Prüfer:	Prof. Dr. rer. nat. habil. Paul Levi
Betreuer:	Priv.-Doz. Dr. habil. Andreas Zell

begonnen am:	1. Mai 1995
beendet am:	31. Oktober 1995

CR-Klassifikation:	C.1.3, D.1.3, I.2.8

Studienarbeit Nr. 1478

Parallele genetische Algorithmen auf einem Neurocomputer

Marc Ebner

Fakultät Informatik
Institut für Parallele und
Verteilte Höchstleistungsrechner
Universität Stuttgart
Breitwiesenstraße 20–22
D–70565 Stuttgart

Vorwort

Die vorliegende Arbeit wurde von der Friedrich-Naumann-Stiftung aus Mitteln des Bundesministeriums für Bildung und Wissenschaft gefördert. Ich möchte der Friedrich-Naumann-Stiftung an dieser Stelle ganz herzlich für die materielle und vor allem für ihre ideelle Förderung danken, die sie mir über ein Stipendium zuteil werden läßt.

Aufgabe dieser Arbeit war es, zu untersuchen, wie sich genetische Algorithmen auf einen Neurocomputer abbilden lassen. Als Plattform stand der Neurocomputer CNAPS von Adaptive Solutions mit 512 Prozessorelementen (PEs) zur Verfügung. Aufgrund der speziellen Kommunikationsstruktur und der Ganzzahlenarithmetik ist der Einsatz genetischer Algorithmen auf einem Neurocomputer schwieriger als auf anderen Architekturen. Es wurde ein paralleler genetischer Algorithmus für die CNAPS (CNGA) entwickelt. Die Untersuchungen, die mit CNGA durchgeführt wurden, zeigen, daß Neurocomputer zur Lösung von Optimierungsproblemen mittels genetischer Algorithmen eingesetzt werden können.

Bei der Implementierung des genetischen Algorithmus wurde auf eine maximale Parallelisierung geachtet. Der für diese Arbeit entwickelte genetische Algorithmus wurde an Standard-Benchmark-Problemen getestet und sein Verhalten analysiert. Es ergaben sich im Vergleich zu anderen Implementierungen massiv paralleler genetischer Algorithmen deutlich kürzere Laufzeiten. Das Problem des Handlungsreisenden zeigte jedoch die Grenzen des Neurocomputers auf. Die Ursachen für die Schwierigkeiten, die der Algorithmus bei der Lösung dieses Problems hatte, wurden genauer untersucht.

Die Arbeit ist eingebunden in ein größeres Projekt "Evolutionäre Algorithmen (EvA)" [Baumann 95], [Hummler 95], [Zell et al. 95a], [Zell et al. 95b], [Görzig 95], [Wakunda 95a] und [Hasel 95]. Innerhalb dieses Projektes wurden verteilte genetische Algorithmen auf dem MIMD-Computer Intel Paragon (VEGA) von Baumann [Baumann 95], massiv parallele genetische Algorithmen auf dem SIMD-Computer MasPar MP-1 (MPGA) von Hummler [Hummler 95] untersucht. Massiv parallele Evolutionsstrategien auf dem SIMD-Computer MasPar MP-1 (MPES) wurden von Görzig [Görzig 95] und verteilte Evolutionsstrategien auf dem MIMD-Computer Intel Paragon (VEES) von Wakunda [Wakunda 95a] untersucht. Die evolutionären Algorithmen werden mit einer von Hasel [Hasel 95] entwickelten graphischen Benutzeroberfläche zusammengefaßt.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Kapitel 1

Einleitung

Mutter Natur führt uns täglich vor Augen, welch mächtiger Mechanismus in ihr steckt. Über die Jahre hinweg produziert sie eine Vielzahl von Pflanzen und Lebewesen. Diese Organismen stellen die Individuen einer Population dar, die sich in ihrer Umgebung behaupten müssen. Wenn sich die Individuen als fit erweisen, d.h. sie sind gut angepaßt an ihre Umgebung, so können sie sich verstärkt fortpflanzen. John Holland erkannte bereits in den 70er Jahren [Holland 92], daß sich die Mechanismen der Evolution auf das Gebiet der Informatik übertragen lassen. Die von Holland entwickelten genetische Algorithmen versuchen, ein Optimierungsproblem zu lösen, indem Sie die Vorgänge der natürlichen Evolution auf entsprechende Algorithmen abbilden. In die Klasse der evolutionären Algorithmen fallen auch die von Rechenberg [Rechenberg 73] [Rechenberg 94] entwickelten Evolutionsstrategien. Die Elemente der genetischen Algorithmen werden im folgenden beschrieben. Für eine Einführung siehe [Holland 92], [Goldberg 89] oder [Schöneburg et al. 94].

