            */
/*                                                                   */
/*-------------------------------------------------------------------*/

/* Standardincludes */
#include <X11/Xlib.h>
#include <X11/Xutil.h>
```

```
#include <X11/Xos.h>
#include <X11/Xatom.h>
#include <X11/Intrinsic.h>
#include <X11/Shell.h>
#include <Xm/Xm.h>
#include <Xm/MessageB.h>
#include <Xm/PushB.h>

#include <stdio.h>
#include <curses.h>

/* Globale Festlegungen */
#define BITMAPDEPTH     1
#define TOO_SMALL       0
#define BIG_ENOUGH      1

#define XKOORDINATE     0
#define YKOORDINATE     1

#define WAHR            1
#define FALSCH          0
#define BESTER          0
#define MITTEL          1
#define SCHLECHT        2

#define FIRST_TIME      0
#define NORMAL_USE      1
#define LAST_TIME       2
#define FEHLER          3

#define BREITENSUCHE    0
#define TIEFENSUCHE     1
#define ARIADNE         2

#define MaxNachfolger           2
#define MaxIndividuen           50
#define MaxIndividuenHalbe      25
int             NoCurses;
int             NoX11;
int             MaxGeneration;
int             MaxKnoten;
int             MaxWesen;
int             MaxWesenHalbe;
long float      M_Wahrschein;
```

```
float           Fit_Verteil;

/* Definitionen fuer den Zufallsgenerator */
long            Zufall_r[97];
short           Zufall_iff;

/* Definitionen fuer X11 */
Display         *display;
GC              FB_Grafik, FS_Grafik, FbB_Grafik;
XEvent          Aktion;
XSizeHints      Baum_Size, Statistik_Size;
XIconSize       *size_list;
XFontStruct     *font_info;
unsigned int    icon_width, icon_height;
Window          F_Statistik, F_Baum, Fb_Baum;

/* Daten fuer das Statistik- Fenster */
unsigned int    FS_Hoehe, FS_Breite, Org_FS_Hoehe, Org_FS_Breite;

/* Daten fuer das Baum-Fenster */
unsigned int    FB_Hoehe, FB_Breite, Org_FB_Hoehe, Org_FB_Breite;
int             FB_aktiv;

/* Daten fuer den besten Baum */
unsigned int    FbB_Hoehe, FbB_Breite, Org_FbB_Hoehe, Org_FbB_Breite;
unsigned int    border_width, display_width, display_height;
int             window_size;

short           Generation, Programmende;
int             GesamtFitness, Beste_Fitness, Best_Wesen;
int             Korrektur_Ordnung, Korrektur_Topologie;

int Individuum[100][100][MaxIndividuen];
int Root[MaxIndividuen];
int Tiefe[100][MaxIndividuen], Beste_Tiefe[100];
int Fitness[MaxIndividuen];
int Statistik[100][3];
int Best_Individuum[100][100];
int Ordnungszahl[MaxIndividuen];
```

A.2.1 Die Hauptroutinen

Die Hauptroutinen stellen das Kernstück des programmierten Systems dar. Dieses Kernstück umfasst das Hauptprogramm sowie die Moduln zur Bewertung

und Fortpflanzung.

Das Hauptprogramm

```
/*-------------------------------------------------------------------------*/
/*                                                                         */
/* Modul: Hauptprogram.c                          Version 1.0              */
/*                                                                         */
/* Hauptprogramm fuer GENETICS                                             */
/*                                                                         */
/* Datum :                                                                 */
/*                                                                         */
/*-------------------------------------------------------------------------*/

#include "genetics.h"

void GA_Haupt_Run_Genetics()
  {
  unsigned int Alt_FS_Hoehe, Alt_FS_Breite;
  while(Programmende == FALSCH)
      {
       if (Generation < MaxGeneration)
         {
         GA_Kontrolle_main();
         GA_Ordnung_main();
         GA_Bewert_main();
         GA_Fpflanz_main();
         GA_Zeige_Baum_Zeichne_Statistik();
         Generation++;
         }
       XCheckMaskEvent(display, ExposureMask | ButtonPressMask |
                       StructureNotifyMask, &Aktion);
       switch (Aktion.type) {
       case Expose:
                    if (Aktion.xexpose.count != 0)
                        break;
                    GA_Basicwin_Initialisierung_Grafik();
                    GA_Zeige_Baum_Zeichne_Statistik_neu();
                    GA_Zeige_Baum_Zeichne_Baum();
                    break;
       case ConfigureNotify:
                    FS_Breite = Aktion.xconfigure.width;
                    FS_Hoehe = Aktion.xconfigure.height;
```

```
                    if ((FS_Breite != Alt_FS_Breite) ||
                        (FS_Hoehe  != Alt_FS_Hoehe))
                      {
                      Alt_FS_Breite = FS_Breite;
                      Alt_FS_Hoehe = FS_Hoehe;
                      XClearWindow (display,F_Statistik);
                      }
                    break;
        case ButtonPress:
                    GA_Benutzerschnittstelle (NORMAL_USE, 0, NULL);
                    break;
        default:
                    break;
        }
    }
  }

main(argc,argv)
unsigned int    argc;
char            **argv;
  {
/* Initialisierung des Servers */
  Programmende = FALSCH;
  NoCurses = FALSCH;
  GA_Benutzerschnittstelle (FIRST_TIME, argc, argv);
  GA_Initial_Wesen();
  GA_Basicwin_Initialisierung_Fenster_Baum();
  GA_Basicwin_Initialisierung_Fenster_bester_Baum();
  GA_Basicwin_Initialisierung_Fenster_Statistik();
  GA_Haupt_Run_Genetics();
  }
```

Die folgenden Moduln stellen die beiden Phasen im Ablauf der *Genetischen Algorithmen* dar.

Das Bewertungsmodul

In diesem Modul werden die Funktionen der Bewertungsphase dargestellt.

```
/*-------------------------------------------------------------------*/
/*                                                                   */
/* Modul: bewert.c                          Version 1.0              */
/*                                                                   */
/* Bewertungsabschnitt fuer GENETICS                                 */
```

```
/*                                                                    */
/* Datum : 31.August 1991                                             */
/*                                                                    */
/*--------------------------------------------------------------------*/

#include <stdio.h>
#include "genetics.h"

int GA_Bewert_addiere_Tiefe(Wesen)
int Wesen;
  {
  int x, sum = 0;
  for (x = 0; x < MaxKnoten; x++)
    sum += Tiefe[x][Wesen];
  return(sum);
  }

int GA_Bewert_bewertete_Tiefe(Wesen)
int Wesen;
  {
  int x, sum = 0;
  for (x = 0; x < MaxKnoten; x++)
    sum += MaxKnoten + 1 - Tiefe[x][Wesen];
  return(sum);
  }

int GA_Bewert_MaxOrdnung()
  {
  int x = 0, sum = 0, y = 0;
  for (y = 0; y < MaxKnoten; y++)
    for (x = y; x < MaxKnoten; x++)
      sum++;
  return(sum);
  }

void GA_Bewert_main()
  {
/* OGF - optimale Gesamtfitness
   OEF - optimale Einzelfitness
   PGF - Prozent der Gesamtfitness*/
  int x, y, z, OGF = 0, OEF = 400, MaxWert = 0;
  int BaumFoerderung = 0, Bester_Baum = 0;
  int MinWert;
  double Prozent, PGF, q = 1, p = 0, PMaxWert, PMinWert;
```

```
  float TopologischeFitness, OrdnungsFitness, OEF1, OEF2;
  GesamtFitness = 0;
  OEF1 = (float)((MaxKnoten * MaxKnoten) + BaumFoerderung);
  OEF2 = (float)(GA_Bewert_MaxOrdnung());
  OGF = OEF * MaxWesen;
  MinWert = OEF;
  for (x = 0; x < MaxWesen; x++)
    {
/* Fitnessfunktion Mittlere Weglaenge */
    if (Root[x] < MaxWesen + 1)
      TopologischeFitness = TopologischeFitness + BaumFoerderung;
    if (Root[x] != MaxWesen + 1)
    TopologischeFitness = (int)((OEF1 - GA_Bewert_addiere_Tiefe(x)) *q +
                          (OEF1 - GA_Bewert_bewertete_Tiefe(x)) *p);
    else
      TopologischeFitness = 0;
/* Fitnessfunktion Ordnung */
    OrdnungsFitness = Ordnungszahl[x];
    if ((OrdnungsFitness < 0) || (OrdnungsFitness > OEF2))
      OrdnungsFitness = 0;
/* Skalieren und addieren der Fitnesswerte */
    TopologischeFitness = (TopologischeFitness / OEF1) * 400;
    OrdnungsFitness = (OrdnungsFitness / OEF2) * 400;
    Fitness[x] = (int)(Fit_Verteil * TopologischeFitness + (1 - Fit_Verteil) *
                       OrdnungsFitness);
    GesamtFitness += Fitness[x];
    if (Fitness[x] > Beste_Fitness)
      {
      Best_Wesen = x;
      Beste_Fitness = Fitness[x];
      for (y = 0; y < MaxKnoten; y++)
        for (z = 0; z < MaxKnoten; z++)
          Best_Individuum[y][z] = Individuum[y][z][x];
      for (y = 0; y < MaxKnoten; y++)
        Beste_Tiefe[y] = Tiefe[y][x];
      GA_Zeige_Baum_Zeichne_Baum();
      }
    if (Fitness[x] > MaxWert)
       {
       MaxWert = Fitness[x];
       Bester_Baum = x;
       }
    if (Fitness[x] < MinWert)
       MinWert = Fitness[x];
```

