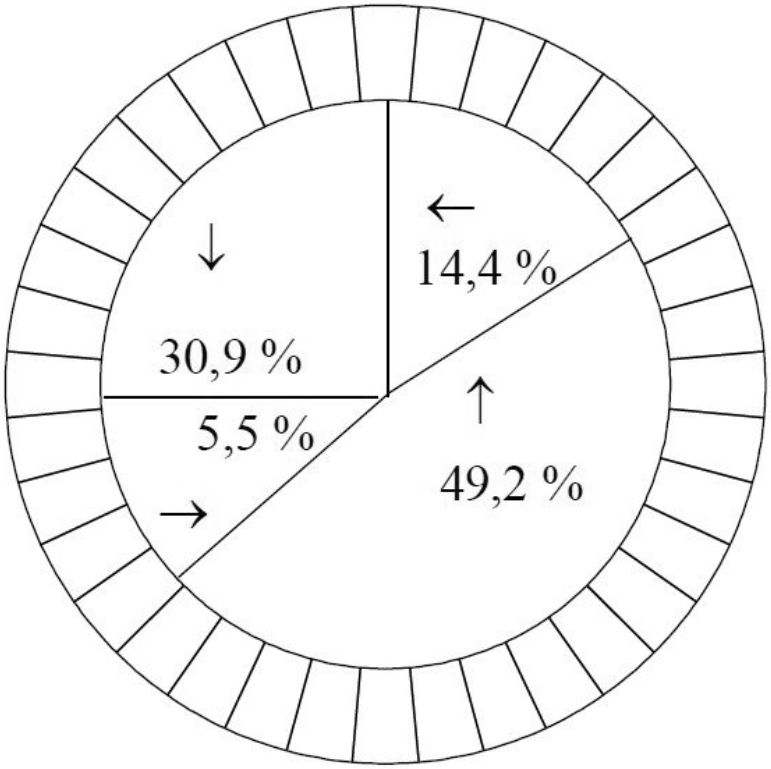
FORTSCHRITTSBERICHT
SIMULATION FBS 2



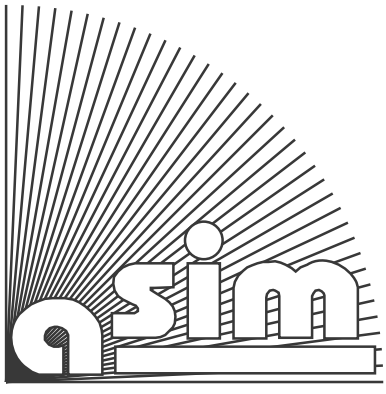
Manfred Salzmann

Genetische Algorithmen in diskreter Simulation

ARGESIM

ISBN Ebook 978-3-903347-02-1
ISBN Print 978-3-901608-52-0

DOI: 10.11128 / fbs.02


ARGESIM

Fortschrittsberichte Simulation

FBS Band 2

Herausgegeben von **ASIM**
Arbeitsgemeinschaft **Sim**ulation, Fachausschuß 4.5 der GI

Manfred Salzmann

Genetische Algorithmen in diskreter Simulation

ARGESIM / ASIM – Verlag, Wien, 1996
ISBN Print 978-3-901608-52-0

Ebook Reprint 2020
ISBN Ebook 978-3-903347-02-1
DOI: 10.11128/fbs.02

ASIM Fortschrittsberichte Simulation / ARGESIM Reports

Herausgegeben von **ASIM**, Arbeitsgemeinschaft Simulation, Fachausschuß 4.5 der **GI** und der **ARGESIM**

Betreuer der Reihe:

Prof. Dr. G. Kampe (ASIM)
Fachhochschule Esslingen
Flandernstraße 101, D-73732 Esslingen
Tel: +49-711-397-3741, Fax: --397-3763
Email: kampe@ti.fht-esslingen.de

Prof. Dr. D.P.F. Möller (ASIM)
Inst. F. Informatik, TU Clausthal-Zellerfeld
Erzstraße 1, D-38678 Clausthal-Zellerfeld
Tel: +49-5323-72-2404, Fax: --72-3572,
moeller@vax.in.tu-clausthal.de

Prof. Dr. F. Breitenecker (ARGESIM / ASIM)
Abt. Simulationstechnik, Technische Universität Wien
Wiedner Hauptstraße 8 - 10, A - 1040 Wien
Tel: +43-1-58801-5374, Fax: +43-1-5874211,
Email: Felix.Breitenecker@tuwien.ac.at

FBS Band 2

Titel: Genetische Algorithmen in diskreter Simulation

Autor: Dipl.-Ing. Dr. M. Salzmann
Rötzergasse 22/1a
A - 1170 Wien
Tel: +43-1-4057880
Email: msalz@osiris.tuwien.ac.at

Begutachter des Bandes:

Prof. Dr. F. Breitenecker TU Wien, Prof. Dr. D.P.F. Möller Univ. Hamburg

ARGESIM / ASIM – Verlag, Wien, 1996
ISBN Print 978-3-901608-52-0

Ebook Reprint 2020
ISBN Ebook 978-3-903347-02-1

DOI: 10.11128/fbs.02

Dissertation

Genetische Algorithmen in diskreter Simulation

ausgeführt zum Zwecke der Erlangung des akademischen Grades eines Doktors der technischen Wissenschaften

eingereicht an der Technischen Universität Wien

Technisch-Naturwissenschaftliche Fakultät

von

Dipl.-Ing. Manfred Salzmann

Lorenz-Müller-Gasse 1/138

A-1200 Wien

Matrikelnummer: 8625638

geboren: 16. April 1968 in A-3340 Waidhofen/Ybbs

Wien, im April 1995

Danksagung & Widmung

Zu Beginn von diversen Diplomarbeiten, Dissertationen, Büchern und anderen Publikationen steht an dieser Stelle eine Dankesadresse, gefolgt von einer Widmung (oder umgekehrt, wir wollen ja schließlich nicht kleinlich sein). Das ist alles ganz gut und schön, und wenn man bedenkt, daß diese Seiten zu den meistgelesenen von vielen Arbeiten zählen, auch sehr wichtig.

Da ich nun damit rechnen muß, daß einige von Ihnen, werte Leser, diese Arbeit nicht weiterlesen werden (aus welchen Gründen auch immer), habe ich mir gedacht, ich biete Ihnen zu Beginn etwas Besonderes! Doch was?

Bei der Danksagung dachte ich daran, eine alphabetische Reihung aller Personen vorzunehmen, die in irgendeiner (sagen wir mal positiver) Weise auf mich und meinen Lebensweg Einfluß genommen haben. Sie, liebe Leser, werden wahrscheinlich schon ahnen, wo das Problem liegt: Wehe man vergißt jemanden!!!

Bei der Widmung tauchen natürlich auch Probleme auf, denn was ist, wenn sich die Widmungsträger im Laufe der Zeit als für die Widmung unwürdig erweisen? Dann will man unter Umständen seine eigene Dissertation nicht mehr aufschlagen und muß doch glatt eine neue schreiben! Kurzfristig dachte ich daran, die Dissertation der Menschheit zu widmen oder dem wissenschaftlichen Fortschritt, aber ersteres ist doch zu pathetisch und zweiteres habe ich ja schon anläßlich meiner Sponsion gelobt (wäre also ein reduntanter Vorgang).

Ich stehe also vor einem Riesendilemma, daß ich vielleicht dadurch lösen könnte, demjenigen zu danken, ohne dem diese Dissertation nie zustande gekommen wäre, nämlich mir, und sie demjenigen zu widmen, dem sie (wahrscheinlich) am meisten nützt, nämlich mir. Aber wie Sie sicher wissen, geneigte (nicht im mathematischen Sinn!) Leser, gehört sich das halt nicht.

Da ich mich nun langsam dem Ende dieser Seite nähere und ich Sie, verehrte Leser, nicht länger auf die Folter spannen will, was so in diesem Werk auf Sie wartet, mache ich es kurz und schmerzlos:

DANKE!!![1]

[1] an all jene, die sich betroffen fühlen und sich auch betroffen fühlen sollten!!!

Kurzfassung

In der vorliegenden Dissertation wird in Anlehnung an die Theorie der naturanalogen Verfahren eine Portierung von Genetischen Algorithmen auf den Bereich der diskreten Computersimulation unternommen.

Im Bereich der Optimierung mit Hilfe von Computern gibt es verschiedene Ansätze. Eine große Gruppe bilden die Monte-Carlo-Methoden, bei denen der gegebene Suchraum stochastisch erforscht wird. Dem gegenüber stehen die Methoden, die sich deterministischer Regeln bedienen und keinerlei Zufallselement beinhalten. Zu den Verfahren, die die Vorteile dieser beiden Bereiche verbinden, zählt unter anderem die Gruppe der naturanalogen Verfahren.

Zu Beginn dieser Arbeit werden unter Berücksichtigung der umfangreichen Literatur und speziell unter Zuhilfenahme der auf John Holland zurückgehenden Theorie der Genetic Algorithms grundsäzliche Eigenschaften der Genetischen Algorithmen dargelegt und anhand eines einfachen Beispiels illustriert.

Anschließend erfolgt eine kurze Einführung in die Arbeitsweise und Problematik der diskreten Simulation, gefolgt von einer eingehenden Analyse der verschiedenen Möglichkeiten der Genetischen Algorithmen, speziell in Hinsicht auf die Verwendung fortgeschrittener Operatoren und Methoden.

Als nächstes werden die Besonderheiten der Anwendung von Genetischen Algorithmen auf Beispiele diskreter Simulation beleuchtet. Darauf aufbauend werden Risken und Probleme diskutiert. Speziell werden auf die Möglichkeiten und Chancen, die sich aus diesen Besonderheiten ergeben, eingegangen.

Weiters werden Implementierungen unter Zuhilfenahme dreier verschiedener Systeme vorgestellt. Als Simulationssprachen gelangen der auf Petrinetzen basierende und unter Microsoft Windows™ laufende Simulator D_SIM, das unter Microsoft DOS™ laufende GPSS/H™ und das unter Microsoft Windows™ laufende MicroSaint™ zum Einsatz. Diese Implementierungen werden durch Anwendungsbeispiele aus dem Bereich der Fertigungstechnik validiert und verglichen.

Abstract

In this dissertation an application of Genetic Algorithms in the domain of discrete event simulation is described in regard to the theory of Evolutionary Algorithms.

In the field of computer optimization there are many different ways of solving a given problem. These different ways can be divided in some main groups such as Monte-Carlo-methods and deterministic methods. Monte-Carlo-methods analyze the search space stochastically whereas the deterministic methods have fixed rules and don't make any use of random search techniques. One group of applications which tries to combine the advantages of both previous mentioned methods is the group of Evolutionary Algorithms.

Basics of Genetic Algorithms used in this dissertation are explained in consideration of the great number of references and especially by means of the theory of classic Genetic Algorithms based on John Holland. Then a short introduction of the theory and the problems of discrete event simulation is presented. This leads to an extensive discussion of further possibilities of Genetic Algorithms, especially with the use of advanced operators and methods.

After that the special issues that arise through the combination of Genetic Algorithms with discrete event simulation are discussed. Risks and problems are shown and ways to avoid these problems are given.

Furthermore the presented theory is implemented in three different simulation systems. The first simulation language that is used for this purpose is D_SIM which is based on the theory of Petri nets and works under Microsoft Windows™. The second one is GPSS/H™ which runs under Microsoft DOS™ and the third is MicroSaint™ which requires Microsoft Windows™. These implementations are validated and compared by means of examples taken from the field of manufacturing.

Inhaltsverzeichnis

Einleitung

Seit jeher versuchen die Menschen die Natur zu analysieren und sich die daraus ergebenden Erkenntnisse zunutze zu machen. Durch Beobachten von Vorgängen in der Natur und geeignetes Abstrahieren und Transferieren auf andere Gebiete gelangen im Verlaufe der Geschichte schon einige bemerkenswerte Erfolge, speziell der Bereich der Technik wäre ohne diese „Knowledge-Transfers“ um einige erfolgreiche Entwicklungen ärmer. Eines der augenscheinlichsten Beispiele ist der Bereich der Flugtechnik. Beginnend mit den - zugegebenermaßen nicht allzu wissenschaftlichen - Versuchen eines Ikarus über die Erfolge eines Otto Lilienthals hin zu den modernen Errungenschaften in dieser Sparte ist doch augenscheinlich das Vorbild der Vögel dafür verantwortlich, daß die meisten Fluggeräte Flügel besitzen.

Dies soll allerdings nicht in der Richtung mißverstanden werden, in der Natur erfolgreiche Konzepte bedingungslos auf Bereiche anzuwenden, ohne sich zu vergewissern, ob die ausgewählten Methoden in diesen Bereichen auch wirklich so funktionieren, wie man sich das vorstellt.

Die Natur stand auch John Holland als Ideenlieferant zur Seite, als er 1975 in einer Veröffentlichung (siehe /HOL75/) naturanaloge Verfahren zur Adaption von Systemen in technischen Problemen vorstellte. Diese Verfahren ließen sich auch zur Optimierung einsetzen, wobei er nicht der erste war, der versuchte, naturanaloge Verfahren für technische Anwendungen nutzbringend zu verwenden, da es schon in den 50er und 60er Jahren Ansätze und Theorien in dieser Richtung gab (vgl. z.B. /REC73/), aber er schaffte es, diesen Zweig der Wissenschaft - basierend auf seiner Veröffentlichung - in den USA - und im englischsprachigen Raum allgemein - zu etablieren. Dadurch wurde er zum Begründer der Genetic Algorithms, die, um es noch einmal zu betonen, von ihm nie als Optimierungswerkzeug entwickelt wurden.

Die Genetic Algorithms gerieten in der darauffolgenden Zeit etwas in Vergessenheit, aber mit wachsender Leistungsfähigkeit der Computer kam es zu einer Wiederentdeckung dieser Verfahren in den verschiedensten Sparten der wissenschaftlichen Forschung. Speziell David Goldberg hat 1989 mit einer

umfassenden Darstellung (vgl. /GOL89/) dieser Theorie einen großen Anteil am derzeitigen Boom.

Wie in vielen anderen Bereichen der Wissenschaft gibt es auch im Bereich der naturanalogen Verfahren verschiedene Ausprägungsformen, abhängig vom Anwendungsgebiet und nicht zuletzt von der Provenience der betreffenden Wissenschaftler. So ist im deutschen Sprachraum die Theorie der Evolutionsstrategien sehr verbreitet (vgl. /REC89a/), wo hingegen diese im englisch-amerikanischen Sprachraum kaum vertreten sind. Neben diesen Verfahren gibt es noch eine Reihe anderer Theorien, wie etwa Genetic Programming (GP, vgl. /KOZ92/) oder Simulated Annealing (SA, vgl. /DAV87/). Auf diese oder verwandte Verfahren wird in dieser Dissertation bei Bedarf eingegangen.

Die vorliegende Arbeit lehnt sich in der Diktion zwar an die Diktion der Genetic Algorithms an, ist aber nicht als eine bloße Übertragung der Genetic Algorithms ins Deutsche zu sehen, da der Bereich der diskreten Simulation zu verschieden von herkömmlichen Anwendungsgebieten der Genetic Algorithms ist. Aus diesem Grund ist in dieser Arbeit der Begriff „Genetische Algorithmen“ nicht als Übersetzung des Begriffes „Genetic Algorithms“ zu verstehen, sondern vielmehr als eigenständiger Begriff, der allerdings ein großes Naheverhältnis zu „Genetic Algorithms“ besitzt.

Für umfassende Darstellungen der Theorien naturanaloger Verfahren sei auf das Literaturverzeichnis verwiesen, wo unter anderem /DAV91a/, /GOL89/, /HOL75/, /KOZ92b/, /NIS94/ und /REC73/ hervorragende Dienste leisten.

Einer Entwicklung unserer Zeit, und damit ist speziell das Computerzeitalter gemeint, Rechnung tragend, wurden in das Literaturverzeichnis nicht nur Bücher, Zeitschriften und Artikel aufgenommen, sondern auch Mailing-Lists, Newsgroups, WWW-Homepages und FTP-Adressen (siehe die Einträge der Form */COMP_**/*, vgl. /HEBE94/). Dort findet der interessierte Leser und/oder Anwender üblicherweise die neuesten Forschungsergebnisse und/oder kann an interessanten Diskussionen teilnehmen. Es werden dort auch Plattformen geboten, um eigene Ideen einer Begutachtung zu unterwerfen, und unter Umständen manche existierende Fehler vor einer offiziellen Veröffentlichung einer Arbeit noch auszumerzen.

Genetische Algorithmen

Genetische Algorithmen (GA) unterscheiden sich von anderen Optimierungs- und Suchverfahren durch einige wichtige Punkte. Im Mittelpunkt jedes GA steht eine gewisse Anzahl von Lösungen, die in Anlehnung an die Natur Population genannt wird. Es wird also nicht nur eine Ausgangslösung herangezogen und versucht mit irgendwelchen Mitteln zu verbessern, sondern es wird vielmehr eine Menge von Ausgangslösungen betrachtet, die erst durch verschiedenen Gütekriterien und durch geeignetes Zusammenspiel im Zuge des Optimierungsprozesses ihre ganze spezifische Macht entfalten. Diese spezielle Eigenschaft sorgt einerseits dafür, daß oft noch brauchbare Ergebnisse erzielt werden, wo andere Verfahren versagen, andererseits führt es auch zu aufwendigen In-Evidenz-halten von Lösungen, die retrospektiv betrachtet wenig zum Erfolg des Verfahrens beitragen.

Die angesprochene Population wird in jeder Generation durch neue Lösungen (= Nachkommen) bereichert, um daraufhin eine gewisse Auswahl durchführen zu können. Dadurch wird ähnlich wie in der Natur eine Selektion erreicht, die bessere Lösungen begünstigen soll. Schlechte Lösungen (= nicht gut angepaßte Lebewesen) geraten dabei ins Hintertreffen und sterben langsam aus.

Interessant ist weiters, daß Genetische Algorithmen sich ähnlich der Vorgehensweise bei Monte-Carlo-Verfahren stochastischer Elemente bedienen, aber sehr wohl auch ähnlich diverser Gradientenmethoden vorgegebene Optimierungsrichtlinien beachten. Damit kann bei geeigneter Implementierung eine Verknüpfung der Vorteile beider Optimierungsarten erreicht werden, da durch GA sowohl einerseits gewährleistet wird, daß jeder Punkt des Suchraums zumindest eine theoretische Chance hat, gefunden zu werden, als auch andererseits eine Verarbeitung der Information der schon gefundenen Lösungen dahin gehend möglich ist, daß eine Art gerichteter Suche erfolgen kann.

Genetische Algorithmen sind natürlich nicht für jede Art von Optimierung gleich gut geeignet. Es läßt sich hier in Anlehnung an /GOL89/ sagen, daß der beste Wirkungsgrad bei Problemen erreicht wird, die eine Zielfunktion aufweisen, die nur einen gewissen Grad an Homogenität besitzen.

Um das soeben Gesagte etwas zu veranschaulichen, stellt man sich die Zielfunktion als mathematische Funktion vor. Geht man der Einfachheit halber davon aus, daß die Zielfunktion von zwei Variablen abhängig ist, ergibt sich als Graph der Funktion eine „Landschaft“ im dreidimensionalen Raum mit den Werten der Zielfunktion als Höhenmeter.

Falls diese Landschaft anschaulich gesprochen eher kontinuierlich ansteigende Hügeln aufweist, sind traditionelle Gradientenmethoden sicher schneller als Genetische Algorithmen, da sie nicht unnötigen Ballast in Form von zusätzlichen Lösungen mitschleppen, sondern direkt dem steilsten Anstieg folgen und somit schneller den Gipfel finden. Bei Landschaften, die durch ständigen Wechsel von An- und Abstiegen charakterisiert sind, würden Gradientenmethoden zu oft in lokalen Maxima hängenbleiben. Bei Sprungstellen versagen diese Methoden meistens ohnehin.

Es kann allerdings sein, daß die Landschaft so unregelmäßig und zerklüftet ist, daß eine herkömmliche Monte-Carlo-Methode schneller zum Ziel führt, wenn man bedenkt, daß für die Implementierung eines Genetischen Algorithmus im allgemeinen mehr Aufwand getrieben werden muß.

Generell läßt sich sagen, daß der große Vorteil der Genetischen Algorithmen darin liegt, universell einsetzbar zu sein, ohne eine zu komplizierte Struktur aufzuweisen. Ein GA benötigt zur Optimierung keine weitergehenden Informationen als die Zielfunktion des betrachteten Problems. Es ist also zum Beispiel nicht notwendig, Ableitungen oder ähnliches zu kennen.

Ein einfacher Genetischer Algorithmus

Zur Erklärung der Theorie der Genetischen Algorithmen wird im folgenden die Struktur eines einfachen GA erläutert, wobei sich das „einfach“ auf die Anzahl der verwendeten genetischen Operatoren bezieht.

Kodierung

Um das Funktionieren eines GA zu verstehen, ist es notwendig, sich über die Form der Verschlüsselung eines gegebenen Problems Gedanken zu machen. Ausgehend von einer bestimmten Optimierungsaufgabenstellung ist als erstes zu beachten, welche

Eingangsdaten variabel sind und daher für den Optimierungsprozeß in Frage kommen. Diese Variablen sollten möglichst in Parameterform vorliegen, anderenfalls muß man sich um eine geeignete Repräsentationsform des Problems kümmern.

Auf die Fragestellung, welchen Umfang das zur Kodierung herangezogene Alphabet am besten besitzen soll, wird an anderer Stelle eingegangen, der Einfachheit wegen wird hier angenommen, daß ein Binäralphabet herangezogen wird. Bei *n* Parametern, die im Falle der Kodierung im Dualsystem jeweils die Länge l_k besitzen, erhält man also eine Gesamtstringlänge $\sum_{k=1}^{n} l_k$. Dieser String repräsentiert nun die zu lösende Aufgabe und man spricht in Anlehnung an die Natur von einem Chromosom. Das Äquivalent zu einem Parameter ist in der Natur ein Gen und daher spricht man auch im Bereich der GA des öfteren von einzelnen Genen, die ein Chromosom bilden. Wenn bei der hier gewählten Repräsentationsform ein Parameter mehr als zwei mögliche Zustände besitzt, setzt sich das entsprechende Gen aus mehreren sogenannten Allelen zusammen, die von Elementen des verwendenten Alphabets gebildet werden.

Weitere wichtige Anlehnungen an die Natur treten in diesem Fall bei der Unterscheidung des Genotyps und des Phänotyps auf. Der oben erwähnte String im Binärsystem entspricht dem Genotyp eines Problems, wo hingegen die Werte der Parameter, welche hinter dieser Kodierung stehen, als Phänotyp bezeichnet werden. Vereinfachend kann man sagen, daß der Phänotyp die konkrete Entsprechung eines Genotyps ist.

Zielfunktion

Grundlage jeglicher Optimierung ist die Vorgabe einer Zielfunktion, deren Maximierung oder Minimierung beabsichtigt ist. Dies Funktion ist durch die Problemstellung determiniert und im Falle der Genetischen Algorithmen ist darauf zu achten, einen eindimensionalen Wert zu erhalten, d.h. es werden nur Funktionen zugelassen, die als Ergebnis einen reellwertigen Ausdruck liefern.

Aus obigen Betrachtungen geht hervor, daß bei dieser Art der Optimierung sicher eine wichtige Aufgabe der geeigneten Bestimmung der Zielfunktion zukommt, speziell wenn es sich um Probleme handelt, wo gleichzeitig zwei oder mehrere Zielwerte, bzw.

deren Zusammenspiel, verbessert werden soll. In diesem Fall ist es notwendig, bei der Formulierung der Zielfunktion auf eine ausgewogene Bewertung aller relevanten Kenngrößen zu achten und eine geeignete Formulierung zu erreichen. Dies wird in den konkreten Anwendungen dem Benutzer solcher Routinen überantwortet werden müssen.

Um bei den Analogien zur Natur zu bleiben, spricht man bei der Zielfunktion von einer Fitneßfunktion, um darauf hinzuweisen, daß Lösungen mit einem besseren Ergebnis (= höherer Fitneß) bessere Chancen haben, zu überleben.

Reproduktion

Nachdem bei jedem Element einer Population die Fitneß bestimmt worden ist, wird entsprechend dieser Fitneß eine Auswahl dahin gehend getroffen, daß Elemente bestimmt werden, die ihre „Eigenschaften“ an die nächste Generation weitergeben dürfen. Dies soll natürlich ähnlich wie in der Natur geschehen, was bedeutet, daß gute Lösungen bessere Chancen haben sollen als schlechte. Allerdings soll auch das schlechteste Element zumindest eine theoretische Chance besitzen, Informationen weiterzugeben. Dies ist wichtig, weil es möglich ist, daß schlechte Lösungen einen guten Teilstring besitzen, der durch das später beschriebene Crossover-verfahren erst zur Geltung kommt.

Für den Prozeß der Reproduktion kommen mehrere Verfahren in Frage, für diesen Abschnitt wollen wir kurz die sogenannte Rouletteradmethode erläutern. Bei diesem Verfahren wird jedem Element ein Wahrscheinlichkeitswert direkt proportional zur Fitneß zugewiesen. Dieser Wert steht für die Wahrscheinlichkeit, zur Reproduktion herangezogen zu werden. Der Name dieser Methode kommt daher, daß man sich alle Elemente einer Population auf einem Rouletterad angeordnet denkt und zwar soll jedes Element ein der Höhe der Fitneß entsprechend großes Feld erhalten. Dann denkt man sich eine Roulettekugel geworfen, was klarerweise guten Lösungen bessere Chancen einräumt, ausgewählt zu werden.

Für den eigentlichen Vorgang der Reproduktion sind unzählige Operatoren entwickelt worden, die auf die oft einzigartigen Kriterien der Aufgabenstellung genau abgestimmt worden sind. Von all diesen wollen wir uns an dieser Stelle zwei der einfachsten zuwenden, nämlich dem Einpunkt-crossover und der Mutation.

Beim Einpunkt-crossover wählt man zwei Elemente der Elterngeneration aus, bestimmt zufällig einen Punkt, an dem beide getrennt werden, und vertauscht den ersten Teilstring des ersten Elements mit dem ersten Teilstring des zweiten. Seien zum Beispiel folgende zwei Lösungen ausgewählt worden:

Lösung 1: $(a_1,a_2,a_3,...,a_n)$

Lösung 2: $(b_1,b_2,b_3,...,b_n)$

Nach Auswahl eines Trennpunktes ergibt sich dann folgendes Bild:

Lösung 1: $(a_1,a_2,a_3,...,a_s)(a_{s+1},,...,a_l)$

Lösung 2: $(b_1,b_2,b_3,...,b_s)(b_{s+1},,...,b_l{}_l)$

Nach Vertauschen erhält man folgende neue Lösungen:

neue Lösung 1: $(a_1,a_2,a_3,...,a_s,b_{s+1},,...,b_l)$

neue Lösung 2: $(b_1,b_2,b_3,...,b_s,a_{s+1},,...,a_l)$

Bei der Mutation wird ein Element zufällig ausgewählt und an einer (selten an mehreren) zufällig bestimmten Stelle verändert. Im Fall eines Binäralphabets ist dies sehr leicht zu verwirklichen, da einfach das entsprechende Bit invertiert wird. Sei das Element $(a_1,a_2,a_3,...,a_s,,...,a_l)$ zur Mutation an der Stelle *s* bestimmt worden, so erhält man dann das neue Element $(a_1,a_2,a_3,...,k,...,a_l)$.

Nachdem neue Elemente erzeugt worden sind, ist es notwendig sich Gedanken zu machen, welche Eingang in die neue Population finden sollen. Hier gibt es wiederum unterschiedliche Ansichten und oft je nach Anwendung unterschiedliche Praktiken. Eine Möglichkeit in diesem Zusammenhang ist jedenfalls die, einfach nur die soeben erzeugten Elemente zu berücksichtigen und die Elemente der Elternpopulation zu vergessen. Dabei muß natürlich Sorge getragen werden, daß eine ausreichende Anzahl von „Kindern“ (im Englischen oft als „Offsprings“ bezeichnet) generiert worden ist.

Abbruchkriterium

Wie jedes Verfahren benötigt auch ein Genetischer Algorithmus eine Vorgabe, wann das Verfahren abzubrechen ist. Dabei spricht man von einem sogenannten Abbruchkriterium, welches oft nicht so leicht zu finden ist, da man selten weiß, wie

das globale Optimum aussieht. Allerdings sind oft durch die Rahmenbedingungen (CPU-Zeit, Projektzeit, etc.) oder durch offensichtliches Verhalten des Prozesses (keine Verbesserung der Ergebnisse über eine größere Anzahl von Populationen) gewisse Richtlinien abgesteckt. Die einfachste Abbruchbedingung ist natürlich die, bei der nach einer fixen Anzahl von Populationen der GA beendet wird. Als weiteres Kriterium ist auch das Erreichen eines gewissen vorgegebenen Wertes der Zielfunktion denkbar.

Ablaufschema

Ein Ablaufschema für einen einfachen Genetischen Algorithmus kann daher folgendes Aussehen in Pseudocomputercode besitzen (vgl. /BBM93/):

```
BEGIN *GA*
        generiere Anfangspopulation
        ermittle für jedes Element die Fitneß
        WHILE NOT fertig DO
        BEGIN *produziere neue Population*
                FOR Populationsgröße/2 DO
                BEGIN *Reproduktionszyklus*
                        mutiere ein Individuum mit gegebener Wahrscheinlichkeit
                        bestimme gegebenenfalls Fitneß des mutierten Individuums
                        füge mutiertes Individuum in neue Population ein
                        wähle zwei Individuen aus der Elterngeneration aus
                                *unter Berücksichtigung der individuellen Fitneß*
                        rekombiniere diese zwei Eltern zu zwei neuen Kindern
                        bestimme die Fitneß der  zwei Kinder
                        füge die zwei Kinder in neue Population ein
                END
                IF Abbruchkriterium ist erfüllt THEN
                        fertig := TRUE
        END
END
```

Ein einfaches Beispiel

Zur Erläuterung des bisher gesagten soll folgendes einfaches Beispiel dienen (vgl. /GOL89/):

Wir suchen das Maximum der Funktion $f(x) = x^2$ im Definitionsbereich [0,31], wobei nur ganze Zahlen berücksichtigt werden. Die Zahlen werden binär dargestellt, daher werden Strings mit einer Länge von 5 Bits benötigt. Die Größe der Population wird während der Optimierung konstant auf 4 gesetzt. Als genetische Operatoren

werden Mutation und Crossover verwendet und die Auswahl erfolgt durch die Rouletteradmethode.

Zuerst wird die Ausgangspopulation durch zufällige Bestimmung der Bits erzeugt. Als Fitneßfunktion verwenden wir einfach die Funktion, die wir optimieren wollen ($f(x) = x^2$), wodurch sich für die Initialpopulation folgendes Bild ergibt:

Nummer	String	*x*-Wert	Fitneß (x^2)	% der Summe
1	01101	13	169	14,4
2	11000	24	576	49,2
3	01000	8	64	5,5
4	10011	19	361	30,9
Summe			1170	100,0
Mittelwert			293	
Maximum			575	

Nachdem die Anfangspopulation bestimmt wurde, werden die Strings ausgewählt, welche für die Reproduktion in Betracht gezogen werden. Das geschieht mittels eingangs erwähnter Rouletteradmethode über den Anteil, den die einzelnen Strings an der Summe der Fitneß haben. Strings mit einem guten Wert der Fitneßfunktion haben also eine größere Chance, als Eltern der nächsten Generation verwendet zu werden.

Durch dieses Auswahlverfahren ergibt sich, daß in der nächsten Generation eine Kopie des Strings Nummer 1, eine Kopie des Strings Nummer 4 und zwei Kopien des Strings Nummer 2 verwendet werden. String Nummer 3 wird für die nächste Generation nicht mehr verwendet.

Es werden nun für die einzelnen Stringpaare Crossoverstellen mittels einer Zufallszahl bestimmt. Weiters wählen wir die Wahrscheinlichkeit einer Mutation mit 0,001, was bedeutet, daß pro Generation durchschnittlich *4*5*0,001=0,02* Bits mutiert werden. Durch diese Annahmen erhält man folgendes Ergebnis:

Ausgewählte Eltern	Partner (zufällig ausgewählt)	Stelle für Crossover	Element der neuen Population	*x* Wert	*f(x)*
01101	2	3	01000	8	64
11000	1	3	11101	29	841
11000	4	2	11011	27	729
10011	3	2	10000	16	256
Summe					1890
Mittelwert					472,5
Maximum					841

In diesem Fall sind keine Mutationen aufgetreten. Wie in der Tabelle ersichtlich ist, hat sich, trotz der zufälligen Auswahl der Partner und der Stellen an denen das Crossover durchgeführt worden ist, sowohl eine deutliche Verbesserung des Maximums von *576* auf *841*, als auch eine Verbesserung des Mittelwertes von *293* auf *472,5* ergeben. Zwei weitere Iterationen bewirken folgendes:

Ausgewählte Eltern	Partner	Elem. d. neuen Pop.	*x* Wert	*f(x)*
11101	4	11000	24	576
11101	3	11101	29	841
11011	2	11011	27	729
10000	1	10101	21	441
11000	3	11101	29	841
11101	4	11111	31	961
11101	1	11010	26	676

11011	2	11001	25	625

Der höchste hier vorkommende Wert ist *31*, was auch wirklich das Maximum darstellt.

Obwohl die Suche durch zufällige Auswahl gesteuert wird, ist der Ablauf der Suche selbst nicht völlig willkürlich. Durch Auswahl von immer besseren Lösungen als Basis für neue Lösungen wandert der Algorithmus in Richtung der optimalen Lösung.

Konvergenz von GA

Im Laufe der Entwicklungen der verschiedenen Anwendungen der Genetischen Algorithmen ist immer wieder die Frage in den Vordergrund getreten, warum und ob die GA konvergieren, was heißt, daß sich gegen Ende des Verfahrens in einer Population hauptsächlich Vertreter eines Genotyps befinden. Es sei an dieser Stelle zu Bedenken gegeben, daß es für die Zwecke der Optimierung im Prinzip ausreicht, wenn am Ende eines GA eine gute - wenn möglich die beste - Lösung bekannt ist, was nicht bedeuten muß, daß die meisten der zu diesem Zeitpunkt noch vorhandenen Lösungen die gleiche oder eine ähnliche Form wie diese Lösung besitzen müssen.

Zum Themenkomplex der Konvergenz gibt es eine Reihe von Untersuchungen und genau so eine Reihe von Theorien. Zu einer allgemein akzeptierten geschlossenen Theorie ist es bis jetzt noch nicht gekommen (vgl. /BBM93/), allerdings sollten folgende Ansätze hinreichend für die Verwendung von Genetischen Algorithmen sprechen. Außerdem ist ein Hauptargument für die Verwendung von GA wohl die große Anzahl von erfolgreichen Implementierungen auf verschiedensten Gebieten der Forschung. Obwohl dieses Argument rein induktiven Charakter besitzt, sollte man es doch nicht zu leicht vom Tisch wischen.

Schema-Theorem

Als erster versucht Holland (vgl. /HOL75/, /GOL89/, /BBM93/) 1975 eine Erklärung zu finden, warum „seine“ Genetischen Algorithmen funktionieren. Als Basis seines Schematheorems wählt er ein Alphabet der Form *{0,1,#}* mit dem er Untersuchungen für Problemstellungen, die im Dualsystem implementiert sind, anstellt.

Er bezeichnet als ein Schema eine Abfolge von Elementen des vorhin erwähnten dreielementigen Alphabets, und zwar fungiert # als Platzhalter für entweder *1* oder *0*. Das bedeutet, daß zum Beispiel ein Schema der Form *10#0#1* für folgende vier konkrete Entsprechungen stehen könnte: *101011, 101001, 100001* oder *100011*. Oder umgekehrt betrachtet könnte man sagen, daß ein Chromosom der Form *11010* unter anderem folgende Schemata beinhaltet: *#1#1#, #1##0, #1010, ##0##, #101#,* etc.

Als Ordnung eines Schemas wird die Anzahl der nicht #-Symbole bezeichnet. Das heißt, daß zum Beispiel *1####1* die Ordnung 2 und *0110#1* die Ordnung 5 hat. Weiters wird noch der Begriff der definierenden Länge eines Schemas eingeführt, was nichts anderes als den maximalen Abstand zweier nicht #-Symbole bedeutet. So haben zum Beispiel die Schemata *1###1#, ###1##, 1#00#0* definierende Längen von 5, 1 beziehungsweise 6.

Zur Erklärung des Verhaltens von Genetischen Algorithmen wird von Holland folgende Argumentation ins Treffen geführt: Die Vermehrung funktioniert so, daß bessere Individuen eine überproportinal gute Chance haben, Kinder zu produzieren, wodurch natürlich auch die in diesen Elementen enthaltenen Schemata eine überproportionale Möglichkeit besitzen, erhalten zu bleiben. Da nun ein Individuum mit hoher Fitneß gut an die Umgebung angepaßt scheint, liegt es nahe, anzunehmen, daß das auch für die in jedem Individuum enthaltenen Schemata gilt. Wenn man sich vor Augen führt, daß beim Crossover jedes Element gespalten wird, um daraufhin wieder mit Teilen eines anderen Elements zu neuen Individuen zusammengesetzt zu werden, liegt es nahe, anzunehmen, daß nicht allein die Fitneß des ganzen Elements für die Fitneß der Nachkommen entscheidend ist, sondern vielmehr die Fitneß der einzelnen Komponenten.

Um einen Analogieschluß zur Natur zu machen, könnte man sagen, daß sich einzelne positive Eigenschaften der Individuen erhalten, um schließlich in einem Individuum die Kulmination zu erreichen. Anschaulich ist klar, daß dieser Prozeß in der Natur immer noch im Gang ist und zum Beispiel der Mensch in der derzeitigen Erscheinungs- und Ausprägungsform nicht die mögliche Kulmination darstellt, aber es sei auch darauf verwiesen, daß die Natur nach anderen Gesichtspunkten optimiert. In der Natur zählt nicht so sehr reine Geschwindigkeit der Konvergenz hin zu einem Optimum, sondern vielmehr ein energieoptimaler Verlauf des Optimierungsprozesses.

Daher ist das Erreichen des bisherigen Entwicklungsniveaus des Menschen als Erfolg zu werten und möge daher auch als Vorbild für die Optimierung künstlicher Systeme dienen. Daher ist es durchaus plausibel, daß die Vererbung positiver Eigenschaften auf das Gesamtergebnis überdurchschnittlich positive Auswirkungen hat.

Geht man von zwei Elementen aus, die sich dadurch auszeichnen, aus jeweils einer „guten“ und einer „schlechten“ Eigenschaft zu bestehen, kann man bei einem Crossover durch geeignete Wahl des Crossover-Punktes zu folgendem kommen:

Es sei also Lösung 1 gegeben mit $(g_1\ s_1)$ und Lösung 2 mit $(s_2\ g_2)$, wobei g_1 und g_2 „gute“ und s_1 und s_2 „schlechte“ Eigenschaften bezeichnen. Nach einem Crossover erhält man also im Idealfall die Elemente $(g_1\ g_2)$ und $(s_1\ s_2)$. Wie man also ohne Problem erkennt, sind in einem Element der Kindgeneration die „guten“ Eigenschaften und in einem die „schlechten“ Eigenschaften erhalten worden. Durch die Art der Implementierung von Genetischen Algorithmen ergibt sich aber, daß das Element $(g_1\ g_2)$ die weitaus besseren Chancen besitzt, seine Eigenschaften weiterzugeben. Dies soll als Heuristik dienen, warum das Schema-Theorem funktioniert, für genauere Analysen sei auf /HOL75/ verwiesen.

Bausteinhypothese

Goldberg zeigt in einer seiner Arbeiten (/GOL89/, vgl. /BBM93/) daß die Theorie von Holland noch weiter ausbaufähig ist. Er führt den Begriff der Bausteine (Building Blocks) ein, unter denen er nichts anderes versteht, als Schemata mit kurzer definierender Länge. Seiner Ansicht nach entwickelt sich bei fortschreitendem Verlauf der Optimierung die Tendenz hin zur Zusammenarbeit solcher Bausteine, um eine akzeptable Lösung zu erhalten. Allerdings räumt er ein, daß Grundbedingung für seine Hypothesen ein geeignetes Kodierungsverfahren sei.

Solch eine Kodierungsform soll gewährleisten, daß „verwandte“ Gene benachbart am Chromosom angeordnet sind und daß die Eigenschaften, die durch die einzelnen Gene kodiert werden, möglichst unabhängig voneinander sind.

Die Eigenschaft der Unabhängigkeit läßt sich am besten so erklären: Gegeben sei ein Element mit den Eigenschaften *(abcdefg)* und durch einen genetischen Operator (zum Beispiel: Mutation) ergibt sich das Element *(abczefg)*. Wenn nun das Element

(abczefg) eine größere Fitneß aufweist als *(abcdefg)* und gleichzeitig gilt, daß die Fitneß für Elemente der Form *###z###* höher ist als die Fitneß der Form *###d###* kann man sagen, daß die Eigenschaft, die an vierter Position steht unabhängig ist von den restlichen Eigenschaften. Falls man das für alle Eigenschaften sagen kann, sind die einzelnen Gene voneinander unabhängig.

Goldberg behauptet nun, daß bei Erfüllung der oben genannten Bedingungen das Schema-Theorem von Holland die gewünschten Resultate zeitigt. Allerdings sind diese Vorbedingungen nicht immer leicht zu erfüllen. Speziell die Eigenschaft der Unabhängigkeit ist oft schwer zu erfüllen, da dem Anwender zum Zeitpunkt der Implementierung die Interdependenzen der einzelnen Komponenten nicht bekannt sind. Soweit sie allerdings bekannt sind, sollte darauf Rücksicht genommen werden und „verwandte“ Gene nach Möglichkeit benachbart plaziert werden, um eine möglichst effektive Arbeitsweise der Genetischen Algorithmen zu ermöglichen.