Bei genetischen Algorithmen werden die Individuen durch Bit-Strings gleicher Länge repräsentiert. Der Bit-String eines Individuums wird Chromosom genannt. Das Chromosom stellt die codierte Form des Individuum, den Genotyp dar. Als Phänotyp wird die decodierte Lösung bezeichnet. Der Bitstring stellt eine Lösung des Problems dar und entspricht der Desoxyribonucleinsäure (DNS) in der Natur. Der Crossover-Operator (Abbildung 1.1) der genetischen Algorithmen versucht die Mechanismen der natürlichen Fortpflanzung nachzubilden. Dabei werden aus zwei Individuen zwei neue Individuen erzeugt, indem der Genotyp der In-

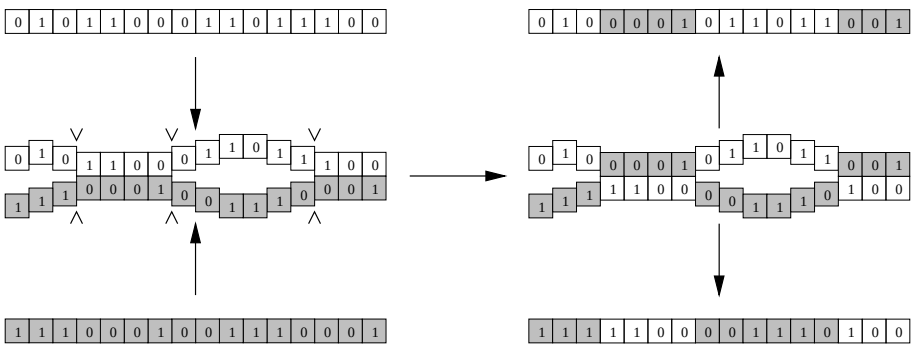

Abbildung 1.1: 3-Punkt-Crossover zweier Individuen

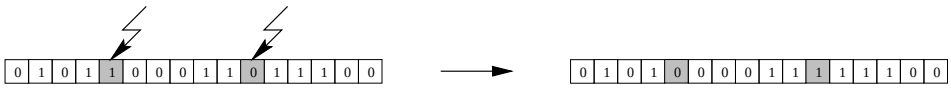

Abbildung 1.2: Mutation eines Individuums

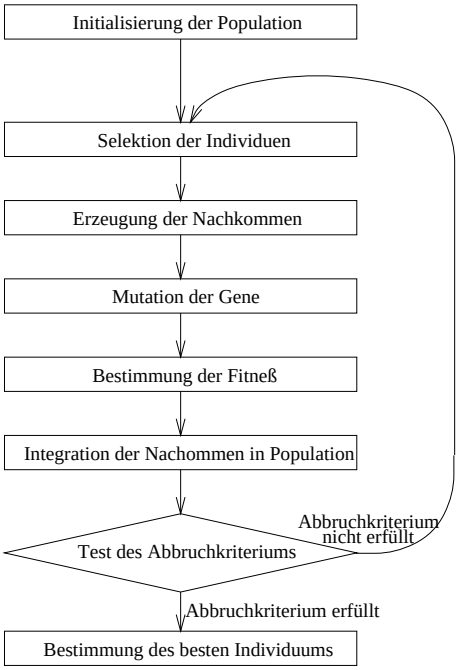

Abbildung 1.3: Flußdiagramm eines typischen genetischen Algorithmus

dividuen an n Punkten aufgetrennt wird, und die dabei entstehenden Stücke des Genotyps, wie in Abbildung 1.1 gezeigt, ausgetauscht werden. Dieser Operator wird n-Punkt Crossover genannt und wird mit der Crossover-Wahrscheinlichkeit p_{cross} eingesetzt.

Um festzustellen, wie fit ein Individuum der Population ist, benötigt der genetische Algorithmus einen Mechanismus zur Bewertung der Individuen. Diese Bewertung wird durch eine Fitneßfunktion durchgeführt. Die Lösung des Problems ist durch ein in der Regel globales Optimum der Fitneßfunktion gegeben. Die Fitneßfunktion wird eingesetzt, um eine verstärkte Vermehrung der fitten Individuen zu ermöglichen. Aus den fitten Individuen wird also eine neue Generation erzeugt.