```
    }
  GA_Zeige_Baum_Zeichne_bester_Baum(Bester_Baum);
  Statistik[Generation][BESTER]  = MaxWert;
  Statistik[Generation][MITTEL]  = GesamtFitness / MaxWesen;
  Statistik[Generation][SCHLECHT]= MinWert;
  PGF = (float)GesamtFitness / (float)OGF * 100;
  PMinWert = ((float)MinWert / (float)GesamtFitness) * 100;
  PMaxWert = ((float)MaxWert / (float)GesamtFitness) * 100;
  GA_Benutz_Statusausgabe(Generation, OGF, GesamtFitness, PGF,
                          Beste_Fitness, MaxWert, MinWert, PMaxWert,
                          PMinWert);
/* Ausgabe ohne Curses */
  if (NoCurses == WAHR)
    {
    system("tput clear");
    printf("Generation : %i \n",Generation);
    printf("Auswertung der Baeume :\n");
    printf(" Optimale Werte :\n");
    printf(" Gesamtfitness (topologisch): %i, Einzelfitness: %i \n",OGF,OEF);
    printf(" erreichte Fitness:           %i\n",Beste_Fitness);
    printf(" Gesamtfitness (topologisch): %i -- %2.2f v.H.\n",GesamtFitness,PGF);
    printf("Einzelwerte :\n");
    for (x = 0; x < MaxWesen; x++)
      {
      Prozent = ((float)Fitness[x] / (float)GesamtFitness) * 100;
      printf(" Wesen %2i : Fitness = %i -- %2.2f v.H.\n",x,Fitness[x],Prozent);
      }
    }
/*  print_best_tree(); */
  }
```

Modul zur Kontrolle der Ordnung

In diesem Modul wird die Ordnung in dem Baum kontrolliert und protokolliert.
Das Ergebnis wird im Bewertungsmodul beurteilt.

```
/*-------------------------------------------------------------------*/
/*                                                                   */
/* Modul: Ordnung.c                          Version 1.0             */
/*                                                                   */
/* Feststellung der Ordnung des Graphen                              */
/*                                                                   */
/* Datum :                                                           */
```

```
/*                                                                    */
/*--------------------------------------------------------------------*/

#include "genetics.h"

void GA_Ordnung_Initial()
  {
  int x;
  for (x = 0; x < MaxWesen; x++)
    Ordnungszahl[x] = 0;
  }

void GA_Ordnung_Feststellen(Wesen)
int Wesen;
  {
  int x, y;
  for (x = 0; x < MaxKnoten; x++)
    for (y = 0; y < MaxKnoten; y++)
      {
      if ((Individuum[x][y][Wesen] == 1) && (x < y))
        Ordnungszahl[Wesen]++;
      if ((Individuum[x][y][Wesen] == 1) && (x > y))
        {
        Ordnungszahl[Wesen]--;
        if (Korrektur_Ordnung == WAHR)
          Individuum[x][y][Wesen] = 0;
        }
      }
  if (Ordnungszahl[Wesen] < 0)
    Ordnungszahl[Wesen] = 0;
  }

void GA_Ordnung_main()
  {
  int x;
  GA_Ordnung_Initial();
  for (x = 0; x < MaxWesen; x++)
    GA_Ordnung_Feststellen(x);
  }
```

Das Kontrollmodul

In diesem Abschnitt werden die verschiedenen internen Kontrollen und Prüfungen durchgeführt. Außerdem werden mehrere interne Tabellen, wie z.B. die Tie-

fentabelle, besetzt. Dieses Modul ist für die Bewertung der Individuen notwendig.

```
/*-------------------------------------------------------------------------*/
/*                                                                         */
/* Modul: Kontrolle                              Version 1.0               */
/*                                                                         */
/* Kontrollfunktionen im Ablauf                                            */
/*                                                                         */
/* Datum :                                                                 */
/*                                                                         */
/*-------------------------------------------------------------------------*/
#include "genetics.h"
int Neu_Individuum[100][100][MaxIndividuen];

int GA_Kontrolle_ist_Wurzel(Nachfolger,Wesen)
int Nachfolger, Wesen;
  {
  int x, sum = 0;
  for (x = 0; x < MaxKnoten; x++)
    {
    sum = sum + Neu_Individuum[x][Nachfolger][Wesen];
    }
  if (sum == 0)
    return (WAHR);
  else
    return (FALSCH);
  }


void GA_Kontrolle_ist_Baum (Wesen)
int Wesen;
  {
  int x, y, AnzWurzel = 0, AnzKnoten, Wurzel, lokale_Tiefe;
/* Initialisierung der Tiefentabelle */
  for (x = 0; x < MaxKnoten; x++)
    Tiefe[x][Wesen] = MaxKnoten + 1;
/* Finden der Wurzel */
  for (x = 0; x < MaxKnoten; x++)
    if (GA_Kontrolle_ist_Wurzel (x, Wesen) == WAHR)
       {
       AnzWurzel++;
       Tiefe[x][Wesen] = 0;
```

```
       Root[Wesen] = x;
       }
/* Wenn weniger als eine Wurzel vorhanden ist, dann ist es ein Graph,
   wenn genau eine Wurzel vorhanden ist, dann ist es ein Baum
   ansonsten ist es ein Wald */
  if (AnzWurzel < 1)
    Root[Wesen] = MaxWesen + 1;
  else
    {
    if (AnzWurzel == 1)
      {
      Wurzel= Root[Wesen];
      lokale_Tiefe = 1;
      for (x = 0; x < MaxKnoten; x++)
        if (Neu_Individuum[Wurzel][x][Wesen] ==  1)
          Tiefe[x][Wesen] = lokale_Tiefe;
      }
    else
      {
      Root[Wesen] = MaxWesen + 2;
      for (x = 0; x < MaxKnoten; x++)
        if (Tiefe[x][Wesen] == 0)
          Tiefe[x][Wesen] = 1;
      }
    }
/* naechste Schicht bearbeiten */
  AnzKnoten = 0;
  lokale_Tiefe = 1;
  do {
    AnzKnoten = 0;
    for (x = 0; x < MaxKnoten; x++)
      if (Tiefe[x][Wesen] == lokale_Tiefe)
        {
        AnzKnoten++;
        for (y = 0; y < MaxKnoten; y++)
          if (Neu_Individuum[x][y][Wesen] == 1)
            if  (Tiefe[y][Wesen] == MaxKnoten + 1)
              Tiefe[y][Wesen] = lokale_Tiefe + 1;
            else
              Neu_Individuum[x][y][Wesen] = 0;
/* Wenn ein Knoten von mehr als einer Wurzel angesprochen wird, dann wird
die entsprechende tiefere Verbindung gekappt, da sie eine Rueckfuehrung
darstellt. */
       }
```

```
    lokale_Tiefe++;
    } while (AnzKnoten > 0);
  }

int GA_Kontrolle_Rest_Graph (Wesen)
  int Wesen;
  {
  int x, y;
  for (x = 0; x < MaxKnoten; x++)
    {
    if (Tiefe[x][Wesen] == MaxWesen + 1)
      {
      for (y = 0; y < MaxKnoten; y++)
        Neu_Individuum[y][x][Wesen] = 0;
      return (WAHR);
      }
    }
  return (FALSCH);
  }

void GA_Kontrolle_main()
  {
  int Wesen, x, y, z;
  for (x = 0; x < MaxKnoten; x++)
    for (y = 0; y < MaxKnoten; y++)
      for (z = 0;  z < MaxWesen; z++)
        Neu_Individuum[x][y][z] = Individuum[x][y][z];
  for (Wesen = 0; Wesen < MaxWesen; Wesen++)
    {
    GA_Kontrolle_ist_Baum (Wesen);
    while (GA_Kontrolle_Rest_Graph (Wesen) == WAHR)
      GA_Kontrolle_ist_Baum (Wesen);
/*      printf("Graph: %i Nicht vollstaendig\n",Wesen); */
    }
/*  for (x = 0; x < MaxKnoten; x++) printf(" %2i ",x); printf("\n");
  for (Wesen = 0; Wesen < MaxWesen; Wesen++)
    {
    for (x = 0; x < MaxKnoten; x++)
      printf(" %2i ",Tiefe[x][Wesen]);
    printf("\n");
    }
  printf("\n");*/
  }
```

Das Fortpflanzungsmodul

Dieses Modul beinhaltet die Prozeduren zur Fortpflanzung der Individuen.