Auch bei Nichterfüllung der oben genannten Bedingungen sind erfolgreiche Anwendungen möglich, wie Goldberg in seinem Beitrag über die kleinste irreführende Aufgabenstellung für GA darlegt („minimal deceptive problem“, vgl. /GOL87/. Er untersucht in diesem Beitrag ausführlich Aufgabenstellungen, die den obengenannten Bedingungen nicht genügen und er kommt zu dem Ergebnis, daß selbst bei ungünstigen Voraussetzungen ein einfacher Genetischer Algorithmus im allgemeinen vernünftige Resultate liefert, ohne eine Umstrukturierung des Kodierungsverfahrens (Vertauschen von Genen) vornehmen zu müssen.

Allerdings sind nach wie vor umfangreiche Untersuchungen über das Konvergenzverhalten von Genetischen Algorithmen unter verschiedenen Voraussetzungen im Gange und Gegenstand zahlreicher Veröffentlichungen und Diskussionsbeiträge (siehe u.a. Diskussionen in /COMP_1/).

Auffinden interessanter Bereiche und Erforschung dieser Gebiete

Ein Optimierungsalgorithmus muß, um ein globales Optimum zu finden, zwei Eigenschaften miteinander kombinieren können. Als erstes muß er fähig sein, erfolgversprechende Bereiche des Suchgebietes aufzufinden und dann im zweiten Schritt die so gefundenen Bereiche effektiv einer genaueren Untersuchung zu

unterwerfen. Diese beiden Eigenschaften sind leider - wie nicht anders zu erwarten ist - von gegensätzlicher Natur.

Ein zufallsdominierter Suchalgorithmus ist gut beim Auffinden erfolgversprechender neuer Suchgebiete, versagt dann aber beim genaueren Analysieren dieses Gebietes. Im Gegensatz dazu stehen deterministische Methoden wie Gradientenmethoden, deren Stärken im Auswerten von Zielfunktionen liegen, bei denen ein klar erkenntlicher Anstieg hin zu einem Optimum (lokal oder global) erkennbar ist. Dadurch bleiben diese Methoden aber bei manchen Anwendungen in suboptimalen lokalen Extrema hängen.

Es ist daher eine Verknüpfung dieser beiden Methoden wünschenswert, allerdings sind auch in diesem Fall Rahmenbedingungen für den Verlauf des Optimierungsprozesses vorzugeben. Goldberg hat in einer Arbeit (/GOL89a/) nachgewiesen, daß Genetische Algorithmen beide Eigenschaften in sich vereinen, er hat dafür allerdings einige in der Praxis schwer zu erfüllende Bedingungen als Voraussetzung angenommen:

- Die Populationsgröße ist unendlich
- Die Fitneßfunktion ist präziser Maßstab für den Wert der erhaltenen Lösungen
- Die Gene weisen untereinander keine nennenswerten Abhängigkeiten auf

Offensichtlich kann die erste Bedingung in der Praxis nie erfüllt werden und so kann es zu einem sogenannten genetischen Drift kommen. Unter genetischem Drift versteht man die durch Kumulierung stochastischer Fehler bewirkte Konvergenz einzelner Gene zu nicht optimalen Punkten. Wenn in einer Population in einer Generation eine bestimmte Ausprägungsform eines Gens (bzw. eines Allels) dominant wird, ist es für die nächste Generation wahrscheinlich, daß diese Belegung dieses Gens (Allels) noch dominanter wird. So kann es nach mehreren Generationen passieren, daß ein Gen in der gesamten Population nur mehr eine Erscheinungsform besitzt. Klarerweise wird ab diesem Zeitpunkt mittels des Crossovers oder ähnlicher Techniken für dieses Gen keine alternativen Varianten generiert, was bedeuten kann, daß der GA nie das globale Optimum erreicht.

Diese Phänomen kann natürlich nicht nur für ein Gen auftreten, vielmehr ist möglich, daß mit der Zeit alle oder die meisten Gene davon betroffen sind. Das würde nichts

anderes bedeuten, als daß der GA sich mit dem Auffinden eines wahrscheinlich nicht schlechten, aber keinesfalls optimalen Punktes zufrieden geben muß. Abhilfe würde hier eine verstärkte Verwendung von Mutationen schaffen, allerdings ist das mit der Gefahr verbunden, zu sehr das stochastische Element zu betonen und mit dem GA in Richtung Random-walk abzugleiten.

Wie man leicht sieht, sind zwar die Operatoren eines Genetischen Algorithmus ziemlich einfach, allerdings tritt nicht selten die eigentliche Problematik im Zusammenhang mit dieser Form der Optimierung beim Tunen der Parameter auf.

Die restlichen vorhin angesprochenen Bedingungen sind für theoretische Beispiele üblicherweise nicht so schwer zu erfüllen, aber das Problem liegt auch hier darin, daß sich die Aufgabenstellungen der realen Welt nicht immer an diese Vorgaben halten. So ist es nicht verwunderlich, daß die zweite Bedingung (die Fitneßfunktion ist direkter Maßstab für die Nützlichkeit einer Lösung) in der realen Welt nicht so leicht einzuhalten ist.

Wie in einem vorangegangenen Abschnitt kurz dargelegt, ist das richtige Kalibrieren der Zielfunktion eine mitunter heikle Angelegenheit, was natürlich zwangsläufig dazu führt, daß Kompromisse gemacht werden müssen. Sobald aber von Kompromissen gesprochen wird, muß klar sein, daß oben genannte Bedingung nur mehr näherungsweise erfüllt sein kann. Das sollte in gängigen Anwendungen zwar kein Problem darstellen, sollte dem Anwender aber immer klar sein und ihn davor bewahren, blind und unkritisch irgendwelche Verfahren anzuwenden.

Auch die Voraussetzung, daß Gene untereinander nicht signifikant korreliert sind, läßt sich in der Praxis nicht immer halten. Man denke nur an ein Modell im Bereich der Produktionsplanung, bei dem das Gen a die Geschwindigkeit der Bearbeitung eines Produktes an der Maschine m_a bedeutet und b die Geschwindigkeit der auf Maschine m_a folgenden Maschine m_b bedeutet. Nehmen wir weiters an, daß die zu optimierende Funktion in nicht unerheblichem Maße von den entstehenden Wartezeiten im Produktionsprozeß abhängt. Falls nun die Geschwindigkeit a hoch und die Geschwindigkeit b niedrig gewählt wird, ist nicht unwahrscheinlich, daß sich zwischen den beiden Maschinen ein Stau bildet, welcher sich signifikant auf den Wert

der Zielfunktion auswirkt. Dadurch ist klar, daß die beiden erwähnten Gene keinesfalls unabhängig voneinander sind.

Andererseits wird in vielen Literaturstellen darauf hingewiesen, daß gerade in den Bereichen die Stärke der genetischen Algorithmen liegt, wo es darum geht, Funktionen zu optimieren, deren einzelne Komponenten leichte bis mittlere Abhängigkeiten voneinander aufweisen (vgl. z.B. /BBM93b/). Sollten solche Interdependenzen nämlich nicht auftreten, wäre es leicht möglich, das gegebene Problem in Teilprobleme zu zerlegen und jedes Gen für sich zu optimieren, oder wenigstens zuerst das erste Gen zu optimieren, dann das zweite und so weiter. Das würde zu einer wesentlichen Arbeitserleichterung führen und herkömmlichen Verfahren die besten Chancen einräumen.

Selektion

Eines der wichtigsten Probleme im Zusammenhang mit Genetischen Algorithmen ergibt sich aus der Tatsache, daß gewisse Elemente einer Population ausgewählt werden müssen, um ein Anwachsen der Populationsgröße in nicht gewünschte Bereiche zu verhindern. Einen Beitrag über vergleichende Betrachtungen mehrerer Ansätze mit dazugehörigen statistischen Vergleichen bietet unter anderem Peter J. B. Hancock (vgl. /HAN94/). An dieser Stelle soll für diesen Bereich der Genetischen Algorithmen eine motivierende Einleitung geboten werden (vgl. /GOL89/.

Selektion in der Natur

Eine hervorstechende Eigenschaft der Natur in Bezug auf Evolution ist auf jeden Fall die Auswahl von besser angepaßten Lebewesen zur bevorzugten Weitergabe der Erbmerkmale. Es ist jedenfalls so, daß die verschiedenen Elemente einer Population darum kämpfen, sich zu vermehren. Dieser Kampf wird durch gute Anpassung an die jeweiligen Lebensumstände nicht unerheblich beeinflußt. Denken wir nur an den Bereich der Tiere, die als Beute für andere Lebewesen dienen. Bei ihnen haben Arten, die sich geschickter tarnen können oder Arten, die spezielle Schutzmechanismen (wie etwa Panzer oder Krallen) gegen ihre Hauptwidersacher besitzen oder Arten, die besondere Eigenschaften entwickelt haben (z.B. besonders wendiges Verhalten), allgemein bessere Chancen, länger zu leben als andere.

Es ist einleuchtend, daß Lebewesen, die länger leben, mehr Chancen haben, ihre Erbmerkmale weiterzugeben als solche, die nur kurz leben. Das ganze ist natürlich relativ zu den natürlichen Lebens- und Vermehrungszyklen der einzelnen Arten zu sehen.

Durch das vorhin gesagte wird allerdings kein Automatismus in der Art und Weise impliziert, daß besser angepaßte Lebewesen auf jeden Fall länger leben und sich immer auf Grund ihrer Erbmerkmale behaupten. Daß es immer wieder Fehlentwicklungen gibt, die sich lange halten und dann ziemlich plötzlich verschwinden, mag durch das Aussterben der Dinosaurier dokumentiert werden.

Es ist somit in der Natur der Zwang zur Selektion gegeben, was üblicherweise als natürliche Auslese bezeichnet wird. Klarerweise ergibt sich in der Natur keinesfalls (oder auf jeden Fall nur selten) eine eindimensionale Zielfunktion und weiters ist durch den biologischen Alterungsprozeß klar, daß in der Natur der Trend dahin geht, ältere Mitglieder einer Population durch jüngere zu ersetzen, auch wenn die älteren besser angepaßt wären.

Da es in der Technik keinen Alterungsprozeß eines Elements im herkömmlichen Sinn gibt, haben sich im Laufe der Geschichte der naturanalogen Verfahren einige grundlegende Richtungen entwickelt:

Aussterben der Elterngeneration

In Analogie zur Natur wird hier hauptsächlich Wert auf die neu entstandene Population gelegt, was in der Praxis nichts anderes bedeutet, als daß die Elemente der Kindgeneration nur aus den neuentstandenen Elementen bestehen und in manchen Fällen sogar nur genau soviel Nachkommen erzeugt werden wie Elemente in der Elterngeneration vorhanden waren und die neue Generation komplett die alte Generation ersetzt.

Das bringt klarerweise den Nachteil, daß gute Lösungen unter Umständen von schlechteren verdrängt werden und vielleicht sogar für immer verloren gehen. Dem kann man allerdings entgegen halten, daß dafür der Auswahldruck in Richtung guter Lösungen nicht allzu groß sein muß. Dieser Aspekt kann sich durchaus als Vorteil erweisen, wenn man bedenkt, daß oft die „schlechten“ Lösungen im Endeffekt den

entscheidenden Anteil am Finden einer wirklich guten Lösung haben. Das ist am besten so zu verstehen, daß durch schlechte Lösungen ein Verlassen eines Bereiches ermöglicht wird, der keine wirklichen Verbesserungsmöglichkeiten mehr bietet. Dadurch kann ein Bereich eines lokalen Optimums verlassen werden.

Es soll aber nicht verheimlicht werden, daß es dadurch auch zu dem unerwünschten Effekt kommen kann, daß die langfristige Devise zur Verbesserung der Lösungen verletzt wird und dadurch eine Annäherung an das globale Optimum über viele Generationen hinausgezögert wird. Daß das Rechenzeit kostet, ist unmittelbar einsichtig. Aber gerade in der Rechenzeit kann auch ein Vorteil dieses Verfahrens liegen, da man sich durch die deterministische Annahme, nur die Kindgeneration weiter zu verwenden, unter Umständen ausgiebige Vergleichsoperationen mit den übrigen Elementen der Elterngeneration (und weitere Selektionsschritte) erspart.

Weiters benötigt diese Art der Auswahl im Fall, daß nur genauso viele Elemente erzeugt werden, wie die Populationsgröße beträgt, keine komplizierten Strategien zur Auswahl, da ohnehin klar ist, welche Populationsmitglieder „überleben“. Klarerweise könnte man durch ausgefeilte Strategien genauer auf spezifische Problemstellungen eingehen, aber je komplexer eine Strategie ist, desto leichter kann man Fehler begehen und damit genau das Gegenteil des Beabsichtigten erreichen.

Wie man daraus sieht, ist speziell in solchen Fällen ein Genetischer Algorithmus ziemlich kontroversiell und oft schwer berechenbar, aber auch bei vielen anderen Verfahren weiß man oft erst am Schluß, ob sie sinnvoll eingesetzt sind. Ein Hauptargument der Befürworter dieses Verfahrens ist die große Analogie zur Selektion in der Natur und um das Vergessen der besten Lösung zu verhindern, kann man als Kompromiß die beste Lösung gesondert speichern. Damit ist jedenfalls gewährleistet, daß das Endergebnis nicht schlechter sein kann als ein Zwischenergebnis.

Auswahl deterministisch nach Fitneß

Bei diesem Ansatz werden jeweils für die benötigten Elemente der nächsten Generation nur die besten Elemente von Elterngeneration und den Elementen, die durch genetische Operatoren aus dieser hervorgegangen sind, genommen. Dadurch ist der Trend zur Auswahl der besten Elemente am stärksten implementiert und es kommt

neben den Überlegungen, die schon im vorigen Abschnitt zu diesem Thema angestellt worden sind, noch zu der höheren Wahrscheinlichkeit einer frühzeitigen Konvergenz hin zu einem lokalen Optimum. Man kommt nicht umhin, die Verwandtschaft dieser Selektionsmethode mit herkömmlichen Gradientenmethoden zu bemerken, da hier auch in Richtung des momentan größten Erfolgs gesucht wird.

Auswahl stochastisch nach Fitneß

Im Unterschied zum vorigen Verfahren werden nicht einfach die besten Elemente genommen, sondern es kommt zu Auswahlmechanismen, bei denen gute Lösungen bessere Chancen haben als schlechte. Da solche Auswahlmechanismen Zeit benötigen, ist in jedem Fall mit mehr CPU-Zeit zu rechnen als in den beiden erstgenannten Fällen. Dafür hat man aber das Überleben von guten Lösungen gewährleistet und gleichzeitig für eine Erhaltung einer gewissen Vielfalt gesorgt, womit ein Haupterfolgsmerkmal der GA erfüllt bleibt. In diesem Fall ist das Auftreten einer frühzeitigen Konvergenz unwahrscheinlicher.

Kombination der Auswahlkriterien

Es liegt auf der Hand, die bisher aufgeführten Verfahren zu kombinieren, um eine Vereinigung der jeweiligen Vorteile zu erhalten und die Nachteile zu minimieren. Dies ist am leichtesten möglich, indem man einen Prozentsatz festlegt, mit dem bestimmt ist, wieviel Elemente der Elterngeneration und wieviel Elemente der neuerhaltenen Lösungen in die Kindgeneration Eingang finden. Die Auswahl innerhalb der Elterngeneration kann nach Kriterien erfolgen, die entweder im zweiten oder dritten Verfahren angedeutet sind. Analog verfährt man mit den neu hinzugekommenen Lösungen.

Das führt allerdings zu etwas komplizierteren und nicht so leicht durchschaubaren Selektionsverfahren, denen meist eine aufwendige Bestimmung der Parameter (Bestimmung des Prozentsatzes, etc.) vorausgeht. Die allgemein gültige Antwort wird sicher nicht zu finden sein, aber es gibt für eine große Anzahl von Anwendungsbeispielen unzählige Präzedenzfälle mit mehr oder weniger geglückten Selektionsmechanismen.

Robustheit

Wenn man vom Begriff der Robustheit spricht, muß man verschiedene Bereiche unterscheiden. Einer davon ist jener, den Goldberg (vgl. /GOL89/) mit seinem Beitrag über das kleinste irreführende Problem dargelegt hat und gezeigt hat, daß Genetische Algorithmen in diesem Sinn als robust zu erachten sind. Es gibt aber in diesem Zusammenhang noch einige weitere Aspekte, die nachfolgend kurz betrachten werden.

Allgemeine Einsetzbarkeit

Man kann durchaus sagen, ein allgemein einsetzbarer Algorithmus ist ein robuster Algorithmus und bei Genetischen Algorithmen hat man es mit einer sehr allgemeinen Kodierungsform zu tun, die es ermöglicht, eine große Menge unterschiedlicher Aufgabenstellungen zu implementieren.

Wenn man sich vor Augen hält, daß jede Aufgabenstellung, die man implementieren will, eigentlich nur eine parameterisierbare Entsprechung benötigt, um geeignet für einen GA verschlüsselt zu werden, wird man sich der Vielzahl der möglichen Anwendungen bewußt. Das reicht noch nicht für die Aussage der allgemeinen Einsetzbarkeit, wenn man sich allerdings die verschiedenen möglichen genetischen Operatoren mit den unzähligen Einstellungsmöglichkeiten der zugehörigen Parameter vorstellt, kommt man der Sache schon ziemlich nahe. Die entgültige Beweisführung sei hier (unvollständig) induktiv durch einen Verweis auf die unzähligen erfolgreichen Anwendungen in Praxis und Theorie geführt.

Portabilität

Ein Begriff der mit dem vorigen nahe zusammenhängt ist jener der Portabilität, worunter man versteht, daß ein bestimmter Genetischer Algorithmus, der mit seinem spezifischen Parametersetting ein Problem zur Zufriedenheit gelöst hat, auch bei verwandten Problemen akzeptable Ergebnisse liefert. Es soll also gewährleistet sein, mit einem gut getunten GA ein ganzes Bündel von ähnlichen Aufgabenstellungen zu bewältigen.

Das ist um so mehr notwendig, als der Vorgang des Parametertunens oder sogar allgemeiner, der Prozeß des Ausfindigmachens des richtigen GA, meist eine im Verhältnis zur Laufzeit des eigentlichen Algorithmus sehr aufwendige Angelegenheit

darstellt, da sie üblicherweise von Hand durchgeführt wird. Es gibt zwar sogenannte Meta-GA, die vor dem eigentlichen GA ausgeführt werden, um ein möglichst optimales Tunen der beteiligten Parameter zu ermöglichen, allerdings sind diese Verfahren nicht sehr verbreitet und auf jeden Fall mit hohem Aufwand verbunden.

Optimierungsziele

Gegen Ende dieses Kapitels sei noch kurz der Begriff der Optimierung näher beleuchtet. Wie eingangs mehrmals erwähnt, hat Holland bei der Entwicklung der Genetic Algorithms hauptsächlich (oder ausschließlich) die Adaption künstlicher Systeme an bestehende Systeme im Auge gehabt. Daß man diese Theorie auch auf Optimierungsziele erfolgreich anwenden kann, zeigt die Verwandschaft dieser beiden Bereiche.

Die klassische Optimierungsform ist jene der Maximierung (oder gegebenenfalls die der Minimierung) gewisser Probleme. Zu diesen gehören unter anderem die Lösung des Traveling Salesman Problems (TSP), die Gewinnmaximierung wirtschaftlicher Probleme, Lösung technischer Probleme, etc.

Bei dieser Art kann man zwei verschiedene Ziele verfolgen:

- Suchen des globalen Maximums
- Finden einer akzeptablen Lösung in einer akzeptablen Zeit

Das erste Ziel wird oft nur zu erreichen sein, wenn man ausreichend Zeit zur Verfügung hat, außerdem kommt noch hinzu, daß oft der Beweis schwer fällt, wirklich **das** globale Maximum gefunden zu haben. Das zweite Ziel wird oft dann angestrebt, wenn man daran interessiert ist, eine bestehende Situation zu verbessern und der Suchraum so groß und inhomogen ist, daß man das Auffinden des globalen Maximums als nahezu unmöglich erachtet.

Es kann allerdings auch das Zeitproblem im Vordergrund stehen, wenn man eine vernünftige Lösung möglichst schnell benötigt und Rechenzeit, die eine umfangreichere Suche sicherlich benötigen würde, nicht zur Verfügung hat.

Beide Arten sind in der Implementierung ähnlich und bedürfen im Allgemeinen keiner speziellen Fallunterscheidungen.

Das Ziel der Approximation wird dann angestrebt, wenn man vorhandene Strukturen möglichst gut durch andere darstellen will oder verrauschte Signale identifizieren will.

Anwendungsbeispiele sind unter anderem Funktionsapproximationen oder Pattern Recognition. Bei diesen Zielen geht es ausschließlich um das Finden guter Entscheidungskriterien und nicht um Maximierung oder Minimierung.

Bei Vorhersagesystemen geht es, wie der Name schon sagt, um Verfahren, die aus eingegebenen Daten Signifikanzwerte ermitteln und dadurch den Zusammenhang zwischen einzelnen Faktoren erklären sollen. Daraus soll dann durch Extrapolation eine Vorhersage für die Zukunft ermöglicht werden. Ein Anwendungsgebiet ist z. B. die Feststellung der Kreditwürdigkeit von Kunden einer Bank. Das angewendete Verfahren muß in diesem Fall eine möglichst gute Erklärungsmöglichkeit der Wirklichkeit gewährleisten.

Simulation

In den letzten Jahren hat sich auf Grund steigender Komplexität der Aufgabenstellungen immer mehr in technischen Bereichen die Computersimulation als Werkzeug zur Entscheidungsfindung durchgesetzt.

Der Begriff „Simulation“ selbst ist sehr vielschichtig, es ist daher notwendig, ihn einzugrenzen. Das Wort Simulation leitet sich vom lateinischen Verb „simulare“ ab, was unter anderem „nachbilden“ oder „nachahmen“ bedeutet. Dadurch ergibt sich die nachfolgende Definition des VDI, welche als VDI-Richtlinie 3633 veröffentlicht ist:

> Simulation ist die Nachbildung eines dynamischen Prozesses in einem Modell, um zu Erkenntnissen zu gelangen, die auf die Wirklichkeit übertragbar sind.

Erkenntnisse können natürlich auch direkt am Prozeß oder durch Versuchsaufbauten gewonnen werden. Beides ist jedoch oft sehr teuer und manchmal auch gar nicht möglich. Simulation wird daher unter anderem dann sinnvoll eingesetzt, falls

- Neuland beschritten werden soll
- die Grenzen analytischer Methoden erreicht sind
- Versuche technisch nicht möglich sind
- Versuche nicht vertreten werden können
- Versuche sehr zeitaufwendig sind
- Versuche sehr teuer sind

Aufbau einer Simulation

Da es mannigfache Arten der Anwendung der Simulation gibt, kann man zwar kein allgemein gültiges Schema für den Aufbau einer Simulation angeben, allerdings kann man für viele Applikationen vereinfachend sagen, daß das Verfahren einer Simulation methologisch grob betrachtet aus folgenden drei Teilen besteht (vgl. folgende Abbildung):

1. Modellieren

2. Experimentieren

3. Interpretieren

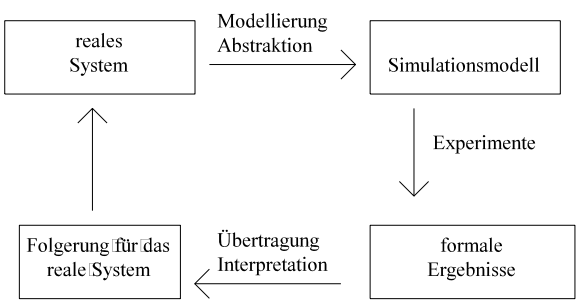
Beim Modellieren eines Problems wird zuerst je nach Anwendungsfall entweder der IST- oder der SOLL-Zustand der abzubildeten Aufgabenstellung erhoben. Es geht in dieser Phase darum, möglichst genau die Eigenheiten des Systems zu erfassen, um später langwierige Korrekturen zu vermeiden.

Es sei an dieser Stelle auch darauf verwiesen, daß nicht immer alle relevanten Daten zur Verfügung stehen, wodurch klar wird, daß sich der Leiter einer Simulation schon zu Beginn genau einen Überblick über alle benötigten Werte verschaffen soll. Er kann so schon rechtzeitig Entscheidungen treffen, wie die fehlenden Daten entweder zu beschaffen oder zu substituieren sind. Dies gilt speziell in dem Fall, wo als Überbau für die Simulation noch eine Optimierung mit einem Genetischen Algorithmus geplant ist.

In der Phase des Experimentierens geht es um gezieltes Verwenden des erstellten und validierten Modells, um die Anforderungen, die an das Modell gestellt werden, zu überprüfen. Es ist hier speziell ein gezieltes Arbeiten notwendig, um oft aufwendige Simulationsläufe nicht unnötig durchzuführen und die Aussage des Ergebnisses der Gesamtoptimierung nicht schon in dieser Phase durch Ungenauigkeiten in Frage zu stellen.

Am Schluß jeder Simulation steht der Prozeß des Interpretierens. Hier geht es um das Umsetzen der am Modell gewonnenen Erkenntnisse in die Praxis des gestellten Problems.

Simulationskreislauf



Schema einer Simulation

In einer Verfeinerung des oben gesagten kann man zu folgendem Ablaufschema einer Computersimulation mit Optimierung kommen:

1. Problemformulierung
2. Mathematische Modellbildung
3. Mathematische Umformungen
4. Numerische Analyse
5. Implementation des Modells
6. Verifizierung des Programms
7. Validierung des Modells
8. Identifikation des Modells
9. Simulation des Modells
10. Optimierung mit geeigneten Verfahren (Genetische Algorithmen) oder Experimente „von Hand“
11. Interpretation der Ergebnisse

Dies stellt klarerweise nur ein Grundgerüst dar, daß je nach Bedarf abgewandelt werden kann, allerdings kann ohne weiteres behauptet werden, daß der Erfolg einer Simulation hauptsächlich durch die Zusammenarbeit zwischen Planer und Simulationsexperte bestimmt wird. Daher ist es notwendig, obige Systematik wenigstens in groben Zügen einzuhalten.

Eine wichtige Frage in diesem Zusammenhang ist auch die Wahl der richtigen Simulationssprache und der richtigen oder am besten angepaßten Strategie. In letzter Zeit hat sich gerade auf diesem Gebiet einiges getan und es sind die verschiedensten Ansätze wie Fuzzy Logik, Parallele Simulationssysteme, Künstliche Intelligenz, Neuronale Netze, Simulated Annealing oder natürlich auch Genetische Algorithmen aufgetaucht.

Professor Bernd Schmidt meint in diesem Zusammenhang (siehe /REG94/, Seite 3): „Die Entscheidung wird um so schwieriger, da viele neue Ansätze zur Zeit nicht mit der erforderlichen Zurückhaltung auftreten. Geradezu marktschreierisch spielen sich neue Gebiete in den Vordergrund, ohne daß sie ihren Wert und ihre Brauchbarkeit bereits unter Beweis gestellt hätten. Mit Sorge beobachtet man eine Situation, die etwas Unseriöses an sich hat und im Wissenschaftsbetrieb eigentlich nicht geduldet werden sollte!“

Wie man sieht, ist es keine leichte Aufgabe, hier das richtige Instrument zu wählen, es bleibt allerdings immer die Möglichkeit, sich durch etablierte Veröffentlichungen eine meinungsbildende Unterstützung angedeihen zu lassen.

Um einem weit verbreiteten Irrtum zu begegnen, sei hier erwähnt, daß das Instrument der Simulation kein Ersatz für die Planung sein soll, viel mehr soll durch gezielte Anwendung eine Unterstützung in der Entwicklungsphase eines Projekts bereit gestellt werden. Um diese Zielsetzung zu erreichen, muß das Ziel einer Simulation vor ihrem Beginn quantitativ formuliert werden. Fragestellungen, die zu global sind oder auf Grund ihrer Komplexität nicht befriedigend lösbar erscheinen, sind weder für den Anwender noch für die Durchführenden hilfreich.

Die Zielsetzung beeinflußt den Umfang und die Detailtiefe von Datenbeschaffung und Modellierung, die Anzahl der Experimente und den Aufwand für Ergebnisinterpretation und -darstellung. Bei der Formulierung der Zielvorstellungen muß der Aufwand bei der Datenbeschaffung und der Simulation gegenüber dem Nutzen der Einzelergebnisse sorgfältig abgewogen werden.

Kontinuierliche Simulation

Da naturanaloge Verfahren und speziell Genetic Algorithms gerade in solchen Bereichen, die mit Hilfe der kontinuierlichen Simulation untersucht werden, verwendet werden, sei hier ein kleiner Exkurs in diesen Bereich durchgeführt.

In der kontinuierlicher Simulation werden Modelle untersucht, die einer permanenten Änderung relevanter Kenngrößen unterworfen sind. Typischerweise - aber nicht notwendigerweise - werden derlei Systeme mittels Differentialgleichungen

beschrieben. Im folgenden seien einige wichtige Beispiele für zeitkontinuierliche Simulationsanwendungen geschildert.

Fahrzeugentwicklung

In letzter Zeit hat der Bereich der Fahrzeugentwicklung in der Simulation seinen festen Platz. Heutzutage ist es undenkbar ohne Computerunterstützung ein Fahrzeug zu planen, da der Kostenaufwand für die Erstellung eines 1:1-Modelles (sprich eines Autos oder anderer Fahrzeuge bzw. Fahrzeugkomponenten) bei weitem zu teuer käme. Interessant sind hier die Bereiche

- Interaktion Fahrzeug - Fahrbahn
- Simulation der Dynamik bei Unfällen
- Simulation der Antriebsstränge
- Simulation der Hydraulik und Elektrik
- Simulation von Prüfständen.

Energiewirtschaft

Im Energiebereich sind die Hauptinteressen auf

- Kraftwerksimulation
- Störfallsimulation
- Ausbildung von Bedienpersonal
- Zeitantwort bei Erdbeben
- Simulation von Schockwellen, Verdampfern, Ventilen
- Steuerung von Rohrleitungssystemen

konzentriert.

Flugsimulation

Die Flugsimulation legt ihr Augenmerk hauptsächlich auf

- Regelung und Steuerung

- Flugbahnsimulation
- Trainingssimulatoren

Elektrotechnik & Elektronik

Die Elektrotechnik verfolgt eher die Simulation der Regeltechnik in allen Gebieten, wo hingegen die Elektronik

- nichtlineare elektronische Schaltungen
- Parameteroptimierung mit Bedingungen im Zeitbereich

untersucht.

Zu den bis jetzt erwähnten Bereiche kommen noch die folgenden mit ihren Subbereichen:

Rechnerherstellung

- Entwicklung und Optimierung von Schnelldruckern,
- Magnetbandlesegeräten und Plattenlaufwerken

Chemie

- Wachstumsprozesse, Diffusionsvorgänge
- Prozeßsteuerung

Medizin

- Medikamenteneinfluß auf den menschlichen Körper
- Tumorwachstum
- Herzschrittmacher
- Simulation des Kreislaufs

Umwelt

- Schadstoffeinflüsse auf Pflanzen und Tiere
- Wachstumsuntersuchungen

- Diffusionsvorgänge

In all diesen Bereichen ist eine Verknüpfung der Simulation mit einem Optimierungstool wie den Genetischen Algorithmen denkbar und manche Aussagen dieser Dissertation sind keinesfalls auf den Bereich der diskreten Simulation beschränkt, sondern lassen sich vielmehr auch auf Problemstellungen der kontinuierlichen Simulation anwenden.

Die kontinuierliche Simulation hat im Vergleich zur diskreten die weitaus längere Tradition und ist eher im „Kernbereich“ der Technik angesiedelt, was heißen soll, daß es hier eher um Darstellung technischer Zusammenhänge geht.

Diskrete Simulation

In der heutigen Zeit trifft man häufig auf komplexe, strukturähnliche Abläufe, deren Planungen sich mit der Komplexität zunehmend schwieriger gestalten und wo die diskrete Simulation wertvolle Dienste leisten kann. Zur Verdeutlichung des Unterschieds zwischen kontinuierlicher und diskreter Simulation mögen die folgenden Überlegungen dienen:

In der Mathematik stehen sich zwei grundverschiedene Begriffe gegenüber, nämlich die Begriffe „diskret" und „stetig". Während „stetig" Strukturen meint, die sich „unendlich" nahe kommen, wie etwa eine stetige, reelle Funktion, wo überabzählbar viele Zahlen, die einander beliebig nahe kommen können, anderen Zahlen zugeordnet werden, meint „diskret" wohlunterscheidbare Elemente, die einzeln für sich stehen.

Hierzu seien zwei Beispiele aus dem Bereich der Wahrscheinlichkeitstheorie angeführt:

- Stetiges Problem: Wie groß ist die Wahrscheinlichkeit, daß die Lösung der Gleichung $x*x+2px+q=0$ reell ist, wenn p und q gleichwahrscheinlich aus zwei bestimmten Intervallen gewählt werden?

- Diskretes Problem: Wie groß ist die Wahrscheinlichkeit, mit einem Würfel einen Sechser zu würfeln?

Bei der Unterscheidung beliebiger Prozesse unterteilt man nun ebenfalls in diese zwei Kategorien, wobei bei stetigen Prozessen zu jedem Zeitpunkt über bestimmte

Intervalle hinweg etwas „passiert", wobei sich diese Zeitpunkte beliebig nahe kommen können.

Als Beispiel denke man an den Bremsvorgang beim Autofahren, wo die negative Beschleunigung zu jedem Zeitpunkt während des Bremsvorganges wirkt. Dieser Vorgang kann etwa mit Differentialgleichungen beschrieben werden und die Untersuchung solcher Gleichungen kann mit rein mathematischen Mitteln geschehen.

Bei diskreten Prozessen „passiert“ etwas zu bestimmten, wohl unterscheidbaren Zeitpunkten. Als Beispiel sei hier ein Aufzug in einem Hochhaus erwähnt, wo zu bestimmten Zeitpunkten Benutzer kommen und den Aufzug rufen bzw. benutzen. Der Aufzug selbst ist zu bestimmten Zeiten belegt bzw. frei. Dieses Problem kann nicht mit rein mathematischen Mitteln beschrieben werden, will man jetzt dennoch herausfinden, wie viele Aufzüge in einem Hochhaus mindestens eingebaut werden sollen, damit es nicht zu überlangen Wartezeiten kommen kann, muß eine andere Form der Problembeschreibung gewählt werden.

Bei Modellen diskreter Prozesse ändern sich die Systemgrößen also nur zu diskreten Zeitpunkten, den sogenannten Entscheidungszeitpunkten. Aufgrund der Festlegung der Zeitpunkte können nun die Modelle und Prozesse folgenderweise katalogisiert werden:

Zeitdauer zwischen Entscheidungszeitpunkten	Prozeß- bzw. Modellart
vorgegeben	deterministisch
berechnet	algorithmisch
zufällig	stochastisch

Die wohl häufigsten Modelle in der Simulation von diskreten Systemen sind stochastische Modelle, was mit der Natur der Daten zusammenhängt, da z. B. Verarbeitungszeiten oder ähnliches meistens einer gewissen Streuung unterliegen. Es ist aber natürlich einsichtig, daß die drei oben genannten Grundarten oft gemischt werden müssen, um ein annähernd getreues Abbild der Wirklichkeit zu ermöglichen.

Diskrete Simulation wird unter anderem für folgende Bereiche verwendet:

- Fertigungssysteme
- Montagesysteme
- Materialflußsysteme
- Werkstattsteuerung
- Arbeitsstrukturierung

Datenbeschaffung

Gerade im Bereich der diskreten Simulation ist im Ablauf einer Simulationsstudie die Beschaffung des Datenmaterials ein wichtiger Schritt. Die Art der Datenbeschaffung hängt davon ab, ob ein bereits bestehendes oder ein geplantes System modelliert werden soll oder ob ein System erst entworfen werden soll oder ein geplantes Modell auf seine Funktionstüchtigkeit überprüft werden soll.

Bei existierenden Anlagen liegen in der Regel Daten und Erfahrungswerte vor. So enthalten Arbeitspläne wichtige Informationen bezüglich Bearbeitungs- und Rüstzeiten sowie der Beschaffungsreihenfolge. Weitere Daten lassen sich über gezielte Analysen am produzierenden System (Störzeitanalysen) oder über entsprechende Fachabteilungen, z. B. Qualitätssicherung oder Instandhaltung beschaffen.

Die Ermittlung von Daten, die in Planung befindliche Anlagen charakterisieren, ist schwieriger und fehleranfälliger. Es hat sich gezeigt, daß über Analogiebetrachtungen zu ähnlichen, vorhandenen und damit analysierbaren Systemen, geeignete Kenngrößen zu beschaffen sind.

Datenarten

Grundsätzlich lassen sich Simulationsdaten in zeit- und mengenorientierte Größen einteilen. Zu den Zeitparametern zählen im wesentlichen Takt- und Bearbeitungszeiten, Handling-, Transport- und Stördauern. Mengenspezifische Kenngrößen sind z. B. Stückzahlen, Pufferkapazitäten, Transportmittelanzahl oder Störungshäufigkeit.

Die Art der Daten beeinflußt in entscheidender Weise ihre Beschaffung. Gerade die Ermittlung des Störverhaltens von Anlagen verlangt unter Umständen einen sehr großen Aufwand, um hier repräsentatives Material zu erhalten. Diese Überlegungen führen uns zum Begriff der Inputdatenanalyse, die oft ungerechtfertigt vernachlässigt wird (vgl. /KEL84/).

Um vom Begriff der Inputdatenanalyse sprechen zu können, ist zuerst prinzipiell festzuhalten, daß zwei Phasen bei der Modellierung einer Simulation zu unterscheiden sind. In der ersten Phase ist man bemüht, die Struktur des Problems zu analysieren, indem man feststellt, welche physikalischen und logischen Zusammenhänge zwischen den einzelnen Komponenten des Systems bestehen. Dies impliziert natürlich, daß man sich bereits einen Überblick verschafft hat und über die Funktion und Anordnung aller wichtigen Komponenten unterrichtet ist.

Es ist dann Sache des Simulationsexperten einen groben Ablaufplan zu erstellen, der bereits Details über die Elemente der Simulationssprache enthalten soll. In der zweiten Phase werden die numerischen Aspekte der geplanten Realisierung näher beleuchtet. Zu diesem Bereich gehören z. B. die Verteilungen von Service- und Bearbeitungszeiten, die Anzahl von parallelen Bearbeitungsstationen oder die Art des Ankunftsprozesses von Einheiten im System.

Wie vorhin ausgeführt, gibt es im Gebiet der diskreten Simulation unter anderem die deterministisch vorgegebenen und die stochastischen Eingangsdaten. Die deterministisch vorgegebenen Werte sind ungleich leichter zu bestimmen, als jene, die Zufallsgesetzen gehorchen, die sich unserer genauen Kenntnis entziehen.

Dieses Problem mit stochastischen Prozessen ist zweigeteilt. Zuerst muß man festlegen, welche Verteilung man vor sich hat und dann müssen die notwendigen Parameter dieser Verteilung bestimmt werden. In diesem Zusammenhang stellt sich die Frage, aus welchen Quellen Datenmaterial zu bekommen ist. Einige gangbare Möglichkeiten sind folgende:

- Beobachtungen vor Ort, was bei bestehenden, überschaubaren Systemen zu empfehlen ist

- Vergleiche mit äquivalenten Systemen was bei nicht existierenden Systemen eine mögliche Form darstellt
- Befragung von Experten (Vorarbeiter, Meister, etc.), was bei fehlenden Daten unerläßlich ist

Bei all diesen Methoden ergeben sich verschiedene Probleme, die unterschiedliche Konsequenzen nach sich ziehen. So ist z. B. bei eigenen Beobachtungen stets die Frage zu stellen, ob die Beobachtungen repräsentativ sind. Es könnte leicht sein, daß man einen Beobachtungszeitraum wählt, der aus mannigfaltigen Gründen ein atypisches Verhalten zeigt.

Beim Vergleich mit anderen Systemen muß gewährleistet sein, daß die relevanten Kenngrößen mit dem zu modellierenden Problem auch wirklich kompatibel sind. Nicht selten kommt es vor, daß Systeme, die scheinbar gleich funktionieren doch signifikante Unterschiede aufweisen.

Das größte Problem mit der dritten Art der Datenrequirierung ist ein Problem der Präzision. Es ist zwar normalerweise leicht für Beschäftigte Auskunft über z. B. die durchschnittliche Dauer eines Arbeitsvorgangs zu geben, allerdings ist es damit noch nicht getan, da man ja meistens eine statistische Verteilungsfunktion benötigt. Es ist somit Sache des Modellbauers, durch geschicktes Befragen nähere Informationen zu erhalten.

In solchen Fällen erhält man als Ergebnis meist eine Gleichverteilung in einem bestimmten Intervall oder als Verfeinerung dieser eine Dreiecksverteilung oder eine Beta-Verteilung, die zusätzlich zum wahrscheinlichsten Wert (in der Dreiecksverteilung verwirklicht) noch den Durchschnittswert berücksichtigt. Das bewirkt natürlich einen weitaus flexibleren Verlauf der Verteilungskurve, bedarf allerdings einer größeren Genauigkeit beim Erstellen, um keine Verteilungen zu erhalten, die unnötig von den wahren Verhältnissen abweichen.

Neben den schon kurz angesprochenen Verteilungen sind natürlich auch alle anderen bekannten (oder unbekannten) Verteilungen ins Auge zu fassen. Die Grundfrage, die schon Generationen von Mathematikern beschäftigt hat, ist jene, ob die Daten unabhängig identisch verteilt sind. Denn nur wenn diese Bedingung erfüllt ist, lassen

sich einige grundlegenden Vorgangsweisen der Statistik auf Probleme der Praxis anwenden. Doch leider ist diese Forderung nur allzu oft nicht erfüllt, man denke nur an Ausfallswahrscheinlichkeiten einer Maschine.

Kann man zu Beginn unter Umständen noch von einer Exponentialverteilung mit gegebenem Erwartungswert für die Arbeitszeit sprechen, so wird sich diese möglicherweise bedingt durch Reparaturen verändern. Es wird also manchmal unerläßlich sein, einen geeigneten Test anzuwenden, um herauszufinden, ob der Forderung nach unabhängiger identischer Verteilung in ausreichendem Maße genüge getan ist.