Der Mutations-Operator (Abbildung 1.2), ebenfalls der Natur nachempfunden, verändert einzelne Bits des Genotyps mit der Mutations-Wahrscheinlichkeit p_{mut}. Dieser Operator ist sehr wichtig, da er sicherstellt, daß der gesamte Problemraum für die Individuen erreichbar bleibt. Denn weder die Selektion noch der Crossover-Operator ist in der Lage, ein konvergiertes Bit der Population zu verändern. Ein Bit an einer bestimmten Stelle (Lokus) im Genotyp nennt man konvergiert, wenn es bei allen Individuen der Population den gleichen Wert hat.

Abbildung 1.3 zeigt das Flußdiagramm eines typischen genetischen Algorithmus.

Kapitel 2

Parallele genetische Algorithmen

2.1 Modelle paralleler genetischer Algorithmen

Durch die Struktur der genetischen Algorithmen bieten sich Parallelrechner zur Berechnung an. Eine Übersicht verschiedener Parallelrechnerarchitekturen wird von Almasi et al. in [Almasi et al. 94] gegeben. Bräunl [Bräunl 94] gibt eine Einführung in die parallele Programmierung. Es gibt eine Reihe von Modellen (siehe [Schöneburg et al. 94]), wie die genetischen Algorithmen parallelisiert werden können. Dorigo et al. [Dorigo et al. 93] beschreiben zwei Klassen, in die parallele genetische Algorithmen eingeteilt werden können:

- Das Insel-Modell (Abbildung 2.1)

  Beim Insel-Modell coexistieren verschiedene Subpopulationen. Die Subpopulationen entwickeln sich parallel. Zwischen den Subpopulationen werden periodisch geeignete Individuen ausgetauscht.

- Das Nachbarschafts-Modell (Abbildung 2.2)

  Das Nachbarschafts-Modell plaziert jedes Individuum auf ein Gitter und definiert, welche Individuen als benachbart gelten. Die einzelnen Strategien und Operatoren des genetischen Algorithmus werden lokal durchgeführt.

Das Insel-Modell ist vor allem zur Implementierung verteilter genetischer Algorithmen auf MIMD-Computern geeignet. MIMD-Rechner besitzen oft eine kleine Anzahl an Prozessoren, die sehr Leistungsfähig sind. Somit kann von jedem PE eine Subpopulation berechnet werden. Der MIMD-Rechner Intel Paragon wurde von Baumann [Baumann 95] eingesetzt, um

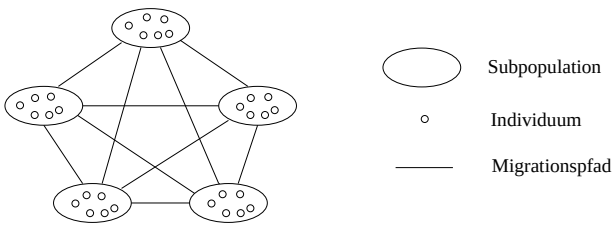

Abbildung 2.1: Insel-Modell (nach [Baumann 95])

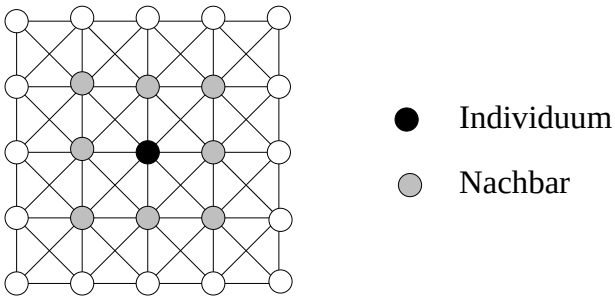

Abbildung 2.2: 8er Nachbarschafts-Modell

einen verteilten genetischen Algorithmus zu realisieren. Entsprechend des Insel-Modells plaziert Baumann auf jedes PE eine Subpopulation und tauscht in regelmäßigen Abständen geeignete Individuen aus. Crossover, Mutation und Selektion werden innerhalb einer Subpopulation ausgeführt.

Verteilte Systeme können ebenfalls zur Berechnung genetischer Algorithmen eingesetzt werden. Auch hier bietet sich wieder das Insel-Modell an, bei dem periodisch geeignete Individuen über ein Kommunikationsnetzwerk [Tanenbaum 89] ausgetauscht werden.