```
/*-------------------------------------------------------------------------*/
/*                                                                         */
/* Modul: Fpflanz                                  Version 1.0             */
/*                                                                         */
/* Fortpflanzungsphase                                                     */
/*                                                                         */
/* Datum : 12.11.1991                                                      */
/*                                                                         */
/*-------------------------------------------------------------------------*/
#include "genetics.h"

void GA_Fpflanz_Roulette()
  {
  int Wheel[100], Prozent, t, u, x, y = 0, z;
  int Neu_Individuum[100][100][MaxIndividuen];
/* Vorbereiten des Roulette- Rades */
  for (x = 0; x < MaxWesen; x++)
    {
    Prozent = (int)(((float)Fitness[x] / (float)GesamtFitness) * 100);
    for (z = 0; z < Prozent; z++)
      Wheel[y++] = x;
    }
/* Werfen des Roulette- Rades */
  for (x = 0; x < MaxWesen; x++)
    {
    z = GA_Zufall_main() % y;
    for (t = 0; t < MaxKnoten; t ++)
      for (u = 0; u < MaxKnoten; u++)
        Neu_Individuum[t][u][x] = Individuum[t][u][Wheel[z]];
    }
/* Uebertragen der Ergebnisse */
/* ohne Generation Gapping
  for (x = 0; x < MaxWesen; x++) */
/* mit Generation Gapping */
  for (t = 0; t < MaxKnoten; t++)
    for (u = 0; u < MaxKnoten; u++)
      Individuum[t][u][0] = Individuum[t][u][Best_Wesen];
  for (x = 1; x < MaxWesen; x++)
    for (t = 0; t < MaxKnoten; t++)
      for (u = 0; u < MaxKnoten; u++)
```

```
        Individuum[t][u][x] = Neu_Individuum[t][u][x];
  }

void GA_Fpflanz_Paarung()
  {
  int Paare[MaxIndividuenHalbe], t, u, x, y = 0, z, Besetzt;
  int Neu_Individuum[100][100][MaxIndividuen];
  for (x = 0; x < MaxWesenHalbe; x++)
    Paare[x] = MaxWesen;
  for (x = 0; x < MaxWesenHalbe; x++)
    {
    t = (GA_Zufall_main() % MaxWesenHalbe) + MaxWesenHalbe;
    do {
      Besetzt = FALSCH;
      for (y = 0; y < x; y++)
        if (Paare[y] == t)
          {
          Besetzt = WAHR;
          t = (GA_Zufall_main() % MaxWesenHalbe) + MaxWesenHalbe;
          }
      } while (Besetzt == WAHR);
    Paare[x] = t;
    }
  for (x = 0; x < MaxWesenHalbe; x++)
    {
    for (y = 0; y < MaxKnoten; y++)
      for (z = 0; z < MaxKnoten; z++)
        {
        Neu_Individuum[y][z][x] = Individuum[y][z][x];
        Neu_Individuum[y][z][Paare[x]] = Individuum[y][z][Paare[x]];
        }
    t = GA_Zufall_main() % MaxKnoten;
    u = GA_Zufall_main() % MaxKnoten;
    for (y = 0; y < t; y++)
      for (z = 0; z < u; z++)
        {
        Neu_Individuum[y][z][x] = Individuum[y][z][Paare[x]];
        Neu_Individuum[y][z][Paare[x]] = Individuum[y][z][x];
        }
    for (y = t; y < MaxKnoten; y++)
      for (z = 0; z < MaxKnoten; z++)
        {
        Neu_Individuum[y][z][x] = Individuum[y][z][Paare[x]];
        Neu_Individuum[y][z][Paare[x]] = Individuum[y][z][x];
```

```
        }
    }
/* Uebertragen der Ergebnisse */
  for (x = 0; x < MaxWesen; x++)
    for (y = 0; y < MaxKnoten; y++)
      for (z = 0; z < MaxKnoten; z++)
        Individuum[y][z][x] = Neu_Individuum[y][z][x];
  }

void GA_Fpflanz_Mutation()
  {
  long float Elemente;
  int Nr, x, y, z;
  Elemente = MaxKnoten * MaxKnoten * MaxWesen;
  Elemente *= M_Wahrschein;
  if (Elemente > 0)
    for (Nr = 0; Nr < Elemente; Nr++)
      {
      x = GA_Zufall_main() % MaxKnoten;
      y = GA_Zufall_main() % MaxKnoten;
      z = GA_Zufall_main() % MaxWesen;
      if (Individuum[x][y][z] == 0)
        Individuum[x][y][z] = 1;
      else
        Individuum[x][y][z] = 0;
      }
  }

void GA_Fpflanz_main()
  {
  GA_Fpflanz_Roulette();
  GA_Fpflanz_Paarung();
  GA_Fpflanz_Mutation();
  }
```

Das Initialisierungsmodul

In diesem Abschnitt werden die verschiedenen Individuen initialisiert. Bei dieser Initialisierung werden Bäume erzeugt.

```
/*-------------------------------------------------------------------------*/
/*                                                                         */
/* Modul: Initial                          Version 1.0                     */
```

```
/*                                                                    */
/* Initialisierung der Adjazenzmatrix                                 */
/*                                                                    */
/* Datum : 20.Juli 1991                                               */
/*                                                                    */
/*--------------------------------------------------------------------*/
#include <stdio.h>
#include "genetics.h"

/* ein Element ist genau dann Blatt, wenn es nur Vorgaenger hat */
int GA_Initial_ist_Blatt(Wurzel,Wesen)
int Wurzel, Wesen;
  {
  int x, sum = 0;
  for (x = 0; x < MaxKnoten; x++)
     sum = sum + Individuum[Wurzel][x][Wesen];
  if (sum == 0)
     {
     for (x = 0; x < MaxKnoten; x++)
        sum = sum + Individuum [x][Wurzel][Wesen];
     if (sum != 0)
        return (WAHR);
     }
  return (FALSCH);
  }

/* ein Element ist genau dann Wurzel, wenn es keine Nachfolger hat */
int GA_Initial_ist_Wurzel(Nachfolger,Wesen)
int Nachfolger, Wesen;
  {
  int x, sum = 0;
  for (x = 0; x < MaxKnoten; x++)
     {
     sum = sum + Individuum[x][Nachfolger][Wesen];
     }
  if (sum == 0)
     return (WAHR);
  else
     return (FALSCH);
  }

/* ein Element ist genau dann frei, wenn es weder Wurzel noch Knoten ist */
int GA_Initial_ist_freies_Element(Wesen)
int Wesen;
```

```
{
  int x, y, sum = 0;
  for (x = 0; x < MaxKnoten; x++)
    {
    for (y = 0; y < MaxKnoten; y++)
      sum = sum + Individuum[x][y][Wesen] + Individuum[y][x][Wesen];
    if (sum == 0)
      return(WAHR);
    else
      sum = 0;
    }
  return (FALSCH);
}

void GA_Initial_Baue_Baum(Wesen)
int Wesen;
{
  int Wurzel, Nachfolger, x;
/* Suche eine Wurzel */
  Wurzel = GA_Zufall_main() % MaxKnoten;
  Root[Wesen] = Wurzel;
/* TESTLAUF
  printf("Wurzel des Baumes: %i\n",Wurzel);
*/
  do {
/* Suche Nachfolger entsprechend eingestellter Rang */
    Nachfolger = Wurzel;
    for (x = 0; x < MaxNachfolger; x++)
      {
       while ((Nachfolger == Wurzel) ||
              (GA_Initial_ist_Wurzel(Nachfolger,Wesen) == FALSCH))
         Nachfolger = GA_Zufall_main() % MaxKnoten;
       if (Root[Wesen] != Nachfolger)
         {
/* TESTLAUF
          printf("Nachfolger : %i\n",Nachfolger);
*/
          Individuum[Wurzel][Nachfolger][Wesen]=1;
         }
       Nachfolger = Wurzel;
      }
/* Neue Wurzel suchen */
    while (GA_Initial_ist_Blatt(Wurzel,Wesen) == FALSCH)
        Wurzel = GA_Zufall_main() % MaxKnoten;
```