Ein Test ist ein Verfahren zur Überprüfung von Annahmen über Verteilungen, die das Zustandekommen von Beobachtungsdaten beschreiben. Solche Annahmen heißen statistische Hypothesen. Liegen die Beobachtungsdaten in Form einer Meßreihe vor, so soll aufgrund eines Tests entschieden werden, ob eine bestimmte Annahme als widerlegt zu betrachten (zu verwerfen) ist.

Ein solcher Test ist daher bereits durch die Angabe eines Ablehnungsbereiches (oder kritischen Bereiches) beschrieben, falls vereinbart wird, daß die Hypothese immer dann abzulehnen ist, wenn das n-Tupel der Beobachtungswerte in diesem Bereich liegt.

Als einer von vielen möglichen Tests sei hier der χ^2-Unabhängigkeitstest , der für eine zweidimensionale Zufallsvariable *(X,Y)* überprüft, ob ihre Komponenten *X* und *Y* unabhängig sind, vorgestellt (vgl. /LEWE92/).

Zuerst zerlegen wir den Wertebereich von *X* in disjunkte Mengen $I_1,...,I_k$ und den Wertebereich von *Y* in disjunkte Mengen $J_1,...,J_l$.

Es seien
$$p_{ij} = P(X \in I_i, Y \in J_j),$$
$$p_{j.} = \sum_{j=1}^{l} p_{ij} = P(X \in I_i)$$

sowie $p_{.j} = \sum_{i=1}^{k} p_{ij} = P(Y \in J_j)$ für $i=1,...,k$ und $j=1,...,l$.

Im Falle der Unabhängigkeit von X und Y gilt

$$P(X \in I_i, Y \in J_j) = P(X \in I_i) * P(Y \in J_j),$$

also $p_{ij}=p_{i.}*p_{.j}$ für $i=1,...,k$ und $j=1,...,l$.

Die Eigenschaft der Unabhängigkeit von *X* und *Y* wird auf die Gültigkeit dieser Gleichungen reduziert. Wir testen die Nullhypothese, daß diese Gleichungen für alle Paare *(i,j)* erfüllt sind, bei der Gegenhypothese, daß mindestens eine dieser Gleichungen nicht erfüllt ist.

Seien $(X_1,Y_1),...,(X_n,Y_n)$ unabhängige, identisch wie *(X,Y)* verteilte zweidimensionale Zufallsvariablen und $(x_1,y_1),...,(x_n,y_n)$ eine Realisierung. Die Zufallsvariable N_{ij} sei definiert als die zufällige Anzahl der Zahlenpaare in der Realisierung, die in $I_i \times I_j$ liegen. Dann ist die relative Häufigkeit $\frac{1}{n} N_{ij}$ ein Schätzer für p_{ij}. Ist die Nullhypothese H_0 gültig, so wird die Testgröße

$$Q((X_1,Y_1),...,(X_n,Y_n)) = \sum_{i=1}^{k}\sum_{j=1}^{l} \frac{(N_{ij} - n \cdot \frac{1}{n} N_{i.} \cdot \frac{1}{n} N_{.j})^2}{n \cdot \frac{1}{n} N_{i.} \cdot \frac{1}{n} N_{.j}}$$

kleine Werte annehmen. Diese Testgröße ist unter der Nullhypothese für großes *n*, wie man zeigen kann, näherungsweise $\chi^2_{(k-1)(l-1)}$-verteilt.

Der χ^2-Unabhängigkeitstest wird folgendermaßen durchgeführt: Aus den Beobachtungsergebnissen $(x_1,y_1),...,(x_n,y_n)$ wird für jedes Paar *(i,j)* die Anzahl n_{ij} ermittelt, in eine sogenannte Kontingenztafel eingetragen und die Spalten- und Zeilensummen $n_{.j}$ bzw. $n_{i.}$ berechnet. Zu gegebenem α wird aus einer Tafel das Quantil $\chi^2_{(k-1)(l-1);1-\alpha}$ bestimmt. Gilt

$$\sum_{i=1}^{k}\sum_{j=1}^{l} \frac{(n \cdot n_{ij} - n_{i.} \cdot n_{.j})^2}{n \cdot n_{i.} \cdot n_{.j}} \geq \chi^2_{(k-1)(l-1);1-\alpha}$$

so wird die Nullhypothese abgelehnt (das heißt, daß die Komponenten nicht unabhängig sind), ansonsten wird nichts gegen die Nullhypothese eingewendet.

Zu den wichtigsten stetigen Verteilungen zählen die Exponentialverteilung und die Gaußverteilung (vgl. /BRE91/ und (BRBG90/). Besonders in den Bereichen der

diskreten Simulation gibt es viele Prozesse, die näherungsweise eine Exponentialverteilung aufweisen. Zu diesen gehören etwa Zwischenankunftszeiten, Bedienungszeiten, störungsfreie Betriebszeiten einer Maschine und ähnliches. Diese Exponentialverteilung ist durch folgende Dichtefunktion charakterisiert:

$$f(x) = \begin{cases} 0 & \text{für } x \leq 0 \\ \lambda \cdot e^{-\lambda \cdot x} & \text{für } x \geq 0 \end{cases}$$

Daraus resultiert die Verteilungsfunktion

$$F(x) = \lambda \cdot \int_{\tilde{x}=0}^{x} e^{-\lambda \cdot \tilde{x}} \cdot d\tilde{x} = 1 - e^{-\lambda \cdot x}$$

Diese Verteilung hat den Mittelwert $\mu = 1/\lambda$ und die Streuung $\sigma^2 = 1/\lambda^2$. Die Gaußverteilung oder Normalverteilung wurde von Carl Friedrich Gauß im Zusammenhang mit der Theorie der Meßfehler eingeführt. Sie ist unter anderem aus folgenden Gründen die wichtigste stetige Verteilung:

- Viele Zufallsvariable, die bei Experimenten und Beobachtungen in der Praxis auftreten, sind normalverteilt
- Andere Zufallsvariable sind annähernd normalverteilt
- Gewisse nicht normalverteilte Variable lassen sich auf verhältnismäßig einfache Weise derart transformieren, daß die sich ergebende Variable normalverteilt ist
- Manche komplizierte Verteilungen lassen sich in Grenzfällen durch die Normalverteilung brauchbar annähern

Die Normalverteilung ist durch folgende Gleichung gekennzeichnet:

$$Y = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}(X-\mu)^2/\sigma^2}$$

wobei μ der Mittelwert und σ die Standardabweichung ist.

Neben diesen jedermann bekannten Verteilungen existieren auch noch benutzerdefinierte Verteilungen. Zu diesen ist als erstes festzustellen, daß es dem Simulationsexperten überlassen bleibt, ob er versucht in einen vorhandenen Datensatz eine Standardverteilung einzupassen oder nicht. Die Alternative ist, mittels der

vorliegenden Daten eine eigene (diskrete oder stetige) Verteilung zu definieren. Dies bietet den Vorteil, flexibler zu sein und ohne große statistische Untersuchungen ermittelbar zu sein. Dagegen spricht allerdings, daß der größte Wert, der generiert werden kann, durch den größten Wert des Datensatzes determiniert ist. Dies kann in manchen Fällen eine unzulässige Einschränkung bedeuten. Dem kann abgeholfen werden, indem man an die eigendefinierte Verteilung eine Standardverteilung anhängt, durch die mit einer gewissen Wahrscheinlichkeit Werte außerhalb des beobachteten Bereiches erreicht werden können (vgl. /KEL84/).

Neben den bereits erwähnten Verteilungen stehen noch viele andere (wie z. B. Gamma- und Binomialverteilung) zur Verfügung und es ist ebenso eine ausreichende Anzahl an statistischen Tests zur Überprüfung vorhanden. Man muß sich immer bewußt sein, daß die Entscheidung, ob Daten in Form einer Verteilung vorgegeben werden, oder ob die Abbildungsgenauigkeit des Systemverhaltens mit einem Mittelwert ausreicht, unmittelbar die Qualität der Ergebnisse und den Simulationsaufwand beeinflußt.

Ablaufregeln

Neben den qualitativen Größen, die ein System charakterisieren, müssen auch die Ablaufregeln wie z. B. Vorfahrtsstrategien, Bearbeitungsreihenfolgen und Dispositionsregeln definiert werden. Bei bestehenden Systemen werden diese Regeln meist aus dem täglichen Betrieb abgeleitet und oft erstmalig formuliert. Für Neuplanungen müssen diese Regeln vom Anwender verbindlich festgelegt werden, um die Simulation zu ermöglichen und sinnvoll zu gestalten.

Die Aufbereitung beginnt mit der Selektionsphase, in der die modell- und zielrelevanten Daten ausgewählt werden. Nachdem der Planer das Datenmaterial auf Plausibilität und Richtigkeit geprüft hat, erfolgt eine Klassifizierung und Verdichtung und es kommt zur eigentlichen Modellbildung, deren Aufwand durch die Komplexität des abzubildenden Systems und durch den Grad der Detailierung bestimmt ist. Hierauf kann die abstrahierte Darstellung entsprechend der benutzten Simulationssoftware auf dem Computer implementiert werden.

Als nächstes kommt es zur Validierung, bei der überprüft wird, ob das implementierte Modell mit dem abzubildenden System übereinstimmt. Dieser Schritt ist unerläßlich,

um eine korrekte Interpretation der Ergebnisse zu ermöglichen. Nach erfolgter positiver Validierung kommt es zu der Durchführung von Experimenten.

Experimentieren bedeutet, sinnvolle Versuchsreihen mit der Variation einzelner Parameter aufzustellen, durchzuführen und Ergebnisse zu erzeugen. Es bedeutet nicht, experimentieren im Sinne von ausprobieren. Für eine sinnvolle Experimentierarbeit, die durchaus auch schon die Verwendung eines Genetischen Algorithmus einschließen kann, sind folgende Punkte zu empfehlen:

- Experimentreihen mit dem Anwender sinnvoll abstimmen
- Nicht mehrere Parameter gleichzeitig ändern, da sonst eine eindeutige Interpretation der Ergebnisse unmöglich ist
- Dokumentation der Experimente, um eine Reproduzierbarkeit der Ergebnisse zu gewährleisten
- Änderungen im Simulationsmodell erfordern einen neuen Zyklus Implementieren-Validieren-Experimentieren

Interpretation

Bei der Interpretation der Experimente ist gesteigerter Wert auf eine sorgfältige Outputdatenanalyse (vgl. /KEL83/) zu legen, da sie in vielen Fällen sträflich vernachlässigt wird, weil man sich meistens schon mit der Implementierung des Modells und einigen Experimenten zufriedengibt. Dabei steht und fällt jede diskrete Simulation, die auf stochastischen Prozessen basiert, mit der Beurteilung der Allgemeingültigkeit der Ergebnisse. Dies gilt natürlich umso mehr, wenn man noch den Oberbau des Genetischen Algorithmus in Betracht zieht.

Es kann vorkommen, daß verschiedene Testläufe desselben Programms nicht nur abweichende, sondern sich diametral gegenüberliegende Ergebnisse liefern. Man stelle sich nur ein Problem vor, bei dem es darum geht, von mehreren gegebenen Lösungen die günstigere zu ermitteln. Es genügt in diesem Fall schon, wenn zwei oder mehrere Lösungen „benachbart“ sind, um mit einer gewissen Wahrscheinlichkeit die ungünstigere Lösung auszuwählen. Es liegt in der Natur von stochastischen Prozessen, daß man derlei nie mit 100%-iger Sicherheit ausschließen kann, aber es gilt für den

Simulationsexperten einiges zu berücksichtigen, um nicht fahrlässig Verfälschungen des Ergebnisses in dieser Richtung zu provozieren.

Bei jeder Outputdatenanalyse sollte man davon ausgehen können, daß die Eingangsverteilungen stimmen, allerdings muß man sich der Tatsache bewußt sein, daß Simulationssprachen Verteilungen nicht mathematisch behandeln, sondern nur Zahlenwerte generieren, die einer bestimmten Realisierung der Zufallsvariablen entsprechen. Im Gegensatz zu statischen Systemen tritt bei dynamischen Systemen eine Zeitabhängigkeit der Ergebnisse auf, die das Analysieren nicht einfacher macht. Das bedeutet, daß die relevanten Größen sich mit der Zeit ändern und es daher wichtig wäre, zu wissen, ob es einen Gleichgewichtszustand gibt oder nicht.

Hauptsächlich werden sich die Bemühungen darauf konzentrieren, für die Parameter von Interesse ein Konfidenzintervall zu ermitteln, mittels dem man sagen kann, daß die gesuchten Parameter mit einer gewissen Wahrscheinlichkeit in einem bestimmten Bereich liegen.

Zur Erklärung des Begriffes eines Konfidenzintervalls gehen wir von Schätzwerten $T_n(x_1,...,x_n)$, die in der Regel vom zu schätzenden Parameter θ abweichen, aus (vgl. /LEWE92/). Es stellt sich dann die Frage nach Schranken, zwischen denen der unbekannte Parameter mit einer gewissen Sicherheit liegt. Man soll daher ein möglichst kleines Intervall $I(x_1,...,x_n)$ berechnen, daß zu vorgegebenem α mit einer Wahrscheinlichkeit $>1-\alpha$ den wahren, aber unbekannten Parameter enthält. Ein solches Intervall heißt Konfidenzintervall für den Parameter θ und $1-\alpha$ heißt Konfidenzniveau.

Diese Vorgehensweise ist aussagekräftiger als das Anwenden von Tests, ob eine Hypothese angenommen oder verworfen werden sollte. Das deshalb, weil bei der Bestimmung von Konfidenzintervallen die Varianz (und nicht nur der Erwartungswert) eine bedeutende Rolle spielt. Die Varianz ist es auch, die ein Ziel von Bemühungen sein kann, wenn es darum geht, möglichst aussagekräftige Ergebnisse zu erhalten.

Es gibt nämlich einige spezielle Techniken, die Varianz zu reduzieren, was klarerweise einen positiven Einfluß auf das Konfidenzintervall hat. Unter diesen Techniken ist jene am leichtesten zu verstehen, die sich antithetischer Zufallszahlen

bedient. Bei dieser Methode werden zusätzlich Simulationsläufe durchgeführt, bei denen die Zufallszahlen genau entgegengesetzt gewählt werden.

Wenn man sich vor Augen führt, daß üblicherweise ein Zufallszahlengenerator eine gleichverteilte Zahl aus dem Intervall *[0,1]*, die dann durch computerinterne Algorithmen in eine Zahl aus dem gewünschten Bereich umgewandelt wird, liefert, so wird bei einem antithetischen Lauf zum Beispiel der Wert *0.221* genommen falls ursprünglich *0.779* Verwendung gefunden hat. Dadurch läßt sich im allgemeinen eine Varianzreduktion erreichen, da folgendes gilt:

$$Var(X+Y) = Var(X) + Var(Y) + 2*Cov(X,Y)$$

wobei die Kovarianz folgendermaßen definiert ist:

$$Cov(X,Y) = E([X-E(X)]*[Y-E(Y)]).$$

Je negativer nun der sogenannte Korrelationskoeffizient

$$\rho(X,Y) = \frac{Cov(X,Y)}{\sqrt{Var(X)\cdot Var(Y)}}$$

ist, desto besser ist es für die Summenvarianz der beiden Simulationsteilläufe. Es kann allerdings auch der (in derlei Anwendungen) seltene Fall eintreten, daß die beiden Teilläufe positiv korreliert (d.h. *Cov(X,Y)* positiv) sind, was eine Varianzerhöhung bedeuten würde.

Ein weiteres Problem der Outputdatenanalyse ist auch jenes, daß bei gleichzeitiger Betrachtung mehrerer wichtiger Outputkenngrößen das Konfidenzniveau der Konfidenzintervalle nachhaltig beeinflußt wird. Nehmen wir z. B. an, daß uns vier Größen interessieren und daß wir für jede ein Konfidenzintervall ermitteln, so daß wir nur mit einer Wahrscheinlichkeit von 5% den richtigen Wert nicht abdecken (d. h. Konfidenzniveau 0.95). Dann ergibt sich allerdings, daß beim Gesamtkonfidenzintervall die Wahrscheinlichkeit 0.20 ist, daß wir nicht das richtige „Parameterquartett“ abdecken. Ein Gesamtniveau von 0.95 kann man nur durch Erhöhung der Einzelkonfidenzniveaus auf 0.9875 erreichen. Das bedeutet im Gegenzug aber entweder eine Vergrößerung der Konfidenzintervalle oder eine Erhöhung der Datenmenge.

Ergebnisdarstellung

Die Simulationsergebnisse liegen zunächst in tabellarischer Form vor, die für die Interpretation verdichtet und aufbereitet werden müssen. Gängige Arten der Ergebnisdarstellung sind unter anderem:

- Liniendiagramme für die Veranschaulichung von sich zeitlich ändernden Größen
- Balkendiagramme zur Darstellung von absoluten und relativen Häufigkeiten
- Kreisdiagramme zur Visualisierung relativer Gewichtsanteile

Sehr anschauliche Information über das Einsatzverhalten von Anlagen liefert die Prozeßvisualisierung oder Animation. Sie unterstützt den Simulationsexperten bei der Validierung, bietet dem Planer und Betreiber eine Hilfestellung zum Begreifen der Vorgänge in seinem System und dient als Kommunikationsmittel zwischen Simulationsexperten und Nutzer. Es sei darauf hingewiesen, daß die Qualität der Animation nichts über die Güte und Richtigkeit der Simulation aussagt.

Eine richtig durchgeführte Simulationsstudie liefert in jedem Fall ein Ergebnis, sei es den Nachweis der Funktionalität oder den der Nichtfunktionalität bei bestimmten Betriebszuständen. Die Aufgabe der Ergebnisinterpretation ist die Untersuchung der Zusammenhänge zwischen den Ergebnissen einzelner Simulationen. Erst durch sorgfältige Durchführung dieses letzten Schrittes wird eine Simulation abgeschlossen.

Wie man sieht, sind die Möglichkeiten mannigfaltig, und trotzdem werden sie häufig aus unterschiedlichen Gründen nicht oder nicht ausreichend ausgeschöpft:

- Einsatzmöglichkeiten und Nutzungspotentiale sind zu wenig bekannt
- Das Wissen über Simulationsmethoden und -werkzeuge ist zu wenig verbreitet
- Planer beteiligen sich zu wenig
- Simulationsmodelle werden selten mehrfach eingesetzt; es gibt viele „Wegwerfmodelle"
- Simulationsuntersuchungen werden oft zu spät, d. h. kurz vor Planungsabschluß in Auftrag gegeben

Der Aufwand für Simulationsprojekte ist je nach Art oder Einsatzfall sehr unterschiedlich und eine gezielte Kostenanalyse gestaltet sich oft sehr schwierig, der Nutzen kann sich unter anderem in folgenden Punkten äußern:

- Frühzeitiges Erkennen von Fehlpanungen und damit Minderung des Planungsrisikos
- Gewinnen quantifizierbarer Ergebnisse für eine objektive Argumentation und Diskussion
- Erlangen eines Gesamtverständnisses für das System

Als Resümee läßt sich sagen, daß ein erfolgreicher Einsatz der Simulation wesentlich davon abhängt, wie gut die Zusammenarbeit zwischen Planer und Simulationsexperten ist, es kommt daher darauf an, die Simulation rechtzeitig, d.h. planungsbegleitend und in laufender Abstimmung durchzuführen. Der Einsatz der Simulation zwingt zu genauerem Durchdenken der Planungsaufgabe und zu genaueren Daten, wodurch die Qualität der Planung im Normalfall erhöht wird.

Simulationswerkzeuge

Wenn wir von Simulationswerkzeugen sprechen, so meinen wir im Normalfall speziell adaptierte Computerprogramme, die für den Zweck der Simulation besonders geeignet sind. Im verallgemeinerten Sinn kann man darunter natürlich auch Modelle verstehen, die für die Analyse relevanter Vorgänge konstruiert wurden. Im Extremfall ist auch der menschliche Verstand (unter Umständen in Verbindung mit Papier und Bleistift) als Simulationswerkzeug zu verstehen, da die Modelle schlicht und einfach durchkalkuliert werden können, was dem Sinn einer Simulation durchaus entspricht.

Üblicherweise sind die Probleme mit denen man es zu tun hat allerdings so rechenintensiv und/oder unüberschaubar, daß man sich gerne der Hilfe von Computern bedient. Mit steigender Leistungsfähigkeit (Geschwindigkeit, Speicherkapazität, etc.) der Computer(systeme) änderten sich auch die zugehörigen Anwenderprogramme, die in steigendem Maße benutzerfreundlicher wurden (oder werden wollten).

Es gibt nun im Bereich der Simulationssprachen einige Arten von Klassifizierungen, die parallel zueinander existieren, wovon eine die Einteilung nach Generationen ist

(vgl. u.a. /REG94/). Solche Klassifizierungen sind immer schwer zu treffen und jede Form der Kategorisierung im Bereich der Simulation hat üblicherweise mehr Gegner als Anhänger. Exemplarisch sei hier dennoch diese Klassifizierung wiedergegeben, wonach bis jetzt fünf Generationen von Simulationssystemen existieren.

Der ersten Generation wird keinerlei Unterstützung der Modellbildung und Erleichterung der Programmierarbeit unterstellt. Es sind herkömmliche Programmiersprachen, wie z. B. FORTRAN oder C. Das impliziert natürlich ein langwieriges und fehleranfälliges Vorgehen.

Die zweite Generation besteht aus einfachen Simulationssprachen, die streng zwischen kontinuierlichem und diskretem Zeitverhalten unterscheiden. Es werden einfache Verfahren zur statistischen Ergebnisanalyse und -darstellung sowie Zeitfortschaltung in der Ablaufkontrolle und Generierung von Zufallszahlen angeboten. Repräsentanten dieser Spezies sind unter anderem SIMSCRIPT, SIMULA und GPSS.

In der dritten Generation wird Augenmerk auf die Kombination von diskreten und kontinuierlichen Elementen gelegt. Damit wird es möglich, Unstetigkeitsstellen besser zu modellieren (üblicherweise in sogenannten „Derivative Sections“). Als Vertreter sind hier hauptsächlich ACSL, GASP IV und GPSS-FORTRAN zu nennen.

Die darauffolgende vierte Generation stellt bereits sehr leistungsfähige Systeme zur Verfügung, die meistens aber auf ein bestimmtes eng begrenztes Anwendungsfeld konzentriert sind. Als einer der herausragenden Neuerungen ist hier die erstmalige Einführung einer Animation zu sehen, die als nicht zu unterschätzende Brückenfunktion zwischen Ausführer und Auftraggeber einer Simulationsstudie zu bewerten ist. Dadurch wird es möglich, Leuten mit wenig Interesse in unübersichtliche Zahlenkolonnen mittels motivationsunterstützender Grafiken die Zusammenhänge und Problematiken auf simplifizierte Art schneller näher zu bringen, als dies sonst möglich wäre.

Dies ist um so mehr wichtig, wenn man die Akzeptanzprobleme, mit denen die Simulation an sich in manchen Kreisen zu kämpfen hat, bedenkt. Als Vertreter sind hier unter vielen anderen GPSS/H in Verbindung mit PROOF, MicroSaint, XCELL, SIMFACTORY oder SEE WHY angeführt.

Vertreter der fünften Generation, wie etwa SIMPLE++, CREATE!, SIMPLEX II, ARENA, werden seit ungefähr 1990 vertrieben. Sie zeichnen sich durch Bereitstellung einer integrierten Modellerstellungsumgebung, objektorientierter Modellspezifikation, grafischer Benutzeroberfläche und durch die Konzeption als offene Systeme aus. Das alles soll zu mehr Transparenz bei der Modellerstellung und mehr Portabilität einzelner Systemkomponenten beitragen.

Weiters ist heutzutage die direkte Verknüpfung der Simulation mit anderen Bereichen, wie etwa Datenbanken oder Steuereinrichtungen, wichtiger denn je, daher gibt es die Bestrebung, neue Simulationssysteme als offene Systeme zu konzipieren. Dem Trend der Zeit folgend ist auch die Entwicklung von benutzerfreundlichen Bedienungsoberflächen logisch. Man denke nur an die steigende Zahl von Anwenderprogrammen aus allen Bereichen der Computertechnologie, die vermehrt auf „Windows-ähnlichen“ Oberflächen aufsetzen.

Daß der Übergang zwischen den einzelnen Generationen fließend ist und oft eine genaue Einteilung nicht möglich ist, bräuchte nicht extra erwähnt zu werden, ist aber trotzdem eine Tatsache, die von Anwendern gerne übersehen wird. Wenn man von Generationen der Simulationssysteme spricht, ist weiters wichtig anzumerken, daß Mitglieder einer anderen als der fünften Generation bei weitem nicht als veraltet gelten müssen. Das dies so sei, ist ein weit verbreiteter Irrglaube, der naturgemäß von Anhängern oder Distributoren der fünften Generation geschürt wird. Allzu oft wird nämlich vergessen, daß Simulationssprachen der ersten vier Generationen nicht zwangsläufig veraltet sein müssen, da auch diese regelmäßige Updates und Verbesserungen erfahren, aber eine andere Arbeitsweise und/oder Sichtweise als die Vertreter der fünften Generation besitzen.

Mit den Systemen der fünften Generation kann man zwar wunderbar einfach Modelle erstellen, ohne eine grundlegende Programmiererfahrung besitzen zu müssen, aber bei Anwendungen, die vom „Mainstream“ der Applikationen abweichen hat man oft mit unerwarteten und/oder unüberwindbaren Schwierigkeiten zu kämpfen. Außerdem verleitet manches System einer höheren Generation oft nur zu leicht zu einfachen „Spielereien“, ohne sich geeignet kritisch mit dem eigentlichen Sinn der Simulation bzw. Optimierung auseinandergesetzt zu haben. Da bietet oft ein „herkömmliches“ Simulationspaket (=Vertreter einer niedrigeren Generation) die einfachere und bessere

Lösung, ist oft aber nur nach längerer Einarbeitungszeit wirklich sinnvoll zu verwenden (man denke an das Erlernen einer Programmiersprache!).

Es gilt also bei der Wahl der Simulationssprache, bzw. des Simulationssystems, einen Kompromiß zwischen Benutzerfreundlichkeit, Kosten (es sei erwähnt, daß Systeme der fünften Generationen oft das Vielfache an Anschaffungskosten haben als Mitglieder einer anderen Generation) und Nutzen zu finden.

Spezielle Optimierungsaspekte

In diesem Kapitel soll auf weitere Aspekte, die den Bereich der Genetischen Algorithmen betreffen, eingegangen werden. Es liegt in der Natur der Sache, daß dies teilweise nur motivierenden Charakter besitzen kann, da in letzter Zeit die Publikationen und Theorien über Genetic Algorithms oder Genetic Algorithms - verwandte Themengebiete sprunghaft zugenommen haben.

Für weiterführende Vertiefungen in einzelnen Teilaspekten sei auf die umfangreiche Literaturliste verwiesen, bei deren Erstellung versucht wurde, für die verschiedenen Anwendungsbereiche und Theoriearbeiten einige Beispiele aufzunehmen.

Auf den folgenden Seiten soll versucht werden, eine Sensibilisierung für die mannigfachen Aspekte und Probleme, die im Zusammenhang mit Genetischen Algorithmen auftauchen, zu erreichen.

Fitneß

Wir haben den Begriff der Fitneß schon in einem vorangegangenen Kapitel eingeführt und wollen uns nun näher mit den dazugehörigen Einzelheiten beschäftigen. Wie unmittelbar einsichtig ist, und wie anhand von zahlreichen Werken belegt ist (vgl. /BBM93a/), ist dies ein Kernbereich der Forschung über GA. Um es plakativ auszudrücken, kann man sagen, daß der beste Algorithmus nichts nützt, wenn die Zielsetzung schlecht ist. Unter dieser Zielsetzung ist nicht nur die eigentliche Zielfunktion gemeint, von der wir hier annehmen können, daß sie geeignet gewählt wurde, sondern vielmehr eine Skalierung dieser Funktion.

Rufen wir uns zur Motivation das Beispiel aus einem vorangegangenen Kapitel in Erinnerung: Es geht um die Optimierung einer quadratischen Funktion in einem bestimmten Intervall. Wir wählen der Einfachheit halber die zu optimierende Funktion als Fitneßfunktion und erreichen damit gute Resultate. Es ist aber nicht unbedingt gesagt, daß wir mit anderen Fitneßfunktionen nicht schneller ans Ziel kommen. Halten wir uns vor Augen, was eine Fitneßfunktion für dieses Beispiel leisten muß und welche Konsequenzen sich daraus ergeben.

Offensichtlich soll ein Element einer Population, daß einen besseren Wert in der Zielfunktion besitzt, auch eine höhere Fitneß aufweisen. Weiters ist natürlich ein Wert aus den positiven reellen Zahlen wünschenswert, um das Auswahlverfahren gemäß der Fitneß einfach gestalten zu können. Sehen wir uns ein paar einfache Beispiele an, die obengenannte Bedingungen erfüllen und somit als mögliche Fitneßfunktion in Betracht zu ziehen sind:

1. Die Funktion $x \rightarrow x^{10}$ erfüllt die oben genannten Bedingungen, wir erreichen dadurch mögliche Fitneßwerte von 1 bis 31^{10}, was einen enormen Unterschied der Werte bedeutet. Wenn wir nun an die Auswahlmechanismen des GA denken, die bessere Elemente klarerweise bevorzugen, wird klar daß hier ein extrem starker Auswahldruck hin zu den guten Elementen einer Population gegeben ist. Da das nicht bei jeder üblichen Anwendung eines Genetischen Algorithmus wünschenswert ist, kann man sagen, daß durch so eine Wahl der Fitneßfunktion dem GA kein guter Dienst erwiesen wird. In dem hier vorgestellten Beispiel würde diese Fitneßfunktion zwar besser funktionieren, allerdings muß man bedenken, daß dieser Fall nicht der Prototyp von typischen GA-Anwendungen darstellt.

2. Betrachten wir nun einmal ein gegenteiliges Beispiel und nehmen wir als Fitneßfunktion $x \rightarrow x+1000$, wodurch wir Werte aus dem Intervall *[1000,1031]* erhalten. Diese Werte liegen nun wieder so nahe beisammen, daß sich die bestmögliche Lösung nur um eine *3,1 %* bessere Fitneß auszeichnet. Das führt natürlich dazu, daß jedes Element einer Population nahezu die gleichen Chancen hat, ausgewählt zu werden und der GA ist defacto ein reiner Zufallsprozeß geworden.

Aus diesen Überlegungen heraus scheint es naheliegend, der Fitneßfunktion und ihrer geeigneten Auswahl erhöhte Aufmerksamkeit zu schenken. Es sind also hauptsächlich zwei Dinge, die wir bei der Wahl der Fitneßfunktion vermeiden sollten:

- Vorzeitige Konvergenz: Darunter versteht man das Verfehlen des globalen Optimums durch Verharren in einem lokalen Optimum oder Aussterben von Elementen, welche Komponenten enthalten, die zum globalen Extremum führen würden. Dieses Verhalten wird durch eine Fitneßfunktion, wie sie im ersten Beispiel konstruiert worden ist, stark begünstigt, da dort der Selektionsdruck hin

zum fitteren so stark ist, daß eine gewisse Diversität, die ein Markenzeichen eines erfolgreichen GA ist, nicht gewährleistet ist.

- Langsames Finish: darunter versteht man die Tendenz bei Genetischen Algorithmen gegen Ende eines erfolgreichen Optimierungsversuches zu lang zu benötigen, um das konkrete Optimum zu lokalisieren, obwohl sich der Algorithmus schon in der Nähe dieses gesuchten Wertes befindet. Dieses Phänomen tritt dadurch auf, daß gegen Ende einer Optimierung durch einen GA angenommen werden kann, hauptsächlich gute Lösungen in der Population zu haben, die sich wenig in ihren Fitneßwerten unterscheiden. Dadurch ist allerdings kein großer Selektionsdruck hin zum Optimum gegeben und der Algorithmus sucht mehr ziellos in der Umgebung, als er gezielt vorgeht.

Um diese unerwünschten Nebeneffekt auszuschalten, gibt es einige Methoden und Theorien, von denen wir hier kurz ein paar näher betrachten wollen.

Fitneßskalierung

Bei dieser Technik wird durch Subtraktion oder Addition einer Zahl eine Verschiebung des Wertebereiches erreicht. Es gibt auch hier mehrere Vorgangsweisen, die sich entweder am fittesten oder am schlechtesten Element orientieren. Man kann zum Beispiel folgendermaßen vorgehen:

Gegeben sei eine Population, deren bestes Element eine Fitneß von f_{max} aufweist und deren durchschnittlicher Fitneßwert bei f_d liegt. Falls nun der Quotient aus $f_{max}:f_d$ zu klein ist, was bedeuten würde, daß ein gutes Element einer Population eine Wahrscheinlichkeit zur Selektion besitzt, die sich nicht um allzu viel von der der schlechteren unterscheidet, kann durch Subtraktion eines Wertes (z.B.: $w = 2* f_d - f_{max}$) von jedem Element der Population (d.h. $f_i - w$ für $i=1,...n$) eine Verbesserung der Situation erreicht werden.

Üblicherweise wird in diesem Zusammenhang ein Verhältnis $f_{max}:f_d$ von 2 angestrebt, wobei aber auf jeden Fall ein negativer Wert für die Fitneß vermieden werden muß. Analoges Vorgehen mit Addition einer geeigneten Konstante ist für den Fall vorgesehen, daß das Verhältnis zu groß ist und dadurch eine übermäßige Betonung der guten Elemente gegeben wäre.

Anstatt sich an den besten Lösungen zu orientieren, kann man auch den umgekehrten Weg gehen und die schlechtesten Elemente der Populationen als Maßstab heranziehen (vgl. /GRE84/):

Man könnte z. B. über eine gewisse Anzahl von Populationen (als Richtwert wird hier von den jeweils zehn letzten Generationen ausgegangen) den schlechtesten auftretenden Fitneßwert bestimmen und diesen von den restlichen Fitneßwerten f_i zum Abzug bringen. Dadurch ändert sich klarerweise laufend die Skalierung und man erreicht dadurch eine gewisse Flexibilität über den Verlauf des Optimierungsprozesses.

Ein beträchtlicher Nachteil der oben erwähnten Methoden ist nun jener, daß die Skalierung (eigentlich kann man nur von einer Verschiebung sprechen) sehr stark von den jeweils besten oder schlechtesten Elementen der Population abhängt. Bei Auftreten extrem fitter oder extrem unfitter Elemente neigen obige Verfahren zu Verzerrungen, deren Vermeidung eigentlich Ziel des Skalierens war.

Als Kompromiß bietet sich an, eine tatsächliche Skalierung, die sowohl die additive Komponente als auch einen multiplikativen Faktor enthält, zu wählen. Das bedeutet nichts anderes, als daß man eine geeignete Abbildungsfunktion der tatsächlich erhaltenen Werte in ein bestimmtes Intervall sucht, die gegebenen Bedingungen genügen soll. Einziger Nachteil dieser Idee ist die etwas komplexer zu programmierende Implementierung dieses Vorgangs.

Reihung-nach-Fitneß

Eine einfache Möglichkeit, allzu komplizierte Berechnungen zu vermeiden und trotzdem eine gewisse Steuerung zu erreichen, ist die, daß man alle Elemente einer Population aufsteigend nach ihrer Fitneß reiht. Anschließend legt man fest, wie groß die Wahrscheinlichkeit zur Selektion für die einzelnen Ränge sein soll. Dies kann in linear absteigender oder auch in anderer geeigneter Form stattfinden.

Hiermit wird erreicht, daß nicht mehr die Fitneß direkt entscheidend ist, sondern der auf Grund der Fitneß ermittelte Rang. Es ist also nebensächlich, um wieviel ein Element einer Population besser ist als ein anderes, es zählt nur mehr, daß es besser

ist. Dadurch wird der verzerrende Effekt, der von einigen extremen Individuen ausgehen kann, kompensiert.

Es gibt noch weitere grundlegende Dinge in diesem Zusammenhang, auf die wir unter anderem in den Abschnitten über Selektionsmethoden und über unzulässige Bereiche zu sprechen kommen.

Crossover-techniken

Im ersten Abschnitt haben wir als Crossover-technik das Einpunktcrossover kennengelernt, bei dem zwei Elemente einer Population an der gleichen Stelle getrennt werden und wieder zu neuen Elementen zusammengesetzt werden. Dieses Verfahren läßt sich natürlich noch verallgemeinern, um spezielle Anwendungen unter Umständen besser zu unterstützen.

Mehrpunktcrossover

Wie der Name schon sagt, wird hier nicht **eine** Trennstelle, sondern mehrere Stellen werden als Trennpunkte gewählt. Außerdem wird üblicherweise davon ausgegangen, daß man sich die Chromosomen als ringförmig geschlossene Einheiten vorstellt, was bedeutet, daß der Anfang eine Chromosoms mit seinem Ende zu einer zyklischen Struktur verbunden wird.

Betrachten wir zur Erläuterung den Fall des bedeutensten Vertreters dieser Art, dem des Zweipunktcrossovers:

Gegeben seien die beiden Strings $(a_1,,...a_k,,...a_p,,...a_n)$ und $(b_1,,...b_k,,...b_p,,...b_n)$ mit den jeweiligen Crossoverstellen k und p. Durch die dieser Vorgehensweise zugrunde liegende Annahme der zyklischen Beschaffenheit der Elemente ist das Bit a_1 dem Bit a_n benachbart und Analoges gilt für den zweiten String. Nach dem das Crossover durchgeführt worden ist, erhält man als neue Elemente $(b_1 ,,...b_{k-1},a_k ,,...a_{p-1} ,b_p ,...,b_n)$ und $(a_1 ,,...a_{k-1},b_k ,,...b_{p-1} ,a_p ,...,a_n)$.

Wie man leicht erkennt, wird auch hier jeweils ein Teilstring des anderen Elements übernommen, der Unterschied besteht hauptsächlich darin, daß bei dieser Vorgehensweise mehr sogenannte Bausteine zur Verfügung stehen, da durch die zyklische Struktur Bausteine über das Ende eines Strings bis zum Beginn desselben

gehen können. Wie wir in einem vorangegangenen Kapitel gesehen haben, wird das ja als einer der Hauptgründe für die erfolgreiche Arbeitsweise von GA angesehen (vgl. Bausteinhypothese).

Da wir vom Mehrpunktcrossover sprechen, darf in diesem Zusammenhang nicht unerwähnt bleiben, daß einige Untersuchungen auf diesem Gebiet gezeigt haben, daß es in den meisten Fällen nicht viel bringt, mehr als zwei Trennstellen zu wählen, da dann der Trend zur Zerstörung erfolgreicher Bausteine zu groß ist. Auch ist der Unterschied in der Performance des Zweipunktcrossovers im Vergleich mit dem Einpunktcrossover nicht allzu bedeutend, aber für manche spezielle Anwendungen doch deutlich meßbar.

Es wird in Fachkreisen zur Zeit davon ausgegangen, daß das Zweipunktcrossover üblicherweise leichte Vorteile genießt, die aber in manchen Applikationen sich umkehren können. Man könnte das Einpunktcrossover folgerichtigerweise auch als Zweipunktcrossover mit einer fixen Trennstelle zu Beginn jedes Strings interpretieren, womit das meiste über den Unterschied der beiden Methoden heuristisch ausgedrückt ist.

Schablonencrossover

Diese Art unterscheidet sich von den bisher vorgestellten Techniken am meisten allein durch die Tatsache, daß keine Teilstrings per se ausgetauscht werden, sondern nur Bits. Es wird vor jedem Crossover-vorgang eine Schablone mittels eines Zufallsgenerators ermittelt, die die Länge der zu kreuzenden Strings besitzt und aus binären Elementen besteht. Diese Schablone denkt man sich über die zwei zu kreuzenden Elemente gelegt, wobei nun eine 1 an einer Position bedeutet, daß das Kindelement an dieser Stelle die Information des ersten Strings erhält, eine 0, daß das Kindelement die Information des zweiten Strings bekommt.

Nachdem man diesen Vorgang für alle Positionen durchgeführt hat, vertauscht man die beiden Elternstrings und ermittelt mittels derselben Schablone das zweite Kindelement. Betrachten wir zur Verdeutlichung folgendes einfaches Beispiel (vgl /BBM93a/):

Schablone 1001011100

Elternteil 1	1010001110
Elternteil 2	0101010011

Daraus erhält man durch oben beschriebenes Procedere die beiden Kinder:

Kind 1	1100001111
Kind 2	0011010010

Es wird deutlich, daß durch ein solches Vorgehen die Vermehrung von erfolgreichen Bausteinen gehemmt wird, da es selten vorkommt, daß längere Passagen unverändert übernommen werden. Hier liegt auch der große Kritikpunkt, den manche dieser Technik vorhalten, da mit vergleichsweise mehr Aufwand für viele Applikationen weniger erreicht wird. Der unbestrittene Vorteil liegt darin, daß man sich um die Anordnung der Gene am Chromosom keine Sorgen machen braucht.

Andere Crossover-techniken

Neben den bereits erwähnten Verfahren gibt es in den unzähligen Applikationen und den dazugehörigen Publikationen eine große Anzahl verschiedener Vorschläge für Crossover-verfahren. Viele davon sind für spezielle Zwecke konstruiert worden und somit für die allgemeine Verwendung nicht so gut geeignet wie die eingangs erwähnten.

Es ist oft der Phantasie des Anwenders überlassen, die geeignete Technik zu wählen, so ist zum Beispiel durchaus denkbar, nicht nur zwei Elemente zu einem Crossover-prozeß zuzulassen, sondern deren mehrere, wodurch sich allein aus diesem Umstand heraus einige neue Perspektiven und beachtenswerte Unterschiede ergeben, auf die wir hier allerdings nicht näher eingehen wollen.