Für SIMD-Rechner bietet sich das Nachbarschafts-Modell an. SIMD-Rechner besitzen meist eine große Zahl an Prozessoren, die jedoch weniger leistungsfähiger sind. Bei SIMD-Rechnern wird meist jedem Individuum ein PE zugewiesen. Die Berechnung der Fitneß wird von allen PEs parallel durchgeführt. Die Selektion der Individuen findet in einer durch die Kommunikationstopologie definierten Umgebung statt. Der Crossover-Operator kann ebenfalls parallel durchgeführt werden, wenn für jedes Individuum ein Partner aus der Nachbarschaft geholt wird. Hummler [Hummler 95] setzte den SIMD-Rechner MasPar MP-1 zur Berechnung massiv paralleler genetischer Algorithmen ein.

2.2 Implementierungen paralleler genetischer Algorithmen

Es existieren bereits eine Vielzahl von Implementierungen paralleler genetischer Algorithmen. Für diese Arbeit wurden einige dieser Implementierungen für einen Vergleich ausgewählt. Sie werden kurz in Kapitel 7 vorgestellt.

Kapitel 3

Der Neurocomputer CNAPS

3.1 Hardware

Der Neurocomputer CNAPS ist ein Single Instruction Multiple Data (SIMD) Parallelrechner mit bis zu 512 Prozessorelementen (PEs). Jeder der 512 PEs hat 32 16-Bit-Register, 4KB lokalen Speicher, eine Schiebe/Logik-Einheit sowie einen Addierer und einen Multiplizierer. Die Prozessoren arbeiten mit Ganzzahlenarithmetik und einer Genauigkeit von 16 Bits. Sie sind mit 20 MHz getaktet. Eine 8x8 oder 8x16 Multiplikation benötigt einen Taktzyklus. Für eine 16x16 Multiplikation werden zwei Taktzyklen benötigt. Ein Divisionsbefehl ist nicht vorhanden.

Jeweils 64 dieser PEs sind in einem CNAPS-1064 Chip (Abbildung 3.1) zusammengefaßt. Befehle erhalten sie über einen 32-Bit-Instruktions-Bus. Ferner stehen den PEs ein 8-Bit-Eingabe- und ein 8-Bit-Ausgabe-Bus zur Kommunikation zur Verfügung. Außerdem ist jeder PE über je einen 4 Bit breiten Kommunikationskanal mit seinem rechten und linken Nachbar-PE verbunden. Dieser 4 Bit breite Kommunikationskanal besteht aus zwei unidirektionalen 2-Bit-Kanälen. Der interne Aufbau eines PEs ist in Abbildung 3.2 gezeigt.

Mit einer CNAPS Platine können bis zu 4 CNAPS-1064 Chips zusammengeschaltet werden. Ein CNAPS Server kann maximal zwei CNAPS Platinen aufnehmen. Somit ergibt sich die größte Konfiguration des Neurocomputers mit 512 PEs (Abbildung 3.3).

Der CNAPS Sequencer Chip (CSC) ist für die Steuerung der PEs zuständig. Er dekodiert die Instruktionen und steuert den Programmfluß. Die Daten für die PEs sendet er ihnen

Klassifikation nach Flynn	SIMD
Anzahl der PEs	64 - 512
Taktfrequenz	20 Mhz
Befehlslänge	64 Bit
Dateispeicher	16 MB
Programmspeicher	512 KB
Datenspeicher je PE	4 KB
Kommunikationstopologie	1D, Bus

Tabelle 3.1: Charakteristische Daten des Neurocomputers CNAPS

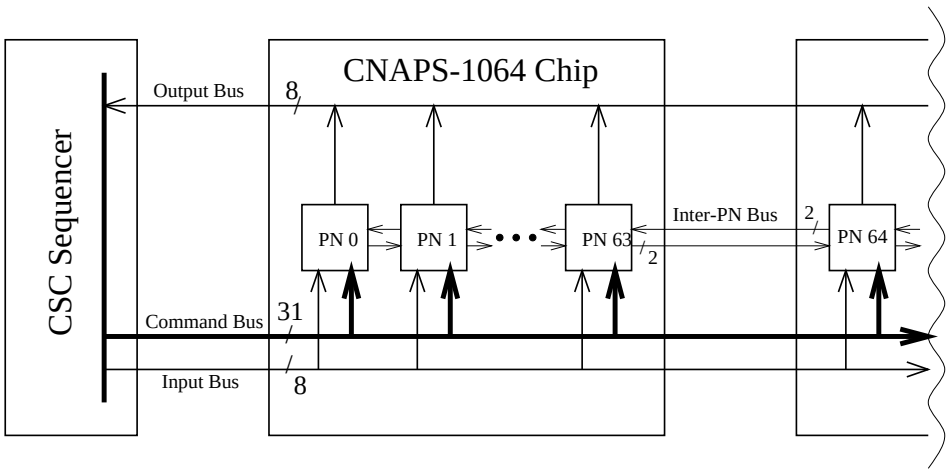

Abbildung 3.1: Aufbau des CNAPS-1064 Chips mit Kommunikationskanälen (nach [Adaptive Solutions 93a])