```
/* TESTLAUF
    printf("Neue Wurzel: %i \n",Wurzel);
*/
  } while (GA_Initial_ist_freies_Element(Wesen) == WAHR);
}

void GA_Initial_Wesen()
  {
   int x,y,z;
/* Vorinitialisierung der Individuen */
   Beste_Fitness = 0;
   for (z = 0; z < MaxWesen; z++)
     {
     for (x = 0; x < MaxKnoten; x++)
       for (y = 0; y < MaxKnoten; y++)
         Individuum[x][y][z] = 0;
     }
/* Aufbau der Baeume */
   for (z = 0; z < MaxWesen; z++)
     {
     GA_Initial_Baue_Baum(z);
/* Testausgabe der erzeugten Baeume
     printf("\n");
     print_tree(z); */
     }
  }
```

A.2.2 Moduln für die graphische Schnittstelle

Diese Moduln sind gesondert gehalten, um eine Anpassung an andere Umgebungen zu ermöglichen.

Das Grundmodul für die Initialisierung der graphischen Ausgabe

```
/*-------------------------------------------------------------------*/
/*                                                                   */
/* Modul: Basicwin                          Version 1.0              */
/*                                                                   */
/* X11- Schnittstelle fuer GENETICS                                  */
/*                                                                   */
/* Datum : 10.Juli 1991                                              */
/*                                                                   */
/*-------------------------------------------------------------------*/
```

```
/* Standardincludes */
#include "genetics.h"
#include <stdio.h>

int screen_num;
char *progname = "genetic";

void GA_Basicwin_Initialisierung_Fenster_Statistik()
{
     int x, y;
/* Text fuer das Icon */
     char *FS_icon = "Statistik";
     char *FS_name = "Genetische Algorithmen: Statistik";
     Pixmap icon_pixmap;
     int count;

     window_size = 0;
     border_width = 4;

/* Uebernahme der Groesse des Bildschirmes aus den Systemdaten */
     screen_num = DefaultScreen(display);
     display_width = DisplayWidth(display, screen_num);
     display_height= DisplayHeight(display,screen_num);

     x = 0;
     y = 400;

     FS_Hoehe = (int)(1.25 * (MaxKnoten * MaxKnoten - MaxKnoten));
     FS_Breite = MaxGeneration + 50;
     Org_FS_Hoehe = FS_Hoehe;
     Org_FS_Breite= FS_Breite;

/* Aufbau eines einzelnen Fensters entsprechend den Voreinstellungen */
     F_Statistik = XCreateSimpleWindow(display, RootWindow(display, screen_num),
             x, y, FS_Hoehe, FS_Breite, border_width, BlackPixel (display,
             screen_num), WhitePixel (display, screen_num));

     XGetIconSizes (display, RootWindow (display, screen_num),&size_list,
                    &count);

     Statistik_Size.flags = PPosition | PSize | PMinSize;
     Statistik_Size.x = x;
     Statistik_Size.y = y;
```

```
    Statistik_Size.width = FS_Breite;
    Statistik_Size.height= FS_Hoehe;
    Statistik_Size.min_width = 300;
    Statistik_Size.min_height = 200;

    XSetStandardProperties (display, F_Statistik, FS_name, FS_icon,
        icon_pixmap,progname,0, &Statistik_Size);
    XSelectInput (display, F_Statistik, ExposureMask |
                ButtonPressMask | StructureNotifyMask);

    GA_Basicwin_erzeuge_Grafik_Umgebung(F_Statistik, &FS_Grafik, font_info);
    XMapWindow(display, F_Statistik);
}

void GA_Basicwin_Initialisierung_Fenster_Baum()
{
    int x, y;
/* Text fuer das Icon */
    char *FB_icon = "bester Graph";
    char *FB_name = "Genetische Algorithmen: Graph";
    Pixmap icon_pixmap;
    int count;

    window_size = 0;
    border_width = 4;

/* Uebernahme der Groesse des Bildschirmes aus den Systemdaten */
    screen_num = DefaultScreen(display);
    display_width = DisplayWidth(display, screen_num);
    display_height= DisplayHeight(display,screen_num);

    x = y = 0;

    FB_Hoehe = MaxWesen * 10;
    FB_Breite = MaxWesen * 10;

/* Aufbau eines einzelnen Fensters entsprechend den Voreinstellungen */
    F_Baum = XCreateSimpleWindow(display, RootWindow(display,screen_num),
            x, y, FB_Hoehe, FB_Breite, border_width, BlackPixel (display,
            screen_num), WhitePixel (display,screen_num));

    XGetIconSizes (display, RootWindow (display, screen_num),&size_list,
                &count);
```

```
     Baum_Size.flags = PPosition | PSize | PMinSize;
     Baum_Size.x = x;
     Baum_Size.y = y;
     Baum_Size.width = FB_Breite;
     Baum_Size.height= FB_Hoehe;
     Baum_Size.min_width = 300;
     Baum_Size.min_height = 200;

     XSetStandardProperties (display, F_Baum, FB_name, FB_icon,
          icon_pixmap,progname,0, &Baum_Size);
     XSelectInput (display, F_Baum, ExposureMask | ButtonPressMask
                    | StructureNotifyMask);

     GA_Basicwin_erzeuge_Grafik_Umgebung(F_Baum, &FB_Grafik, font_info);
     XMapWindow(display, F_Baum);
}

void GA_Basicwin_Initialisierung_Fenster_bester_Baum()
{
     int x, y;
/* Text fuer das Icon */
     char *FB_icon = "Baum";
     char *FB_name = "Genetische Algorithmen: Baum";
     Pixmap icon_pixmap;
     int count;

     window_size = 0;
     border_width = 4;

/* Uebernahme der Groesse des Bildschirmes aus den Systemdaten */
     screen_num = DefaultScreen(display);
     display_width = DisplayWidth(display, screen_num);
     display_height= DisplayHeight(display,screen_num);

     x = y = 0;

     FbB_Hoehe = MaxWesen * 10;
     FbB_Breite = MaxWesen * 10;

/* Aufbau eines einzelnen Fensters entsprechend den Voreinstellungen */
     Fb_Baum = XCreateSimpleWindow(display, RootWindow(display,screen_num),
             x, y, FbB_Hoehe, FbB_Breite, border_width, BlackPixel (display,
             screen_num), WhitePixel (display,screen_num));
```

```
    XGetIconSizes (display, RootWindow (display, screen_num),&size_list,
                   &count);

    Baum_Size.flags = PPosition | PSize | PMinSize;
    Baum_Size.x = x;
    Baum_Size.y = y;
    Baum_Size.width = FbB_Breite;
    Baum_Size.height= FbB_Hoehe;
    Baum_Size.min_width = 300;
    Baum_Size.min_height = 200;

    XSetStandardProperties (display, Fb_Baum, FB_name, FB_icon,
         icon_pixmap,progname,0, &Baum_Size);
    XSelectInput (display, Fb_Baum, ExposureMask | ButtonPressMask
                  | StructureNotifyMask);

    GA_Basicwin_erzeuge_Grafik_Umgebung(Fb_Baum, &FbB_Grafik, font_info);
    XMapWindow(display, Fb_Baum);
}

GA_Basicwin_erzeuge_Grafik_Umgebung(Fenster, Grafik_Umgebung, font_info)
Window Fenster;
GC *Grafik_Umgebung;
XFontStruct *font_info;
{
   unsigned long valuemask = 0;
   XGCValues values;
   unsigned int line_width = 0;
   int line_style = LineSolid;
   int cap_style = CapButt;
   int join_style = JoinRound;

   *Grafik_Umgebung = XCreateGC (display, Fenster, valuemask, &values);
   XSetFont (display, *Grafik_Umgebung, font_info->fid);
   XSetLineAttributes (display, *Grafik_Umgebung, line_width, line_style,
                       cap_style, join_style);
}

GA_Basicwin_load_font (font_info)
XFontStruct **font_info;
{
   char *fontname = "fixed";
   if ((*font_info = XLoadQueryFont (display,fontname)) == NULL)
      {
```