Hingegen erwähnenswert in diesem Zusammenhang scheint noch, daß es Ansätze gibt, Unterschiede in den möglichen Crossover-stellen dahingehend zu machen, daß nicht jeder Punkt mit der gleichen Wahrscheinlichkeit als Trennstelle dienen soll. Wenn man sich den Aufbau eines Chromosoms aus einzelnen Genen, die jeweils einen bestimmten zu optimierenden Parameter repräsentieren, in Erinnerung ruft, so kann man die Argumentation nachvollziehen, daß es Vorteile bringen kann, nicht innerhalb von Genen, also zwischen beliebigen Allelen auf dem Chromosom, eine Trennstelle

zu plazieren, da dadurch ja eine zusammengehörige Ansammlung von Bits unterbrochen wird. Unter Umständen können hier gute Entwicklungen für einen konkreten Parameter gestoppt werden.

Gegenargument ist hier, daß durch eine solche Vorgangsweise die Anzahl der möglichen Bausteine drastisch reduziert wird und somit eine Schwächung des Verfahrens vorliegt. Allgemeingültige Aussagen sind hier schwer zu treffen, es sei aber in diesem Zusammenhang auf den Abschnitt über Anordnung der Gene in einem Chromosom verwiesen.

Selektionsmethoden

Dieser Bereich ist einer der Zentralaspekte der Theorie der Genetischen Algorithmen und nahe mit jenem der Fitneß verwandt. Ein GA steht und fällt mit der Art, wie er seine Auswahl unter den zur Verfügung stehenden Elementen einer Population zur Weitergabe von Information oder zum Weiterverbleib in der nächsten Generation trifft. Peter Hancock hat in dieser Richtung empirische Untersuchungen durchgeführt und eine Einteilung der weitverbreitetsten Methoden gegeben. Für die genauen Ergebnisse sei auf seine Arbeit (/HAN94/) verwiesen.

An dieser Stelle soll nur ein Überblick der verschiedenen Möglichkeiten geboten werden. Neben den bereits im Abschnitt über Fitneß erwähnten Verfahren der fitneßproportionalen Selektion und der Selektion aufgrund von Fitneßwerten ermittelten Reihenfolge sei hier noch die folgende Idee erwähnt.

Wettkampfselektion

Bei dieser Auswahlmethode wird eine bestimmte Anzahl von Elementen der Elternpopulation ausgewählt, die insofern gegeneinander kämpfen, als nur der beste Teilnehmer zur Reproduktion herangezogen wird. Das bedeutet natürlich nicht, daß die anderen Elemente endgültig aus dem Rennen sind, da dieser Vorgang sooft wiederholt wird, bis eine ausreichende Anzahl von Individuen zur Reproduktion ausgewählt wurden. Auf jeden Fall hat das schlechteste Individuum praktisch keine Chancen, in die nächste Generation zu gelangen, außer es gibt mehrere gleichschlechte Elemente, die zufällig zusammengelost werden.

Dieses Verfahren ähnelt der Methode, wo auf Grund der Fitneß eine Reihenfolge ermittelt wird, und dementsprechend ausgewählt wird, allerdings ist der vorhin erwähnte Unterschied zu beachten. Durch die Anzahl der zu einem Wettkampf ausgewählten Individuen wird eine Steuerung des Verfahrens erreicht, eine größere Anzahl bewirkt eine Tendenz hin zu den fitteren Elementen, wo hingegen ein Zweikampf auch schlechteren eine gute Chance läßt, da beim Aufeinandertreffen zweier solcher auf jeden Fall einer zur Reproduktion herangezogen wird.

In diesem Zusammenhang sei noch auf gängige Auswahlverfahren bei den Evolutionsstrategien verwiesen, wo es ($\mu+\lambda$)- und (μ,λ)-Strategien gibt, bei denen μ die Anzahl der Eltern und λ die Anzahl der daraus entstehenden Kinder ist. Im ersten Fall werden die besten μ Individuen von Eltern und Kindern für die nächste Generation herangezogen, im zweiten Fall werden die besten μ Elemente der Kinder als Eltern der nächsten Generation genommen. Dadurch ist ein starker Druck hin zu den fitteren Elementen gegeben, wobei bei der ersten Methode ein einmal gefundenes Optimum nicht mehr „vergessen“ werden kann.

Mutation

Bedeutung der Mutation

Über die Bedeutung der Mutation als genetischer Operator wird in den Fachkreisen oft diskutiert, wobei sich hier keine einheitliche Lehrmeinung bis jetzt durchsetzen hat können. Während manche die Mutation als reinen Hintergrundoperator verstanden wissen wollen, propagieren andere die Mutation als mächtiges Werkzeug, daß dem Crossover bei weitem überlegen ist (vgl. /BBM93b/).

Das geht soweit, daß man eine Verwendung der Genetischen Algorithmen nur mit Selektionsmechanismen und Mutationsoperatoren (als sogenannte naive Evolution in der Literatur bekannt) einer Einbindung der verschiedenen Crossovertechniken vorzieht.

Unbestritten in diesem Bereich ist allerdings die Tatsache, daß die Bestimmung der Mutationsrate eine weitaus diffizilere Aufgabe darstellt, als die Ermittlung anderer, für den Ablauf eines GA relevanter, Werte. Es ist die Mutation, die neue „Impulse“ setzt und Material, das entweder schon verloren war oder im vorliegenden Simulationslauf

noch nicht aufgetreten ist, in die Population einbringt. Daher ist klar, daß man bei Wahl einer zu kleinen Mutationsrate diesen Effekt nicht erreicht und dadurch womöglich aus lokalen Extrema nicht mehr herausfindet. Im umgekehrten Fall läuft man allerdings Gefahr, den Optimierungsprozeß zu sehr in Richtung Zufallssuche zu treiben, was normalerweise auch kein wünschenswerter Effekt ist.

Um jetzt eine konkrete Antwort auf die Frage nach der optimalen Mutationsrate zu erhalten, müßte man viele anwendungsspezifische Dinge berücksichtigen und selbst dann kann man nur von Faustregeln ausgehen. Eine solche populäre Faustregel besagt, daß etwa 1% aller genetischen Operationen mutationsähnlich sein sollen.

Zu bedenken gibt es in diesem Kontext noch, daß Untersuchungen zeigen, daß Mutation gegen Ende eines Optimierungsprozesses bessere Dienste als das Crossover zeitigen, da man annehmen kann, daß manche Elemente der Population nur mehr knapp vom globalen Optimum entfernt sind und durch Mutation der nötige Schritt in die richtige Richtung aufgrund der Struktur des Operators leichter geleistet werden kann.

Durchführen der Mutation

Im Falle einer Kodierung im Binärsystem besteht der Prozeß der Mutation einfach aus dem Ändern des Wertes eines bestimmten Bits in den anderen möglichen Wert (aus 0 wird 1 bzw. umgekehrt). Die Stelle des Bits im ausgewählten String wird üblicherweise zufällig bestimmt, es sei denn man hat berechtigte Annahmen, daß sich Mutationen an bestimmten Stellen besser auswirken können als an anderen. Dies wäre aber nicht nur schwieriger zu programmieren, es läßt sich eine bevorzugte Behandlung mancher Bereiche eines Strings a priori selten rechtfertigen. Eine Ausnahme bilden jene Bereiche, wo man einschlägiges problemspezifisches Wissen zur Verfügung hat und dementsprechend handeln will.

Bei Kodierungsverfahren, die sich nicht dem Binärsystem bedienen, ist dieser Vorgang mit etwas mehr Überlegungsarbeit verbunden. Nehmen wir als erstes den Fall an, daß einfach ein größeres Alphabet (wie z.B. das Dezimalsystem) zur Kodierung herangezogen wird. Dann stellt sich die Frage, in welchen Wert das ausgewählte Bit bei der Mutation gewandelt werden soll, da in diesem Fall nicht nur eine einzige Möglichkeit (sondern bei der Verwendung des Dezimalsystems neun

andere) der Abwandlung zur Verfügung steht. Üblicherweise wird in solchen Fällen jede andere Variante als gleichwahrscheinlich betrachtet und durch Verwendung eines Zufallszahlengenerators dementsprechend ausgewählt. Es sind allerdings auch andere Vorgehensweisen denkbar, wie etwa, daß benachbarte Werte bevorzugt gewählt werden.

Das führt uns zu den Fällen, wo eine Kodierungsform vorliegt, die sich Parameter in der Form bedienen, wie sie in der Wirklichkeit vorliegen. Nehmen wir an, daß eine Aufgabenstellung darin besteht, ein Problem zu optimieren, daß aus *n* zu variierenden Parametern zusammengesetzt ist. Die Parameter können jeweils aus einem bestimmten Intervall alle Zwischenwerte annehmen, wodurch sich Aufgabenstellungen ergeben, die unter anderem von Vertretern der Evolutionsstrategien untersucht werden (vgl. /NIS94/).

Ein Ansatz in diesem Bereich den Vorgang der Mutation zu realisieren, ist einfach den zu mutierenden Parameter mit einem Wert aus einer (z. B. normalverteilten) Zufallsgröße additiv oder subtraktiv zu verändern. Die unterschiedlichsten Auswirkungen der Mutation auf diesem Gebiet werden in den einschlägigen Literaturstellen ausführlich beschrieben.

Populationsgröße

Bei der Bestimmung der Populationsgröße ist ein vernünftiger Ausgleich zwischen ausreichender Vielfalt der verschiedenen Elemente einer Population und Rechenzeitaufwand ins Auge zu fassen. Wie augenscheinlich klar ist, ist bei Verwendung einer großen Population der Rechenaufwand zur Evaluierung enorm und dadurch schleppt sich der GA dem Ziel entgegen.

Bei zu kleiner Wahl dieses Wertes ist keine ausreichende „Artenvielfalt“ gegeben und es kommt wahrscheinlich zu einem Verharren in einem nichtoptimalen Bereich. Die Größe einer Population kann sich auch im Laufe der Zeit ändern, wenn man der Meinung ist, daß gegen Ende der Optimierung andere Prioritäten zu setzen sind.

Eine konkrete Zahl anzugeben fällt in diesem Zusammenhang schwer, da es zu viele unterschiedliche Anwendungsbereiche gibt, es lassen sich allerdings ein paar Rahmenbedingungen festlegen: Durch das Produkt von Populationsgröße mit

erwarteter Anzahl von Iterationen (diese Anzahl vorherzusagen ist aber wieder ein Problem für sich) soll man größenordnungsmäßig weit unter der theoretischen Anzahl an Möglichkeiten bleiben, da sonst der Mehraufwand des GA im Vergleich zu Enumerationsverfahren (als dessen spezielle Abart man die Zufallssuche bezeichnen kann) keine nennenswerten Vorteile bringt.

Andererseits ist ein Mindestwert in Betracht zu ziehen, der ein Vielfaches der Anzahl der Bits pro Chromosom darstellt. Dieses Vielfache hängt klarerweise von der Kodierungsform ab, bei Verwendung des Binärsystems kann man aber durchaus von einem Faktor größer zwei ausgehen.

Bijektivität von Genotyp und Phänotyp

In konkreten Anwendungen und bei Verwendung eines endlichen Alphabets zur Darstellung des Genotyps ist es mehr als nur wahrscheinlich, daß aufgrund der gewählten Kodierungsform einige mögliche Ausprägungen des Genotyps keine Entsprechungen im Phänotyp besitzen.

Als Veranschaulichung möge folgendes einfaches Beispiel dienen: Gegeben sei eine Aufgabenstellung in einem Suchraum von 43 möglichen Zuständen. Bei einer üblichen Kodierung im Binärsystem benötigt man Strings der Länge 6, um alle 43 Möglichkeiten zu erfassen. Bei dieser Kodierung werden aber insgesamt 64 verschiedene Genotypvarianten bereitgestellt, was bedeutet, daß 21 Elemente keine Entsprechung in der Realität haben. Auch bei der Verwendung anderer Kodierungsformen wird in diesem Beispiel nie vernünftigerweise eine Bijektivität herzustellen sein.

Das hier angedeutete Problem tritt hauptsächlich dann auf, wenn wir uns in einem Suchraum bewegen, der diskrete Strukturen besitzt, obwohl ähnliche Probleme auch im Bereich kontinuierlicher Parameterwerte vorstellbar sind. Die Relevanz dieses Dilemmas liegt nun in der Struktur der Genetischen Operatoren (wie z.B. Mutation und Crossover) begründet, da durch Anwendung dieser Operatoren a priori nicht ausgeschlossen werden kann, daß im Laufe eines Optimierungsvorganges unzulässige (das heißt Genotyprealisierungen ohne zuordenbare Entsprechungen im Phänotyp)

Lösungen erzeugt werden. Als Lösungsansätze werden in der Literatur hauptsächlich drei Varianten vorgeschlagen:

Die erste Möglichkeit ist die am einfachsten zu realisierende, da einfach jedes Element, daß einen unzulässigen Wert entspricht nicht zugelassen wird und die Prozedur solange wiederholt wird, bis ein geeignetes Element entsteht. Dadurch treten natürlich auch unerwünschte Phänomene auf, wie etwa das Nichtberücksichtigen von guten Genwerten an anderen Positionen, nur weil an einer anderen Stelle ein unzulässiger Wert vorkommt.

Die zweite Methode vermeidet das, indem sie das vorübergehende Überleben unzulässiger Elemente gewährleistet, ihnen aber einen geringeren Fitneßwert zuweist, um ein Überhandnehmen zu verhindern. Diese Zuweisung eines Fitneßwertes wird üblicherweise als Straffunktion bezeichnet, da unerwünschte Elemente auf diese Art und Weise „bestraft“ werden. Das Problem ist nun in den meisten solchen Fällen die geeignete Wahl dieser Straffunktion, da sie erhebliche Auswirkungen zeitigt. Darüber gehen nun in den Literaturstellen die Meinungen auseinander und daher wird eigentlich in den meisten Fällen zur dritten Methode gegriffen.

Diese weist den überzähligen Lösungen auf mehr oder weniger einfache Art und Weise Elemente des Phänotyps zu. Hier ist wiederum die einfachste Technik jene, bei der jedem Wert des Genotyps ein Wert des Phänotyps zugeordnet wird, was allerdings dazu führt, daß manche Elemente des Suchraums durch zwei und manche nur durch eine Entsprechung in der Kodierung gekennzeichnet sind, was klarerweise statistisch alles andere als unproblematisch ist.

Einen Ausweg bietet der Ansatz, bei dem überzählige Repräsentationen zufällig auf gültige abgebildet werden. Es sind auch Mittelwege dieser beiden Verfahren denkbar, bei denen nach gewissen Prämissen die Entscheidung in irgendeiner Form getroffen wird.

Zusammenfassend läßt sich zu diesem Abschnitt sagen, daß viele Anwender hier ihre eigenen Methoden und Techniken entwickelt haben und dieser Bereich sicher noch ausführlichen Untersuchungen in der Zukunft Raum bietet.

Allgemeine Kodierungsformen

Wie im Laufe dieser Arbeit schon öfter erwähnt worden ist, ist die Kodierungsform eines Genetischen Algorithmus nicht zwangsläufig auf die Verwendung eines Binäralphabets beschränkt, es bieten sich vielmehr dem Benutzer die Möglichkeiten, für problemabhängige Anwendungen die adäquate Form herauszufinden und sich dieser zu bedienen. Im amerikanisch-englischen Sprachgebrauch wird bei der Verwendung des Begriffes der Genetic Algorithms in erster Linie aber dennoch an Binärkodierung gedacht, was teilweise auf Tradition zurückzuführen ist, aber von dem etablierten Proponenten der Genetic Algorithms David Goldberg auch rational begründet werden kann (vgl. /GOL89/): Er führt ins Treffen, daß die Kodierung im Binäralphabet die größte Anzahl von Schemata (und damit auch von Bausteinen) bietet und auf diese Art dem Algorithmus die Möglichkeit gibt, aus einer großen Anzahl die besten Strukturen zu finden.

Als gängige Alternativen werden in Applikationen Kodierungen in Fließkommazahlen oder in Integerwerten verwendet, wobei die Fließkommarepräsentation hauptsächlich dann verwendet wird, wenn die zu optimierenden Parameter kontinuierlich (nicht zu verwechseln mit dem Bereich der kontinuierlichen Simulation) sind und eine gewisse Feinheit erreicht werden soll. Daß bei diesen Verschlüsselungsverfahren neue Möglichkeiten des Crossovers und der Mutation vorhanden sind, liegt auf der Hand.

Im Bereich des Crossovers könnte man daran denken, nicht nur ein bestimmtes Gen von einem der beiden Eltern zu nehmen, sondern den Mittelwert der jeweiligen Gene als Wert für das Gen des zu erzeugenden Kindes zu verwenden. Auch andere mathematische Operationen, wie etwa der geometrische Mittelwert oder Addition (Subtraktion) der Differenz zum höheren (vom niedrigsten) der beiden Gene (soferne man im erlaubten Bereich bleibt) ist denkbar.

Die Mutation kann im zufälligen Verändern eines Gens zu einem anderen Wert bestehen oder aus der Überlagerung mit einer Realisierung einer normalverteilten Zufallsvariablen. Gerade die letzte Variante bietet den Übergang der GA in den Bereich der Evolutionsstrategien, die von Rechenberg (vgl. /REC73/) vorgestellt worden sind, an, speziell dann wenn man die Mutation als den „wichtigeren“ Operator im Vergleich zum Crossover annimmt.

Einhergehend mit diesen Überlegungen muß eine Nachjustierung anderer Einflußgrößen auf den Genetischen Algorithmus erfolgen, wie sie etwa im Abschnitt über die Populationsgröße geschieht, um auf die geänderten Strukturen der Proplemrepräsentation eingehen zu können.

Umordnung

Genetische Algorithmen finden unter anderem deswegen in den meisten Fällen ihr Ziel, weil sie Mechanismen besitzen, die erfolgreiche Strukturen weitervererben. Es gibt nun Bestrebungen dem Genetischen Algorithmus bei der Suche nach ausgezeichneten Strukturen noch weiter als bis jetzt unter die Arme zu greifen. Man kann in den meisten Fällen davon ausgehen, daß die Abfolge der Gene am Chromosom vom Ausführenden ohne tiefere Absichten aufgesetzt worden ist. Es stellt sich nun die Frage, ob nicht vielleicht eine andere Reihenfolge der Gene die Produktion guter Bausteine besser unterstützt oder erst ermöglicht.

Genau bei diesen Überlegungen setzt der Gedanke der Umordnung der Gene am Chromosom ein. Daß dadurch ein gewisser Mehraufwand verbunden ist, ist klar, muß doch zur Ermittlung des Phänotyps aus einem Genotyp zur Konvertierungsfunktion noch die Position der einzelnen Gene gespeichert werden, um eine korrekte Umrechnung zu ermöglichen. Zur Veranschaulichung dieses abstrakten Sachverhalts betrachten wir ein einfaches allgemeines Beispiel:

Gehen wir von einem Chromosom der Länge 6 aus, wo wir als a-posteriori-Wissen annehmen, daß alle guten Lösungen sich durch das Element *a* an der ersten und das Element *b* an der letzten Stelle auszeichnen, woraus sich ergibt, daß alle solchen Lösungen einen Genotyp der Form *a****b* besitzen. Bringt man nun einen Repräsentanten dieses Genotyps mit einem anderen beliebigen Element zum Crossover, ist die Wahrscheinlichkeit hoch, daß keines der dabei erzeugten Kinder die als gut erkannte Struktur besitzt. Das ist natürlich keinesfalls mit dem Aussterben dieser Struktur oder gar Versagen des Verfahrens gleichzusetzen, aber es ist immerhin ein Irrweg beschritten worden.

Gehen wir nun im vorigen Beispiel davon aus, daß die positiven Interdependenzen zwischen erstem und letztem Gen rechtzeitig erkannt worden sind und daraufhin die

Position (nicht die Bedeutung!) des zweiten mit dem letzten Gen vertauscht worden wäre (ohne davon auszugehen, daß man zu diesem Zeitpunkt gewußt hat, daß *a* und *b* die optimalen Werte für diese Gene sind), hätten wir für alle guten Lösungen Genotypen der Art *ab*****. Diese Repräsentationsform läuft nun weniger Gefahr, beim Crossover zerstört zu werden und damit kann man davon ausgehen, einen Erfolg bei der Verkürzung der Laufzeit des Genetischen Algorithmus erreicht zu haben.

So schön sich das ganze in der Theorie anhört, führen doch Skeptiker dieser Technik einige berechtigte Argumente ins Treffen, die sich hauptsächlich mit dem Mehraufwand der Kodierung beschäftigen. Es muß jedes Gen immer „wissen“, welche Eigenschaft der Lösung es verschlüsselt und außerdem ist der GA in diesem Fall nicht nur mit der Suche nach der optimalen Lösung des Problems beschäftigt, viel mehr ist er am Beginn mit dem Finden einer optimalen Anordnung der Gene beschäftigt, was im Endeffekt mehr Zeit kosten kann, als es bringt. Daher empfehlen viele Experten, diese Technik nicht ohne besonderen und berechtigten Grund zu verwenden, um eine Überlastung des GA zu vermeiden.

Adaption von Operatoren

Im Laufe eines Optimierungsverfahrens kann es notwendig sein, Justierungen bei den das Verfahren determinierenden Parametern vorzunehmen, um ein besseres Verhalten eines Algorithmus zu erreichen. Ist diese Adjustierung bei den Gradientenmethoden unter anderem als eine Steuerung der Schrittweite implementiert, kann man im Bereich der GA eine Vielzahl von sinnvollen und vielleicht auch weniger sinnvollen Änderungen durchführen.

Wie schon in einem vorausgehenden Abschnitt angedeutet, kann man verallgemeinert sagen, daß ein GA zuerst erfolgsversprechende Bereiche des Suchraums aufzuspüren hat und dann diese genauer nach einem potentiellen globalen Optimum zu erforschen hat. Daß das Eigenschaften sind, die jeweils etwas andere Voraussetzungen an das angewendete Verfahren stellen, ist einsichtig, wenn man sich vor Augen hält, daß gegen Ende eines Optimierungsprozesses durch Operatoren, die eher dazu neigen, neue Suchgebiete zu erschließen, nicht ein so schneller Erfolg erzielt werden kann, als durch verstärktes Verwenden von Operatoren, die zur lokalen Erforschung gewisser

Bereiche des Suchraums besser geeignet sind. Es sollten klarerweise solche Bereiche erforscht werden, in denen gute Lösungen gehäuft auftreten.

In letzter Konsequenz kann das durchaus bedeuten, einen GA mit einem herkömmlichen Suchverfahren zu kombinieren, um die Vorteile von zwei grundverschiedenen Techniken gewinnbringend zu vereinen. Diese herkömmliche Technik kann in gewissen Anwendungsgebieten ein Gradientenverfahren sein, daß, nachdem der GA erfolgversprechende Gebiete lokalisiert hat, diese Bereiche schnell und effektiv nach dem lokalen Extremum absucht, um ein zu langes „Probieren“ des GA zu vermeiden. Solche und andere Hybridtechniken sind auf jeden Fall eine Überlegung Wert und in den meisten Fällen bei geeigneter Kombination jedem der Einzelverfahren überlegen.

Doch auch ohne Kombination mit anderen Techniken kann durch Steuerung der Optimierungsparameter und/oder Hinzufügen anderer Genetischer Operatoren eine positive Beeinflussung erreicht werden. In diversen Literaturstellen wird zu diesem Thema stets angemerkt, daß diese Adaption sehr problemabhängig ist und es verschiedenste Ansätze gibt (beginnend vom simplen Veränderung der Mutationsrate hin zur Erfassung von erfolgreichen Operatoren im Verlauf des Optimierungsprozesses und verstärkter Einsatz derselbigen), um eine Verbesserung des Verfahrens zu erreichen.

Diploidität und Dominanz

Um zu veranschaulichen, wie sehr die Natur in gewissen Bereichen als Vorbild für manche Verfahren dienen kann, sei hier exemplarisch die Diploidität und die daraus resultierende Dominanz erwähnt. Die meisten höherentwickelten Lebewesen haben zur Kodierung ihrer Erbeigenschaften nicht nur einen „Datensatz“, sondern einen zweiten, wobei für jedes Gen entschieden wird, welche Belegung für die konkrete Ermittlung des Phänotyps genommen wird. Dieses Gen heißt dann im üblichen Sprachgebrauch dominant im Vergleich zu seinem rezessiven Kollegen.

In der Theorie der Genetischen Algorithmen und auch allgemeiner gesprochen im Bereich der naturanalogen Verfahren wird im Gegensatz dazu die haploide Kodierungsform bevorzugt, wo jeder gespeicherte Wert eines Gens für eine

Eigenschaft des Phänotyps zuständig ist. Bei Kodierungen in diploiden Strukturen ist üblicherweise der Aufwand im Verhältnis zum Erfolg zu hoch, da immer ein zweiter Datensatz abgespeichert und darüber hinaus noch in Evidenz gehalten werden muß, welche Belegung dominant oder rezessiv ist.

Trotz dieses Nachteils kann es für manche Anwendungen von Vorteil sein, jeweils einen zweiten „Datensatz“ in Reserve zu haben und zwar speziell in solchen, wo mit häufig wechselnden Randbedingungen zu rechnen ist. Durch das gleichzeitige Bereithalten von zwei Genotypen pro Individuum kann durch simple Änderung der Dominanz ein und dasselbe Individuum an verschiedene Umgebungen gut angepaßt sein. Das bedeutet nicht, daß der GA Lösungen sucht, die gute „Allroundeigenschaften“ besitzen, sondern vielmehr solche, die wechselnde Bedingungen erkennen und den besser angepaßten Genotyp zur Ermittlung des Phänotyps heranzieht.

Wissensbasierte Techniken

Dieser Bereich zerfällt hauptsächlich in zwei Bereiche und zwar in jenen, wo versucht wird, für jede Applikation möglichst geeignete Operatoren zu finden und in jenen, wo durch Vorwissen des Anwenders eine Initialisierung gefunden werden soll, die einen besseren Startpunkt liefert als eine Zufallsinitialisierung.

Bei der Konstruktion von Operatoren für bestimmte Anwendungen geht die Robustheit im Sinne der Portabilität auf andere Applikationen mehr oder weniger verloren, dafür erreicht man allerdings ein besseres Verhalten für den konkreten Anwendungsfall. Es gilt speziell hier Aufwand und Nutzen geeignet abzuschätzen und dann die Entscheidung zu treffen, ob es sich lohnt, sich dieser Mehrarbeit zu unterziehen.

Im Fall der Initialisierung unter Verwendung von Vorwissen ist lediglich darauf zu achten, dem System nicht durch persönliche Vermutungen über das Ziel zuviel an Information mitzugeben und damit in Wahrheit den Optimierungsprozeß zu verlangsamen oder gar zu verhindern, wenn dadurch eine vorzeitige Konvergenz hin zu einem lokalen Extremum erreicht wird. Bei richtiger Anwendung wird dem GA

aber viel unnötige Suche in wenig erfolgsversprechenden Gebieten erspart und dadurch ein Zeitgewinn erzielt.

Spezielle genetische Operatoren

Manche Aufgaben erfordern spezielle genetische Operatoren, ohne die ein Funktionieren des Verfahrens nicht möglich wäre. Ein besonders wichtiges Anwendungsgebiet ist in diesem Zusammenhang die Bestimmung von optimalen Lösungen bei Reihenfolgenproblemen. Dies sind besonders in Bereichen des Operations Research benötigte Applikationen, die üblicherweise einen enormen Rechenaufwand erfordern, um akzeptable Lösungen zu erhalten.

Das bekannteste dieser Problemklasse ist wohl das „Problem des Handlungsreisenden“ (Travelling Salesman Problem - TSP), bei dem es darum geht, eine vorgegebene Anzahl von Städten zu besuchen, ohne eine Stadt doppelt oder eine Stadt gar nicht zu erreichen. Als Zielfunktion ist eine Minimierung des Weges vorgesehen. Für dieses und andere große und wichtige Anwendungsgebiete wurden eigene Operatoren entworfen, die sich grundlegend von den bis jetzt vorgestellten unterscheiden.

Crossover-Operator

Bei Verwendung der herkömmlichen Crossover-Operatoren würden hauptsächlich unzulässige Lösungen erzeugt werden, wie nachstehendes einfaches Beispiel einfach illustriert:

Gegeben seien die beiden Eltern *ABCDE* und *AEBDC*, die gültige Lösungen repräsentieren mögen. Wählt man nun die Stelle 2 als Crossoverposition erhält man als Kinder *ABBDC* und *AECDE*, die beide ungültige Lösungen darstellen, da zum Beispiel beim ersten „Kind“ die Stadt *B* zweimal und die Stadt *E* nicht besucht wird. Man könnte nun diese Lösungen mit einer Straffunktion belegen und das Verfahren weiterlaufen lassen, allerdings stellt sich das für diese Arten der Anwendungen als wenig sinnvoll heraus.

Aus diesem Grund sind Operatoren entwickelt worden, die diesen speziellen Anforderungen genügen, wie zum Beispiel der PMX (partially matched crossover)-

Operator (vgl. /GOL89/), bei dem es darum geht, beim Crossover die Abfolge der Gene so weit wie möglich unverändert zu lassen.

Das bedeutet im konkreten Fall, daß von einem Elternteil ein Teilstring unverändert übernommen und um die fehlenden Städte so ergänzt wird, daß die Abfolge dieser Städte der Reihenfolge im anderen Elternteil weitgehend entspricht. Im Fall des obrigen Beispiels könnten sich dadurch also folgende Kinder ergeben: *ABDCE* und *AECDB*. Man beachte, daß in diesem Fall die Crossovervorschriften nicht so strikt vorgegeben sind wie in anderen Fällen, daß also auch andere Kinder bei dieser Vorgehensweise entstehen können.

Mutation

Ähnliches wie im vorigen Abschnitt über Crossover läßt sich über die Mutation sagen, was anhand des vorigen Beispiels verdeutlicht werden soll: Gegeben sei das Element *ABCDE*, welches einer Mutation an der zweite Position unterzogen werden soll. Als mögliche Alternativen für die Stadt *B* stehen die restlichen vier Städte zur Auswahl, wodurch wiederum zwangsläufig ein unzulässiges Element erzeugt werden würde. Als Lösung ist es in diesem Fall möglich, eine zweite Stelle zu bestimmen und dann die Werte dieser zwei Positionen zu vertauschen, wodurch zum Beispiel nach einer solchen zulässigen Operation die mutierte Reihenfolge *ACBDE* entstehen könnte.

Anwendungen

Zum Abschluß seien noch kurz einige Anwendungsfelder aufgelistet, in denen in letzter Zeit durch die Verwendung von naturanalogen Verfahren im Allgemeinen und von Genetic Algorithms im Besonderen erfolgreiche Applikationen erfolgt sind:

- Funktionsoptimierung
- Bilderkennung
- Kombinatorische Optimierung
- Machine learning
- Steuerungsmechanismen

Neben diesen Gebieten gab und gibt es noch eine Vielzahl anderer Anwendungen, die oft in andere Bereiche wissenschaftlicher Forschung vordringen, wie etwa in den Bereich der Künstlichen Intelligenz oder des Simulated Annealing. Für ausführlichere Übersichten sei auf das Literaturverzeichnis verwiesen, bzw. auf manche darin enthaltenen Stellen (z. B. /COMP_1/, /NIS94/, /COMP_8/, etc.).

Zusammenwirken von Genetischen Algorithmen und diskreter Simulation

Dieser Abschnitt setzt sich mit den Besonderheiten der Anwendung von Genetischen Algorithmen auf den Bereich der diskreten Simulation auseinander. Es werden Überlegungen angestellt, auf welche speziellen Aspekte in diesem Fall Rücksicht zu nehmen sind. Weiters werden Konzepte präsentiert, wie eine Anwendung auf diesen Bereich aussehen könnte und auf welche Dinge besonders im Vergleich zu Applikationen von GA in anderen Bereichen zu achten sind.

Unterschiede zu anderen Bereichen

Als den Hauptunterschied ist sicher die Tatsache zu sehen, daß üblicherweise die Dauer der Evaluierung im Bereich der Simulation deutlich höher ist als die Methoden zur Bestimmung der Fitneßfunktion in anderen Bereichen, wo oft eine mehr oder weniger simple Funktionsauswertung diese Rolle übernimmt.

Speziell in der diskreten Simulation, wo meistens noch zusätzlich stochastische Überlagerungen vorhanden sind, die es notwendig machen, für die Evaluierung eines Elements einer Population mehrere Simulationsläufe durchzuführen (siehe Kapitel über diskrete Simulation), ist die Dauer des Optimierungsprozesses im größeren Maße von der Dauer eines Einzelschrittes (= Evaluierung) abhängig.

Wie im weiteren Verlauf dieses Kapitels noch dargestellt werden wird, kann die Verwendung von unterschiedlichen Simulationssprachen enorme Auswirkungen auf die Dauer der Optimierung haben. Dadurch ist größeres Augenmerk auf die richtige Auswahl der Simulationssprache zu legen, die, wie durch die Bezeichnung „Sprache“ schon angedeutet wird, einerseits Alternativen (= Verwendung anderer Sprachen) und andererseits durch geschicktes Verwenden (Vermeidung komplizierter Konstrukte oder anderer aufwendiger Programmierstile) Perspektiven für eine gelungene Optimierung bietet. Bei „herkömmlichen“ Einsätzen von Genetischen Algorithmen steht die Art der Bestimmung der Fitneßfunktion üblicherweise nicht in derlei dramatischem Ausmaß im Vordergrund.

Dies führt uns direkt zum nächsten wichtigen Punkt, der eng mit dem vorhergehenden zusammenhängt, nämlich der Tatsache, daß im hier besprochenen Anwendungsbereich die Struktur des Genetischen Algorithmus nicht der belastende „Overhead“ des Systems ist, sondern, daß vielmehr die Einzelevaluierung eines Populationsmitglieds die zeitaufwendige Komponente darstellt. Damit soll natürlich auf keinen Fall gesagt werden, daß die Steuerungsroutinen des Verfahrens, wie etwa Art der Crossovertechnik, nebensächliche Bedeutung haben, es soll vielmehr darauf hingewiesen werden, daß Sorge zu tragen ist, die Anzahl der Fitneßwertevaluierungen möglichst gering zu halten, da hier der größere Zeitverlust geschehen kann.

Aus diesem Grund kann eine komplizierte Struktur des Genetischen Algorithmus herangezogen werden, wenn dadurch nur gewährleistet wird, seltener die Simulation (= Bestimmung der Fitneßfunktion) bemühen zu müssen. Unter komplizierterer Struktur ist nicht notwendigerweise eine unüberschaubarere Struktur gemeint, sondern lediglich eine, bei der öfter Randbedingungen und Zielvorgaben ins Kalkül gezogen werden.

Gerade im Bereich der diskreten Simulation, die sich oft am Schnittpunkt zwischen Technikern und Managern befindet, ist auf eine klare und einleuchtende Verfahrensweise zu achten, um eine gewisse Grundakzeptanz möglich zu machen. Manager (oder andere entscheidungsbefugte Personen) werden selten durch mathematische Beweise überzeugt, sondern vielmehr durch eine Präsentation nachvollziehbarer Gedankengänge.

Aus diesem Grund ist es für den Ausführenden einer solchen Optimierungsstudie notwendige Voraussetzung, seinen „wissenschaftlichen Elfenbeinturm“ zu verlassen, um sich auf die Denkweisen Anderer einzustellen, was natürlich auf keinen Fall um den Preis der Scharlatanerie geschehen darf!

Ein weiterer wesentlicher Unterschied zu anderen Bereichen, in denen Genetische Algorithmen zur Optimierung herangezogen werden, liegt direkt im Namen „diskrete Simulation“ verborgen, und zwar im Wort „diskret“. Wir haben es hier mit einem Suchraum zu tun, der in vielen Fällen mit keinen Gradientenmethoden oder verwandten Techniken durchforstet werden kann, da keinerlei Ableitungen gebildet werden können.

Ausnahmen bestätigen auch hier, wie sooft im Leben, die Regel, und daher kann es auch geschehen, daß eine differenzierbare Teilstruktur vorliegt. Dies ist zum Beispiel dann der Fall, wenn ein zu optimierender Parameter „stufenlos“ variierbar ist und sich durch diesen Parameter eine Änderung des Fitneßwertes ergibt, die in einschlägiger mathematischer Weise von diesen abhängig dargestellt werden kann.

Im allgemeinen ist aber ein solcher Zusammenhang nur schwer oder gar nicht herzustellen, wodurch die Situation eintritt, daß man von den bekannten Optimierungsverfahren die meisten nicht verwenden kann. Hier liegt nun der große Vorteil der GA, da diese Verfahren theoretisch immer, aber klarerweise nicht immer mit der gleichen Effizienz, einsetzbar sind und daher dem Anwender die aufwendige Suche nach komplizierten, verborgenen Zusammenhängen erspart bleibt.

Durch die eben erwähnten Zusammenhänge ergibt sich in der Folge des Optimierungsverfahrens, daß eine Kombination von Genetischen Verfahren mit anderen Techniken nur bedingt möglich ist. Wie bereits in einem vorangehenden Kapitel dargelegt ist, wird des öfteren ab einem gewissen Zeitpunkt der GA von Verfahren abgelöst, die für das genaue Lokalisieren eines Optimums die besseren Eigenschaften besitzen. In den meisten der hier betrachteten Fälle bietet sich als Ablöse für den GA eigentlich nur ein lokales Enumerationsverfahren an, beziehungsweise ein Zurückgreifen auf eine lokale Monte-Carlo-Methode.

Vorgehensweise

In weiterer Folge dieser Arbeit ist geplant, die Simulation für den Genetischen Algorithmus als eine Art „Black Box“ aufzufassen und darauf aufbauend flexible Optimierungskonzepte darzustellen, beziehungsweise Problematiken in diesen Bereich anzudiskutieren. Ähnlich wie in anderen Bereichen der Optimierung kann man auch im Bereich der diskreten Simulation einige völlig unterschiedliche Bereiche identifizieren, was dazu führen müßte, für jeden dieser Bereiche eigene Verfahren zu entwickeln. Mit eigenen Verfahren ist in diesem Zusammenhang hauptsächlich die Auswahl anderer Genetischer Operatoren oder die Verwendung eines anderen Parametersettings (z. B. andere Mutationsrate) gemeint.

In diesem Abschnitt sollen allgemeine Ansätze dargestellt werden, die prinzipiell geeignet sein sollen, mehrere Problemstellungen im Bereich der diskreten Simulation zu lösen. Dadurch ist dem Anwender eine gewisse Verantwortung übertragen, in seinem Anwendungsgebiet eine geeignete Adaption der allgemeinen Theorie vorzunehmen. Das soll auch dazu führen, Anwender in der diskreten Simulation vom „blinden“ Verwenden von Verfahren abzuhalten und zum bewußten Verwenden dieses Tools hin zuführen, da gerade in diesem Bereich oft durch sorgloses Verwenden irgendwelcher Schemata mehr Schaden als Nutzen angerichtet wird.

So wird meistens eine statistisch relevante Input- und Outputdatenanalyse vernachlässigt oder erst gar nicht durchgeführt. Gerade in einem Bereich, der von stochastischen Überlagerungen gekennzeichnet ist, kann auf eine sorgsame Verwendung der Resultate nicht oft genug hingewiesen werden. Durch das Bereitstellen von Rahmenüberlegungen im Gegensatz zum Liefern von fertigen „Kochrezepten“ ist der Benutzer gezwungen, sich mit der Aufgabenstellung näher auseinander zu setzen und dadurch ist die Gefahr voreiliger Schlüsse etwas gemildert.

Parametrisierung

Um ein Problem nun geeignet für die Verwendung von Genetischen Algorithmen aufzubauen sind einige Vorüberlegungen notwendig. Es ist an eine geeignete Darstellung der zu optimierenden Größen zu denken. In den hier unter anderem zu Grunde gelegten Anwendungsmöglichkeiten sind die Variablen Größen unterschiedlichster Natur. Sie können von Kapazitäten einer Maschine über Anzahl von Arbeitern bis hin zu Ablauffolgen (Prioritäten, etc.) so ziemlich alles beinhalten, was im Bereich der diskreten Simulation auftritt. Da ein GA eine Parameterdarstellung zum Funktionieren benötigt, ist also an eine geeignete Parametrisierung zu denken.

Erster Schritt in diesem Zusammenhang ist die Auswahl eines Alphabets. Wie schon erwähnt, ist mit der Verwendung eines bestimmten Alphabets die Inkaufnahme gewisser Vor- und Nachteile verbunden. Für manche Anwendungen ist die Verwendung einer bestimmten Kodierungsmethode auf der Hand liegend, in manchen hingegen ist durch die unterschiedliche Struktur und Beschaffenheit der zu optimierenden Größen keine triviale Kodierung gegeben.

Es ist daher in diesem Zusammenhang nicht zu abwegig, von der Idee, für die Repräsentation des gesamten Optimierungsproblems eine einheitliche Darstellungsform zu verwenden, abzugehen und statt dessen innerhalb eines Strings (= Element einer Population) unterschiedliche Alphabete zuzulassen.

Zur Veranschaulichung sei hier ein kleines abstraktes Beispiel angeführt, bei dem eine fiktive Fertigung in Hinblick auf irgendein Gütekriterium zu optimieren sei. In dieser Fertigung seien drei Maschinen aufgestellt, die mit stufenlos regelbarer Intensität betrieben werden können. Weiters sei die Option offen, einen oder zwei Arbeiter in dieser Abteilung zu beschäftigen und die Arbeitszeit in Stunden könne die Werte 7, 7.25, 7.5, 7.75, 8, 8.25, 8.5, 8.75 und 9 annehmen. In diesem Fall wäre eine Repräsentation folgendermaßen vorstellbar:

$$(r_1,r_2,r_3,b,t_1,t_2)$$

Es handelt sich bei r_1,r_2 und r_3 um Variablen, die reelle Werte sind, b ist eine Binärzahl, die die Anzahl der Arbeiter bedeuten soll (0 entspreche einem Arbeiter und 1 entspreche zwei Arbeitern) und t_1 und t_2 seien die Kodierungen der möglichen Arbeitszeit mit einem Alphabet der Länge drei (00 entspreche 7, 01-7.25, 02-7.5, 10-7.75, 11-8, 12-8.25, 20-8.5, 21-8.75 und 22-9 Stunden). In diesem Beispiel werden dadurch drei Effekte gleichzeitig erreicht:

1. jedes Element des Genotyps hat eine eindeutige Entsprechung im Phänotyp, ohne daß aufwendige Abbildungsverfahren notwendig wären
2. reelle Werte müssen nicht diskretisiert werden
3. die genotypische Darstellung ist für Außenstehende leichter nachzuvollziehen

Aussage 1 ist natürlich nicht immer erreichbar bzw. erstrebenswert, aber man kann dadurch die Anzahl der genotypischen Elemente, die keine Entsprechung im Phänotyp besitzen, in einem gewissen Ausmaß steuern.