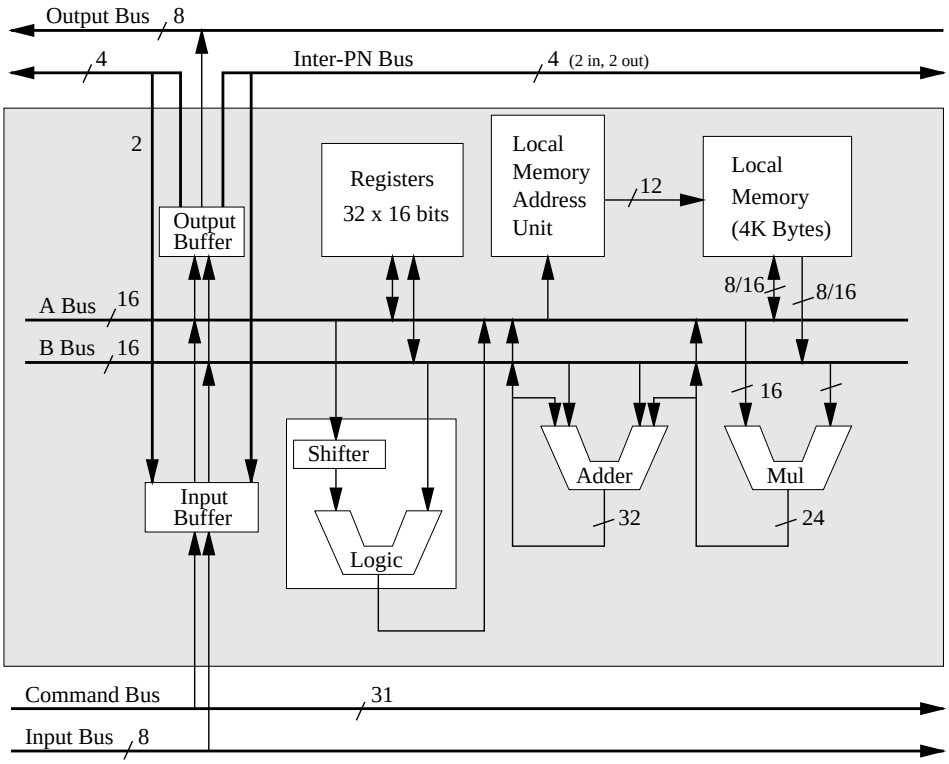

Abbildung 3.2: Interner Aufbau eines PEs (nach [Adaptive Solutions 93a])

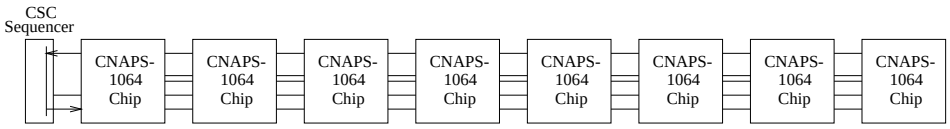

Abbildung 3.3: Der Neurocomputer CNAPS besteht aus maximal 8 CNAPS-1064 Chips

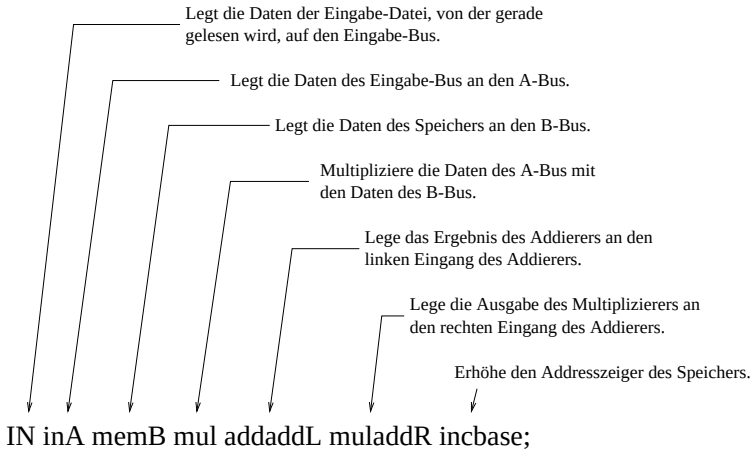

Abbildung 3.4: Ein CPL-Befehl (nach [Adaptive Solutions 94c])