```
      GA_Benutzerschnittstelle(FEHLER,2,NULL);
      }
}

GA_Basicwin_Initialisierung_Grafik()
{
  int x, Dehnungsfaktor_X, Dehnungsfaktor_Y;
  int len10, len50, len100;
  char *Marke10 = "10";
  char *Marke50 = "50";
  char *Marke100= "100";
  len10 = strlen(Marke10);
  len50 = strlen(Marke50);
  len100= strlen(Marke100);
  Dehnungsfaktor_Y = FS_Hoehe / Org_FS_Hoehe;
  Dehnungsfaktor_X = FS_Breite/ Org_FS_Breite;
  if (Dehnungsfaktor_X < 1)
    Dehnungsfaktor_X = 1;
  if (Dehnungsfaktor_Y < 1)
    Dehnungsfaktor_Y = 1;

  XDrawLine(display,F_Statistik,FS_Grafik, 30, 0, 30, FS_Hoehe);
  XDrawLine(display,F_Statistik,FS_Grafik, 0,(FS_Hoehe - 30),
            FS_Breite, (FS_Hoehe - 30));
  for (x = 30; x < FS_Breite; x += (5 * Dehnungsfaktor_X))
    XDrawLine(display,F_Statistik,FS_Grafik, x, (FS_Hoehe - 30),
            x, (FS_Hoehe - 25));
  for (x = 30; x < FS_Breite; x += (50 * Dehnungsfaktor_X))
    XDrawLine(display,F_Statistik,FS_Grafik, x, (FS_Hoehe - 30),
            x, (FS_Hoehe - 20));
  for (x = (FS_Hoehe - 30); x > 0; x -= (5 * Dehnungsfaktor_Y))
    XDrawLine(display,F_Statistik,FS_Grafik, 25, x,
            30, x);
  for (x = (FS_Hoehe - 30); x > 0; x -= (50 * Dehnungsfaktor_Y))
    XDrawLine(display,F_Statistik,FS_Grafik, 20, x,
            30, x);
  XDrawString (display, F_Statistik,FS_Grafik,
               30 + (10 * Dehnungsfaktor_X), FS_Hoehe - 15,
               Marke10, len10);
  XDrawString (display, F_Statistik,FS_Grafik,
               30 + (50 * Dehnungsfaktor_X), FS_Hoehe - 15,
               Marke50, len50);
  XDrawString (display, F_Statistik,FS_Grafik,
               30 + (100 * Dehnungsfaktor_X), FS_Hoehe - 15,
```

```
                Marke100, len100);
  XDrawString (display, F_Statistik,FS_Grafik,
                1, (FS_Hoehe - 30 - (10 * Dehnungsfaktor_Y)),
                Marke10, len10);
  XDrawString (display, F_Statistik,FS_Grafik,
                1, (FS_Hoehe - 30 - (50 * Dehnungsfaktor_Y)),
                Marke50, len50);
  XDrawString (display, F_Statistik,FS_Grafik,
                1, (FS_Hoehe - 30 - ( 100 * Dehnungsfaktor_Y)),
                Marke100, len100);
}

GA_Basicwin_TooSmall (win, gc, font_info)
Window win;
GC gc;
XFontStruct *font_info;
{
   char *string1 = "Zu klein !";
   int y_offset, x_offset;

   y_offset = font_info->ascent + 2;
   x_offset = 2;

   XDrawString (display, win, gc, x_offset, y_offset, string1,
                strlen(string1));
}
```

Graphische Ausgabe

In diesem Modul wird die laufende Ausgabe der Zustände realisiert.

```
/*-------------------------------------------------------------------------*/
/*                                                                         */
/* Modul: Zeige_Baum                          Version 1.0                  */
/*                                                                         */
/* Graphische Ausgabe fuer GENETICS                                        */
/*                                                                         */
/* Datum : 10.Juli 1991                                                    */
/*                                                                         */
/*-------------------------------------------------------------------------*/

#include <stdio.h>
#include "genetics.h"
```

```
void GA_Zeige_Baum_print_tree(Wesen)
int Wesen;
  {
  int x, y;
  for (x = 0; x < MaxKnoten; x++)
    {
    for (y = 0; y < MaxKnoten; y++)
      printf(" %i ",Individuum[x][y][Wesen]);
    printf("\n");
    }
  }

GA_Zeige_Baum_print_best_tree()
  {
  int x, y;
  for (x = 0; x < MaxKnoten; x++)
    {
    for (y = 0; y < MaxKnoten; y++)
      printf(" %i ",Best_Individuum[x][y]);
    printf("\n");
    }
  }

void GA_Zeige_Baum_Zeichne_Statistik_neu()
  {
  int Dehnungsfaktor_X, Dehnungsfaktor_Y, x, y, z;
  Dehnungsfaktor_Y = FS_Hoehe / Org_FS_Hoehe;
  Dehnungsfaktor_X = FS_Breite/ Org_FS_Breite;
  if (Dehnungsfaktor_X < 1)
    Dehnungsfaktor_X = 1;
  if (Dehnungsfaktor_Y < 1)
    Dehnungsfaktor_Y = 1;
  for (z = 0; z < (Generation - 2); z++)
    {
    x = 30 + z * Dehnungsfaktor_X;
    y = (FS_Hoehe - 30) - (Statistik[z][BESTER] * Dehnungsfaktor_Y);
    XDrawPoint(display,F_Statistik,FS_Grafik, x, y);
    y = (FS_Hoehe - 30) - (Statistik[z][MITTEL] * Dehnungsfaktor_Y);
    XDrawPoint(display,F_Statistik,FS_Grafik, x, y);
    y = (FS_Hoehe - 30) - (Statistik[z][SCHLECHT] * Dehnungsfaktor_Y);
    XDrawPoint(display,F_Statistik,FS_Grafik, x, y);
    }
  }
```

```
void GA_Zeige_Baum_Zeichne_Statistik()
  {
  int Dehnungsfaktor_X, Dehnungsfaktor_Y, x, y;
  Dehnungsfaktor_Y = FS_Hoehe / Org_FS_Hoehe;
  Dehnungsfaktor_X = FS_Breite/ Org_FS_Breite;
  if (Dehnungsfaktor_X < 1)
    Dehnungsfaktor_X = 1;
  if (Dehnungsfaktor_Y < 1)
    Dehnungsfaktor_Y = 1;
  x = 30 + Generation * Dehnungsfaktor_X;
  y = (FS_Hoehe - 30) - (Statistik[Generation][BESTER] * Dehnungsfaktor_Y);
  XDrawPoint(display,F_Statistik,FS_Grafik, x, y);
  y = (FS_Hoehe - 30) - (Statistik[Generation][MITTEL] * Dehnungsfaktor_Y);
  XDrawPoint(display,F_Statistik,FS_Grafik, x, y);
  y = (FS_Hoehe - 30) - (Statistik[Generation][SCHLECHT] * Dehnungsfaktor_Y);
  XDrawPoint(display,F_Statistik,FS_Grafik, x, y);
  }

void GA_Zeige_Baum_Zeichne_Baum()
  {
  int Koordinate[MaxIndividuen][2], x, y, z, Zeichenlaenge;
  char *Zeichen[4];
  for (x = 0; x < MaxWesen; x++)
    {
    Koordinate[x][XKOORDINATE] = 0;
    Koordinate[x][YKOORDINATE] = 0;
    }
  XClearWindow (display, F_Baum);
  y = x = 0;
  for (y = 0; y < MaxKnoten; y++)
    {
    for (z = 0; z < MaxKnoten; z++)
      {
      if (Beste_Tiefe[z] == y)
        {
        Koordinate[z][XKOORDINATE] = (x++ * 30 + 15);
        Koordinate[z][YKOORDINATE] = (y * 50 + 15);
        }
      }
    x = 0;
    }
  for (y = 0; y < MaxKnoten; y++)
    {
```