Durch diese „gemischte“ Kodierung muß noch Sorge getragen werden, beim Mutationsoperator eine geeignete Methode zu implementieren. Der Crossover-Operator funktioniert in normaler Art und Weise, es ist nur denkbar, für den Bereich der reellen Werte eine zusätzliche Crossover-Komponente zu definieren, bei dem die

Kindergenerationen einen berechneten Wert aus den entsprechenden Positionen der Eltern erhalten.

Dieser Wert kann der Mittelwert der entsprechenden Allele oder zum Beispiel ein durch Überlagerung einer geeigneten Realisierung einer Wahrscheinlichkeitsverteilung ermittelter Wert sein. Bei oftmaliger Verwendung des Mittelwertes kann es zur Konzentration der Werte in einem bestimmten Bereich kommen, da die Randbereiche durch die Art der Berechnung naturgemäß vernachlässigt werden. Auf diese Tatsache ist noch gesondert acht zu geben.

Aufbereitung für den GA

Bei bereits vorhandenen Simulationen ist es notwendig, das Programm für die Verwendung von Genetischen Algorithmen speziell aufzubereiten. Damit ist eine geeignete Parameterdarstellung gemeint und unter Umständen ist an eine geeignete Schnittstelle zu einem externen Programm zu denken. Es ist in manchen Simulationssprachen aber durchaus auch möglich und sinnvoll, die Implementierung des GA mit einem Konstrukt dieser Simulationssprache vorzunehmen (vgl. die Implementierungen im folgenden Kapitel in D_SIM und MicroSaint™).

Wie schon mehrfach erwähnt, spielt im Bereich der Genetischen Algorithmen die Anordnung der einzelnen Gene am Chromosom eine große Bedeutung (Bausteinhypothese!). Aus diesem Grund ist nach Möglichkeit eine gute Anordnung erstrebenswert. In manchen Applikationen wird in diesem Fall auf einen sogenannten Meta-GA zurückgegriffen, der unter anderem eine gute Anordnung der Gene am Chromosom zum Ziel hat. Da so eine Vorgehensweise extrem zeitaufwendig ist, kann darauf nur zurückgegriffen werden, wenn der eigentliche GA öfters benötigt wird und daher der Zeitaufwand für den Meta-GA gerechtfertigt ist.

Üblicherweise kann davon ausgegangen werden, daß die Anordnung vom Ausführenden des Optimierungsvorganges nach Heuristiken festgelegt wird. Da im Bereich der diskreten Simulation von längerdauernden Optimierungsprozessen ausgegangen werden kann, kann auch an eine „händische Nachjustierung“ der Abfolge der Gene im Laufe des Genetischen Algorithmus gedacht werden.

In diesem Fall ist vom Beginn an eine geeignete Eingriffsmöglichkeit zu denken und es ist eine Transparenz der internen Vorgänge nötig (Beobachtungsmöglichkeit des Optimierungsfortschrittes, Feststellung von bereits konvergierten Teilbereichen, Überwachung der spezifischen Zusammensetzung einer Population, etc.).

Dies führt direkt zum nächsten wichtigen Punkt in diesem Zusammenhang, nämlich der Kontrolle des Prozeßablaufes. Da nicht immer die Optimierung vom Ausführenden überwacht wird, ist auch an geeignete Kontrollmechanismen zu denken, um nicht unnötig Rechenaufwand in Kauf zu nehmen. Insbesonders wichtig in diesem Zusammenhang ist die rechtzeitige Erkennung von vorzeitiger Konvergenz.

Dieser Fall tritt bekanntlich dann auf, wenn viele (oder alle) Elemente einer Population das gleiche Aussehen besitzen und es sich dennoch noch nicht um das globale Maximum handelt. Zu diesem Zweck ist es prinzipiell einmal nötig, Konvergenz zu erkennen. Ein taugliches Instrument dafür ist zum Beispiel die Hammingdistanz, die bekanntlich als Ergebnis die Anzahl unterschiedlicher Stellen von jeweils zwei Chromosomen liefert.

Im Fall der Binärkodierung ist dieses Maß - angewendet auf je zwei Elemente der Population und anschließend aufsummiert - sicher eine gewisse Richtgröße, an der sich das Verhalten des Gesamtprozesses ablesen läßt. Im Falle anderer Kodierungsformen muß man sich über die Signifikanz dieses Maßes Gedanken machen, allerdings bleibt auf jeden Fall eine gewisse Restsignifikanz. Der hiermit getriebene Aufwand rechnet sich auf jeden Fall, wenn man sich vor Augen hält, daß zusätzliche Evaluationen der Fitneßfunktion (= Simulationsläufe) ungleich mehr Zeit verbräuchten und dem Ziel des Optimierungsvorganges unter Umständen abträglicher wären.

Populationsverwaltung

Als eine große Chance im Bereich der Applikation der Genetischen Algorithmen auf Beispiele der diskreten Simulation ist die Tatsache, mehr Aufwand für den GA treiben zu können, und daß dadurch unter Umständen die Möglichkeit besteht, alle bisher gefundenen Lösungen zu speichern. Das impliziert das Speichern des Phänotyps mit

zugehörigem Fitneßwert. Dieser Zeitaufwand ist im Vergleich mit der Evaluierung eines Mitgliedes einer Population üblicherweise nahezu vernachlässigbar gering.

Der Vorteil, der durch solch eine Vorgehensweise entsteht, liegt darin, Evaluierungen, die in diesem Fall zeitaufwendigen Simulationen gleichzusetzen sind, zu verhindern, wenn schon Fitneßwerte für diese Phänotypen bekannt sind.

In diesem Zusammenhang muß man sich auch über eine möglichst ressourcenschonende Speichermethode Gedanken machen. Dies bedeutet, daß beim Überprüfen, ob Elemente der aktuellen Population im Verlauf der Evaluierungen vorangegangener Generationen schon bewertet wurden, möglichst wenig Zeit verbraucht wird. Eine Möglichkeit stellt die Speicherung aller im Verlauf des gesamten Optimierungsprozesses aufgetretenen Lösungen in einem geordneten Feld dar. In diesem Zusammenhang ist darauf zu achten, daß im Falle von Umordnungen der Gene am Chromosom eine eindeutige Behandlungsweise garantiert ist.

Diese eben beschriebene Technik bindet den Vorteil eines Enumerationsverfahrens in den GA ein, da auf diese Weise keine bereits gefundenen Elemente wieder „vergessen“ werden können. Das ist ja nach wie vor einer der Hauptvorwürfe an die Familie der naturanalogen Algorithmen, daß nämlich gute Lösungen wieder verworfen werden und unnötige Evaluierungen von Elementen stattfinden, deren Fitneßwerte bekannt sein müßten, da sie in früheren Generationen schon bestimmt worden sind.

Falls das Merken aller bisher aufgetretenen Lösungen aus programmtechnischen Gründen zu aufwendig ist oder eine drastische Erhöhung der Laufzeit des Optimierungsvorganges bedeuten würde, können als Kompromiß auch nur die Lösungen der vorangegangenen Generation gespeichert werden. Dies ist programmtechnisch meistens leichter zu bewerkstelligen und kann oft auch schon einen merkbaren Einfluß haben, da in aufeinanderfolgenden Generationen einige gleiche Elemente auftreten können.

Auf jeden Fall ist es ratsam, jene Lösung zu speichern, die das beste Ergebnis erzielt hat, damit am Ende des Prozesses als Endergebnis wirklich das beste gefundene Element präsentiert werden kann. Innerhalb einer Generation sollte ebenfalls darauf geachtet werden, daß beim Auftreten gleicher Elemente innerhalb dieser Generation

die Simulation als Evaluierungstool für solche Elemente jeweils nur einmal gestartet wird.

Gerade im Bereich der diskreten Simulation ist es wichtig, für Akzeptanz und Transparenz zu sorgen. Daher ist es als eine wichtige Richtlinie zu erachten, den Optimierungsprozeß möglichst durchschaubar zu gestalten. Dies impliziert, nachvollziehbare Strukturen zu verfolgen und unter Umständen den Verzicht auf den höchsten Grad an Effizienz, wenn es dadurch möglich wird, die Akzeptanz zu erhöhen. Da in den meisten Fällen der Ausführende der Simulation keine Personalunion mit dem Entscheidungsträger bildet, ist es oft günstiger, eine ausreichend gute Lösung zu erreichen, als zu versuchen, eine aufwendigere Methode, die zwar bessere Aussichten auf Erfolg verspricht, aber beim Entscheidungsträger keine Akzeptanz erzielt, durchsetzen zu wollen.

Perspektiven

Gerade Genetische Algorithmen bieten durch ihre Vielfalt an Strategieparametern eine große Anzahl von möglichen Ausprägungsformen. Daher muß man Sorge tragen, Strategieparameter wie etwa Mutationsrate, Crossoverrate, Populationsgröße nicht zu oft unmotiviert zu ändern. Dadurch könnte man schon im Anfangsstadium des Verfahrens zuviel Zeit verlieren. Hingegen ist eine Änderung des Selektionsdrucks in den meisten Fällen ein probates Mittel, um vorzeitige Konvergenz zu verhindern, beziehungsweise gewünschte Konvergenz zu erzielen.

Um eine Möglichkeit zu schaffen, Strategieparameter automatisch anzupassen, kann daran gedacht werden, ähnlich wie im Falle der Evolutionsstrategien, eigene Gene zu verwenden, in denen zur Disposition stehende Strategieparameter kodiert werden können. Diese Gene werden dann jedem einzelnen Element einer Population hinzugefügt und genauso dem Optimisierungsprozeß unterworfen. So könnte zum Beispiel ein Bit angehängt werden, bei dem eine 0 bedeutet, daß für dieses Element keine Mutation stattfinden kann und eine 1, daß eine Mutation möglich ist.

Ähnlich könnte mit unterschiedlichen Crossover-verfahren vorgegangen werden, nur muß in diesem Fall eine Konvention getroffen werden, welche Ausprägung dieses Bits

dominant ist, wenn Elemente mit unterschiedlichen Crossover-bits gepaart werden sollen.

Dies führt in die Richtung der Evolutionsstrategien, wodurch es möglich wird, Vorteile dieser Verfahren in den Optimierungsprozeß mit Genetischen Algorithmen miteinzubeziehen.

Implementierungen

In diesem Kapitel werden anhand einfacher Modelle aus dem Bereich der Fertigung Beispiele vorgestellt, die, basierend auf unterschiedlichen Simulationssprachen, erfolgreiche Anwendungen der Theorie der Genetischen Algorithmen auf den Bereich der diskreten Simulation darstellen.

Implementierung in D_SIM

D_SIM basiert auf der Theorie der Petrinetze und kann vom anonymous ftp-server der Abteilung Simulationstechnik (/COMP_24/) bezogen werden. Es ist im Laufe einiger Diplomarbeiten entwickelt worden (vgl. /HLA93/, /WRI92/ und /WAG92/) und für einige praktische Aufgaben eingesetzt (vgl. /GS94/, /SB95/).

Petrinetze

Petrinetze wurden zur Simulation paralleler Systeme entwickelt. Ihre Struktur läßt sich mit einigen Erweiterungen auf viele in der Realität auftretende diskrete Prozesse abbilden. In ihrer einfachsten Form, den Bedingungs-Ereignisnetzen, können Systeme, die sich, wie der Name schon sagt, durch das Eintreten gewisser Bedingungen in andere Zustände begeben (Automaten), auf ihr dynamisches Verhalten und etwa auf die Möglichkeit von Systemverklemmungen (Deadlocks), die das System zum Stillstand bringen, getestet werden. Die nächste Stufe, die sogenannten Stellen-Transitionennetze bilden eine Übermenge der Bedingungs-Ereignisnetze und können schon mit gewissen Einschränkungen zur Simulation diskreter Prozesse herangezogen werden.

Der diesem Abschnitt zugrunde liegende Simulator verwendet gefärbte Stellen-Transitionennetze, wobei aber viele zusätzliche Features ein möglichst genaues Abbilden der Realität ermöglichen sollen. Auch ist es bei D_SIM möglich, in einem Zeitschritt alle möglichen Transitionen zu feuern, was in der allgemeinen Petrinetztheorie nicht möglich ist.

Ein Petrinetz besteht bekanntlich aus Places (Stellen), Transitionen und Verbindungen (Connections) zwischen diesen, wobei ein Place nur mit Transitionen und eine

Transition nur mit Places verbunden werden darf. Zusätzlich muß eine Anfangsmarkierung angegeben werden, damit das Netz auch etwas tun kann, da ohne Anfangsmarkierung das Netz stillsteht. Die Marken (Token, Werkstücke) befinden sich hierbei in Places, sie werden durch Transitionen in andere Places weitergeschoben bzw. erzeugt oder vernichtet.

Um die nun doch sehr generellen Petrinetze sinnvoll für eine Simulation einsetzen zu können, erwies es sich als notwendig, einige Erweiterungen an diesen vorzunehmen. Es ist nicht sehr sinnvoll nur eine Art von Token zu erlauben. Das führt zur Einführung von „farbigen“ Token, doch auch dies stellt nur einen Übergang zu Token dar, die unterschiedliche Eigenschaften, sogenannte Attribute, aufweisen. Eine solche Eigenschaft könnte beispielsweise die Zeit sein, wie lange sich ein solches Token bereits im System aufhält. Diese Gedanken führen dazu, daß nun die Token im System Werkstücken in einem Modell entsprechen. Gibt es unterschiedliche Werkstücke, so werden diese durch Token mit verschiedenen Attributen repräsentiert.

Aber nicht nur das ist möglich. Können nämlich solche Attribute während der Simulation im Netz verändert werden, ist es auch möglich den Arbeitsfortschritt in den Eigenschaften darzustellen. Soll z. B. ein bestimmtes Werkstück gebohrt werden, erhält das entsprechende Token das Attribut „soll bohren“. Mit diesem Attribut kommt das Token zu einem Teilnetz, das eine Bohrmaschine verkörpert, wo das Kennzeichen in „wurde gebohrt“ verwandelt wird. Damit ist es möglich, die Arbeitsschritte „am Werkstück“ zu verfolgen.

In den Transitionen ist es weiters möglich, bestimmte Eigenschaften der Token zu ändern, aber auch neue hinzuzufügen, falls dies notwendig erscheint. Des weiteren könnte auch die Verarbeitungszeit, die für die Durchführung eines Arbeitsschrittes notwendig ist, berücksichtigt werden, in dem der Weitertransport des Token um genau diese Zeit verzögert wird. Damit wird nun die wichtige Dimension der Zeit in die Theorie der Petrinetze eingeführt.

Als weitere Spezialisierung soll eine Transition nur feuern, wenn auch in den folgenden Places Platz für die Token ist. Weil ein unendlich großer Buffer, wie er in den ursprünglichen Petrinetzen selbstverständlich ist, in der Realität nur in Ausnahmefällen zu realisieren ist. Diese Places entsprechen den Buffern im Modell,

was heißt, daß sie nur eine beschränkte Kapazität haben. Ein weiteres Problem, welches sich aus den modifizierten Token in den Buffern ergibt, ist die Reihenfolge in der die im Buffer liegenden Token zu den nächsten Transitionen befördert werden.

Bei der Modellbildung mit Petrinetzen hat sich gezeigt, daß ähnliche Verarbeitungsmaschinen auch ähnliche Petrinetzteile ergeben. Das geht sogar soweit, daß sich bestimmte Maschinen nur bei den gewünschten Parametern unterscheiden, d. h. die gezeichneten Subpetrinetze unterscheiden sich gar nicht. Das führt zur Idee, eine Art von Bibliothek für Maschinen, Förderwerkzeugen und ähnlichen, in der Modellbildung immer wiederkehrenden Subnetzen, zu erstellen, bei der dann nur noch die entsprechenden Parameter einzusetzen sind.

Eine weitergehende Abstrahierung führt zum Gedanken, dem Benutzer, der gar keine Ahnung von Petrinetzen zu haben braucht, ein Computerprogramm zu geben, mit dem er nur noch seine Maschinen, Förderwerkzeuge oder ähnliches, so auf dem Bildschirm zu plazieren hat, wie es dem darzustellenden Modell entspricht. Es werden die Maschinen am Bildschirm als kleine Graphiken („Ikons“) dargestellt. Die plazierten Elemente werden durch gerichtete Linien, die der Verarbeitungsreihenfolge entsprechen, verbunden. Von sich aus erfragt der Rechner alle notwendigen Informationen, um damit eine entsprechende Nachbildung in Form eines Petrinetzes anfertigen zu können.

Um auch das Problem zu lösen, für den Fall gerüstet zu sein, wenn eine neue Maschine angeschafft wird, für die kein vorgegebenes Subpetrinetz existiert, wurde ein weiteres Programm entwickelt, das sogenannte SIMPAINT, mit dem es möglich ist, neu erstellte Subpetrinetze in das bestehende System aufzunehmen. Aber nicht nur das neue Petrinetz, sondern auch eine entsprechende Graphik für den Graphikeditor ist notwendig. Mit SIMPAINT kann auch diese generiert werden.

Ist nun der statische Teil des Modells aufgebaut, können dem System noch Arbeitsschritte bekannt gegeben werden, die von den verschiedenen Arbeitsmaschinen ausgeführt werden sollen. Des weiteren müssen noch die unterschiedlichen Werkstücke definiert werden, d. h. ein Werkstück entsteht durch eine Determinierung von verschiedenen Arbeitsschritten an diesem Werkstück. Es besteht auch noch die

Möglichkeit, bestimmte vorgefertigte Werkstücke zu Beginn in das System einzuführen (Initialisierungen).

Sind all diese Vorarbeiten erledigt, erfolgt der Start der Simulation. Während die Simulation läuft ist es möglich, die Zustände, bzw. die Auslastung der Buffer über eine sich ständig aktualisierende Grafik am selben Modell mitzuverfolgen. Für genauere Beobachtungen kann auch bei einer Unterbrechung der Simulation der vorherige Zustand der Buffer durch die Betrachtung der Statistikdaten analysiert werden. Diese stehen natürlich auch nach der Simulation zur Darstellung bzw. zur weiteren Verarbeitung zur Verfügung.

Eine weitere Möglichkeit zur Kontrolle der Simulation ist eine Überwachung der Linien. Wird ein „Supervise“ (Überwachungsfunktion) der Linie eingeschaltet, so erscheint immer dann ein Ausgabefenster, wenn ein Werkstück über diese Linie zu einer anderen Verarbeitungseinheit bewegt wird. In einem Fenster erscheinen die Informationen um welches Werkstück es sich handelt, welche Verarbeitungszeit bereits für das Werkstück verwendet worden ist und welche Arbeiten noch auszuführen sind. Damit wird es möglich, ständig Informationen aus der Simulation zu ziehen.

Die eigentliche Simulation, also die Abarbeitung der Petrinetze, wird von einem, für den Benutzer nicht sichtbaren, Simulationsprogramm durchgeführt, welches alle notwendigen Informationen zurückliefert. SIMSHELL bereitet dann diese Daten grafisch - in animierter Form - auf.

Der Petrinetzsimulator SIMDISC

SIMDISC.EXE bzw. SIMDISC.DLL ist ein Petrinetzsimulator, der beliebige, gefärbte Stellen-Transitionennetze abarbeiten kann. SIMDISC.EXE ist die Stand-Alone-Version, ab der Version 2.0 ist SIMDISC eine Dynamische Link Library, die von der Windowsapplikation SIMSHELL geladen und mit Petrinetzinformation versorgt wird.

Da SIMDISC zur Simulation diskreter Prozesse entwickelt wurde, werden Marken auch Werkstücke bzw. Token genannt. Werkstücke können nicht nur verschiedene Typen besitzen, sondern auch Attribute, die durch simple Ja/Nein-Darstellung repräsentiert werden. Stellen (Places) werden als Puffer aufgefaßt, in denen

Werkstücke abgelegt und wieder entnommen werden. Transitionen bewegen die Werkstücke von Place zu Place oder erzeugen bzw. vernichten Werkstücke. Transitionen setzen auch die Typen bzw. Attribute von Marken.

Bei der Aktivitätsüberprüfung von Transitionen werden auch noch sogenannte Transitionskonditionen verwendet, die mittels binärer Logik die Typen und Attribute von Marken prüfen können. Die Transition ist nur dann aktiviert, wenn in den Vorgängerstellen genug Tokens sind, die die Transitionskondition erfüllen. Tokens, die sie nicht erfüllen, werden quasi nicht gesehen. Es können zwar Inhibitorkanten, aber keine Testkanten verwendet werden.

Zur Angabe einer Anfangsmarkierung gibt es 2 Möglichkeiten:

- Definition direkt im Modellfile durch Angabe vom Schlüsselwort _TOKEN (Sinnvoll etwa bei einem Generator).
- Plazieren von Marken in beliebige Stellen von SIMSHELL.EXE aus.

Ab der Version 4.0 können noch sogenannte Epsilonstellen definiert werden. Dies wird durch ein 'e' als ersten Buchstaben der Verzögerungszeit einer Stelle ausgedrückt. Es ist nämlich oft wünschenswert, komplexe Steuer- und Regelungsmechanismen in ein Netz einzubauen, die aber alle innerhalb eines einzigen Schrittes abgearbeitet werden müssen. Dies wird mit der Epsilontik erreicht.

Hat eine Transition nur Epsilonstellen als Vorgänger, ist sie eine Epsilontransition und wird erst nach der Bearbeitung aller normalen Transitionen auf Aktivität getestet und eventuell gefeuert. So kann ein komplettes Steuerungsunternetz, welches nur aus Epsilontransitionen und Epsilonstellen besteht, zwischen den einzelnen Simulationsschritten abgearbeitet werden, ohne die Statistik zu beeinflussen.

Bei einem normalen Simulationsschritt werden die Epsilontransitionen jedoch nicht gefeuert. Während der Bearbeitung des Epsilonsubnetzes wird statt der globalen Variablen Simulation_Time die Variable Epsilon_Time erhöht, die Tokens in den Stellen werden zwar weiterhin angezeigt, aber nicht als Statistik abgespeichert.

SIMDISC.DLL

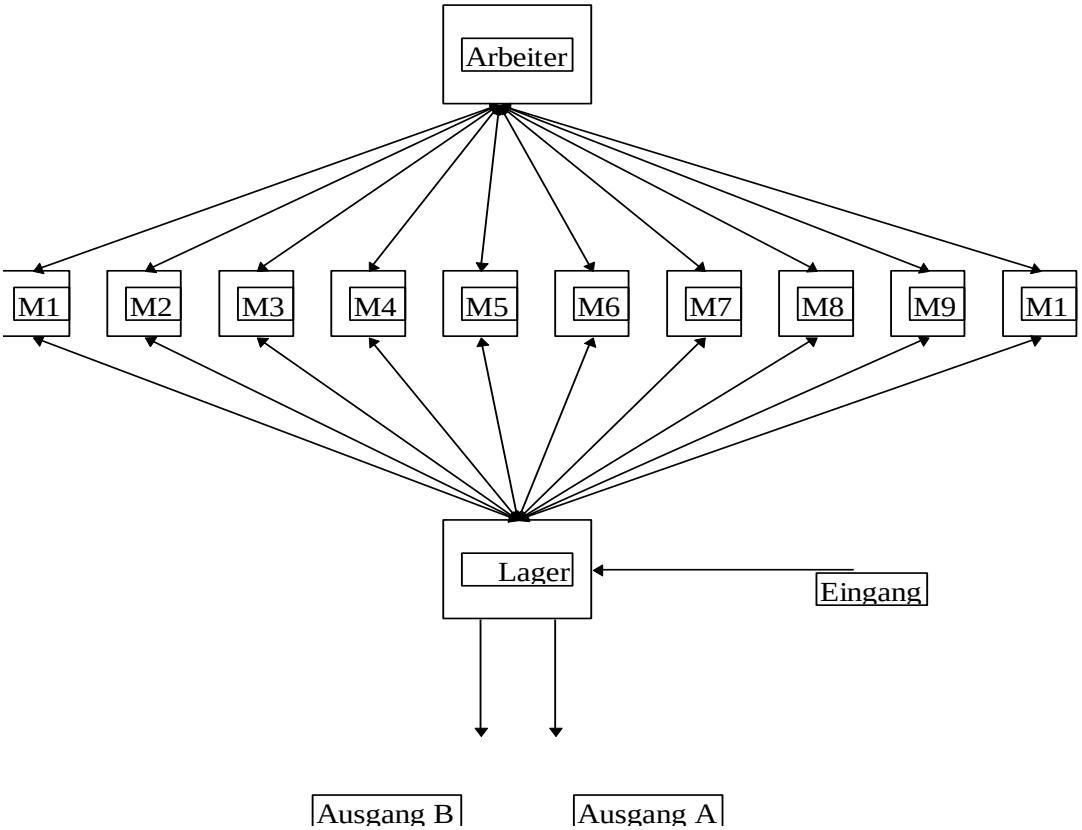
SIMDISC.DLL ist eine Dynamic Link Library, die mit der Windows Applikation SIMSHELL.EXE zusammenarbeitet. Die DLL wird von dieser Applikation geladen und gestartet.

Die Schlüsselworte PLACE, TRANSITION, CONNECTION und _TOKEN definieren dabei die Struktur des Netzes. Dieser String wird auch in einem File mit der Extension .MOD abgespeichert. Es können statt natürlicher Zahlen teilweise beliebige arithmetische Terme (Ausdrücke, expressions) übergeben werden, die ihrerseits nur konstante Zahlen enthalten dürfen. Die Terme werden dann bei der Erstellung des Netzes ausgewertet. Auf diese Weise können komplexere Zusammenhänge als Formel abgespeichert werden. CONNECTION und _TOKEN können auch dann übergeben werden, wenn die durch sie verbundenen Knoten bzw. die Stellen, in denen sie erzeugt werden sollen, noch gar nicht definiert worden sind.

SIMDISC.DLL durchsucht den angegebenen Stringbuffer nach speziellen Schlüsselworten. In der Version 4.1 sind dies die Worte PLACE, TRANSITION, CONNECTION und _TOKEN. Der Underscore vor TOKEN wird dazu verwendet, um TRANSITION von TOKEN unterscheiden zu können, da eigentlich nur der erste Buchstabe des Schlüsselwortes verglichen wird. Nach den Schlüsselworten müssen jeweils Parameter folgen, die die übergebenen Netzelemente näher beschreiben. Werden diese Parameter nicht gefunden, so resultiert dies in einer Fehlermeldung.

Anwendungsbeispiel

Für den Zweck der Implementierung eines Genetischen Algorithmus in D_SIM wurde folgendes fiktives Fertigungsmodell entworfen:



Es sollen zwei Arten von Werkstücke produziert werden. Für das Werkstück vom Typ A sind die Arbeiten I, III, I (in dieser Reihenfolge) notwendig. Für das Werkstück vom Typ B werden die Arbeiten II, I, IV, III, I benötigt. Es gibt vier unterschiedliche Maschinentypen, die jeweils die Arbeit I, II, III, oder IV verrichten können. Vom Maschinentyp I existieren vier Maschinen, von allen anderen Typen jeweils zwei Stück. Außerdem gibt es für jede Maschine noch vier mögliche Zustände, denen eine progressive Kostenentwicklung zugrunde liegt.

- Zustand 1: Maschine ausgeschaltet, Kosten = 0
- Zustand 2: geringe Geschwindigkeit, Kosten sehr gering
- Zustand 3: mittlere Geschwindigkeit, Kosten mittel
- Zustand 4: hohe Geschwindigkeit, Kosten sehr hoch

Weiters gibt es acht Arbeiter, wobei immer einer während der Bearbeitung eines Werkstückes bei der Maschine stehen muß. Während einer 8-Stunden Schicht sollen 30 Werkstücke vom Typ A und 20 Werkstücke vom Typ B produziert werden. Die Rohmaterialien gelangen über den Eingang in das Lager. Abhängig vom Typ des Werkstücks und von den bereits verrichteten Arbeiten wird, wenn ein Arbeiter frei ist,

eine für die nächste zu erledigende Arbeit passende Maschine ausgewählt. Nach der Bearbeitung gelangt das Werkstück wieder ins Lager und der Arbeiter wird wieder frei. Sind an dem Werkstück alle Arbeiten verrichtet, so gelangt es, abhängig vom Typ, zu einem der beiden Ausgänge.

Implementierung der Fertigungsanlage in D_SIM

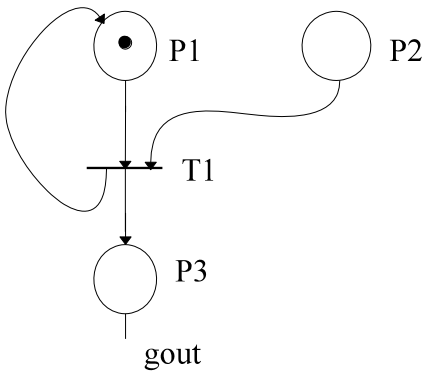
Für jedes Objekt, das im Modell verwendet wird, muß ein Petrinetz erstellt werden. Es entsteht dadurch eine Art Bibliothek von Maschinen, Förderwerkzeugen, Lagern und ähnlichen Ressourcen, die beliebig oft in Modelle eingesetzt werden können. Es ist allerdings möglich, für jede Instanz einer Ressource unterschiedliche Parameter einzustellen. Der Benutzer selbst muß keine Ahnung mehr von Petrinetzen haben, seine Aufgabe besteht darin, die Objekte auf dem Bildschirm zu plazieren und entsprechend der Modellvorgabe zu verbinden.

Zu Beginn muß aber erst eine Modellbibliothek erstellt werden, das heißt, man muß die Aufgabe der jeweiligen Ressource in Form eines Petrinetzes definieren. Dabei ist es sinnvoll die Petrinetze zuerst grafisch zu entwerfen und auf ihre Funktionstüchtigkeit theoretisch zu überprüfen. Anschließend kann man das Petrinetz in das nicht so übersichtliche Textformat übersetzen. Als Beispiele seien hier zwei Petrinetze vorgestellt.

Generator

Das Petrinetz des Generators ist klein und übersichtlich. Es besteht nur aus 3 Places und einer Transition. Im Place P1 muß sich immer eine Marke befinden. Laut Modellbeschreibung werden in einer Schicht 30 Werkstücke vom Typ A und 20 Werkstücke vom Typ B produziert. Das Place P2 muß daher mit 30 Token vom Type wp1 und mit 20 Token vom Typ wp2 vorbelegt werden. Zur Laufzeit kann eine Verzögerungszeit für P1 angegeben werden, die die Geschwindigkeit bestimmt, mit der die unfertigen Werkstücke den Generator verlassen. Nachdem die Verzögerungszeit von P1 abgelaufen ist, und sich in P2 noch Token befinden, kann

die Transition T1 feuern. Von P1 und P2 wird je ein Token abgezogen, wovon eines nach P3 und eines nach P1 gelangt.

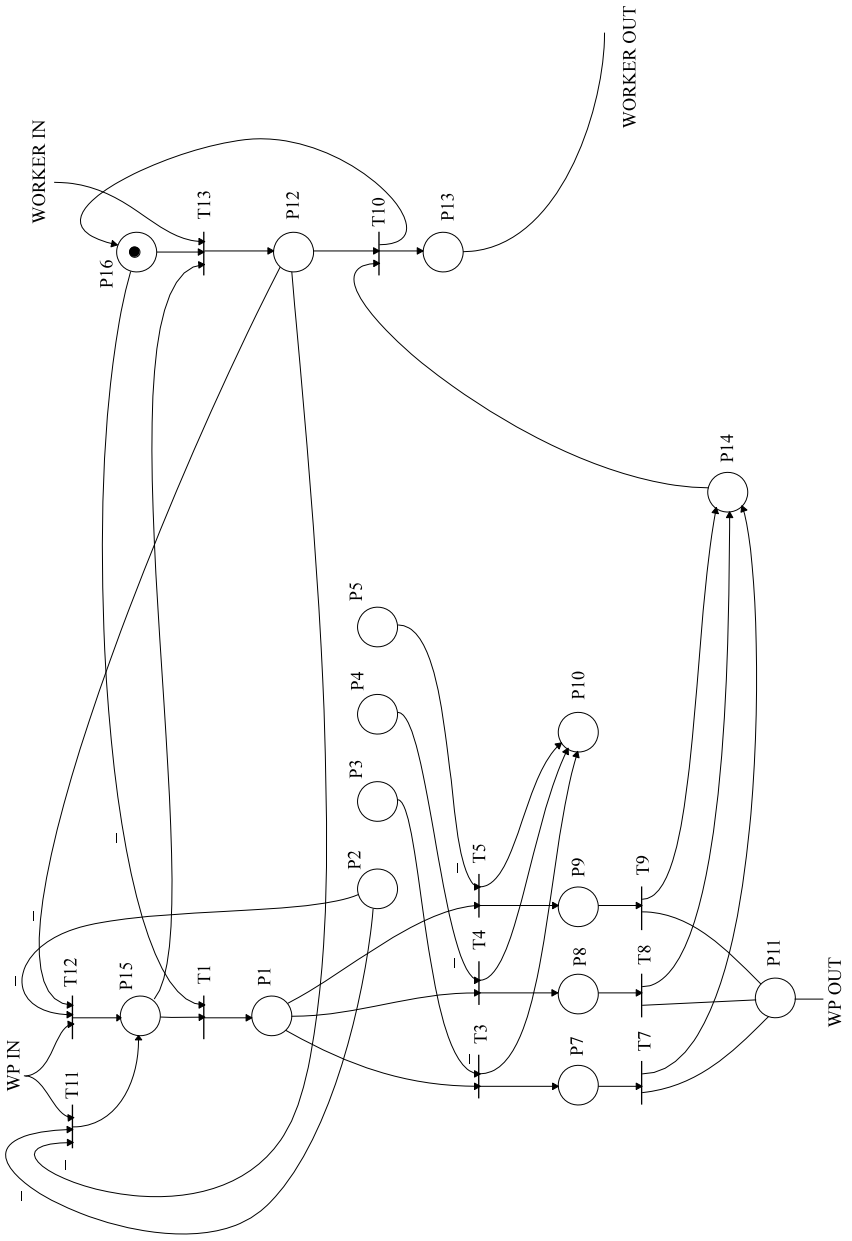


Der grafischen Darstellung entspricht folgende in textueller Form:

```
places:
e:p1        GeneratorInput
e:p2        GeneratorBuffer
end:
attribute:
MObType=generate
GenDelay  1
GenBuffer 1
end:
In_Output:
gout        Output
end:
description:
PLACE           e:p1    2       a:GenDelay
PLACE           e:p2    a:GenBuffer 1
PLACE           e:p3    1 1
TRANSITION      e:t1    -1      ""      a:TYPES
CONNECTION      p1      1       t1
CONNECTION      p2      1       t1
CONNECTION      t1      1       p1
CONNECTION      t1      1       p3
CONNECTION      p3      1       gout
end:
```

Maschinen vom Typ I

Das Petrinetz der Maschine vom Typ I ist wesentlich komplizierter als das des Generators.



Auf die textuelle Darstellung, sowie auf die Darstellung der restlichen Teilpetrinetze des Systems, sei an dieser Stelle verzichtet und auf den Anhang A verwiesen.

Implementierung des Genetischen Algorithmus in D_SIM

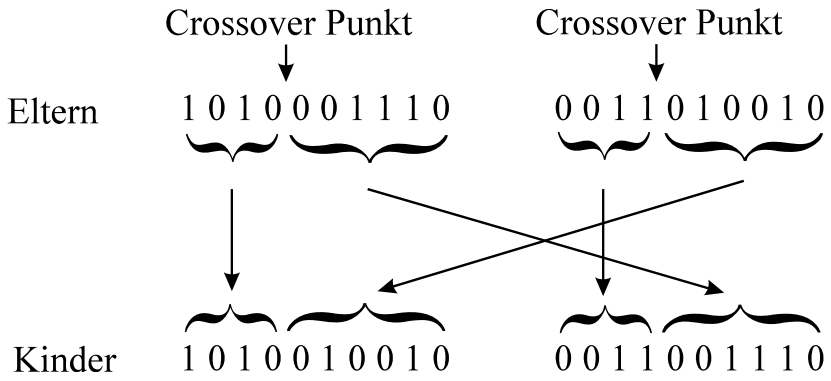
Die Implementierung des genetischen Algorithmus in D_SIM gestaltet sich nicht sehr einfach, da D_SIM ein bereits bestehender, komplexer Simulator ist. Zusätzlich zur Entwicklung des eigentlichen Genetischen Algorithmus müssen viele Teile des schon

vorhandenen Programmcodes komplett überarbeitet werden, da sie den neugestellten Anforderungen des Genetischen Algorithmus nicht entsprechen.

Crossovermethoden

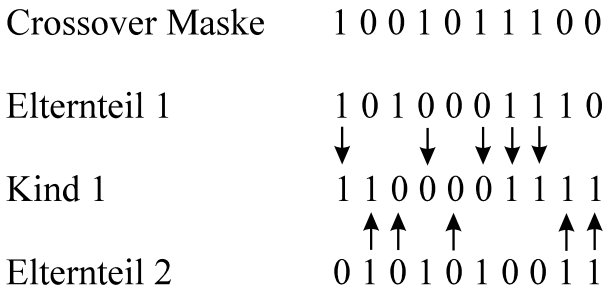
Derzeit sind 2 verschiedene Crossovermethoden implementiert. Es steht entweder das Einpunktcrossover oder das Schablonencrossover zur Auswahl. Zur besseren Verständlichkeit seien hier diese beiden Techniken nochmals grafisch dargestellt.

Beim Einpunktcrossover werden mit Hilfe einer Selektionsmethode zwei Eltern bestimmt, die an einem zufällig ausgewählten Crossover-Punkt getrennt werden.



Die erste Hälfte des ersten Elternteils wird mit der zweiten Hälfte des zweiten Elternteils zusammengesetzt. Die erste Hälfte des zweiten Elternteils wird mit der zweiten Hälfte des ersten Elternteils verbunden. Dabei entstehen zwei neue Chromosomen, deren Fitneß durch die anschließende Simulation ermittelt wird.

Das Schablonencrossover ist nicht mit dem soeben beschriebenen Verfahren zu vergleichen. Hier gibt es eine zufällig erstellte Crossovermaske, die bestimmt, welches Gen von welchem Elternteil vererbt wird.



Steht in der Crossovermaske eine 1, so wird das korrespondierende Gen vom Elternteil 1 vererbt, steht dort eine 0, wird es vom Elternteil 2 vererbt. Für das zweite Kind wird die Crossovermaske invertiert und der Vorgang wiederholt.

Bei den Testläufen des oben beschriebenen Fertigungsmodells hat sich gezeigt, daß das Einpunktcrossover schneller zur optimalen Lösung führt als das Schablonencrossover. Der Grund dafür könnte darin zu suchen sein, daß beim Schablonencrossover durch die viel höhere Anzahl von Trennungspunkten bereits gefundene, optimale Gensequenzen, mit höherer Wahrscheinlichkeit wieder zerstört werden (vgl. Bausteinhypothese).

Selektionsmethoden

Im Genetischen Algorithmen von D_SIM wurden als Selektionsmethoden eine Variante der Reihung-nach-Fitneß und die Rouletteradmethode implementiert. Diese beiden Verfahren seien hier kurz grafisch dargestellt.

Bei der Reihung-nach-Fitneß ist die Population sortiert nach der Fitneß, wobei das Gen mit dem größten Fitneßwert an erster Stelle steht. Zur Auswahl der Elternteile wird nun die erste Hälfte der Population herangezogen. Es wird dabei sequentiell vorgegangen, d.h. es werden das 1. und 2. Gen, dann das 3. und 4. Gen ausgewählt und mit Hilfe einer der Crossovermethoden gekreuzt. Die neu entstandenen Gene werden in der zweiten Hälfte der Population abgelegt.

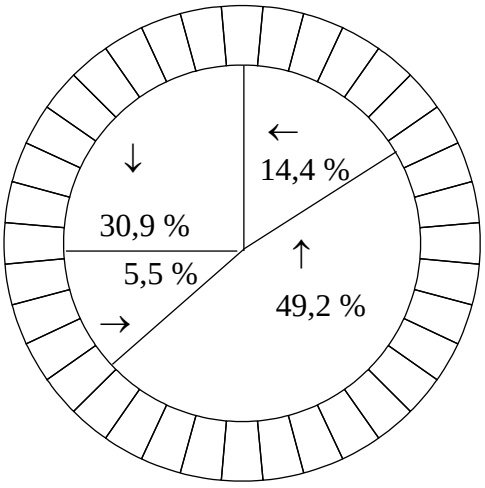
Population	Fittness	Populationsgröße = 8
10001001	500	Elternpaar 1
00111011	420	
01011001	390	Elternpaar 2
11010111	200	
01010000	150	
11111000	98	
11001111	65	
11100001	43	

Anders als bei der Reihung-nach-Fitneß haben bei der Rouletteradmethode auch Gene mit schlechteren Fitneßwerten eine Chance ausgewählt zu werden. Man kann sich dabei ein Rouletterad vorstellen, wo jedes Gen einen Abschnitt des Rades proportional zu seiner Fitneß belegt. In dem unten angeführten Beispiel sind vier Gene mit ihren

zugehörigen Fitneßwerten aufgeführt. Die Fitneßwerte aller Gene werden addiert und anschließend die prozentuale Verteilung berechnet.

Es ergibt sich ein Rouletterad mit vier Bereichen, deren Größe proportional zur Fitneß des jeweiligen Gens ist. Um ein Gen auszuwählen, kann man sich vorstellen, daß man das Rouletterad dreht und darauf wartet, in welchem Bereich die Kugel zu liegen kommt. Das korrespondierende Gen wird dann als Elternteil ausgewählt. Durch diese Methode werden zwar gute Gene bevorzugt, aber auch Gene mit nicht so guten Fitneßwerten haben die Chance sich fortzupflanzen.