durch den Eingabe-Bus und erhält die Ergebnisse über den Ausgabe-Bus. Der CSC besitzt eine Arithmetische/ Logische-Einheit (ALU), 64 32-Bit Register zur allgemeinen Verwendung und eine Engabe-Ausgabe-Pipeline. Die Eingabe-Ausgabe-Pipeline wird zur Verteilung der Informationen eines PEs an andere PEs eingesetzt. Durch die Pipeline werden die Daten des einen PEs an den Ausgabe-Bus angelegt und gelangen über die Eingabe-Ausgabe-Pipeline an den Eingabe-Bus und somit zu den PEs, die diese Daten benötigen.

Das gesamte CNAPS System besteht aus einer Workstation und dem CNAPS Server. Host und Server sind über eine Ethernet-Leitung miteinander verbunden. Die Befehle des Host werden vom Server mit dem sogenannten Control Processor Board entgegengenommen. Es besteht aus einem 680x0 Controller, mit 4 MB Speicher für den Datentransport zwischen Host und Server.

3.2 Software

3.2.1 Die CNAPS Programming Language CPL

Der Neurocomputer CNAPS kann mit der Programmiersprache CPL programmiert werden. Bei CPL handelt es sich um die Assembler-Sprache des Neurocomputers. Mit ihr hat der Programmierer die vollständige Kontrolle über den Datenfluß innerhalb der PEs. Die Befehle werden zu einem Very-Long-Instruction-Word (VLIW), also zu einem sehr langen Instruktionswort zusammengefaßt. Ein Befehl besteht aus 64 Bit, von denen die oberen 32 Bit für den

CNAPS Sequencer bestimmt sind. Von den restlichen 32 Bit werden 31 Bit an den Command-Bus und somit an die PEs angelegt. Ein Bit bleibt unbenutzt. Ein typischer Befehl ist in Abbildung 3.4 zu sehen. Eine kurze Einführung in CPL ist durch [Adaptive Solutions 93c] und eine genauere Darstellung durch [Adaptive Solutions 94c] gegeben.

3.2.2 CNAPS-C

Mit der Programmierung in CPL kann der CNAPS Server optimal eingesetzt werden. Allerdings ist der Programmcode schwer zu lesen. Daher ist auch die Wartung von CPL-Code erschwert. Adaptive Solutions bietet aber auch eine parallele Programmiersprache CNAPS-C an. CNAPS-C ist in weiten Teilen kompatibel zu ANSI-C. CNAPS-C enthält jedoch einige Erweiterungen, während wesentliche Eigenschaften aus ANSI-C fehlen. Zu den Erweiterungen von CNAPS-C gehören unter anderem folgende Funktionen.

- Sprachkonstrukte, um die parallel arbeitenden PEs des Servers auszunutzen.
- Festkommaarithmetik
- Minimum und Maximum Operatoren
- Unäre Reduktionsoperatoren
- In-Line CPL Programmcode
- Laufzeitkonstanten und sogenannte Entry Points
- Spezielle Kommandos für den Dateitransfer

Zu den wichtigsten derzeit nicht vorhandenen Funktionen von ANSI-C gehören:

- Division
- Modulo
- Fließkommazahlen
- Shift-Operation um eine Variable Anzahl von Bits
- Pointer auf Funktionen
- Standard C Bibliotheken

Die Programmierung mit der Sprache CNAPS-C ist ausführlich in [Adaptive Solutions 94b] beschrieben. In [Adaptive Solutions 94a] sind alle wesentlichen Funktionen von CNAPS-C zusammengefaßt.

Kapitel 4

Parallele genetische Algorithmen auf einem Neurocomputer

4.1 Motivation

Untersuchungen von Siegmund [Siegmund 92] und Hummler [Hummler 95] zeigten bereits, daß sich SIMD Computer wie z.B. die MasPar hervorragend zur Berechnung von genetischen Algorithmen eignen. Der Neurocomputer CNAPS ist ebenfalls ein SIMD Computer. Somit stellt sich die Frage, ob er nicht nur zum extrem schnellen Training und Einsatz Neuronaler Netze [Zell 94], sondern auch zur Berechnung genetischer Algorithmen geeignet ist.