```
    (void) sprintf(Zeichen,"%d",y);
    Zeichenlaenge = strlen(Zeichen);
    XDrawString(display, F_Baum, FB_Grafik, Koordinate[y][XKOORDINATE],
                                            Koordinate[y][YKOORDINATE],
                                            Zeichen, Zeichenlaenge);
    }
  for (x = 0; x < MaxKnoten; x++)
    for (y = 0; y < MaxKnoten; y++)
      if (Best_Individuum[x][y] == 1)
        {
        if (Koordinate[x][YKOORDINATE] < Koordinate[y][YKOORDINATE])
          XDrawLine(display,F_Baum,FB_Grafik,Koordinate[x][XKOORDINATE],
                                            Koordinate[x][YKOORDINATE],
                                            Koordinate[y][XKOORDINATE],
                                            Koordinate[y][YKOORDINATE] -10);
        if (Koordinate[x][YKOORDINATE] > Koordinate[y][YKOORDINATE])
          XDrawLine(display,F_Baum,FB_Grafik,Koordinate[x][XKOORDINATE],
                                            Koordinate[x][YKOORDINATE] -10,
                                            Koordinate[y][XKOORDINATE],
                                            Koordinate[y][YKOORDINATE]);
        if (Koordinate[x][YKOORDINATE] == Koordinate[y][YKOORDINATE])
          XDrawLine(display,F_Baum,FB_Grafik,Koordinate[x][XKOORDINATE],
                                            Koordinate[x][YKOORDINATE] +3,
                                            Koordinate[y][XKOORDINATE],
                                            Koordinate[y][YKOORDINATE] +3);
        }
  }

void GA_Zeige_Baum_Zeichne_bester_Baum(bester_Baum)
int bester_Baum;
  {
  int Koordinate[MaxIndividuen][2], x, y, z, Zeichenlaenge;
  char *Zeichen[4];
  for (x = 0; x < MaxWesen; x++)
    {
    Koordinate[x][XKOORDINATE] = 0;
    Koordinate[x][YKOORDINATE] = 0;
    }
  XClearWindow (display, Fb_Baum);
  y = x = 0;
  for (y = 0; y < MaxKnoten; y++)
    {
    for (z = 0; z < MaxKnoten; z++)
      {
```

```
      if (Tiefe[z][bester_Baum] == y)
        {
        Koordinate[z][XKOORDINATE] = (x++ * 30 + 15);
        Koordinate[z][YKOORDINATE] = (y * 50 + 15);
        }
      }
    x = 0;
    }
  for (y = 0; y < MaxKnoten; y++)
    {
    (void) sprintf(Zeichen,"%d",y);
    Zeichenlaenge = strlen(Zeichen);
    XDrawString(display, Fb_Baum, FbB_Grafik, Koordinate[y][XKOORDINATE],
                                              Koordinate[y][YKOORDINATE],
                                              Zeichen, Zeichenlaenge);
    }
  for (x = 0; x < MaxKnoten; x++)
    for (y = 0; y < MaxKnoten; y++)
      if (Individuum[x][y][bester_Baum] == 1)
        {
        if (Koordinate[x][YKOORDINATE] < Koordinate[y][YKOORDINATE])
          XDrawLine(display,Fb_Baum,FbB_Grafik,Koordinate[x][XKOORDINATE],
                                               Koordinate[x][YKOORDINATE],
                                               Koordinate[y][XKOORDINATE],
                                               Koordinate[y][YKOORDINATE] -10);
        if (Koordinate[x][YKOORDINATE] > Koordinate[y][YKOORDINATE])
          XDrawLine(display,Fb_Baum,FbB_Grafik,Koordinate[x][XKOORDINATE],
                                               Koordinate[x][YKOORDINATE] -10,
                                               Koordinate[y][XKOORDINATE],
                                               Koordinate[y][YKOORDINATE]);
        if (Koordinate[x][YKOORDINATE] == Koordinate[y][YKOORDINATE])
          XDrawLine(display,Fb_Baum,FbB_Grafik,Koordinate[x][XKOORDINATE],
                                               Koordinate[x][YKOORDINATE] +3,
                                               Koordinate[y][XKOORDINATE],
                                               Koordinate[y][YKOORDINATE] +3);
        }
  }
```

A.2.3 Moduln für die Benutzerschnittstelle

Benutzerschnittstelle

```
/*-------------------------------------------------------------------*/
```

```
/*                                                                    */
/* Modul: Benutz.c                              Version 1.0           */
/*                                                                    */
/* Benutzerschnittstelle fuer GENETICS                                */
/*                                                                    */
/* Datum : 20.August 1991                                             */
/*                                                                    */
/*--------------------------------------------------------------------*/

#include "genetics.h"
#include "domenu.h"

char *menutab[5] = {
    "Starten des Systems",
    "Aendern der Parameter",
    "",
    "Abbrechen",
    };

struct menu Menu = {
    4, 25,
    "Genetische Algorithmen: Auswahlmenu",
    &menutab[0]
    };

static XmStringCharSet OSF_CharSet = (XmStringCharSet) XmSTRING_DEFAULT_CHARSET;

void GA_Benutz_error(Fehlerart)
int Fehlerart;
  {
  attron(A_BLINK|A_REVERSE);
  switch(Fehlerart) {
    case 1:
      mvaddstr(12,10,"Genetics: Keine Verbindung mit dem X-Server moeglich\0");
      break;
    case 2:
      mvaddstr(12,10,"Genetics: Definierter X11-Font nicht auffindbar\0");
    default:
      mvaddstr(12,10,"Genetics: Nicht klassifizierbarer Fehler\0");
      break;
    }
  attroff(A_BLINK|A_REVERSE);
  refresh();
  endwin();
```

```
  exit (1);
  }

void GA_Benutz_ClearScreen()
  {
  clear();
  mvaddstr(0,5,"Genetische Algorithmen: Selbstorganisierende Datenstrukturen\0");
  refresh();
  }

void GA_Benutz_Vorinitial()
  {
  Korrektur_Ordnung = FALSCH;
  Korrektur_Topologie = FALSCH;
  MaxWesen = 20;
  MaxWesenHalbe = 10;
  MaxGeneration = 300;
  MaxKnoten = 20;
  M_Wahrschein = 0.0005;
  }

void GA_Benutz_Einstellung()
  {
  int Eingabe, weiter = FALSCH;
  float FEingabe, Rechfeld;
  long float LfEingabe;
  while (weiter == FALSCH)
    {
    GA_Benutz_ClearScreen();
    mvaddstr(1,50,"Eingabe : 0 = Nein, 1 = Ja\0");
    mvaddstr(3,5,"Systemparameter :\0");
    mvaddstr(4,5,"Maximale Generationen :\0");
    mvaddstr(4,45,"Anzahl der Knoten :\0");
    mvaddstr(5,5,"Mutationswahrscheinlichkeit :\0");
    mvaddstr(5,45,"Anzahl der Individuen :\0");
    mvaddstr(7,5,"Fitnessfunktionen:\0");
    mvaddstr(8,5,"Verteilung Fitness:\0");
    mvaddstr(21,5,"Korrektur :\0");
    mvaddstr(22,5,"Ordnung   :\0");
    mvaddstr(22,45,"Topologie :\0");
    mvaddstr(23,5,"O.K. ?\0");
    refresh();
/* Eingabe Anzahl der Generationen */
    do {
```

```
      move(4,34); scanw("%i",&Eingabe);
      printw("%i",Eingabe);
      } while ((Eingabe < 1) || (Eingabe > 999));
    MaxGeneration = Eingabe;
    refresh();
/* Eingabe Anzahl der Knoten */
    do {
      move(4,65); scanw("%i",&Eingabe);
      printw("%i",Eingabe);
      } while ((Eingabe < 10) || (Eingabe > 99));
    MaxKnoten = Eingabe;
    refresh();
/* Eingabe der Mutationswahrscheinlichkeit */
    do {
      move(5,34); scanw("%lf",&LfEingabe);
      printw("%lf",LfEingabe);
      } while ((LfEingabe >= 1.0) || (LfEingabe < 0.000005));
    M_Wahrschein = LfEingabe;
    refresh();
/* Eingabe der Populationsgroesse */
    do {
      move(5,69); scanw("%i",&Eingabe);
      printw("%i",Eingabe);
      } while ((Eingabe < 2) || (Eingabe > 50));
    MaxWesen = Eingabe;
    MaxWesenHalbe = MaxWesen / 2;
    refresh();
/* Verteilung der Fitnessfunktionen */
    do {
      move(8,34); scanw("%f",&FEingabe);
      Rechfeld = 1 - FEingabe;
      printw("%1.2f:%1.2f",FEingabe,Rechfeld);
      } while ((FEingabe > 1.0) || (FEingabe < 0.01));
    Fit_Verteil = FEingabe;
    refresh();
/* Eingabe Schalter fuer Ordnungs-Korrektur */
    do {
      move(22,17); scanw("%i",&Eingabe);
      printw("%i",Eingabe);
      } while ((Eingabe != 0) && (Eingabe != 1));
    if (Eingabe == 1)
      Korrektur_Ordnung = WAHR;
    else
      Korrektur_Ordnung = FALSCH;
```