Nr.	Gen	Fitneß	Anteil in %
1	10001010	576	49,2
2	00101111	361	30,9
3	10101110	169	14,4
4	01011100	64	5,5
Total		1170	100,0



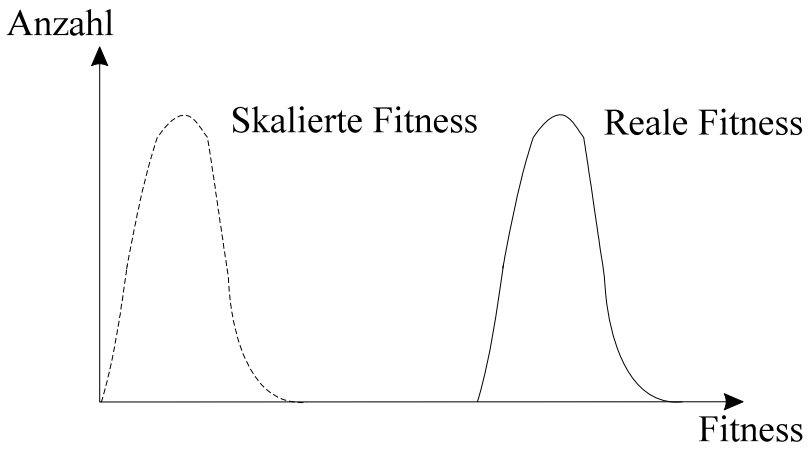
Mutation

Mutation ist ein sehr wichtiger Bestandteil der Genetischen Algorithmen. Dabei werden zufällig Teile eines Gens verändert. Die Wahl der Mutationswahrscheinlichkeit hat sehr großen Einfluß auf die Effizienz des Genetischen Algorithmus. Im oben beschriebenen Fertigungsmodell liefert die Simulation mit einer Mutationswahrscheinlichkeit von 0,1% keine optimalen Ergebnisse. Mit einer Einstellung von 1% wird fast jedes Mal ein Optimum erreicht.

Fitneßskalierung

Wie in einem vorangegangenen Kapitel erwähnt und erläutert ist, spielt die Fitneßskalierung unter Umständen eine wichtige Rolle beim Verlauf eines Genetischen Algorithmus. Beim Start einer Simulation sind die Fitneßwerte der Gene einer Population zufällig verteilt. Während der Simulation beginnen sich einzelne Werte durchzusetzen, mit fortschreitender Konvergenz reduziert sich die Bandbreite der Fitneßwerte in der Population. Dadurch kann es vorkommen, daß sich die Fitneßwerte vom schlechtesten und besten Gen nur wenig unterscheiden und die Wahrscheinlichkeit, daß tatsächlich ein Teil der besten Gene ausgewählt wird, sinkt.

Daraus ergeben sich die Probleme der verfrühten Konvergenz und des langsamen Vorankommens am Ende der Simulation. Deshalb verwendet man die Methode der Fitneßskalierung. Man skaliert also die erhaltenen Werte so, daß sich das beste Gen vom schlechtesten wieder deutlich unterscheidet.



Eine Möglichkeit besteht darin, von den berechneten Fitneßwerten den kleinsten Wert der Population zu subtrahieren. Eine andere wäre, einen fixen Wert zu subtrahieren, wobei man das Entstehen von negativen Werten vermeiden sollte. Zusätzlich besteht noch die Möglichkeit, die verschobenen Fitneßwerte mit einer fixen Größe zu multiplizieren.

Im genetischen Algorithmus von D_SIM hat der Benutzer die Wahl zwischen einer automatischen Skalierung oder einer manuellen Eingabe des Multiplikators und des (möglicherweise negativen) Summanden.

Gewinnfunktion

Die Gewinnfunktion wird verwendet, um nach der Evaluierung eines Gens den erreichten Gewinn zu berechnen. Je höher der Gewinn ist, desto höher ist die Fitneß des Gens. Im Simulator D_SIM hat der Anwender die Möglichkeit eine beliebige Gewinnfunktion einzugeben.

Ergebnisse

Die Ergebnisse der einzelnen Optimierungsläufe sind in unten stehender Tabelle zusammengefaßt, wobei der maximal zu erreichende Profit 3720 Geldeinheiten beträgt. Die Abbruchbedingung ist dadurch gegeben, daß wenn innerhalb der letzten zwölf Generationen keine Verbesserung des Gewinns der besten Lösung eingetreten ist, eine Beendigung des Optimierungsprozeßes herbeigeführt wird.

Populationsgröße	Selektion	Crossover	Anz. d. Gen.	Profit
52	Rouletterad	Schablonen	22	3720
52	Reihenfolge	Schablonen	19	3640
52	Rouletterad	Einpunkt	19	3720
24	Rouletterad	Schablonen	44	3720

Wie man sieht, erreicht in drei von vier Fällen der hier präsentierten Beispiele der Genetische Algorithmus das globale Optimum und er bleibt nur bei Verwendung der Reihenfolge-nach-Fitneß-Selektionsmethode in einem lokalen Optimum hängen.

Dieses Beispiel ist auf einem 80486/33 MHz PC implementiert, wo die Evaluierung einer Generation der Größe 52 circa 30 Minuten dauert, was zu einem Gesamtrechenaufwand von circa zehn Stunden führt.

Implementierung in GPSS/H unter Verwendung von C++

GPSS/H™ ist eine alteingeführte Simulationssprache (vgl. /WOLV91/, /SCH91/ und /BCN89/), die unter anderem auf der Plattform MS-DOS™ läuft. Zur Implementierung eines Genetischen Algorithmus wird im Prinzip das

Fertigungsmodell aus dem vorigen Unterkapitel übernommen, es ist allerdings der Einfachheit halber als zu optimierender Wert die Anzahl der gefertigten Werkstücke gewählt, obwohl es in der Praxis sicherlich sinnvoller ist, den Gewinn - unter Einbeziehung des Erlöses aus dem Verkauf der Werkstücke und der Kosten für die Maschinen - zu maximieren. Diese Änderungen bedeuten jedoch keinen großen Programmieraufwand (vgl. /RUD95/ und /SB95/).

Der Genetische Algorithmus in C++

Jedes Chromosom wird in einer Struktur gespeichert, die ein Feld von vier Integerwerten für die Zustände der Maschinentypen und einen Integerwert für die Fitneß umfaßt. Die Codierung als String erfolgt durch die Aneinanderreihung der vier Zustände in binärer Darstellung. (Da nur vier verschiedene Zustände angenommen werden können, reichen zwei Bit zur Darstellung eines Zustands => ein Chromosom wird durch eine Kette von acht Bit dargestellt.) Diese Codierung ist jedoch nur theoretisch und nicht ausprogrammiert; die entsprechenden Operationen wie Crossover und Mutation finden direkt in den Structs statt.

Als Selektionsmethode wird das Rouletteradverfahren verwendet. Dabei wird die Gesamtfitneß der Population, d. h. die Summe der Fitneßwerte über alle Chromosomen, berechnet und der prozentuelle Anteil jedes Chromosoms daran ermittelt. Dieser Prozentsatz bestimmt dann die Wahrscheinlichkeit, mit der ein Chromosom in den mating-pool kommt.

Für das Crossover werden aus der Elterngeneration zufällig zwei Chromosomen gewählt, aus denen wiederum zwei „Kinder“ entstehen, und zwar so lange, bis eine neue Population der gleichen Größe vorhanden ist (also *POPSIZE/2* mal). Dabei können manche Chromosomen auch öfter ausgesucht werden, sie können sich nur nicht mit sich selbst paaren.

Für das Crossover selbst werden in den gedachten Strings ab einer zufälligen Stelle die Enden vertauscht, d. h. in den Chromosomenstructs werden die entsprechenden Zustandsfelder ausgetauscht. Sollte die Trennstelle „innerhalb eines Zustands“ liegen, wird nur das letzte Bit dieses Zustands getauscht (und alle Zustände, die hinter der Trennstelle liegen).

Bei der Mutation wird an einer zufällig gewählten Stelle im gedachten String ein Bit invertiert, d. h. von 0 auf 1 bzw. von 1 auf 0 gesetzt. Auch diese Operation findet direkt am Struct statt. Die Mutationsrate ist mit 1% festgelegt. Als Abbruchbedingung dient das Vollenden einer bestimmten Anzahl von Durchläufen.

Die Schnittstelle

Wie die Skizzierung eines allgemeinen GA zeigt, wird üblicherweise die Evaluierungsfunktion innerhalb des GA aufgerufen. Das würde in diesem Fall bedeuten, aus dem C-Programm heraus über die DOS-Ebene das GPSS/H-Modell zu starten, von dem der Fitneßwert in einer Datei abgelegt würde; dort könnte ihn das C++-Programm lesen. Speziell in früheren Versionen von GPSS/H wird ein Modell jedoch bei jedem Start neu übersetzt. Das würde aber einen hohen Zeitaufwand in jeder Generation bedeuten.

Andererseits verspricht das GPSS/H-Handbuch die Verwendbarkeit einer CALL-Funktion, mit deren Hilfe extern programmierte und übersetzte Funktionen und Prozeduren in einem Modell aufgerufen werden können. Die Kommunikation des Modells mit der Funktion erfolgt dabei über die Parameterliste; eine Übergabe der Parameteradressen (und nicht nur der Werte) ist möglich und damit auch eine Änderung der Parameterwerte durch die externe Funktion.

Diese Voraussetzungen lassen es vernünftig erscheinen, den umgekehrten Weg zu gehen und die Evaluierungsfunktion (also das GPSS/H-Modell) den GA (also das C++-Programm) aufrufen zu lassen, mit einem Flag für die Initialisierung und die Beendigung des Programms. Um mit dem CALL-Statement nun externe C++-Funktionen aufrufen zu können, sind zwei Dinge notwendig:

- Der Watcom™ C++ 32 Bit Compiler, da externe Funktionen nur mit diesem übersetzt werden dürfen
- Eine neue Version von GPSS/H™ unter MS-DOS™, da die bisherigen das CALL-Statement auf dem PC nicht unterstützen

Neu in Version 2.05 von GPSS/H™ ist hingegen ein Statement mit Namen SYSCALL, das dem Programmierer erlaubt, Befehle an die DOS-Ebene zu schicken und damit zwar keine externe Funktion, aber ein Programm aufzurufen. Daraus ergibt

sich übrigens der Vorteil, daß zur Erstellung dieses Programms nicht mehr unbedingt der Watcom™ Compiler notwendig ist. Ein Nachteil bei dieser Methode ist, daß dem Programm keine Parameteradressen übergeben werden können, so daß die Kommunikation zwischen GPSS/H™ und dem Genetischen Algorithmus nur als Einbahnstraße funktioniert. Der Datenaustausch muß daher über Dateien erfolgen, aus denen sich GPSS/H™ die Parameterwerte (in diesem Beispiel die Maschinenzustände) holt und in die es die berechneten Fitneßwerte einträgt. Auch das C++-Programm braucht Dateien, in denen es alle Werte speichert, die beim nächsten Aufruf wieder gebraucht werden:

- Parameterwerte und Fitneßwerte der vorigen Population
- Parameterwerte der aktuellen Population
- Populationszähler für die Abbruchbedingung
- für statistische Zwecke das Chromosom mit der besten Fitneß aus jeder Population.

Das Verwenden von Dateien verlangsamt den Ablauf des C++-Programms etwas, gemessen an der Laufzeit ohne Filebenutzung, da aber ein Durchlauf des GA mit oder ohne Öffnen von Dateien wiederum nur einen Bruchteil der Zeit für die Evaluierung der Fitneßwerte für eine Population in Anspruch nimmt (Größenordnung Sekunden im Vergleich zu Minuten), fällt diese Verzögerung im Gesamtablauf kaum ins Gewicht.

Der Programmablauf

Wie bereits beschrieben ist das Hauptprogramm das GPSS/H™-Modell. Noch vor dem ersten Durchlauf der Blöcke, also der eigentlichen Simulation, wird in eine Datei mit Namen *BRIDGE*, die als Datenkanal zwischen GPSS/H™-Modell und GA dient, ein Wert eingetragen, der dem nachfolgend aufgerufenen C- Programm signalisiert, daß die erste Population mit Hilfe des Zufallsgenerators erzeugt werden soll. Das Programm wird beendet, nachdem es diese Werte in die Datei *BRIDGE* eingetragen und für den nächsten Aufruf in der Datei *SAVEVAL* gespeichert hat. GPSS/H™ startet für jeden dieser Werte einen Blockdurchlauf, schreibt die Ergebnisse (= Fitneßwerte) in die Datei *BRIDGE* und ruft wieder den GA auf. Diese Schleife (GA aufrufen, Werte lesen, Fitneß bestimmen) wird solange durchlaufen, bis der GA durch ein Flag in der Datei das Signal zum Abbruch gibt.

Nach Beendigung des Programms stehen zur Analyse des Ergebnisses zwei Dateien zur Verfügung: in der Datei *CHAMPION* findet man Parameterwerte und Fitneß des besten Chromosoms aus jeder Population, und die Datei *SAVEVAL* enthält Parameter- und Fitneßwerte aus der vorletzten Population sowie die Parameterwerte der aktuellen Population, die jedoch nicht mehr bewertet wurden.

Für genauere Darstellungen ist im Anhang B das entsprechende GPSS/H™-Programm und das zugehörige C++-Programm aufgelistet.

Ergebnisse

Obwohl in diesem Beispiel ein sehr einfacher GA implementiert ist, zeigt sich in Abhängigkeit von der gewählten Populationsgröße und der Anzahl der Generationen (wobei die Populationsgröße eine wichtigere Rolle spielt) eine gute Konvergenz gegen eine optimale Lösung. Bei einer zu kleinen Populationsgröße hängt das Ergebnis sehr stark von der Anfangspopulation ab, da insgesamt nur sehr wenig Erbinformation zur Verfügung steht, wird sich die Fitneß im Normalfall nicht wesentlich verbessern und auf einen nicht optimalen Wert einpendeln.

Diese theoretisch bekannte Tatsache wird durch einen Durchlauf mit Populationsgröße zehn verifiziert. Trotz einer relativ hohen Anzahl von Generationen (80) haben am Ende alle Chromosomen einen Fitneßwert, der 20% unter dem des Optimums liegt. Populationsgrößen von 25 und 40 ergeben mit jeweils 50 Generationen das gleiche Ergebnis, nämlich einen Wert 12% unter dem Optimum. Eine Erhöhung der Generationsanzahl würde in diesem Fall keine weitere Verbesserung bringen, da bereits alle Chromosomen gleich sind. Selbst eine Mutation würde keine dauerhafte Verbesserung bedeuten, da die „Übermacht“ der anderen Chromosomen einfach zu groß ist.

Ein gutes Ergebnis bringt eine Populationsgröße von 50. Nach 30 Generationen haben nur mehr neun Chromosomen einen Fitneßwert 12% unter dem Optimum, alle anderen haben die bestmögliche Fitneß; bereits nach 40 Generationen haben alle Chromosomen die optimale Fitneß.

Zeitaufwand

Alle Programmläufe sind auf einem PC mit 80486 DX/2 - Prozessor (Taktfrequenz 66 MHz) mit 16 Megabyte Hauptspeicher implementiert. Die Dauer eines solchen Laufs hängt sehr stark von der Populationsgröße ab, wie man aus folgender Übersicht entnehmen kann:

Populationsgröße 10	80 Generationen	15 Minuten
Populationsgröße 25	50 Generationen	22 Minuten
Populationsgröße 40	50 Generationen	36 Minuten
Populationsgröße 50	30 Generationen	26 Minuten
Populationsgröße 50	40 Generationen	35 Minuten

Dies bedeutet, daß die Verbindung von GPSS/H™ mit C++ (oder ähnlichen Programmiersprachen) vom zeitlichen Aufwand eine vielversprechende Perspektive darstellt. Insbesonders wenn man bedenkt, daß in diesem Beispiel noch nicht die schnellste mögliche Variante implementiert ist, sondern noch der Umweg über ein File verwendet wird.

Implementierung in MicroSaint

Für die Implementierung eines Genetischen Algorithmus in MicroSaint™ wird ein ähnliches Beispiel wie in den vorangegangenen Unterkapiteln herangezogen (vgl. /SB94b/ und /SB95/):

Innerhalb einer Fertigung werden zwei Produkte erzeugt (Produkt A, Produkt B). Vom Produkt A werden 30 Stück hergestellt, vom Produkt B 20 Stück. Die Fertigung wird mit vier unterschiedlichen Maschinenarten betrieben (M1-M4). Vom Maschinentyp M1 gibt es vier Stück, von M2, M3 und M4 jeweils zwei Stück mit unterschiedlichen Kostenmultiplikatoren, die die Abnutzung der Maschinen beschreiben.

Maschinentyp	Maschinennummer	Kostenmultiplikator
I	1	1,2
I	2	1,4
I	3	1,6
I	4	1,8

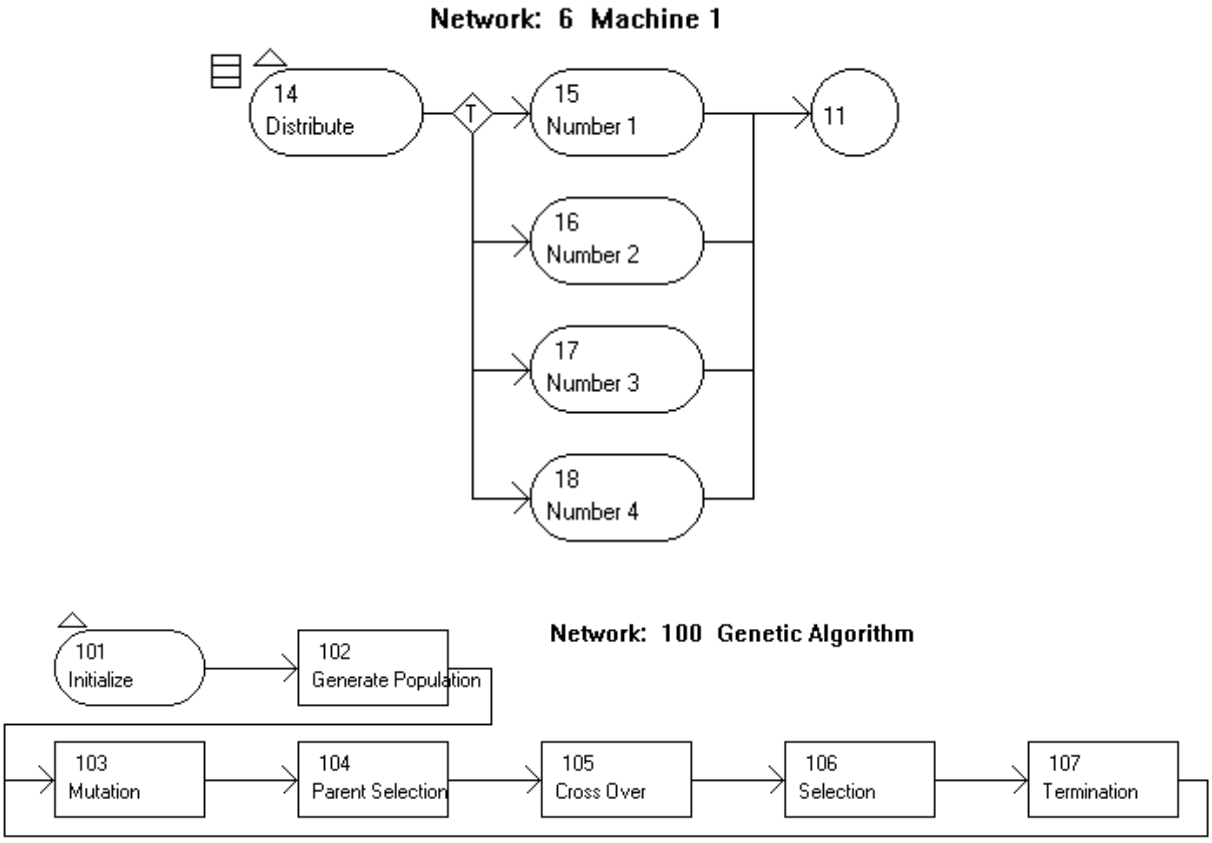
II	5	1,3
II	6	1,7
III	7	1,1
III	8	1,5
IV	9	1,6
IV	10	1,9

Die Kostenmultiplikatoren werden verwendet, da ansonsten zu viele nicht unterscheidbare Zustände auftreten können und es daher nicht nur **ein** globales Optimum gäbe. Weiters sind acht Arbeiter in der Abteilung beschäftigt, wobei jede Maschine von einem Arbeiter bedient werden muß. Das bedeutet zwangsläufig, daß jeweils zwei Maschinen zu einem bestimmten Zeitpunkt nicht sinnvoll genutzt werden können. Zur Fertigstellung benötigt das Produkt A die Maschinentypen I, III, I in genau dieser Reihenfolge und B hat die Arbeitsreihenfolge II, I, IV, III, I. Der betrachtete Zeitraum entspricht einer 8-Stunden-Schicht.

Spätestens nach 5,5 Stunden sind alle Produkte des Typs A im Modell vorhanden, nach 4 Stunden alle Produkte des Typs B. Wenn ein Produkt innerhalb der Schicht nicht fertiggestellt werden kann, muß es zugekauft werden, um den Auftrag zu erfüllen.

Programmierung in MicroSaint

Im Gegensatz zum vorigen Modell wird in der Lösungsvariante mit MicroSaint sowohl der Genetische Algorithmus als auch das eigentliche Simulationsprogramm in MicroSaint™-typischen Networks realisiert. Durch die Tatsache, daß sich der gesamte Optimierungsprozeß unter Microsoft Windows™ abspielt, nimmt man zwar einen Geschwindigkeitsverlust in Kauf, aber dafür steigt der Komfortfaktor beträchtlich. Zur Veranschaulichung sei hier exemplarisch ein Network einer Maschine und das Network des eigentlichen Genetischen Algorithmus abgebildet:



Für die Darstellung der restlichen benötigten und entworfenen Networks sei an dieser Stelle auf den Anhang C verwiesen.

Beschreibung des Genetischen Algorithmus

Generate Population

Hier wird die Initial Population mit Größe *g_size* generiert. Die Variable *g_pop[i,j]* enthält das Bitmuster dieser Population. Wobei *i* die Nummer des Mitglieds, *j* die Nummer des Bits angibt. An der Stelle *g_pop[i,0]* steht das Bitmuster umgerechnet in eine Integer Zahl. In *g_fitness[i]* wird die Fitneß gespeichert, wobei jedes Mitglied der Population evaluiert (die Fitneß berechnet) wird.

Evaluation:

Das Bitmuster wird in den M-Vektor übersetzt, der die Schnittstelle zwischen GA und Modell bildet. Nach Beenden der Evaluation wird die Fitneß zugewiesen.

Mutation:

Jedes Mitglied der Population wird mit Wahrscheinlichkeit *g_mutate* mutiert. Das passiert folgendermaßen: Ein Bit des Bitmusters wird zufällig „geflippt“, danach wird überprüft, ob das neu erhaltene Bitmuster in der Elterngeneration bereits vorhanden ist. Wenn das der Fall ist, wird die Fitneß übernommen, ansonsten wird es neu evaluiert.

Parent Selection

Aus der gesamten Population werden *g_size/2* Elternpaare ausgewählt, wobei kein Mitglied mit sich selbst ein Paar bilden darf, gleiche Elternpaare jedoch durchaus zulässig sind. Die verwendete Auswahlmethode ist die Rouletteradmethode.

Crossover

Die Elternpaare werden nun zum Crossover gebracht, wodurch *g_size* „neue“ Bitmuster entstehen. Diese werden danach sofort evaluiert, sofern nicht dasselbe Bitmuster bereits in der Elterngeneration vorhanden ist.

Selection

Aus den nun vorhandenen Eltern und Kindern (Anzahl = *g_size * 2*) wird die neue Elternpopulation ausgewählt. Die Auswahlmethode ist abermals die Rouletteradmethode.

Termination

Falls sich nach *g_steps* keine Verbesserung ergibt, wird abgebrochen. Nach jeder Generation wird der beste Fitneßwert gespeichert.

Testen des Genetischen Algorithmus

Die getesteten Populationen haben die Größen *g_size* = 12, 24, 48, 96 und 100. Erst bei einer Größe von 96 kommen zufriedenstellende Ergebnisse zustande. Kleinere Populationen liefern keine optimalen Werte. Das Modell ist derart modifiziert, daß man die optimale Lösung im Vorhinein kennt. Der erwartete optimale Wert ist 1820. Dieser Wert wird bei Populationen von *g_size*=12 bis 48 auch nach 100 Generationen kein einziges Mal erreicht.

Die anfängliche Fitneßfunktion sieht folgendermaßen aus:

m_result + 400 = fitness

Dabei stellt der Wert *400* den Skalierungssummanden dar. Bessere Ergebnisse wurden mit einem Summanden von *1000* erzielt. Bei *g_size=96* taucht der optimale Wert das erste Mal nach 86 Generationen auf und die Population verläßt nach 150 Generationen diesen Bereich des globalen Optimums praktisch nicht mehr.

Zeitaufwand

Dieses Beispiel ist auf einem PC 80486 mit 33 MHz Taktfrequenz implementiert und benötigt dort folgende Zeiten:

- Durchschnittliche Dauer einer Evaluation: 21 Sekunden
- Dauer der GA-Mutation: 0.5 Sekunden
- Parent Selection: 0.5 Sekunden
- Crossover: 0.5 Sekunden
- Selection: 0.5 Sekunden
- Initialisierung: 31 Minuten
- Generierung der 2.Generation: 21 Minuten

In höheren Generationen treten öfter gleiche Bitmuster auf, daher wird auch weniger oft evaluiert und es ergibt sich daher mit steigender Generationsanzahl eine kürzere Evaluierungszeit pro Generation. Generell läßt sich noch sagen, je mehr Fenster während der Simulation gleichzeitig geöffnet sind, desto langsamer wird MicroSaint™ und damit klarerweise der gesamte Optimierungsprozeß.

Vergleich und Validierung der Implementierungen

Die drei hier vorgestellten Ansätze werden auf sehr ähnliche Beispiele der Fertigungstechnik mit Erfolg angewendet. Jedes der Modelle findet den optimalen Punkt des Suchraums und doch gibt es große Unterschiede zwischen den Implementierungen, da den Simulationssprachen zu unterschiedliche Konzepte zu Grunde liegen.

D_SIM ist sehr allgemein einsetzbar, benötigt aber bei der Adaptierung an neue Erforderungen - wie es die Einbindung einer Optimierung mit Genetischen Algorithmen darstellt - einen großen Aufwand, der sich am Schluß bezahlt macht, da man eine anspruchsvolle Oberfläche zur Verfügung hat. Da D_SIM unter Windows™ läuft, hat man es zwar mit den Windows™-typischen Vorteilen zu tun, nimmt aber klarerweise längere Laufzeiten in Kauf.

GPSS/H™ liefert - wie nicht anders zu erwarten ist - das von der Laufzeit her beste Ergebnis, allerdings ist hier keine so gute Überwachung des Optimierungsfortschritts gegeben und man verzichtet auf die Annehmlichkeiten von visuellen Umgebungen. Da für die meisten Anwendungen der Zeitfaktor nicht unerheblich ist und wenn man bedenkt, daß Modelle in der realen Anwendungssituation meistens umfangreicher sind, kann man sagen, daß für größere Optimierungen im Bereich der diskreten Simulation diese Variante von den drei vorgestellten die aussichtsreichste darstellt.

MicroSaint™ liefert von der CPU-Zeit her keine besonders guten Werte, ist aber in der Modellbildung sicher von den drei Alternativen das am einfachsten zu handhabende Tool und daher sicher für Optimierungen vernünftig einzusetzen, wo der Zeitfaktor nicht so gravierend ins Gewicht fällt und wo unter Umständen ein Computer einige Zeit für das Verfahren zur Verfügung gestellt werden kann

Zusammenfassung & Ausblick

Wie das bis jetzt Dargelegte beweist, ist eine sinnvolle Verwendung naturanaloger Verfahren in der Gestalt von Genetischen Algorithmen auf den Bereich der diskreten Computersimulation sinnvoll. Es zeigt sich, daß die hier vorgebrachten Implementierungen brauchbare Ergebnisse unter den unterschiedlichsten Voraussetzungen liefern.

In dieser Arbeit sind hauptsächlich Anregungen und Ideen zum erfolgreichen Einsatz dargelegt, da naturgemäß der Bereich der diskreten Simulation so mannigfaltige Bereiche besitzt, daß es unmöglich ist, eine wirklich vollständige Darstellung zu präsentieren. Auch der Bereich der naturanalogen Verfahren wird immer umfangreicher und viele der dort eingesetzten Methoden sind sicher auch mit geeigneter Adaption auf den Bereich der Simulation übertragbar.

Gerade der Bereich der Implementierungen in diesem Bereich verspricht für die Zukunft ein lohnendes Aufgabenfeld, da sich nur durch eine größere Anzahl von konkreten Beispielen aus den daraus möglichen Vergleichen die notwendigen allgemeinen Rückschlüsse ziehen lassen.

Zum Abschluß und als Abrundung dieser Dissertation sei hier noch ein Medium und dessen Verwendung angeführt, das in letzter Zeit in aller Munde ist und gerade für das wissenschaftliche Arbeiten in der Zukunft (nicht nur in Bereichen der Computertechnologie und -wissenschaften oder der naturanalogen Verfahren) ungeahnte Perspektiven bietet:

Das Internet als essentielle Kommunikationsform

Im Laufe der letzten Jahre hat die Bedeutung der weltweiten Kommunikation mittels Computernetzwerken einen rasanten Aufstieg erlebt. Kaum ein Gebiet kann sich diesem Boom verschließen. Auch wissenschaftliche Kommunikation, sowohl die einseitige in Form von Wissensbeschaffung, als auch die zweiseitige in Form von Diskussion und Gedankenaustausch läuft immer mehr über derlei Kanäle.

Anschließend ist eine Kurzübersicht über die wichtigsten Bereiche gegeben, da gerade so expandierende Bereiche wie Genetische Algorithmen oder Simulation naturgemäß von diesen Entwicklungen sehr stark profitieren.

Email

Email ist die Abkürzung für Electronic Mail und dient zur Übermittlung von Nachrichten (meistens Texten) zwischen Benutzern miteinander verbundener Rechner. Damit diese Kommunikation funktionieren kann, benötigt man ähnlich wie im herkömmlichen Postwesen eine genau definierte Adresse, an welche man mit den Befehlen des verwendeten Mailprogramms eine Nachricht schicken kann.

Im einfachsten Fall verbirgt sich hinter einer solchen Adresse eine Einzelperson, mit der Botschaften ausgetauscht werden können. Da die meisten Mailsysteme nach wie vor ASCII-dominiert sind (in vielen Fällen ist nur der amerikanische Standardzeichensatz zur Übertragung geeignet) ist eine im Vergleich zu herkömmlichen schriftlichen Kommunikationsformen andere Schreibweise anzuwenden, um dem Geschriebenen die nötige Betonung zu geben.

So ist durch das Fehlen von Unterstreichungen, Fettdruck und anderen Hervorhebungstechniken eine andere Kennzeichnung von wichtigen Textteilen notwendig. Üblicherweise wird solcherlei durch die Verwendung von Sonderzeichen wettgemacht (so kann z.B.: *wichtig* als email-Variante von ***wichtig*** vorkommen). Die Verwendung von sprachspezifischen Sonderzeichen (im Deutschen: ö, ä, ü, ß) ist zu vermeiden, da manche Mailsysteme diese Zeichen nicht richtig interpretieren können (Ersatz: oe, ae, ue, ss oder “o, “a, “u, “s).

Die nächste wichtige Angelegenheit ist es - ähnlich wie in anderen geschriebenen Kommunikationsformen auch- die Intention des Schreibers klarzulegen. Gerade im Bereich des Internet, wo verschiedene Kulturen aufeinanderprallen, ist dies eine notwendige Angelegenheit, um gröbere Mißverständnisse zu vermeiden. So gibt es zum Beispiel zur Kennzeichnung witzig (ironisch, etc.) gemeinter Textpassagen das sogenannte Smiley (oder ‘tongue in cheek’), welches folgendermaßen aussehen kann: **:-)**.

Neben anderen zusammengefügten Symbolen ist das Internet von einer Sucht nach Abkürzungen jeglicher Art erfaßt, die teilweise allgemeingültig (ASAP - as soon as possible, IMHO - in my humble opinion, etc.) oder Benutzer(gruppen)spezifisch (SGA-C - simple Genetic Algorithm written in C) sind.

Hinter einer email-Adresse kann auch eine sogenannte mailing list stehen, welches bedeutet, daß die mail, die an diese Adresse geschickt wird, an alle Mitglieder dieser Liste weitergeleitet wird. Da mailing lists immer ein spezielles Gebiet abdecken (nicht immer notwendigerweise rein wissenschaftlicher Natur), hat man hier ein Instrument in der Hand, an andere Menschen zu schreiben, die das gleiche Interesse besitzen, ohne alle mail-Adressen selbst besitzen zu müssen.

Die Verwaltung erfolgt in den meisten Fällen von einem list owner, der mail-Adressen von Interessierten in den Verteilerkreis aufnimmt, diese Adressen bei Bedarf oder auf Wunsch wieder entfernt, bzw. die Gestalt der mails, die weitergeleitet werden, bestimmt, indem er z. B. einen header oder eine signature einfügt oder die einzelnen emails kürzt, kommentiert oder als irrelevant für die Gesamtheit der Liste verwirft.

Um in eine mailing list aufgenommen zu werden, schickt man üblicherweise eine mail an eine sogenannte request-Adresse (z.B. /COMP_1/) mit dem Kennwort ‘subscribe’. Daraufhin erhält man üblicherweise eine automatisch generierte Antwort, die unter Umständen weitere Anweisungen und zusätzliche Information enthält. Wichtig ist in diesem Zusammenhang den Unterschied zwischen der request-Adresse (sie dient administrativen Dingen) und der „normalen“ Adresse zu beachten.

Mails an die „normale“ Adresse mit administrativem Inhalt werden allgemein nicht gern gesehen. Der optische Unterschied sei anhand der Adresse vom Genetic Algorithm Digest dargelegt (/COMP_1/): ga-list@aic.nrl.navy.mil ist die „normale“ Adresse, an die alle Beiträge geschickt werden sollen, die an die weiteren Mitglieder dieser mailing list (Mitgliederstand Ende 1994: mehr als 2800 mail-Adressen) geschickt werden sollen, ga-list-request@aic.nrl.navy.mil ist die administrative Adresse, an die Wünsche an den Betreiber deponiert werden können, die nicht sämtliche Mitglieder angeht oder den Umfang der Liste sprengen könnte. Im Literaturverzeichnis der vorliegenden Arbeit sind nur die administrativen Adressen angegeben, allerdings sind die anderen Adressen leicht daraus abzuleiten.

Wenn man es nun geschafft hat, erfolgreich eine mailing list zu „subscriben“, ist es ratsam, vor der ersten eigenen mail an die ganze Liste den Ton und Inhalt etwas zu studieren, um nicht aus der Rolle zu fallen, bzw. Altbekanntes neu aufzuwärmen. Um sich einen noch besseren Überblick zu verschaffen, bieten die meisten Listen dieser Art ein sogenanntes FAQ (= Frequently Asked Questions) an, in dem häufig gestellte Fragen (und deren Antworten) und noch weitere informative Dinge zu finden sind. Es ist höchst ratsam, dieses genau durchzulesen.

Mailing lists bieten, ähnlich wie andere Internet-Foren, den Vorteil in „relativ“ formloser Weise über neue Entwicklungen zu diskutieren. Dies geschieht mediengemäß in weitaus schnellerer Form als in herkömmlichen Publikationen und kann dazu dienen, eigene Veröffentlichungen vor der Drucklegung noch einmal auf Fehler zu untersuchen. Weiters sind in solchen Listen Querverweise auf andere Internet-sourcen, Tagungsankündigungen, Veröffentlichungshinweise, Quellen von Source-codes oder ähnlichem zu finden.

Newsgroups

Im Gegensatz zu mails, die immer an einen einzelnen Empfänger oder eine klar definierte Empfängergruppe adressiert sind, versteht man unter News bzw. Newsartikeln Nachrichten, die lokal oder weltweit zwischen sogenannten Newsservern ausgetauscht werden und von einem nicht näher definierbaren anonymen Personenkreis gelesen werden. Newsgroups werden daher auch gerne als die Anschlagbretter („schwarzen Bretter“) des Internets bezeichnet.

Zum Zweck der einfacheren Orientierung in der angebotenen Informationsmenge sind die Newsartikel thematisch in Newsgroups eingeteilt, die wiederum in mehrere Hauptkategorien unterteilt werden können. Diese Unterscheidungen sind an den ersten Zeichen des Namens einer Newsgroup zu erkennen, so stehen z.B. comp.ai.genetic und comp.ai für Newsgroups, die spezielle Themen für Computerfachleute (Beginn: *comp*) behandeln und Gruppen wie sci.op-research oder sci.math.num-analysis beschäftigen sich eher mit Themen über spezielle Wissensgebiete (Beginn: *sci*) und deren Anwendungen. Bei vielen Anwendern beliebt sind auch die Newsgroups, die mit *rec* (Hobbys, Spiele und Freizeitgestaltung) oder *alt* (alternativ) beginnen.

Die jeweiligen Artikel der Newsgroups werden von lokalen Mailservern empfangen und für einen bestimmten Zeitraum aufgehoben, so daß sie von den Interessierten gelesen und bei Bedarf gespeichert werden können. Zu den Vorteilen von News gehören das große Angebot an Themenkreisen, das große Wissenspotential der Teilnehmer und die Möglichkeit, auf Anfragen rasch Antworten zu bekommen.

In diesem Zusammenhang sind die Gruppen news.lists, news.announce.newusers, news.groups und - speziell für Österreich - at.news zu empfehlen, in denen Einführungsartikel zur Verwendung des Newsservices, Übersichten über verfügbare Newsgruppen, Informationen über neu eingerichtete Newsgruppen und ähnliches verteilt werden.

World Wide Web

Das sogenannte World Wide Web (WWW) hat in letzter Zeit anderen ähnlich gearteten Informationssystemen, wie etwa dem Gopher, deutlich den Rang abgelaufen. Mit einem geeigneten Anwendungsprogramm wählt man eine Adresse an und erhält dann in ansprechender grafischer Form Information dargeboten. Darüber hinaus können Verweise auf andere Stellen im Internet gesetzt sein, so daß man durch simples Anklicken (ohne zusätzliche Angabe einer neuen Adresse) zu anderen interessanten Bereichen gelangen kann.

Man könnte das WWW als riesige Illustrierte mit sich ständig aktualisierendem Inhalt bezeichnen, welche in letzter Zeit rasant wächst. Als Beispiele für die vielfältigen Anwendungsmöglichkeiten sind im Literaturverzeichnis die Adressen („Homepages“) von einem Simulationsserver (/COMP_27/), von einem Server für Genetische Algorithmen (/COMP_26/) und die Adresse für EUROSIM (/COMP_25/), der Vereinigung europäischer Simulationsgesellschaften, aufgeführt.

FTP

FTP ist die Abkürzung für File Transfer Programm und steht, wie der Name schon sagt, für das Übertragen von Files verschiedenster Provenience. Üblicherweise kann man sich mit Hilfe dieses Tools Source-codes von Programmen, exekutierbare Programme, Demos von Programmen, Vorträge, Auszüge aus wissenschaftlichen

Arbeiten bis hin zu kompletten Dissertationen (meistens im Postscript Format) auf den eigenen Rechner holen.

Dies bedingt natürlich, daß man weiß, wo man was holen kann und da kommen die vorhin erwähnten anderen Internetanwendungsbereiche zum Tragen, da in mailing lists oder Newsgroups häufig von Arbeiten oder Programmen zu lesen ist, die an bestimmten Stellen im Internet per FTP geholt werden können. WWW-Anwendungsprogramme stellen in dieser Hinsicht in vielen Fällen eine direkte Verbindung her, da man mit ihnen die gewünschten Files ohne manuelle Inanspruchnahme eines FTP-Programms auf den eigenen Rechner kopieren kann.