Im Gegensatz zur MasPar hat die CNAPS kein Gitter und auch keinen Router für die Kommunikation, sondern nur die Bus- und 1D-Topologie. Außerdem rechnet die CNAPS (16-Bit-Prozessor) nur mit Ganzzahlen- bzw. Festkommaarithmetik, während die MasPar MP-1 (4-Bit-Prozessor) Fließkommazahlenarithmetik mit bis zu 64 Bit Genauigkeit besitzt. Daher stellt sich die Frage, wie der auf Gleitkommazahlen aufbauende Teil der genetischen Algorithmen auf den Neurocomputer übertragen werden kann.

Ein ebenfalls wichtiger Punkt ist der geringe Speicherplatz (4 KB) je PE. Wieviele Individuen können je PE plaziert werden? Wie kann der genetische Algorithmus implementiert werden, so daß er den vorhandenen Speicherplatz optimal ausnutzt. Der lokale Speicher der MasPar ist um den Faktor 4 bis 16 größer. Ferner besitzt die CNAPS mit maximal 512 PEs deutlich weniger PEs als die MasPar mit maximal 16384. Also können nur kleinere Populationen parallel berechnet werden.

Folgende Parameter bestimmen den genetischen Algorithmus auf der CNAPS:

- Integer und Festkommaarithmetik (16 Bit)
- 1D- und Bus-Kommunikationstopologie
- 4 KB lokaler Speicher je PE
- 512 PEs

Aus diesen gegebenen Parametern gilt es einen möglichst effizienten und zugleich allgemeinen genetischen Algorithmus mit größtmöglichem Parallelisierungsgrad zu entwickeln. Für die Komplexität [Hopcroft 90] [Cormen et al. 90] [Davis et al. 83] des genetischen Algorithmus sind folgende Größen relevant.

- n_x = Zahl der PEs
- n_y = Zahl der Individuen je PE
- l = Länge des Genotyps

Also ergibt sich eine Populationsgröße $n = n_x n_y$.

4.2 Berechnungen mit Fließkommazahlen auf der CNAPS

Zu Beginn der Arbeit wurde die Möglichkeit untersucht, keine Fließkommazahlen im genetischen Algorithmus zu verwenden. Dabei ergab sich, daß der genetische Algorithmus speziell auf die Zielfunktionen zugeschnitten gewesen wäre und somit nicht allgemein einsetzbar. Es hätten lediglich Selektionsverfahren implementiert werden können, die auf einfachem Vergleich basieren. Statistikdaten hätten nur eingeschränkt ermittelt werden können. Denn schon bei $l = 8$ und $n = 512$ kann bei den Crossover-Statistiken die Zahl 65536 auftreten, für deren Darstellung 17 Bits benötigt werden. Außerdem wären bei einer Summation über alle 512 PEs nur 7 Bits für die Codierung der Fitneß eines Individuums übriggeblieben.

Um den genetischen Algorithmus möglichst allgemein zu halten und einen objektiven Vergleich mit anderen Architekturen zu ermöglichen, wurde ein Modul für Fließkommazahlen implementiert. Es arbeitet mit einer Genauigkeit von 16 Bit für die Mantisse und 16 Bit für den Exponenten. Der Einsatz dieses Moduls wurde auf ein Minimum beschränkt. Lediglich der Fitneßwert eines Individuums wurde als Gleitkommazahl repräsentiert. Daher wurde es möglich, auch Verfahren wie z.B. Roulette-Rad-Selektion zu implementieren. Die Statistikdaten werden ebenfalls mit Fließkommazahlenarithmetik ermittelt. Da diese Daten aber nur für die Analyse des genetischen Algorithmus, wie z.B. das Konvergenzverhalten wichtig ist, kann deren Ermittlung aber auch vollständig abgeschaltet werden.

Für die Fließkommazahlenarithmetik wurden neben Addition, Subtraktion und Multiplikation auch die Division implementiert. Die Division wurde iterativ mit dem von Goldberg [Goldberg 90] beschriebenen Verfahren durchgeführt. Um den Quotient $x = \frac{z_1}{z_2}$ zu berechnen, wird erst iterativ $\frac{1}{z_2}$ wie folgt berechnet. Es sei $z_i = m_i 2^{e_i}$. Zunächst wird m_2 auf den Bereich $1 \leq m_2 < 2$ transformiert $b = 2m_2$. Dann wird die Iteration mit $x_0 = 0.6$ begonnen. Anschließend wird bis zum Erreichen der Genauigkeit $0,000091552$, jedoch höchstens fünfmal folgender Iterationsschritt durchgeführt.

$$x_{i+1} = x_i(2 - x_i b)$$

Das Ergebnis $z = z_m 2^{z_e}$ ergibt sich schließlich aus $z = m_1 x_{\max} 2^{e_1 - e_2 + 1}$.