```
    refresh();
/* Eingabe Schalter fuer Topologie-Korrektur */
    do {
      move(22,57); scanw("%i",&Eingabe);
      printw("%i",Eingabe);
      } while ((Eingabe != 0) && (Eingabe != 1));
    if (Eingabe == 1)
      Korrektur_Topologie = WAHR;
    else
      Korrektur_Topologie = FALSCH;
    refresh();
    do {
      move(23,12); scanw("%i",&Eingabe);
      printw("%i",Eingabe);
      } while ((Eingabe != 0) && (Eingabe != 1));
    if (Eingabe == 1)
      weiter = WAHR;
    refresh();
    }
  }

void GA_Benutz_Startmenu()
  {
  int Ergebnis, weiter;
  GA_Benutz_Vorinitial();
  weiter = FALSCH;
  while (weiter == FALSCH)
    {
    GA_Benutz_ClearScreen();
    Ergebnis = GA_domenu_main(&Menu);
    switch(Ergebnis)
      {
      case 0:  weiter = WAHR;
               break;
      case 1:  GA_Benutz_Einstellung();
               break;
      default: resetty();
               endwin();
               exit (0);
      }
    }
  }

void GA_Benutz_Statusausgabe(Generation,OGF,GF,PGF,BF, MaxWert,
```

```
                              MinWert, PMaxWert, PMinWert)
int Generation, OGF, GF, BF, MaxWert, MinWert;
double PGF, PMaxWert, PMinWert;
  {
  GA_Benutz_ClearScreen();
  mvaddstr(5,5,"Generation :\0");
  mvaddstr(7,5,"Erreichbare Fitness:\0");
  mvaddstr(8,5,"Erreichte   Fitness:\0");
  mvaddstr(8,45,"in Prozent :\0");
  mvaddstr(10,5,"Beste Einzelwerte :\0");
  mvaddstr(11,5,"im Ablauf            :\0");
  mvaddstr(12,5,"in dieser Generation :\0");
  mvaddstr(12,45,"prozentualer Anteil :\0");
  mvaddstr(14,5,"Schlechteste Einzelwerte :\0");
  mvaddstr(15,5,"in dieser Generation :\0");
  mvaddstr(15,45,"prozentualer Anteil :\0");
  move(5,20); printw("%i",Generation);
  move(7,26); printw("%i",OGF);
  move(8,26); printw("%i",GesamtFitness);
  move(8,60); printw("%2.2f",PGF);
  move(11,29); printw("%i",BF);
  move(12,29); printw("%i",MaxWert);
  move(12,67); printw("%2.2f",PMaxWert);
  move(15,29); printw("%i",MinWert);
  move(15,67); printw("%2.2f",PMinWert);
  refresh();
  }

void GA_Benutzerschnittstelle (Art,argc,argv)
int Art, argc;
char **argv;
  {
  char          *progname = "Genetic";
  char          *display_name = NULL;
  if (Art == FIRST_TIME)
    {
    initscr(); savetty(); cbreak(); nonl(); echo(); standend();
    GA_Benutz_Startmenu();
    GA_Benutz_ClearScreen();
    mvaddstr(10,12,"Initialisierung X-Windows:\0");
    refresh();
    XtToolkitInitialize();
    if ((display =XtOpenDisplay (NULL, NULL, argv[0], "Mwm", NULL,
                                 0, &argc, argv)) == NULL)
```

```
        GA_Benutz_error(1);
    GA_Basicwin_load_font (&font_info);
    mvaddstr(10,39,"O.K.\0");
    refresh();
    }
  if (Art == NORMAL_USE)
    {
    XUnloadFont (display, font_info->fid);
    XFreeGC (display, FS_Grafik);
    XCloseDisplay (display);
    Programmende = WAHR;
    resetty();
    endwin();
    }
  if (Art == FEHLER)
    {
    GA_Benutz_error(argc);
    }
  }
```

Zusatzmodul zum Einsatz von Curses

Dieses Programmstück ist [Hav87] angelehnt. Es dient zur Benutzung der Unix-Bibliothek CURSES zur Ausnutzung der Full-Screen Kapazitäten.

```
/*-------------------------------------------------------------------------*/
/*                                                                         */
/* Modul: domenu                                  Version 1.0              */
/*                                                                         */
/* Menusystem unter CURSES                                                 */
/*                                                                         */
/* Datum :                                                                 */
/*                                                                         */
/*-------------------------------------------------------------------------*/

#include <curses.h>
#include "domenu.h"

GA_domenu_isblank(s)
char *s;
{
  while(*s == ' ')
    s++;
```

```
  return *s == '\0' ? 1: 0;
}

GA_domenu_main(m)
struct menu *m;
{

  int option, lastoption, j, c, y, x;
  char *p;

  /* save current terminal state, then set required modes */
  savetty();

  cbreak(); nonl(); noecho(); standend();

  /* empty screen */
  clear();

  /* initialize keypad, TRUE is defined in curses.h */
  keypad(stdscr,TRUE);

  /*print centred title on line one */
  move(0, (COLS - strlen(m->m_title))/2);
  addstr(m->m_title);

  /* work out position for top left corner of menu */
  y = (LINES - m->m_height)/2 + 1;
  x = (COLS  - m->m_width)/2;

  /* display menu */
  for(j = 0; j < m->m_height; j++)
    mvaddstr( y+j, x, m->m_data[j]);

  /* initial values for cursor pos. and option setting */
  move( y, x);

  /* this assumes first line in menu isn't blank */
  lastoption = option = 0;

  for (;;) {

    /* remove highlight bar from last option */
    if(lastoption != option)
```

```
    mvaddstr( lastoption+y, x, m->m_data[lastoption]);

  /* put highlight bar on current option */
  standout();
  mvaddstr( option +y, x, m->m_data[option]);
  standend();
  move(option +y, x);

  /* save current option */
  lastoption = option;

  refresh();

  /* process input */
  switch( (c = getch()) ){

    case '\r':                  /* option selected, so return */
    case '\n':
      if(option < 0) {
        beep();
        break;
      }

      /* restore initial state and return */
      resetty();
      return option;

    case KEY_DOWN:              /* move current down if possible */
    case KEY_RIGHT:             /* or wrap around       */
      do {
         option = (++option < m->m_height) ? option: 0;
      }while( GA_domenu_isblank(m->m_data[option]));
      break;

    case KEY_UP:                /* move current line up or wrap */
    case KEY_LEFT:              /* around */
      do{
        option = (--option >= 0) ? option : m->m_height -1;
      }while( GA_domenu_isblank(m->m_data[option]));
    default:
      for(j = 0; j < m->m_height; j++){

        for(p = m->m_data[j]; *p == ' '; p++)
          ;
```

```
          if( *p == '\0')       /* blank line */
            continue;

          if( *p == c){
            option = j;
            break;
          }
        }
        if(j >= m->m_height)
          beep();
        break;

      }
    }
}
```

Die hinzugehörende Includedatei

```
/*menu.h -- contains definition of menu structure */

struct menu {
  int m_height;         /* menu height          */
  int m_width;          /* menu width           */
  char *m_title;        /* menu title           */
  char **m_data;        /* pointer to data      */
  };
```

Schnittstelle zum Zufallsgenerator

Wie im theoretischen Teil ausgeführt, bedeutet ein Zufallsgenerator mit zu kleiner Bandbreite eine Einschränkung in der Funktionalität. Für die Experimente wird ein Zufallsgenerator mit einer Bandbreite von 48 Bit benutzt.

```
/*-------------------------------------------------------------------------*/
/*                                                                         */
/* Modul: Zufall                                Version 1.0                */
/*                                                                         */
/* Schnittstelle zum Zufallsgenerator                                      */
/*                                                                         */
/* Datum :                                                                 */
/*                                                                         */
/*-------------------------------------------------------------------------*/
#include <sys/time.h>
```