Anhang A: D_SIM - Teilpetrinetze

Maschinen vom Typ I

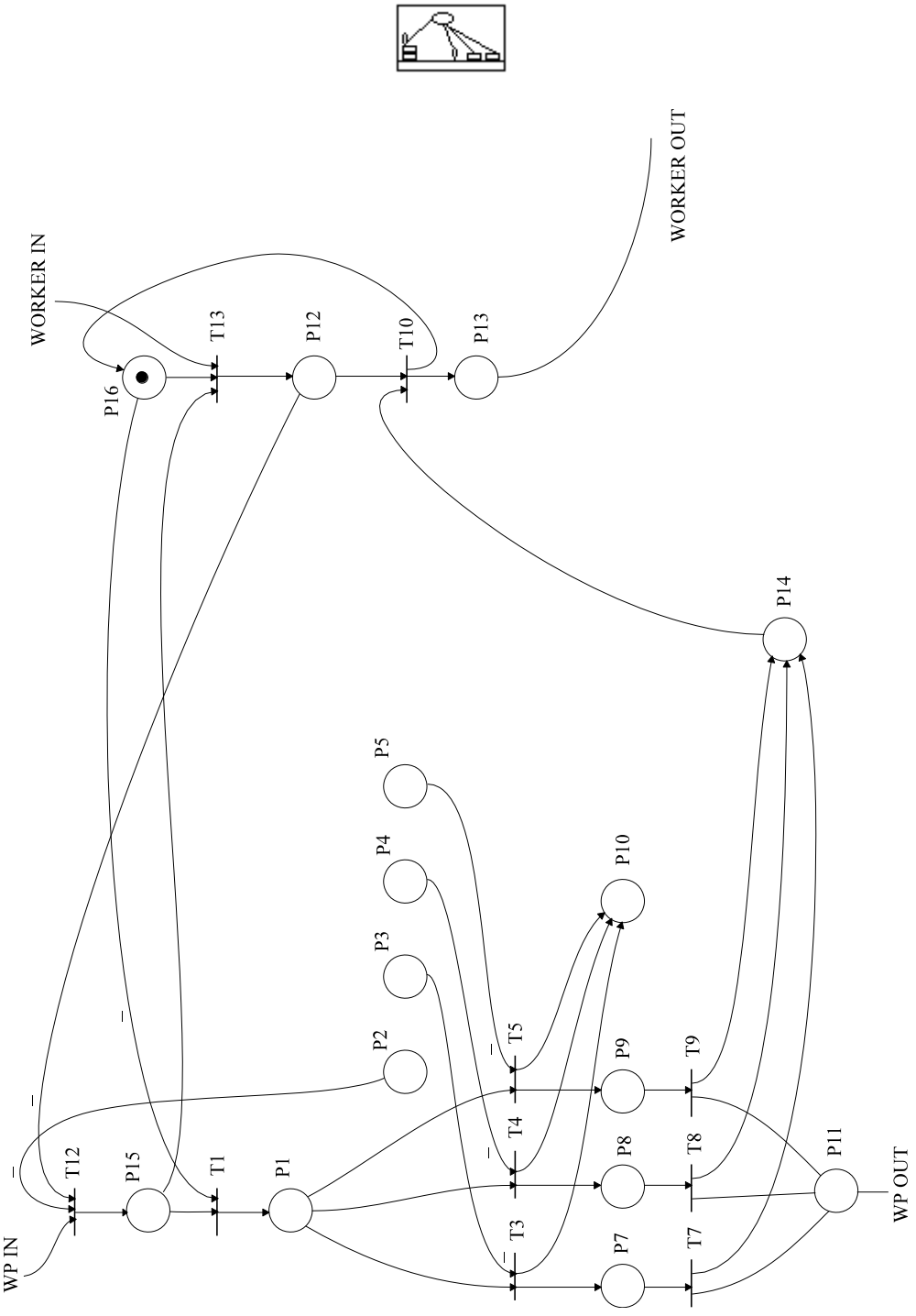
```
places:
e:p15       Eingang
e:p2        Z00
e:p3        Z01
e:p4        Z10
e:p5        Z11
e:p10       Kosten
e:p16       Worker
end:
attribute:
MObType=work
Zeit_Z00    1
Kosten_Z00      1
Zeit_Z01    1
Kosten_Z01      1
Zeit_Z10    1
Kosten_Z10      1
Zeit_Z11    1
Kosten_Z11      1
end:
In_Output:
wpin        Input
wpout       Output
workerin            Input
workerout   Output
end:
description:
TRANSITION      e:t1    -1      ""      -1
TRANSITION      e:t3    -1      ""      -1
TRANSITION      e:t4    -1      ""      -1
TRANSITION      e:t5    -1      ""      -1
TRANSITION      e:t7    -1      ""      -1
TRANSITION      e:t8    -1      ""      -1
TRANSITION      e:t9    -1      ""      -1
TRANSITION      e:t10   -1      ""      -1
TRANSITION      e:t12   a:op    "a:X"   -1
TRANSITION      e:t11   4
                "((0&&2&&~4)&&-1)||((1&&0&&3&&2&&~4)&&-2)"    -1
TRANSITION      e:t13   -1      ""      -1
PLACE           e:p1    1       1
PLACE           e:p2    2       1
PLACE           e:p3    2       1
PLACE           e:p4    2       1
```

```
PLACE          e:p5    2       1
PLACE          e:p7    1       a:Zeit_Z01
PLACE          e:p8    1       a:Zeit_Z10
PLACE          e:p9    1       a:Zeit_Z11
PLACE          e:p10   30000   1
PLACE          e:p11   1       1
PLACE          e:p12   1       1
PLACE          e:p13   1       1
PLACE          e:p14   1       1
PLACE          e:p15   2       1
PLACE          e:p16   1       1
CONNECTION     wpin    1       t11
CONNECTION     wpin    1       t12
CONNECTION     t11     2       p15
CONNECTION     t12     2       p15
CONNECTION     p15     1       t1
CONNECTION     t1      1       p1
CONNECTION     p2      0       t11
CONNECTION     p2      0       t12
CONNECTION     p1      1       t3
CONNECTION     p3      0       t3
CONNECTION     p1      1       t4
CONNECTION     p4      0       t4
CONNECTION     p1      1       t5
CONNECTION     p5      0       t5
CONNECTION     t3      1       p7
CONNECTION     t3      a:Kosten_Z01 p10
CONNECTION     t4      1       p8
CONNECTION     t4      a:Kosten_Z10 p10
CONNECTION     t5      1       p9
CONNECTION     t5      a:Kosten_Z11 p10
CONNECTION     p7      1       t7
CONNECTION     p8      1       t8
CONNECTION     p9      1       t9
CONNECTION     t7      1       p11
CONNECTION     t8      1       p11
CONNECTION     t9      1       p11
CONNECTION     t7      1       p14
CONNECTION     t8      1       p14
CONNECTION     t9      1       p14
CONNECTION     p11     1       wpout
CONNECTION     workerin        1       t13
CONNECTION     t13     1       p12
CONNECTION     p12     1       t10
CONNECTION     t10     1       p13
CONNECTION     p13     1       workerout
CONNECTION     p16     1       t13
CONNECTION     p16     0       t1
CONNECTION     p15     1       t13
```

CONNECTION	p12	0	t12
CONNECTION	p12	0	t11
CONNECTION	t10	1	p16
CONNECTION	p14	1	t10

end:

Maschinen vom Typ II, III, IV



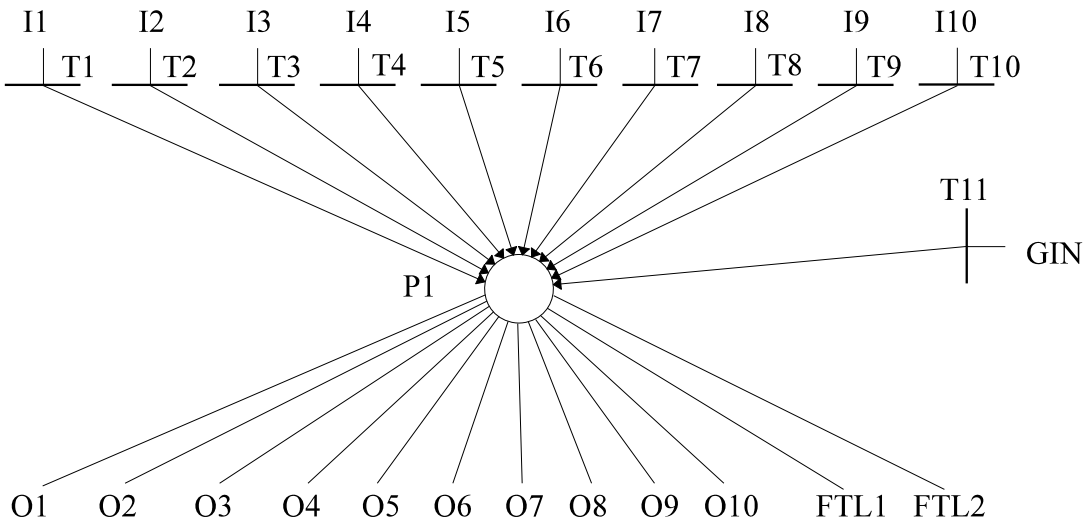
places:

e:p15	Eingang
e:p2	Z00

```
e:p3        Z01
e:p4        Z10
e:p5        Z11
e:p10       Kosten
e:p16       Worker
end:
attribute:
MObType=work
Zeit_Z00    1
Kosten_Z00      1
Zeit_Z01    1
Kosten_Z01      1
Zeit_Z10    1
Kosten_Z10      1
Zeit_Z11    1
Kosten_Z11      1
end:
In_Output:
wpin        Input
wpout       Output
workerin            Input
workerout   Output
end:
description:
TRANSITION      e:t1    a:op    ""      -1
TRANSITION      e:t3    -1      ""      -1
TRANSITION      e:t4    -1      ""      -1
TRANSITION      e:t5    -1      ""      -1
TRANSITION      e:t7    -1      ""      -1
TRANSITION      e:t8    -1      ""      -1
TRANSITION      e:t9    -1      ""      -1
TRANSITION      e:t10   -1      ""      -1
TRANSITION      e:t12   -1      "a:X"   -1
TRANSITION      e:t13   -1      ""      -1
PLACE           e:p1    1       1
PLACE           e:p2    2       1
PLACE           e:p3    2       1
PLACE           e:p4    2       1
PLACE           e:p5    2       1
PLACE           e:p7    1       a:Zeit_Z01
PLACE           e:p8    1       a:Zeit_Z10
PLACE           e:p9    1       a:Zeit_Z11
PLACE           e:p10   30000   1
PLACE           e:p11   1       1
PLACE           e:p12   1       1
PLACE           e:p13   1       1
PLACE           e:p14   1       1
PLACE           e:p15   2       1
PLACE           e:p16   1       1
```

```
CONNECTION  wpin    1       t12
CONNECTION  t12     2       p15
CONNECTION  p15     1       t1
CONNECTION  t1      1       p1
CONNECTION  p2      0       t12
CONNECTION  p1      1       t3
CONNECTION  p3      0       t3
CONNECTION  p1      1       t4
CONNECTION  p4      0       t4
CONNECTION  p1      1       t5
CONNECTION  p5      0       t5
CONNECTION  t3      1       p7
CONNECTION  t3      a:Kosten_Z01 p10
CONNECTION  t4      1       p8
CONNECTION  t4      a:Kosten_Z10 p10
CONNECTION  t5      1       p9
CONNECTION  t5      a:Kosten_Z11 p10
CONNECTION  p7      1       t7
CONNECTION  p8      1       t8
CONNECTION  p9      1       t9
CONNECTION  t7      1       p11
CONNECTION  t8      1       p11
CONNECTION  t9      1       p11
CONNECTION  t7      1       p14
CONNECTION  t8      1       p14
CONNECTION  t9      1       p14
CONNECTION  p11     1       wpout
CONNECTION  workerin        1       t13
CONNECTION  t13     1       p12
CONNECTION  p12     1       t10
CONNECTION  t10     1       p13
CONNECTION  p13     1       workerout
CONNECTION  p16     1       t13
CONNECTION  p16     0       t1
CONNECTION  p15     1       t13
CONNECTION  p12     0       t12
CONNECTION  t10     1       p16
CONNECTION  p14     1       t10
end:
```

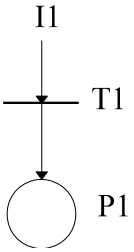
Zwischenlager




```
places:
e:p1        Storage
end:
attribute:
MObType=simple
end:
In_Output:
I1  Input
I2  Input
I3  Input
I4  Input
I5  Input
I6  Input
I7  Input
I8  Input
I9  Input
I10 Input
gin Input
O1 Output
O2 Output
O3 Output
O4 Output
O5 Output
O6 Output
O7 Output
O8 Output
O9 Output
O10         Output
ftl1 Output
ftl2 Output
end:
description:
TRANSITION    e:t1    -1      ""      -1
TRANSITION    e:t2    -1      ""      -1
TRANSITION    e:t3    -1      ""      -1
```

```
TRANSITION    e:t4    -1    ""    -1
TRANSITION    e:t5    -1    ""    -1
TRANSITION    e:t6    -1    ""    -1
TRANSITION    e:t7    -1    ""    -1
TRANSITION    e:t8    -1    ""    -1
TRANSITION    e:t9    -1    ""    -1
TRANSITION    e:t10   -1    ""    -1
TRANSITION    e:t11   -1    ""    -1
PLACE         e:p1    50    1
CONNECTION    p1      1     ftl1
CONNECTION    p1      1     ftl2
CONNECTION    I1      1     t1
CONNECTION    I2      1     t2
CONNECTION    I3      1     t3
CONNECTION    I4      1     t4
CONNECTION    I5      1     t5
CONNECTION    I6      1     t6
CONNECTION    I7      1     t7
CONNECTION    I8      1     t8
CONNECTION    I9      1     t9
CONNECTION    I10     1     t10
CONNECTION    gin     1     t11
CONNECTION    t1      1     p1
CONNECTION    t2      1     p1
CONNECTION    t3      1     p1
CONNECTION    t4      1     p1
CONNECTION    t5      1     p1
CONNECTION    t6      1     p1
CONNECTION    t7      1     p1
CONNECTION    t8      1     p1
CONNECTION    t9      1     p1
CONNECTION    t10     1     p1
CONNECTION    t11     1     p1
CONNECTION    p1      1     O1
CONNECTION    p1      1     O2
CONNECTION    p1      1     O3
CONNECTION    p1      1     O4
CONNECTION    p1      1     O5
CONNECTION    p1      1     O6
CONNECTION    p1      1     O7
CONNECTION    p1      1     O8
CONNECTION    p1      1     O9
CONNECTION    p1      1     O10
end:
```

Lager der fertigen Teile

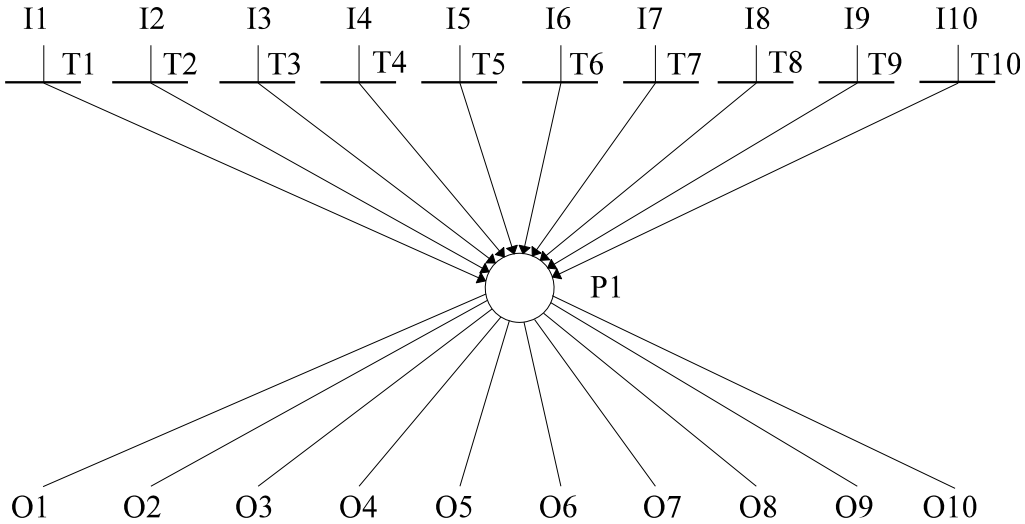


places:
e:p1 Sink
end:
attribute:
MObType=work
end:
In_Output:
I1 Input
end:
description:

TRANSITION	e:t1	-1	"a:X" -1
PLACE	e:p1	50	1
CONNECTION	I1	1	t1
CONNECTION	t1	1	p1

end:

Aufenthaltraum der Arbeiter



places:
e:p1 Workers
end:
attribute:
MObType=simple

```
end:
In_Output:
I1  Input
I2  Input
I3  Input
I4  Input
I5  Input
I6  Input
I7  Input
I8  Input
I9  Input
I10 Input
O1 Output
O2 Output
O3 Output
O4 Output
O5 Output
O6 Output
O7 Output
O8 Output
O9 Output
O10         Output
end:
description:
TRANSITION      e:t1    -1      ""      -1
TRANSITION      e:t2    -1      ""      -1
TRANSITION      e:t3    -1      ""      -1
TRANSITION      e:t4    -1      ""      -1
TRANSITION      e:t5    -1      ""      -1
TRANSITION      e:t6    -1      ""      -1
TRANSITION      e:t7    -1      ""      -1
TRANSITION      e:t8    -1      ""      -1
TRANSITION      e:t9    -1      ""      -1
TRANSITION      e:t10   -1      ""      -1
PLACE           e:p1    10      1
CONNECTION      I1      1       t1
CONNECTION      I2      1       t2
CONNECTION      I3      1       t3
CONNECTION      I4      1       t4
CONNECTION      I5      1       t5
CONNECTION      I6      1       t6
CONNECTION      I7      1       t7
CONNECTION      I8      1       t8
CONNECTION      I9      1       t9
CONNECTION      I10     1       t10
CONNECTION      t1      1       p1
CONNECTION      t2      1       p1
CONNECTION      t3      1       p1
CONNECTION      t4      1       p1
```

```
CONNECTION    t5      1       p1
CONNECTION    t6      1       p1
CONNECTION    t7      1       p1
CONNECTION    t8      1       p1
CONNECTION    t9      1       p1
CONNECTION    t10     1       p1
CONNECTION    p1      1       O1
CONNECTION    p1      1       O2
CONNECTION    p1      1       O3
CONNECTION    p1      1       O4
CONNECTION    p1      1       O5
CONNECTION    p1      1       O6
CONNECTION    p1      1       O7
CONNECTION    p1      1       O8
CONNECTION    p1      1       O9
CONNECTION    p1      1       O10
end:
```

Anhang B: GPSS/H - Source-code

Programmcode des GPSS/H-Modells

```
SIMULATE
INTEGER    &COUNT
INTEGER    &TYPA,&TYPB
INTEGER    &STOPFLAG
INTEGER    &POPSIZE
LET        &POPSIZE=50
INTEGER    &ZUSTAND(50)          Feldgroesse = POPSIZE !
INTEGER    &FITNESS(50)  Feldgroesse = POPSIZE !
INTEGER    &ZUSTMI
INTEGER    &ZUSTMII
INTEGER    &ZUSTMIII
INTEGER    &ZUSTMIV
INTEGER    &LAUFNR
LET        &LAUFNR=0
*
 AUSGABE  PICTURE
 ****************************************_
 ****************************************
*
 ARBZEIT  MATRIX     MX,4,4
INITIAL    MX$ARBZEIT(1,1),0
INITIAL    MX$ARBZEIT(1,2),24
INITIAL    MX$ARBZEIT(1,3),20
INITIAL    MX$ARBZEIT(1,4),12
INITIAL    MX$ARBZEIT(2,1),0
INITIAL    MX$ARBZEIT(2,2),20
INITIAL    MX$ARBZEIT(2,3),18
INITIAL    MX$ARBZEIT(2,4),15
INITIAL    MX$ARBZEIT(3,1),0
INITIAL    MX$ARBZEIT(3,2),24
INITIAL    MX$ARBZEIT(3,3),20
INITIAL    MX$ARBZEIT(3,4),16
INITIAL    MX$ARBZEIT(4,1),0
INITIAL    MX$ARBZEIT(4,2),45
INITIAL    MX$ARBZEIT(4,3),37
INITIAL    MX$ARBZEIT(4,4),30
*
PUTPIC     FILE=BRIDGE,(-1)
 *
CLOSE      BRIDGE
SYSCALL    'genalg'
```

```
GETLIST
FILE=BRIDGE,(&STOPFLAG,(&ZUSTAND(&COUNT),&COUNT=1,&POPSIZE))
CLOSE      BRIDGE
*
 MASCHI   STORAGE    4
 MASCHII  STORAGE    2
 MASCHIII STORAGE    2
 MASCHIV  STORAGE    2
 ARBEITER STORAGE    8
*
*
GENERATE   ,,,30
ADVANCE    0
*
ENTER      MASCHI
ENTER      ARBEITER
ADVANCE    MX$ARBZEIT(1,&ZUSTMI)
LEAVE      ARBEITER
LEAVE      MASCHI
TEST NE    &ZUSTMI,1,SIMENDA
*
ENTER      MASCHIII
ENTER      ARBEITER
ADVANCE    MX$ARBZEIT(3,&ZUSTMIII)
LEAVE      ARBEITER
LEAVE      MASCHIII
TEST NE    &ZUSTMIII,1,SIMENDA
*
ENTER      MASCHI
ENTER      ARBEITER
ADVANCE    MX$ARBZEIT(1,&ZUSTMI)
LEAVE      ARBEITER
LEAVE      MASCHI
*
BLET       &TYPA=&TYPA+1
*
 SIMENDA  TERMINATE  0
*
*
GENERATE   ,,,20
ADVANCE    0
*
ENTER      MASCHII
ENTER      ARBEITER
ADVANCE    MX$ARBZEIT(2,&ZUSTMII)
LEAVE      ARBEITER
LEAVE      MASCHII
TEST NE    &ZUSTMII,1,SIMENDB
*
```

```
ENTER     MASCHI
ENTER     ARBEITER
ADVANCE   MX$ARBZEIT(1,&ZUSTMI)
LEAVE     ARBEITER
LEAVE     MASCHI
TEST NE   &ZUSTMI,1,SIMENDB
*
ENTER     MASCHIV
ENTER     ARBEITER
ADVANCE   MX$ARBZEIT(4,&ZUSTMIV)
LEAVE     ARBEITER
LEAVE     MASCHIV
TEST NE   &ZUSTMIV,1,SIMENDB
*
ENTER     MASCHIII
ENTER     ARBEITER
ADVANCE   MX$ARBZEIT(3,&ZUSTMIII)
LEAVE     ARBEITER
LEAVE     MASCHIII
TEST NE   &ZUSTMIII,1,SIMENDB
*
ENTER     MASCHI
ENTER     ARBEITER
ADVANCE   MX$ARBZEIT(1,&ZUSTMI)
LEAVE     ARBEITER
LEAVE     MASCHI
*
BLET      &TYPB=&TYPB+1
*
 SIMENDB  TERMINATE  0
*
*
GENERATE   480
TERMINATE  1
*
 REPEAT   DO       &COUNT=1,&POPSIZE,1
  LET      &ZUSTMI=&ZUSTAND(&COUNT)/1000
  LET      &ZUSTMII=(&ZUSTAND(&COUNT)@1000)/100
  LET      &ZUSTMIII=(&ZUSTAND(&COUNT)@100)/10
  LET      &ZUSTMIV=&ZUSTAND(&COUNT)@10
  LET      &TYPA=0
  LET      &TYPB=0
  CLEAR    MX$ARBZEIT
  START    1
*         Fitnessberechnung
*         Zuweisung der Fitness an FITNESS(&COUNT)
  LET      &FITNESS(&COUNT)=&TYPA+&TYPB
ENDDO
*
```

```
PUTPIC
FILE=BRIDGE,PICTURE=AUSGABE,(&STOPFLAG,(&FITNESS(&COUNT),&C
OUNT=1,&POPSIZE))
CLOSE      BRIDGE
SYSCALL    'genalg'
GETLIST
FILE=BRIDGE,(&STOPFLAG,(&ZUSTAND(&COUNT),&COUNT=1,&POPSIZE))
CLOSE      BRIDGE
*
IF         &STOPFLAG=0
  LET        &LAUFNR=&LAUFNR+1
  PUTPIC     (&LAUFNR)
 * . Generation
  GOTO       REPEAT
ENDIF
*
END
```

Programmcode des GA

```
#include <stdio.h>
#include <stdlib.h>

#define MAXGENERATIONNUMBER 40
#define POPSIZE 50
#define VERBINDUNGSDATEI "bridge"
#define SPEICHERDATEI "saveval"
#define BESTOFDATEI "champion"

typedef struct  {
                int zust[4];
                int fit;
                } pop;

/* Globale Variablen */

pop parents[POPSIZE];
pop children[POPSIZE];
int stopflag;
int nr_of_gen;

/* Prototypen */

void copy_pop( pop *, pop * );
void children_erzeugen( void );
void parents_reset( void );
void uebergeben( void );
void speichern( void );
void einlesen( void );
void besten_speichern( void );
void select_neue_eltern( void );
void crossover( void );
void mutation( void );


/* Funktionen */

void copy_pop( pop *newp, pop *oldp )
{
        int i;

        for (i=0; i<4; i++)
        {
                newp->zust[i] = oldp->zust[i];
                newp->fit = oldp->fit;
        }
```

```
}



void children_erzeugen()
/* erzeugt mit Zufallsgenerator eine Population der Groesse POPSIZE zu je 4 Maschinen */
{
        extern pop children[POPSIZE];
        int i,j;

        for (i=0; i<POPSIZE; i++)
        {
                for (j=0; j<4; j++)
                        children[i].zust[j] = rand()%4;
                children[i].fit = 0;
        }
}


void parents_reset()
{
        extern pop parents[POPSIZE];
        int i,j;

        for (i=0; i<POPSIZE; i++)
        {
                for (j=0; j<4; j++)
                        parents[i].zust[j] = 0;
                parents[i].fit = 0;
        }
}


void uebergeben()
/* schreibt Zustandssummenvektor der Kinder zur Uebergabe an GPSS/H in Datei bridge */
{
        extern pop children[POPSIZE];
        extern int stopflag;
        int i;
        FILE *fp;

        if ((fp = fopen( VERBINDUNGSDATEI, "w")) == NULL)
        {
                printf( "Kann nicht oeffnen!\n" );
                getchar();
                exit;
        }
```

```
        fprintf( fp, "%d\n", stopflag );

        for (i=0; i<POPSIZE; i++)
                fprintf( fp, "%d\n", (children[i].zust[0] + 1)*1000 +
                                                        (children[i].zust[1]    +
1)*100 +
                                                        (children[i].zust[2]    +
1)*10 +
                                                        children[i].zust[3] + 1 );

        fclose( fp );
}


void speichern()
{
        extern pop parents[POPSIZE];
        extern pop children[POPSIZE];
        extern int nr_of_gen;
        int i;
        FILE *fp;

        if ((fp = fopen( SPEICHERDATEI, "w" )) == NULL)
        {
                printf( "Kann nicht oeffnen!\n" );
                getchar();
                exit;
        }

        fprintf( fp, "%d\n", nr_of_gen );

        for (i=0; i<POPSIZE; i++)
        {
                fprintf( fp, "%d ", (parents[i].zust[0] + 1)*1000 +
                                                (parents[i].zust[1] + 1)*100 +
                                                (parents[i].zust[2] + 1)*10 +
                                                parents[i].zust[3] + 1 );

                fprintf( fp, "%d ", parents[i].fit );

                fprintf( fp, "%d ", (children[i].zust[0] + 1)*1000 +
                                                (children[i].zust[1] + 1)*100 +
                                                (children[i].zust[2] + 1)*10 +
                                                children[i].zust[3] + 1 );

                fprintf( fp, "\n");
        }
        fclose( fp );
```

```
}



void einlesen()
/* holt die Zustaende und die Fitnesswerte der Eltern aus der Datei saveval */
/* und die Fitnesswerte der Kinder aus der Datei bridge */
/* in die Populationsstruktur */
{
        extern pop parents[POPSIZE];
        extern pop children[POPSIZE];
        extern int stopflag;
        extern int nr_of_gen;
        int i, j;
        FILE *fp_speicher, *fp_verbindung;
        int zustvektor[POPSIZE];

        if ((fp_speicher = fopen( SPEICHERDATEI, "r")) == NULL)
        {
                printf( "Kann nicht oeffnen!\n" );
                getchar();
                exit;
        }

        fscanf( fp_speicher, "%d", &nr_of_gen );

        for (i=0; i<POPSIZE; i++)
        {
                fscanf( fp_speicher, "%d", &(zustvektor[i]) );
                parents[i].zust[0] = zustvektor[i]/1000 - 1;
                parents[i].zust[1] = (zustvektor[i]/100)%10 - 1;
                parents[i].zust[2] = (zustvektor[i]/10)%10 - 1;
                parents[i].zust[3] = (zustvektor[i])%10 - 1;

                fscanf( fp_speicher, "%d", &(parents[i].fit) );

                fscanf( fp_speicher, "%d", &(zustvektor[i]) );
                children[i].zust[0] = zustvektor[i]/1000 - 1;
                children[i].zust[1] = (zustvektor[i]/100)%10 - 1;
                children[i].zust[2] = (zustvektor[i]/10)%10 - 1;
                children[i].zust[3] = (zustvektor[i])%10 - 1;
        }

        fclose( fp_speicher );

        if ((fp_verbindung = fopen( VERBINDUNGSDATEI, "r")) == NULL)
        {
                printf( "Kann nicht oeffnen!\n" );
```

```
                getchar();
                exit;
        }

        fscanf( fp_verbindung, "%d", &stopflag );

        for (i=0; i<POPSIZE; i++)
                fscanf( fp_verbindung, "%d", &(children[i].fit) );

        fclose( fp_verbindung );
}



void besten_speichern()
{
        extern pop parents[POPSIZE];
        extern pop children[POPSIZE];
        FILE *fp;
        int i;
        pop the_best;
        pop temp_elem;

        copy_pop( &the_best, &parents[1] );
        for (i=2; i<POPSIZE; i++)
        {
                if (parents[i].fit > the_best.fit)
                {
                        copy_pop( &temp_elem, &the_best );
                        copy_pop( &the_best, &parents[i] );
                        copy_pop( &parents[i], &temp_elem );
                }
        }

        for (i=1; i<POPSIZE; i++)
        {
                if (children[i].fit > the_best.fit)
                {
                        copy_pop( &temp_elem, &the_best );
                        copy_pop( &the_best, &children[i] );
                        copy_pop( &children[i], &temp_elem );
                }
        }

        if ((fp = fopen( BESTOFDATEI, "a" )) == NULL)
        {
                printf( "Kann nicht oeffnen!\n" );
                getchar();
                exit;
```

```
        }

        fprintf( fp, "%d ", nr_of_gen );

        fprintf( fp, "%d ", (the_best.zust[0] + 1)*1000 +
                                                (the_best.zust[1] + 1)*100 +
                                                (the_best.zust[2] + 1)*10 +
                                                the_best.zust[3] + 1 );

        fprintf( fp, "%d\n", the_best.fit );

        fclose( fp );
}



void select_neue_eltern()       /* roulette wheel-Selektion */
{
        extern pop parents[POPSIZE];
        extern pop children[POPSIZE];
        pop temp_pop[POPSIZE];
        int fitnessumme = 0;
        int teilsumme = 0;
        float zufallswert;
        int i, index;

        for (i=0; i<POPSIZE; i++)
        {
                fitnessumme += parents[i].fit + children[i].fit;
        }

        for (i=0; i<POPSIZE; i++)
        {
                teilsumme = 0;
                index = 0;
                zufallswert = (float)(rand())/RAND_MAX * fitnessumme;

                if (zufallswert == 0.0)
                {
                        copy_pop( &temp_pop[i], &parents[0]);
                }
                else
                {
                        while (((float)teilsumme < zufallswert) && (index !=
POPSIZE))
                        {
                                teilsumme += parents[index].fit;
                                if ((float)teilsumme >= zufallswert)
                                        copy_pop( &temp_pop[i], &parents[index] );
```

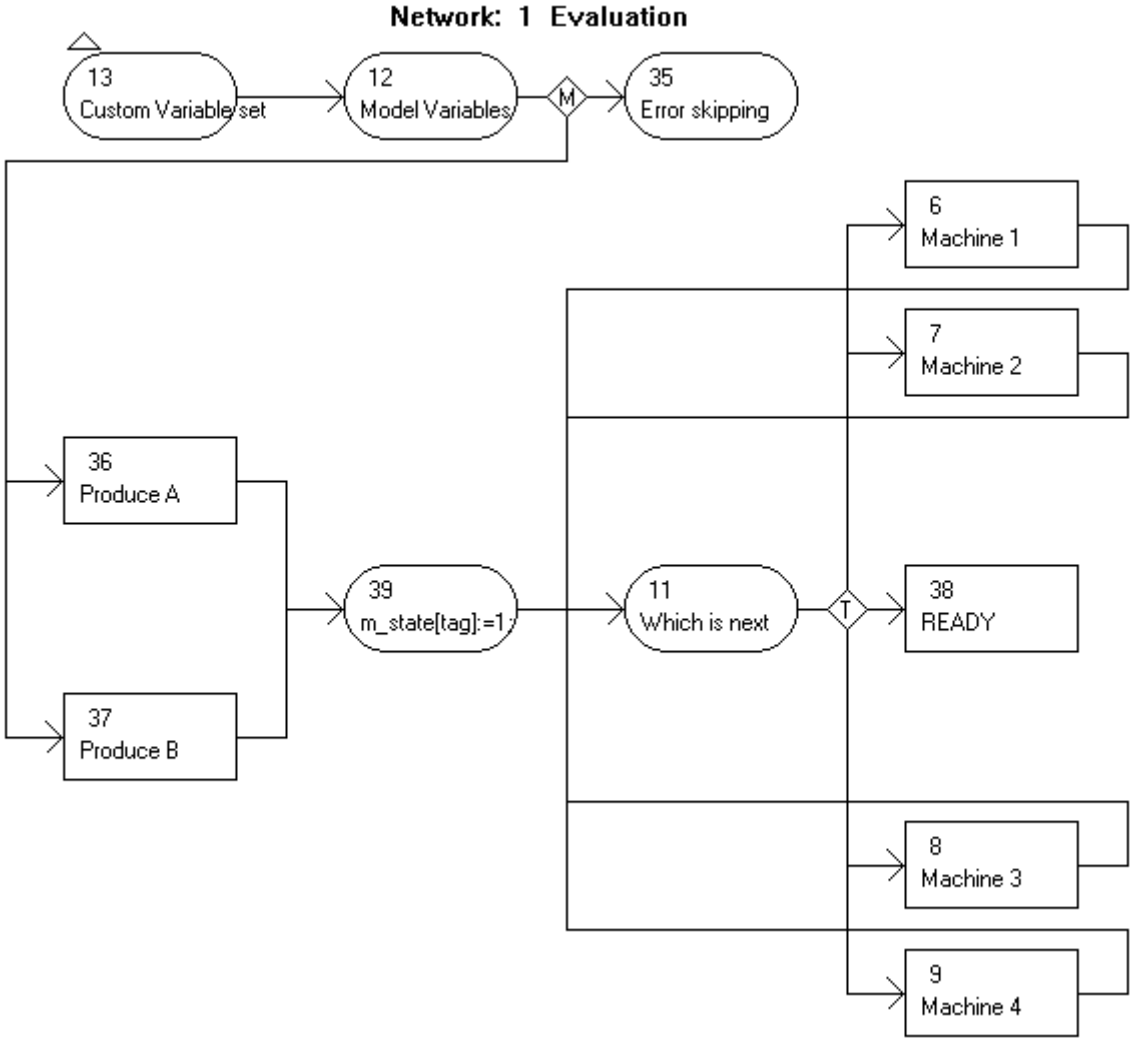
```
                                else
                                {
                                        teilsumme += children[index].fit;
                                        if ((float)teilsumme >= zufallswert)
                                                copy_pop(               &temp_pop[i],
&children[index] );
                                }
                                index++;
                        }
                }
        }

        for (i=0; i<POPSIZE; i++)
                copy_pop( &parents[i], &temp_pop[i] );
}



void crossover()
{
        extern pop parents[POPSIZE];
        extern pop children[POPSIZE];
        int i, j, fath, moth, trenn;

        for (i=0; i<POPSIZE; i+=2)
        {
                /* 2 von den Eltern und Trennstelle zufaellig bestimmen */

                fath = rand() % POPSIZE;
                do
                {
                        moth = rand() % POPSIZE;
                } while (moth == fath);
                trenn = rand() % 8;             /* Trennstelle zwischen 0 und 7 */

                if (trenn % 2 == 1)             /* Trennung zwischen 2 Maschinen */
                {
                        for (j=0; j<=trenn/2; j++)
          {
                                children[i].zust[j] = parents[fath].zust[j];
                                children[i+1].zust[j] = parents[moth].zust[j];
                        }
                        for (j=trenn/2 + 1; j<4; j++)
          {
                                children[i].zust[j] = parents[moth].zust[j];
                                children[i+1].zust[j] = parents[fath].zust[j];
                        }

                        children[i].fit = 0;
```

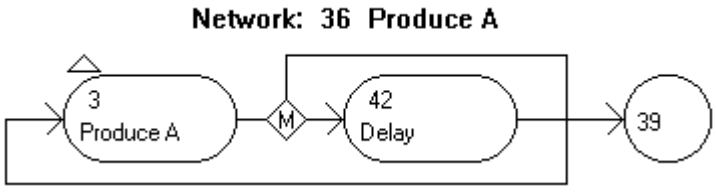
```
                        children[i+1].fit = 0;
                }

                else                            /* Trennung innerhalb einer Maschine */
                {
                        for (j=0; j<=trenn/2 - 1; j++)
          {
                                children[i].zust[j] = parents[fath].zust[j];
                                children[i+1].zust[j] = parents[moth].zust[j];
                        }

                        children[i].zust[trenn/2] = (parents[fath].zust[trenn/2] & (~01)) |

        (parents[moth].zust[trenn/2] & 01);
                        children[i+1].zust[trenn/2]   =   (parents[moth].zust[trenn/2]   &
(~01)) |

        (parents[fath].zust[trenn/2] & 01);

                        for (j=trenn/2 + 1; j<4; j++)
          {
                                children[i].zust[j] = parents[moth].zust[j];
                                children[i+1].zust[j] = parents[fath].zust[j];
                        }

                        children[i].fit = 0;
                        children[i+1].fit = 0;

                } /*else*/
        } /*for i */
}


void mutation()
{
        extern pop parents[POPSIZE];
        int mut_opfer;
        int mut_stelle;

        mut_opfer = rand() % POPSIZE;
        mut_stelle = rand() % 8;

        if (mut_stelle % 2 == 1)
                parents[mut_opfer].zust[mut_stelle/2] ^= 2;
        else
                parents[mut_opfer].zust[mut_stelle/2] ^= 1;
}
```

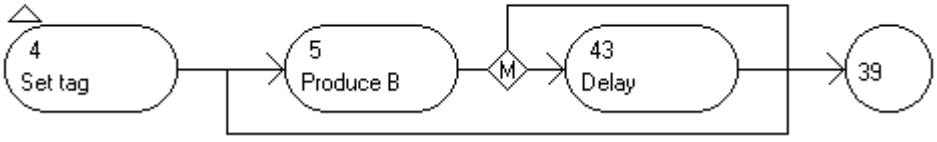
```
/* Hauptprogramm */

main()
{
	extern pop parents[POPSIZE];
	extern pop children[POPSIZE];
	extern int stopflag;
	extern int nr_of_gen;
	FILE *fp;
	int i=0;
	int zuf=0;

	if ((fp = fopen( VERBINDUNGSDATEI, "r" )) == NULL)
	{
		printf( "Kann nicht oeffnen!\n" );
		getchar();
		exit;
	}
	fscanf( fp, "%d", &stopflag );
	fclose( fp );

	if (stopflag == -1)
	{
		fp = fopen( BESTOFDATEI, "w" );
		fclose( fp );
		children_erzeugen();
		parents_reset();
		nr_of_gen = 1;
		stopflag = 0;
		uebergeben(); /* stopflag, children */
		speichern(); /* nr_of_gen, parents, children */
	}
	else
	{
		einlesen();  /* nr_of_gen, stopflag, parents, children */
		besten_speichern();

		if (nr_of_gen == 1)
		{	/* Eltern erzeugen durch Kinder verdoppeln, damit fuer Selektion
genug da sind */
		  for (i=0; i<POPSIZE; i++)
		  {
			copy_pop( &parents[i], &children[i] );
			if ((zuf = rand()) < RAND_MAX/100)		/*
Ereignis mit Wahrsch. < 0.01 */
			{
				printf( "\nMutation\n");
				mutation(); /* parents */
			}
```

```
                }
        }
        select_neue_eltern(); /* parents, children */
        if ((zuf = rand()) < RAND_MAX/100)          /* Ereignis mit
Wahrsch. < 0.01 */
        {
                mutation(); /* parents */
        }
        crossover(); /* parents, children */
        nr_of_gen++;
        if (nr_of_gen == MAXGENERATIONNUMBER+1)     /*
Abbruchbedingung */
                stopflag = 1;
        uebergeben(); /* stopflag, children */
        speichern(); /* nr_of_gen, parents, children */
    }
}
```

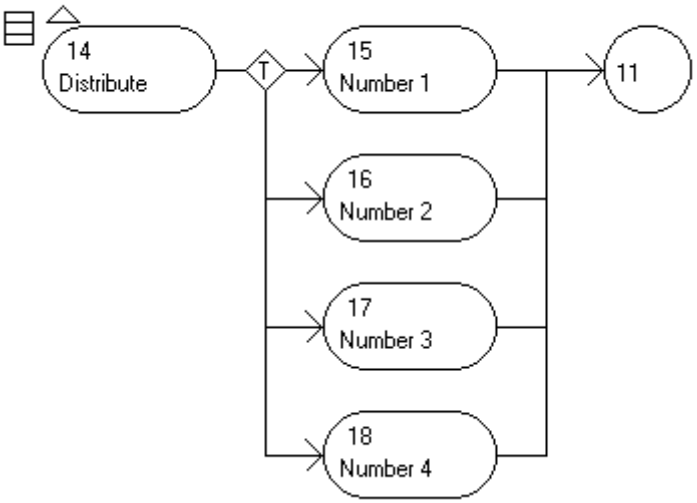
Anhang C - Networks in MicroSaint™



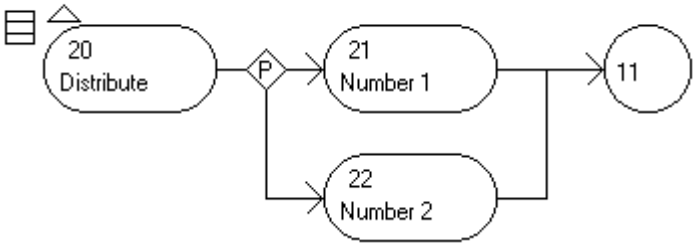




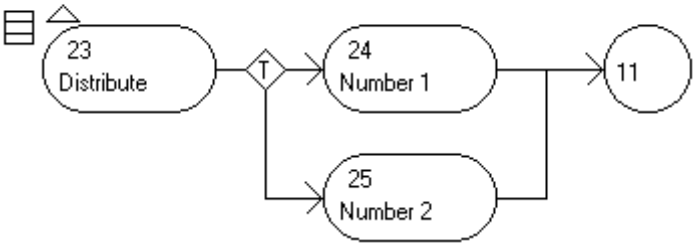
Network: 6 Machine 1



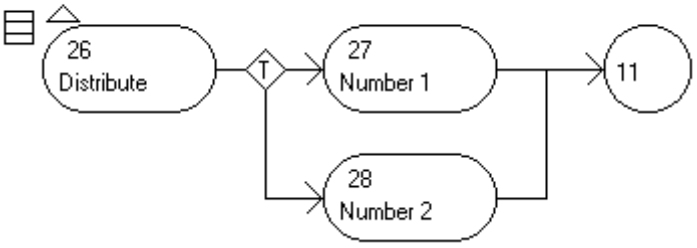
Network: 7 Machine 2

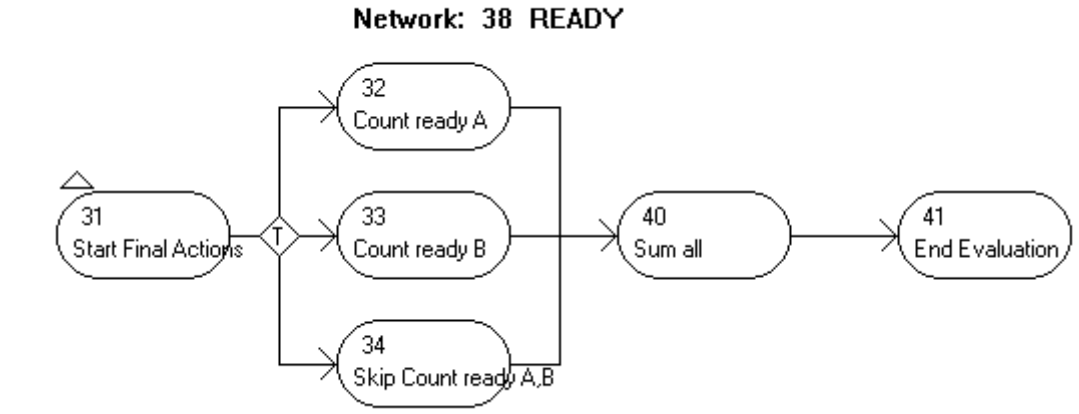


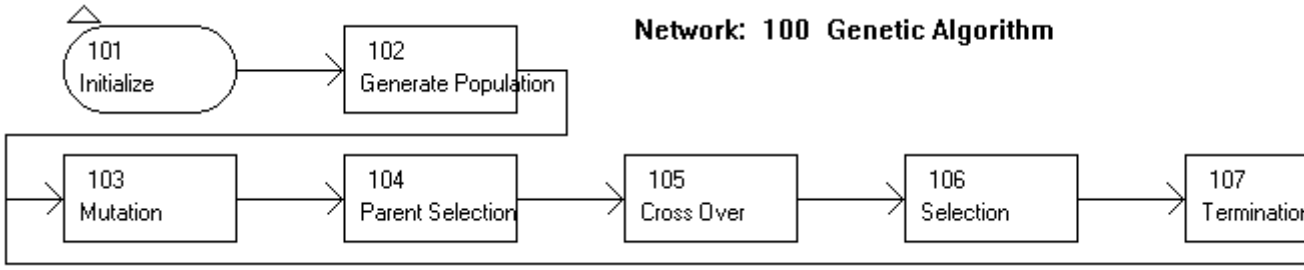
Network: 8 Machine 3

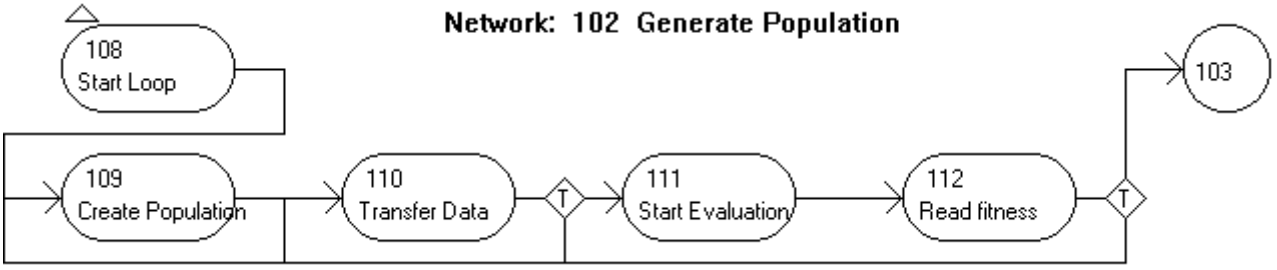


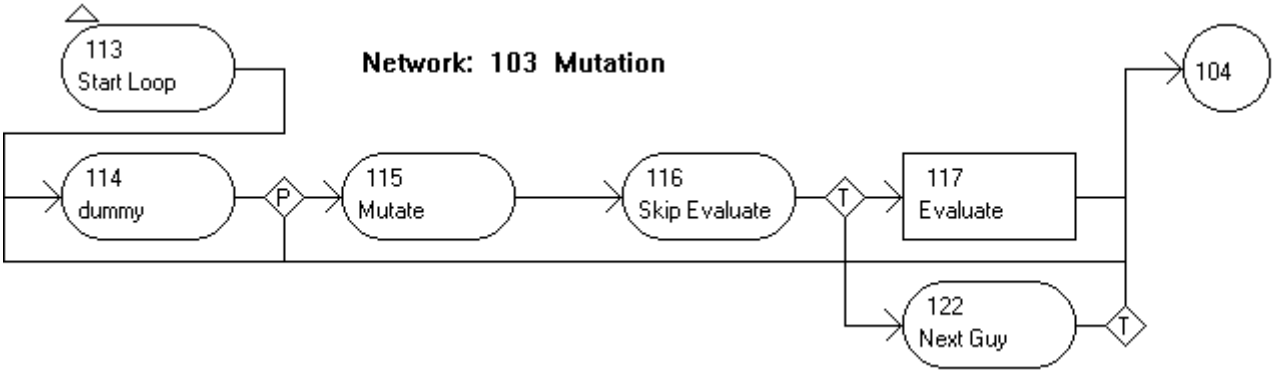
Network: 9 Machine 4

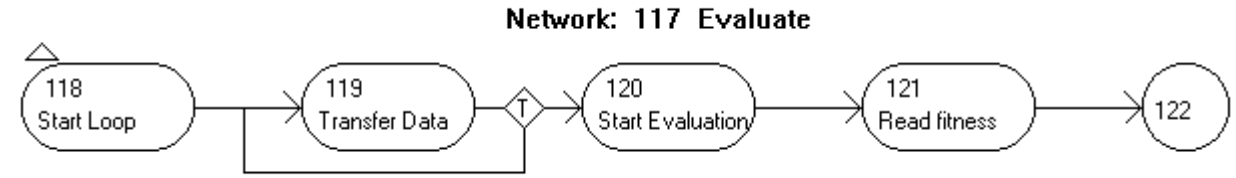


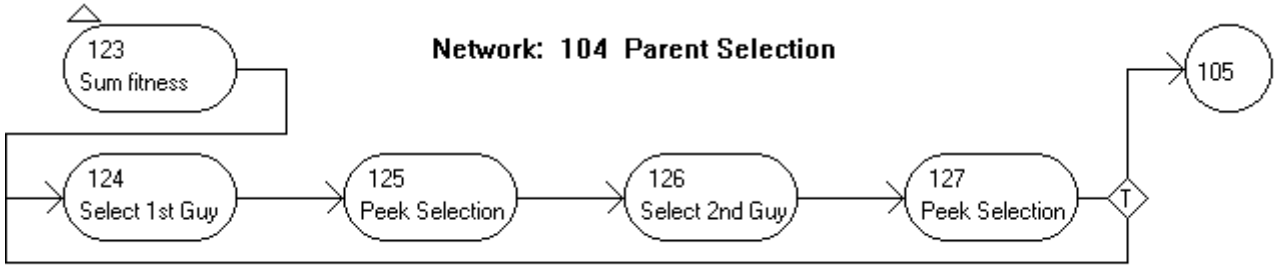

Network: 38 READY
32
Count ready A
31
Start Final Actions
T
33
Count ready B
40
Sum all
41
End Evaluation
34
Skip Count ready A,B

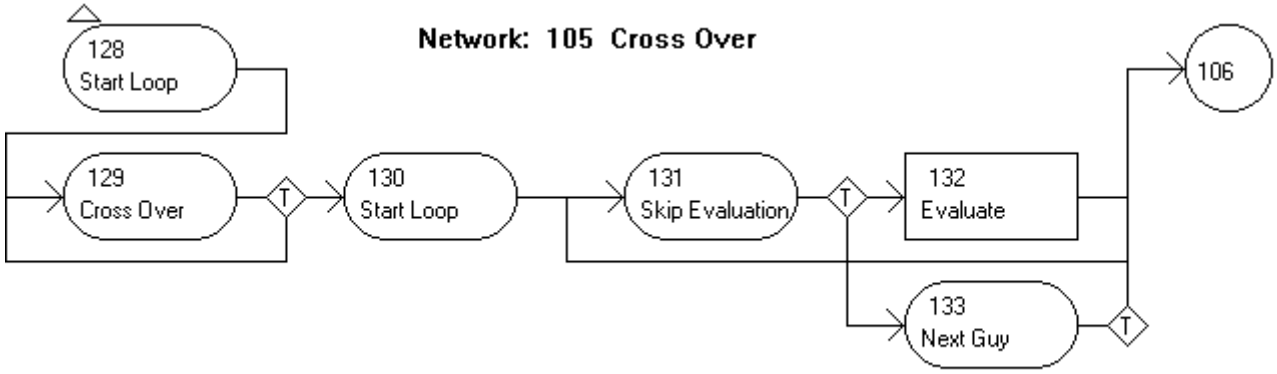

Network: 100 Genetic Algorithm
101
Initialize
102
Generate Population
103
Mutation
104
Parent Selection
105
Cross Over
106
Selection
107
Termination

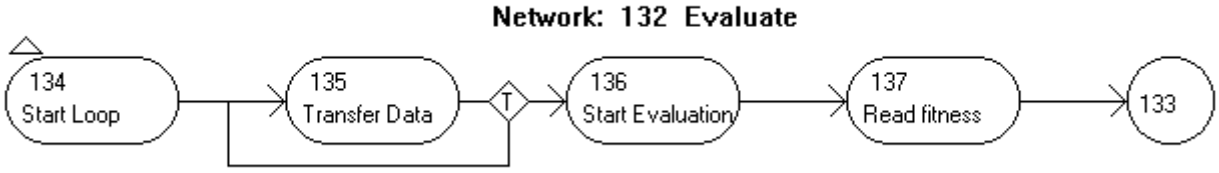

Network: 102 Generate Population
108
Start Loop
103
109
Create Population
110
Transfer Data
T
111
Start Evaluation
112
Read fitness
T

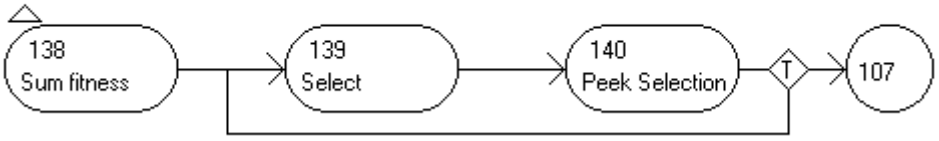

Network: 103 Mutation
113
Start Loop
104
114
dummy
P
115
Mutate
116
Skip Evaluate
T
117
Evaluate
122
Next Guy
T

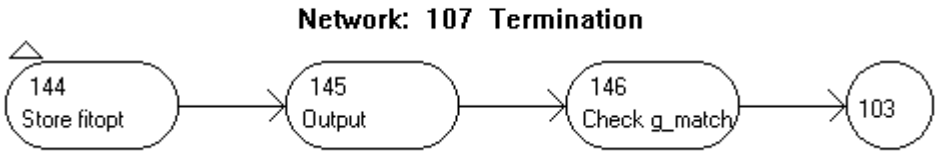

Network: 117 Evaluate
118
Start Loop
119
Transfer Data
T
120
Start Evaluation
121
Read fitness
122

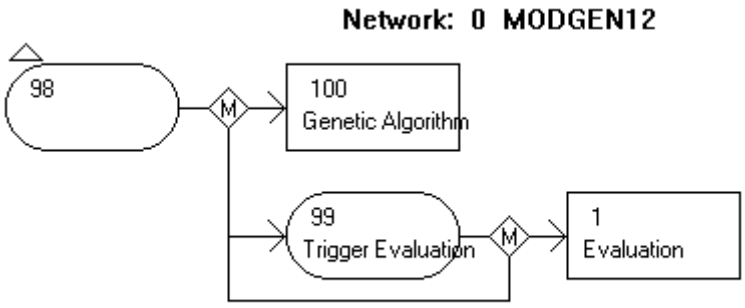

Network: 104 Parent Selection
123
Sum fitness
105
124
Select 1st Guy
125
Peek Selection
126
Select 2nd Guy
127
Peek Selection
T


Network: 105 Cross Over
128
Start Loop
106
129
Cross Over
T
130
Start Loop
131
Skip Evaluation
T
132
Evaluate
133
Next Guy
T


Network: 132 Evaluate
134
Start Loop
135
Transfer Data
T
136
Start Evaluation
137
Read fitness
133


Network: 106 Selection
138
Sum fitness
139
Select
140
Peek Selection
T
107


Network: 107 Termination
144
Store fitopt
145
Output
146
Check g_match
103


Network: 0 MODGEN12
98
M
100
Genetic Algorithm
99
Trigger Evaluation
M
1
Evaluation

Literaturliste

/AC93/ Arunkumar, S.; Chockalingam,T.: Genetic search algorithms and their randomized operators. Computers & Mathematics with Applications, S. 91-100, März 1993

/ACH90/ Ansari, N.; Chen, M.H.; Hou, E.S.H.: Point pattern matching by a genetic algorithm.in IECON 90, volume 2, S. 1233-1238, Pacific Grove, CA, November 1990

/AK92/ Allred, L.G.; Kelly, G.E.: A modified genetic algorithm for extracting thermal profiles from infrared image data.in Proceedings of the SPIE, S. 77-81, San Diego, CA, 1992

/ART93/ Artificial neural nets and Genetic Algorithms, Proceedings, Innsbruck, 1993

/ASIM87/ ASIM-Arbeitskreis: Simulation in der Fertigungstechnik. Mitteilungen aus den Arbeitskreisen, 1987

/ASIM89/ ASIM-Arbeitskreis: Simulation in der Fertigungstechnik - Anwendungsbeispiele. Mitteilungen aus den Arbeitskreisen, 1989

/BÄC91/ Bäck, T.: Self-adaptation in genetic algorithms. in Proceedings of the First European Conference on Artificial Life, S.263-271, MIT Press, Cambridge, MA, 1991

/BB91/ Belew, R.K.; Booker, L.B.: Proceedings of the Fourth International Conference on Genetic Algorithms, San Mateo, CA, Morgan Kaufmann, 1991

/BBM93a/ Beasley, D., Bull, D. R., Martin, R. R.: An Overview of Genetic Algorithms - Part 1, Fundamentals. University Computing Cardiff, 1993

/BBM93b/ Beasley, D., Bull, D. R., Martin, R. R.: An Overview of Genetic Algorithms - Part 2, Research Topics. University Computing Cardiff, 1993

/BCN89/ Banks,J.;Carson,J.;Ngo Sy,J: Getting Started with GPSS/H. Wolverine Software Corporation, Annandale, 1989

/BD90/ Biegel, J.E.; Davern, J.J.: Genetic algorithms and job shop scheduling. Computers & Industrial Engineering, 1990

/BEY93/ Beyer, O: Die Kopplung Simulation und Optimierung - System- und Nutzeraspekte. in /SYD93/

/BGH89/ Booker, L.B.; Goldberg, D.E.; Holland, J.H.: Classifier systems and genetic algorithms. Artificial Intelligence, S. 235-282, 1989

/BINI94/ Biethahn, J., Nissen, V.: Combinations of Simulation and Evolutionary Algorithms in Management Science and Economics. Universität Göttingen, 1994

/BM93/ Bock, S.; Meyer, R.: Akzeptanz der Simulationstechnik - Ergebnis einer Umfrage. in /SYD93/

/BOO87/ Booker, L.: Improving search in genetic algorithms. in /DAV87/

/BP92/ Buckles, B.P.; Petry, F.E.: Genetic Algorithms, IEEE Computer Society, 1992

/BRA91a/ Bramlette, M.F.: Initialisation, mutation and selection methods in genetic algorithms for function optimization. in /BB91/

/BRA91b/ Braun, H.: On solving travelling salesman problems by genetic algorithms. in /SM91/ S. 129-133, Dortmund, Springer-Verlag, Berlin, 1991

/BRE91/ Breitenecker, F. : Diskrete Simulationssysteme. Skriptum, TU Wien, 1991

/BRBG90/ Breitenecker, F.; Bausch-Gall, I. : Kontinuierliche Simulationssprachen. Skriptum, Wien, 1990

/BRU93/ Bruns, R.: Direct Chromosome Representation and Advanced Genetic Operators for Production Scheduling. In /FOR93/

/BS88/ Bronstein, I.N.; Semendjajew, K.A.: Taschenbuch der Mathematik. Verlag Harry Deutsch, Thun - Frankfurt, 1988

/BUC92/ Buckles, P.B.: Genetic Algorithms. IEEE Computer Society Press, Los Alamitos, CA, 1992

/BWJ91/ Beaty, S.; Whitley, D.; Johnson, G.: Motivation and framework for using genetic algorithms for microcode compaction. SIGMICRO Newsletter, S. 20-27, 1991

/CAR85/ Carnegie Mellon University. Proceedings of an International Conference on Genetic Algorithms and Their Applications, 1985

/CDM90/ Colorni, A.; Dorigo, M.; Maniezzo, V.:On the use of genetic algorithms to solve the time-table problem. Technical Report 90-060, Politecnico die Milano, Italy, 1990

/CHMR91/ Cohoon, J.P.; Hegde, S.U.; Martin, W.N.; Richards, D.S.: Distributed genetic algorithms for the floorplan design problem. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, S. 483-492, 1991

/CLA93/ Claus, Th.: Genetische Simulation. Universität Osnabrück, 1993

/COMP_1/ Genetic Algorithm Digest. email: ga-list-request@aic.nrl.navy.mil

/COMP_2/ Artificial Life Digest. email: alife-request@cognet.ucla.edu

/COMP_3/ Evolutionary Programming Digest. email: ep-list-request@magenta.me.fau.edu

/COMP_4/ Genetic Programming mailing List. email: genetic-programming-request@cs.stanford.edu

/COMP_5/ Tierra mailing list. email: tierra-request@life.slhs.udel.edu

/COMP_6/ GA-Molecule mailing list. email: ga-molecule-request@tammy.harvard.edu

/COMP_7/ UK’s Evolutionary-Computation mailing list. email: evolutionary-computing-request@mailbase.ac.uk

/COMP_8/ Genetic Algorithm Research Student mailing list. email: gaphd-list-request@dcs.warwick.ac.uk

/COMP_9/ Genetic Algorithms and Neural Networks -mailing list. email: majordomo@cs.iastate.edu

/COMP_10/ Newsgroup: comp.ai.genetic

/COMP_11/ Newsgroup: comp.ai

/COMP_12/ Newsgroup: comp.ai.fuzzy

/COMP_13/ Newsgroup: comp.ai.jair.announce

/COMP_14/ Newsgroup: comp.ai.jair.papers

/COMP_15/ Newsgroup: comp.ai.neural-nets

/COMP_16/ Newsgroup: comp.theory.self-org-sys

/COMP_17/ Newsgroup: sci.bio.evolution

/COMP_18/ Newsgroup: sci.math.num-analysis

/COMP_19/ Newsgroup: sci.op-research

/COMP_20/ Newsgroup: comp.simulation

/COMP_21/ ENCORE (EvolutioNary COmputation Repository network): ftp://ftp.Germany.EU.net/pub/research/softcomp/EC/Welcome.html

/COMP_22/ ENCORE: ftp://ftp.cs.wayne.edu/pub/EC/Welcome.html

/COMP_23/ ENCORE: ftp://ftp.krl.caltech.edu/pub/EC/Welcome.html

/COMP_24/ anonymous ftp-server der ARGE Simulation News: ftp://simserv.tuwien.ac.at

/COMP_25/ Homepage von EUROSIM: http://eurosim.tuwien.ac.at

/COMP_26/ Homepage für GA: http://www.aic.nrl.navy.mil:80/galist

/COMP_27/ Homepage für Computersimulation: http://piranha.eng.buffalo.edu/simulation

/COMP_28/ scheduling mailing list: email: gasched-request@acse.shef.ac.uk

/DAV85a/ Davis, L.: Applying adaptive algorithms to epistatic domains. In 9th Int. Joint Conf. on AI, S. 162-164, 1985

/DAV85b/ Davis, L.: Job Shop Scheduling with Genetic Algorithms. In /GRE85/

/DAV87/ Davis, L.: Genetic Algorithms and Simulated Annealing. Pitman, 1987

/DAV90/ Davidor, Y.:Epistasis variance: Suitability of a representation to genetic algorithms. Complex Systems, S. 369-383, 1990

/DAV91a/ Davis, L.: Handbook of Genetic Algorithms. 1991

/DAV91b/ Davidor,Y.: A genetic algorithm applied to robot trajectory generation. in /DAV91a/

/DEJ75/ DeJong, K.A.: Analysis of behavior of a class of genetic adaptive systems, University of Michigan, Dept. Computer and Communication Sciences, Diss. Abstr. Int 36(10), 1975

/DEJ85/ DeJong, K.: Genetic Algorithms: A 10 year perspective. in /GRE85/

/DS93/ Dorigo, M.; Schneph, U.: Genetics-based machine learning and behaviour based robotics. IEEE Transactions on Systems, Man and Cybernetics, 1993

/ES93/ Eshelman, L.J.; Schaffer, J.D.: Real-coded genetic algorithms and interval schemata. In L.Darell Whitley, editor, Foundations of Genetic Algorithms, 2, S. 187-202. Morgan Kaufmann, 1993

/FA92/ Fogel, D.B.; Atmar, W.: First Annual Conference on Evolutionary Programming, La Jolla, California, 1992

/FA93/ Fogel, D.B.; Atmar, W.: Second Annual Conference on Evolutionary Programming, La Jolla, California, 1993

/FB91/ Falkenauer, E.; Bouffouix, S.: A genetic algorithm for job shop. In Proceedings 1991 IEEE International Conference on Robotics and Automation, volume 1, S. 824-829, Sacramento, CA, 1991, IEEE Computer Society Press, Los Alamitos, CA

/FH91/ Fogarty, T.C.; Huang, R.: Implementing the genetic algorithm on transputer based parallel processing systems. In /SM91/, S. 145-149

/FOG88/ Fogel, D.B.: An evolutionary approach to travelling salesman problem. Biological Cybernetics, S. 139-144, 1988

/FOG89/ Fogel, D.B.: Evolutionary programming for voice feature analysis. In Proceedings of 23rd Asilomar Conference on Signals, Systems and Computers, S. 381-383, Pacific Grove, California, 1989

/FRC93/ Fang, H.-L.; Ross, P.; Corne, D.: A Prommising Genetic Algorithm Approach to Job-Shop Scheduling, Rescheduuling, and Open-Shop Scheduling Problems. In /FOR93/

/FOR93/ Forrest, S.: Proceedings of the Fifth International Conferrence on Genetic Algorithms, Urbana-Champaign, 1993

/FOW66/ Fogel, L.J.; Owens, A.J.; Walsh, M.J.: Artificial Intelligence trough simulated Evolution, John Wiley & Sons, 1966

/GAN93/ Gangl, P.: Simulation - eine Schlüsseltechnologie der 90er Jahre: Hoher Nutzen aber geringer Kenntnisstand in der Industrie. in /SYD93/

/GDC92/ Goldberg, D.E.; Deb, K.; Clark, J.H.: Genetic algorithms, noise and the sizing of populations. Complex Systems, S. 415-444, 1990

/GMD94/ GMD-Spiegel 2’94. ISSN: 0724-4339

/GOL87/ Goldberg, D. E.: Simple Genetic Algorithms and the Minimal, Deceptive Problem. in /DAV87/

/GOL89/ Goldberg, D.E.: Genetic Algorithms in search, optimization and machine learning. Addison-Wesley, 1989

/GOL90/ Goldberg, D.E.: A note on boltzmann tournament selection for genetic algorithms and populationoriented simulated annealing. Complex Systemx, S. 445-460, 1990

/GOL91/ Goldberg, D.E.: Real-coded genetic algorithms, virtual alphabets, and blocking. Complex Systems, S. 139-167, 1991

/GR91/ Goldberg, D.E.; Rudnick, M.: Genetic algorithms and the variance of fitness. Complex Systems, S. 265-278, 1991

/GRE84/ Greffenstette, J. J.: GENESIS - A system for using genetic search procedures. in Proceedings of the 1984 Conference on Intelligent Systems and Machines, S. 161-165, 1984

/GRE85/ Grefenstette, J.J.: Proceedings of the First International Conference on Genetic Algorithms and their Applications. Pittsburgh, 1985

/GRE87a/ Grefenstette, J.J.: Proceedings of the Second International Conference on Genetic Algorithms and their Applications. Cambridge, 1987

/GRE87b/ Grefenstette, J.J.: Incorporating problem specific knowledge into genetic algorithms. S. 42-60 in /DAV87/

/GS94/ Gabler, Barbara; Strauss, Hannes: Erweiterte Features von D_SIM. Projektpraktikum, TU Wien, 1994

/HAN92/ Hancock, P.J.B.: Coding strategies for genetic algorithms and neural nets. PhD thesis, Department of Computing Science and Mathematics, University of Stirling, 1992

/HAN94/ Hancock, P.J.B.: An empirical comparison of selection methods in evolutionary algorithms. University of Stirling, 1994

/HEI94/ Heistermann, J.: Genetische Algorithmen. Teubner-Texte zur Informatik, Stuttgart-Leipzig, 1994

/HB91/ Hoffmeister, F.; Bäck, T.: Genetic algorithms and evolution stragtegies: similarities and differences. in /SM91/

/HEBE94/ Heitkötter, J.; Beasley, D.: The Hitchhiker’s Guide to Evolutionary Computation. in /COMP_10/, 1994

/HL91/ Hou, E.S.H.; Li, H.Y.: Task scheduling for flexible manufacturing systems based on genetic algorithms. in Proceedings of the 1991 IEEE International Conference on Systems, Man and Cybernetics, Decision Aiding for complex systems, S. 397-402, Charlottesville, VA, 1991, IEEE, New York, 1991

/HLA93/ Hlavacs, Helmut: Entwurf und Implementierung eines Petrinetzsimulators für ein diskretes Simulationssystem. Diplomarbeit, TU Wien, 1993

/HM91a/ Hesser, J.; Männer, R.:An alternative genetic algorithm. S. 33-37 in /SM91/

/HM91b/ Hesser, J.; Männer, R.: Towards an optimal mutation probability for genetic algorithms. in /SM91/, 1991

/HOL75/ Holland, J.H.: Adaption in Natural and Artificial Systems, University of Michigan Press, 1975

/HOL91/ Holland, J.H.: Genetic algorithms, Scientific American, S. 66-72, 1992

/IR92/ Ingber, L.; Rosen, B.: Genetic algorithms and very fast simulated annealing - a comparison. Mathematical and Comuter Modeling, S. 87-100, 1992

/JM91/ Janikow, C.Z.; Michalewicz,Z.: An experimental comparison of binary and floating point representations in genetic algorithms. S. 31-36 in /BB91/

/KAR91/ Karr, C.: Genetic algorithms for fuzzy controllers. AI Expert, S. 26-33, 1991

/KAVO67/ Kadlec, V.; Vodacek, L.: Lineare Optimierung im Transportwesen. VEB Verlag Berlin, 1967

/KAZE94/ Kampe, G.; Zeitz, M.: Fortschritte in der Simulationstechnik Band 9; 9. Symposium Stuttgart, Vieweg Verlag, Braunschweig/Wiesbaden, 1994

/KEL85/ Kelton, D.: Analysis of Simulation Output Data. Lehrtransparente, University of Minnesota, 1985

/KEL84/ Kelton, D.: Input Data Collection. Lehrtransparente, University of Minnesota,1984

/KNI74/ Knippers, R.: Molekulare Genetik - Einführungen zur Molekularbiologie, Band 2. Georg Thieme Verlag Stuttgart, 1974

/KNO93/ Knorr, B: Optimierungsmöglichkeiten mit dem Simulationssystem IDAS. in /SYD93/

/KOP92a/ Kopfer, H.:Progenitor - a genetic algorithm for production scheduling. Wirschaftsinformatik, S. 255-256, 1992

/KOP92b/ Kopher, H.: Genetic algorithms concepts and their applications to freight minimisation in commercial long distance freight transportation. OR Spektrum, S. 137-147, 1992

/KOZ92a/ Koza, J. R.: Evolution of subsumtion using genetic programming. in Proceedings of the First European Conference on Artifical Life, S 110-119, MIT Press, Cambridge MA, 1992

/KOZ92b/ Koza, J.R.: Genetic Programming: On The Programming Of Computers By Means Of Natural Selection. MIT Press,1992

/KQF93/ Kreinovich, V.; Quintana, C.; Fuentes, O.: Genetic algorithms - what fitness scaling is optimal. Cybernetics and Systems, 1993

/KS91/ Kanet, J.J.; Sridharan, V.: Progenitor - a genetic algorithm for production scheduling. Wirtschaftsinformatik, 1991

/KV92/ Kroger, B.; Vornberger, O.: Enumerative vs. genetic oprimization: two parallel algorithms for the bin packing problem. Final report on the DFG Special Initiative, S. 330-362. Springer-Verlag, Berlin, 1992

/LEN93/ Lenneis, G.: Adaptive Algorithmen in der Maschinenbelegungsplanung, Dissertation WU Wien, 1993

/LEV94/ Levine, D.: A Parallel Algorithm for the Set Partitioning Problem. Dissertation, Argonne, 1994

/LEWE92/ Lehn, J.; Wegmann, H.: Einführung in die Statistik. Teubner Studienbücher, Stuttgart, 1992

/MAS92/ Mason, A.J.: Genetic Algorithms and Job Scheduling. PhD thesis, University of Cambridge, Management Studies Group, Department of Engineering, UK, 1992

/MICH92/ Michalewicz, Z.: Genetic Algorithms + Data Structures = Evolution Programs, Berlin, 1992

/MUH91b/ Mühlenbein, H.: Parallel genetic algorithms, population genetics and combinatorial optimization. In B.D. Becker, I. Eisele, and F.W. Mundemann (editors): Parallelism, Learning, Evolution. Workshop on Evolutionary Models and Strategies and Workshop on Parallel Processing: Logic Organization and Technology - WOPPLOT 89, S. 398-406. Springer-Verlag, Berlin, 1991

/MVH91/ Michalewicz, Z.; Vignaux, G.A.; Hobbs, M.: A nonstandard genetic algorithm for the nonlinear transportation problem. ORSA Journal on Computing, S. 307-316. 1991.

/NES93/ Nestraschil, A.: Genetische Algorithmen zur Funktionsoptimierung im Vergleich. Diplomarbeit, WU Wien, 1993

/NIS93/ Nissen, V.: Solving the Quadratic Assignment Problem with Clues fromnature, TNN, Nr. 1, S. 66-72, 1993

/NIS94/ Nissen, V.: Evolutionäre Algorithmen - Darstellung, Beispiele, betriebswirtschaftliche Anwendungsmöglichkeiten. Deutscher UniversitätsVerlag, Wiesbaden, 1994

/NUS89/ Nußbaumer, A.: Vergleich diskreter Simulationssprachen. Diplomarbeit, TU Wien, 1989

/OBER90/ Oberpfalzer, F.-S.: Zufallsgesteuerte Suchstrategien für das TSP. Diplomarbeit, TU Wien 1990

/RAW91/ Rawlins, G.J.E.: Foundations of Genetic Algorithms, 1991

/REC73/ Rechenberg, I.: Bionik, Evolution und Optimierung. In „Naturwissenschaftliche Rundschau“ Vol. 26, S. 465-472, 1973

/REC89a/ Rechenberg, I.: Evolutionsstrategie - Optimierung nach Prinzipien der biologischen Evolution. in „Evolution und Evolutionsstrategien in Biologie, Technik und Gesellschaft“ (Hrsg: Freie Akademie), S. 25-72, 1989

/REC89b/ Rechenberg, I.: Evolution Strategy: Nature's Way of Optimization. Springer-Verlag „Lecture Note in Engineering" Vol. 47, 1989

/REC93/ Rechenberg, I.: Genetische Algorithmen versus Evolutionsstrategien. in „Evolutionsstrategie '93" - Hrsg.: Frommann-Holzbog, 1993

/REG94/ Reger, K.: Simulation in Passau, Heft 1 1994. ISSN-Nr. 0945-3830, Universität Passau, 1994

/ROT88/ Rothe, G.: Two Solvable Cases of the TSP. Dissertation, TU Graz, 1988

/ROT92/ Rothe, G.: Habilitationsschrift. TU Graz, 1992

/RUD95/ Rudolph, Martina: GPSS/H und Genetische Algorithmen - Zwei Welten prallen aufeinander. Projektpraktikum, TU Wien, 1995

/RUH94/ Ruhland, J: Bibliography. Universität Kassel, 1994

/SAFO93/ Saravanan, N.; Fogel, D. B.: A Bibliography of Evolutionary Computation & Applikations. Technical Report No. FAU-ME-93-100, Florida Atlantic University, 1993

/SAL92/ Salzmann, M.: Modellbildung, Simulation, Analyse und Animation einer Fertigung. Diplomarbeit, TU Wien, 1992

/SB93a/ Salzmann, M., Breitenecker, F.: Modellbildung, Simulation, Analyse und Animation einer Fertigung. in /SYD93/

/SB93b/ Schulte, J.; Becker, B.-D.: Optimierung in der Werkstattsteuerung: Simulation und genetische Algorithmen. in /SYD93/

/SB94a/ Salzmann, M., Breitenecker, F.: Optimization in Discrete Event Simulation - An Approach with Genetic Algorithms. Proceedings der 1. MATHMOD Vienna, Wien, 1994

/SB94b/ Salzmann, M., Breitenecker, F.: Optimierung in diskreten Simulationssystemen mit Genetischen Algorithmen. in /KAZE94/

/SB95/ Salzmann, M. Breitenecker, F.: Optimisation in Discrete Event Simulation using Genetic Algorithms. Proceedings of UKSS'95, North Berwick, 1995

/SBH93/ Schwefel, H-P.; Bäck, Th.; Hammel, U.: Modelloptimierung mit evolutionären Algorithmen. in /SYD93/

/SCH89a/ Schaffer, J.D.: editor, Proceedings of the Third International Conference on Genetic Algorithms, S. 2-9, Morgan Kaufmann,1989

/SCH89b/ Schiefer, Ch.: Approximation von minimalen Steinerbäumen mit Hilfe von zufallsabhängigen Suchstrategien. Diplomarbeit, TU Wien, 1989

/SCH91/ Schriber, T.: An Introduction to Simulation. Wiley&Sons, NewYork,1991

/SCH92/ Schriber, T.: Modeling the Un/Availability of Resources in Wolverine GPSS/H. Lehrtransparente, Magdeburg, 1992

/SCH93/ Schmidt, B.: Simulationssysteme der 5. Generation. in /SYD93/

/SD91/ Spears, W.M.; DeJong, K.: An analysis of multi-point crossover. in /RAW91/, S. 301-315, Morgan Kaufmann, 1991

/SHF1994/ Schöneburg, E.; Heinzmann, F.; Feddersen, S.: Genetische Algorithmen und Evolutionsstrategien, Addison-Wesley, 1994

/SHMH90/ Sumida, B.H.; Houston, A.I.; McNamara, J.M.; Hamilton, W.D.: Genetic algorithms and evolution. Journal of Theoretical Biology, S. 59-84, Nov 1990

/SL91/ Shang, Yi; Li, Guo-Jie: New crossover operators in genetic algorithms.in Proceedings of Third International Conference on Tools for Artificial Intelligence TAI 91, S. 150-153, IEEE Computer Society Press, Los Alamitos, CA, San Jose, CA, November 1991.

/SM91/ Schwefel, H.P.; Männer, R.: Parallel Problem Solving form Nature-Proceedings 1st Workshop PPSN 1, volume 496 of Lecture Notes in Computer Science, Berlin, Springer-Verlag, 1991

/SPE93/ Spears, W.M.: Crossover or mutation? In L. Darrell Whitley, editor, Foundations of Genetic Algorithms, 2, S. 221-237, Morgan Kaufmann, 1993

/SYD93/ Sydow, A.: Fortschritte in der Simulationstechnik Band 6; 8. Smposium Berlin, Vieweg Verlag, Braunschweig/Wiesbaden, 1993

/SYS89/ Syswerda, G.: Uniform crossover in genetic algorithms. in /SCH89a/

/SYS91a/ Syswerda, G.: Schedule optimization using genetic algorithms. S. 332-349 in /DAV91/

/TN92/ Thangiah, S.R.; Nygard, K.E.: School bus routing using genetic algorithms. In Proceedings of the SPIE, volume 1707 of Applications of Artificial Intelligence X: Knowledge based systems, S. 387-398, Orlando, FL, April 1992

/VOI86/ Voigt, H.-M.: Evolution und Optimierung - Ein populationsgenetischer Zugang zu kombinatorischen Optimierungsproblemen. Dissertation, Berlin, 1986

/WAG92/ Wagner, Pierre: Entwurf und Implementation eines graphischen Editors zur Modellbildung und eines statistischen Analyseprogramms für ein diskretes Simulationssystem unter Windows 3.0. Diplomarbeit, TU Wien, 1992

/WEN93/ Wenzel, S: Konfigurierbare Simulatoren - Die Werkzeuge der 90er Jahre. in /SYD93/

/WIE90/ Wienholt, W.: Through chance to success. genetic algorithms. Microcomputer Zeitschrift, S. 152-154, S. 156-158, S. 160-163, Deutschland, März 1990

/WOLV89/ Wolverine Software Corporation: GPSS/H Reference Manual. Handbuch, Annandale, 1989

/WOLV91/ Wolverine Software Corporation: System Animation Using PROOF. Tutorial Guide, Annandale, 1991

/WRI92/ Wriesnegger, Walter: Entwurf und Implementation eines Animationssystems für ein diskretes Simulationsmodells unter Windows 3.0. Diplomarbeit, TU Wien, 1992

/YR92/ Yang, H.; Richards, G.: Worst-case analysis of distribution system harmonics using genetic algorithms. In Proceedings of IEEE SOUTHEASTCON 92, volume 2, S. 530-533, Birmingham, AL, April 1992. IEEE, New York

/ZR92/ Zhang, J.; Roberts, F.D.: Use of genetic algorithms in training diagnostic rules for process fault diagnosis. Knowledge-Based systems, S. 277-288, Dezember 1992

Lebenslauf

16.4.1968	geboren in A-3340 Waidhofen/Ybbs
1974-1978	Besuch der Volksschule in A-3334 Gaflenz
1978-1982	Besuch der Unterstufe des BRG Waidhofen/Ybbs
1982-1986	Besuch der Oberstufe (naturwissenschaftliche Richtung)
31.5.1986	Matura
1986	Beginn des Studiums der Technischen Mathematik
	Zweig: Wirtschafts- und Planungsmathematik
1990	1. Diplomprüfung bestanden
1992	2. Diplomprüfung bestanden
26.11.1992	Sponsion zum Diplom-Ingenieur
seit 1992	freier Mitarbeiter der Abteilung Simulationstechnik
	regelmäßiger Besuch wissenschaftlicher Tagungen
1993	Beginn des Doktoratsstudiums der Technischen Wissenschaften
seit 1993	Mitarbeiter der Mitgliederverwaltung von ASIM
seit 1994	Mitarbeiter der ARGE Simulation News
	Mitarbeiter im Organisationskommittee der Tagung EUROSIM’95
seit 1995	Leiter der lokalen Organisation der Tagung EUROSIM’95

Über den Autor …

Dip.-Ing. Dr. Manfred Salzmann hat seine Ausbildung im Bereich der Simulationstechnik gegen Ende seines Studiums der Technischen Mathematik/Wirtschaftsmathematik im Jahre 1991 an der Technischen Universität Wien begonnen und neben Erstellung von Diplomarbeit und Dissertation maßgeblich am Aufbau der ARGE Simulation News mitgearbeitet.
Derzeit ist er im Logistikbereich einer Unternehmensberatung tätig und mit praxisnahen Problemen der Computersimulation konfrontiert.

Über diesen Band …

Genetische Algorithmen in diskreter Simulation ist als eine leicht lesbare Einführung sowohl in den Bereich der Genetischen Algorithmen (GA) als auch in den Bereich der diskreten Simulation gedacht, die auch den Einsatz der GA als Optimierungswerkzeug in der diskreten Simulation diskutiert.

Zu Beginn wird eine mit verständlichen Beispielen unterstützte Einführung in die Struktur und Arbeitsweise der GA geboten, gefolgt von speziellen Aspekten der GA, wie etwa sinnvolle Parameterauswahl, effiziente Implementierung, Überlegungen zur Konvergenz und fortgeschrittene Operatoren und deren Anwendung. Anschließend wird die Computersimulation als Analysewerkzeuge für diskrete Prozesse vorgestellt, um schließlich eine Verbindung zu den GA als Optimierungswerkzeug herstellen zu können. Für diese Verbindung werden einige interessante Konzepte vorgestellt und Implementierungsmöglichkeiten diskutiert.

Über diese Reihe …

Die Bände dieser neuen ASIM - Reihe Fortschrittsberichte Simulation konzentrieren sich auf neueste Lösungsansätze, Methoden und Anwendungen der Simulationstechnik (Ingenieurwissenschaften, Naturwissenschaften, Medizin, Ökonomie, Ökologie, Soziologie, etc.).

ASIM, die deutschsprachige Simulationsvereinigung (Fachausschuss 4.5 der GI - Gesellschaft für Informatik) hat diese Reihe ins Leben gerufen, um ein rasches und kostengünstiges Publikationsmedium für derartige neue Entwicklungen in der Simulationstechnik anbieten zu können.

Die Fortschrittsberichte Simulation veröffentlichen daher: * Monographien mit speziellem Charakter, wie z. B. Dissertationen und Habilitationen * Berichte zu Workshops (mit referierten Beiträgen) * Berichte von Forschungsprojekten * Handbücher zu Simulationswerkzeugen (User Guides, Vergleiche, Benchmarks), und Ähnliches. Die Kooperation mit den ARGESIM Reports der ARGESIM vermittelt dabei zum europäischen Umfeld und zur internationalen Publikation.