```
#include "genetics.h"

long GA_Zufall_main()
  {
  return(lrand48());
  }
```

Literaturverzeichnis

[Arb81] M.A.Arbib, A.J.Kfoury, R.N.Moll: A Basis for Theoretical Computer Science
Springer-Verlag
1.Auflage, New York, 1981

[Bau84] F.L.Bauer, G.Goos: Informatik
Eine einführende Übersicht, Zweiter Teil
Springer-Verlag, Sammlung Informatik Band 91
3.Auflage, Berlin, 1984

[Bos86] K.Bosch: Elementare Einführung in die Wahrscheinlichkeitsrechnung
Serie vieweg studium / Basiswissen
Vieweg Verlag
5.Auflage, Braunschweig, 1986

[Bra91] T.Bratke, J.Gramatzki, T.Wagner, R.Welker: Entwicklung eines Programms zur Stundenplanbelegung mit Hilfe von genetischen Algorithmen
aus: [Joc91]

[Dav91] L.Davis: Handbook of Genetic Algorithms
Verlag Van Nostrand Reinhold
1.Auflage, New York, 1991

[Den77] E.Denert, R.Franck: Datenstrukturen
Wissenschaftsverlag, Bibliographisches Institut
1.Auflage, Berlin,München, 1977

[Dew86] A.K.Dewdney: Computer-Kurzweil
aus: Spektrum der Wissenschaft, Januar 1986
Orginal:
Exploring the field of genetic algorithms in a primordial computer sea full of flips
aus: Scientific American, Mai, 1985

[Dob84] P.Dobrinski, G.Krakau, A.Vogel: Physik für Ingenieure
B.G.Teubner Verlag
6.Auflage, Stuttgart, 1984

[Doe73] W.Dörfler, J.Mühlbacher: Graphentheorie für Informatiker
Verlag Walter de Gruyter
Berlin, 1973

[Doe77] W.Dörfler: Mathematik für Informatiker
Band 1: Finite Methoden und Algebra
Carl Hanser Verlag
1.Auflage, München, 1977

[Dor78] L.L.Dornhoff, F.E.Hohn: Applied Modern Algebra
Verlag MacMillan Publishing Co
1.Auflage, Urbana-Champaign, 1978

[Eng84] G.Engeln-Müllges, F.Reuther: Numerische Mathmatik für Ingenieure
Wissenschaftsverlag, Bibliographisches Institut
5.Auflage, Aachen, 1987

[Eve83] S.Evens: Graph Algorithms
Computer Science Press
2.Auflage, Haifa, 1983

[Gol89] D.E.Goldberg: Genetic Algorithms in Search, Optimization and Machine Learning
Verlag Addison-Wesley
1.Auflage, Ann Arbor, 1989

[Gri90] A.Grimm: Genetische Algorithmen
aus: [Joc91]

[Gri91] A.Grimm: Untersuchungen über den Einsatz von Genetischen Algorithmen bei topologischen Optimierungen
aus: [Joc91]

[Gue91] C.Günthner: Modifiziertes Tourenproblem am Beispiel einer Telefonkette
aus: [Joc91]

[Hav87] K.Haviland, B.Salama: Unix System Programming
Verlag Addison-Wesley
1.Auflage, Wokingham, 1987

[Hec76] J.Heck: Zufallsgraphen
aus: Graphen-Sprachen und Algorithmen auf Graphen
Hrsg. U.Pape

Serie Applied Computer Science,
Berichte zur praktischen Informatik Band 1
Hanser Verlag
1. Auflage, Berlin, 1976

[Hof85] D.R.Hofstadter: Gödel, Escher, Bach: ein endlos geflochtenes Band
Verlag Klett-Cotta
5.Auflage, Stuttgart, 1985

[Hol68] J.H.Holland: Hierarchical description of universal spaces and adaptive systems
aus: Technical Report ORA Projects 01252 and 08226
University of Michigan
Department of Computer and Communication Sciences
Ann Arbor, 1968

[Hop88] J.E.Hopcroft, J.D.Ullman: Einführung in die Automatentheorie, Formale Sprachen und Komplexitätstheorie
Verlag Addison-Wesley
1.Auflage, Bonn, 1988

[Joc88] F.Jochum: Datenbanksysteme, Folien zur Vorlesung
Gummersbach, 1988

[Joc91] F.Jochum (Hrsg.): Genetische Algorithmen - Prinzipien, Anwendungen, Experimente
Fachbereich Informatik
Fachhochschule Köln, Abt. Gummersbach
1.Auflage, Gummersbach, 1991

[Ker88] B.W.Kernigham, D.M.Ritchie: The C Programming Language
Verlag Prentice Hall
2.Auflage, New Jersey, 1988

[Kie90] B.Kießwetter: Experimentelle Untersuchung und Anwendung Genetischer Algorithmen in den Bereichen maschinelles Lernen und Optimierung
Diplomarbeit an der FH Köln, Abt. Gummersbach
Gummersbach, 1990

[Kla83] R.Klar: Digitale Rechenautomaten
Verlag de Gruyter, Sammlung Göschen
3.Auflage, Berlin, 1983

[Kno83] H.Knodel (Hrsg.): Linder Biologie
J.B.Metzlersche Verlagsbuchhandlung
19.Auflage, Stuttgart, 1983

[Knu68] D.E.Knuth: The Art of Computer Programming
Vol.1: Fundamental Algorithms
Verlag Addison-Wesley
1.Auflage, Reading, 1968

[Knu81] D.E.Knuth: The Art of Computer Programming
Vol.2: Seminumerical Algorithms
Verlag Addison-Wesley
2.Auflage, Reading, 1981

[Knu73] D.E.Knuth: The Art of Computer Programming
Vol.3: Sorting and Searching
Verlag Addison-Wesley
1.Auflage, Reading, 1973

[Kuh79] R.Kuhlen: Datenbasen, Datenbanken, Netzwerke
Praxis des Information Retrieval
Band 2: Konzepte von Datenbanken
Verlag K.G.Saur
1.Auflage, München, 1979

[Lip81] J.D.Lipson: Elements of Algebra and Algebraic Computing
Verlag Addison-Wesley
1.Auflage, Reading, 1981

[Luc91] C.B.Lucansius, M.J.J.Blommers, L.M.C.Buydens, G.Kateman: A Genetic Algorithm for Conformational Analysis of DNA
aus: [Dav91], Kap.18, S.251 ff.

[Maj76] M.E.Majster: Datenstrukturen und Operationen
aus: Graphen, Algorithmen, Datenstrukturen
Hrsg. H.Noltemeier
Serie Applied Computer Science,
Berichte zur praktischen Informatik Band 4
Hanser Verlag
1. Auflage, München, 1976

[Meh84] K.Mehlhorn: Data Structures and Algorithms 1:
Sorting and Searching
Serie EATCS Monographs on Theoretical Computer Science
Springer Verlag
1.Auflage, Berlin, 1984

[Meh88] K.Mehlhorn: Datenstrukturen und effiziente Algorithmen
Band 1: Sortieren und Suchen
B.G.Teubner Verlag

2.Auflage, Stuttgart, 1988
vgl.[Meh84]

[Moo74] A.M.Mood, F.A.Graybill, D.C.Boes: Introduction to the Theory of Statistics
Verlag McGraw-Hill
3. Auflage, Auckland, 1974

[Mue73] H.Müller-Merbach: Operations Research
Methoden und Modelle der Optimalplanung
Verlag Franz Vahlen
3. Auflage, München, 1973

[Mue78] H.Müller: Diskrete Algebraische Strukturen I
Arbeitsberichte des Instituts für Mathematische Maschinen und Datenverarbeitung (Informatik) an der Friedrich Alexander Universität Erlangen Nürnberg
Band 6, Nummer 5
4.Auflage, Erlangen, 1978

[Mue79] H.Müller: Diskrete Algebraische Strukturen II
Arbeitsberichte des Instituts für Mathematische Maschinen und Datenverarbeitung (Informatik) an der Friedrich Alexander Universität Erlangen Nürnberg
Band 8, Nummer 2
4.Auflage, Erlangen, 1979

[Pre89] W.H.Press, B.P.Flannery, et al.:Numerical Recipes
The Art of Scientific Computing
(Fortran Version)
Cambridge University Press
1.Auflage, Cambridge, 1989

[Sch83] G.Schlageter, W.Stucky: Datenbanksysteme:
Konzepte und Modelle
Teubner Studienbücher Informatik
2.Auflage, Hagen, 1983

[Tar91] O.Taraszow: Optimierung von Topologien verteilter Systeme und andere verwandte Strukturprobleme
Vortrag an der Fachhochschule Köln, Abt. Gummersbach
Gummersbach, 1991

[Ull83] J.D.Ullman: Principles of Database Systems
Computer Science Press
1.Auflage, Rockville, 1983

[Wed76] H.Wedekind, T.Härder: Datenbanksysteme II
Wissenschaftsverlag, Bibliographisches Institut
1.Auflage, Erlangen, 1976

[Wir81] N.Wirth: Compilerbau
Teubner Studienbücher Informatik
2.Auflage, Zürich, 1981

[Wir83] N.Wirth: Algorithmen und Datenstrukturen
Pascal- Version
Teubner Leitfäden und Monographien der Informatik
3.Auflage, Stuttgart, 1983

[Wir85] N.Wirth, K.Jensen: Pascal
User Manual and Report
Springer-Verlag
3.Auflage, New York, 1985

[XOp89] X/Open Gruppe: X/Open Portability Guide, Stand 1988
Band 1: XSI Commands and Utilities
Band 2: XSI System Interfaces and Headers
Band 3: XSI Supplementary Definitions
Band 4: Programming Languages
Band 6: Window Management
Band 7: Networking Services
Verlag Prentice Hall
1.Auflage, Englewood Cliffs, 1989