Ferner wurden die Funktionen Negation und $\lfloor x \rfloor$ implementiert. Die Funktionen cos und exp können durch Summen approximiert werden [Bronstein et al. 89]. Sie haben aber wegen der endlichen Summe nur in kleinen Intervallen Gültigkeit und sind aufgrund der umfangreichen Berechnung sehr langsam.

4.3 Verteilung der Individuen auf die PEs

Aufgrund des sehr kleinen Speichers von 4 KB pro PE bietet sich zunächst an, nur ein Individuum pro PE zu plazieren. Als Selektionsverfahren kann eine globale Strategie gewählt werden.

Es wird also für alle PEs global entschieden, von welchen Individuen sie eine Kopie erhalten. Alternativ kann auch die 1D-Topologie für eine lokales Selektionsverfahren eingesetzt werden. Da der Crossover-Operator ein binärer Operator ist, benötigt er noch ein weiteres Individuum als Partner. Weil aber nur Platz für ein Individuum je PE zur Verfügung steht, kann der Partner nicht auf das PE kopiert werden, sondern muß in kleinen Stücken von anderen PEs auf das eigene PE transportiert werden.

Wird bei der Partnerwahl eine allgemeine Strategie zugelassen, d.h. jedes der anderen Individuen steht als potentieller Partner zur Verfügung, so kann der Crossover-Operator nur global implementiert werden. Die Plazierung von einem Individuum je PE hat also zur Folge, daß nur die Mutation, Evaluierung der Fitneß und eine lokale Selektions- und Integrationsstrategie parallel durchgeführt werden kann. Eine kleine Verbesserung ist dadurch zu erreichen, die Partnerwahl nur auf das jeweils nächste Individuum zu begrenzen. Dann kann der Kommunikationsring zum Transport verwendet werden und der Crossover-Operator ist auch bei nur einem Individuum je PE vollständig parallelisiert. Dies setzt aber voraus, daß die Individuen so auf die PEs verteilt sind, daß die benachbarten Individuen nicht identisch sind.

Falls nur ein Individuum auf ein PE paßt, so kann der Crossover-Operator auch nur ein neues Individuum erzeugen. Welches von den beiden möglichen Individuen erzeugt wird, kann durch eine Heuristik bestimmt werden. Steht genügend Speicher auf einem PE für zwei oder mehr Individuen zur Verfügung, so läßt sich auch der Crossover-Operator parallelisieren. Außerdem können aus zwei Individuen wieder zwei neue Individuen entstehen. Es sollten also mindestens zwei Individuen je PE plaziert werden. Für den Fall, daß der Genotyp nicht sehr lang ist, können auf jedem PE auch kleine Subpopulationen von Individuen verwaltet werden. Dann lassen sich lokale Selektionsstrategien anwenden. Der Austausch von Individuen mit hoher Fitneß kann über den Kommunikationsring erfolgen. Also ist bei der Verwendung von mindestens zwei Individuen je PE bei einer lokalen Selektionsstrategie eine vollständige Parallelisierung möglich.

4.4 Programmfluß des CNGA

Es wurde ein genetischer Algorithmus entwickelt, der alle oben genannten Möglichkeiten, die Individuen auf die PEs zu verteilen, in einem einzigen Algorithmus vereinigt. Je nachdem, ob eine globale oder lokale Selektionsstrategie gewählt wurde, wird die neue Generation entweder global über alle PEs oder lokal innerhalb der PEs erzeugt. Auch die Crossover-Strategie wird, je nach Wahl, entweder lokal oder global durchgeführt. Da die Selektion über den Bus und die Selektion über den Kommunikationsring zwei verschiedene Mechanismen sind, wurden sie voneinander entkoppelt und sind somit auch gleichzeitig einsetzbar. Der Programmfluß des genetischen Algorithmus ist wie in Abbildung 4.1 aufgebaut.

4.5 Datenstruktur

Für jedes Individuum werden folgende Werte gespeichert:

- Genotyp
- Wert der Zielfunktion
- Fitneß

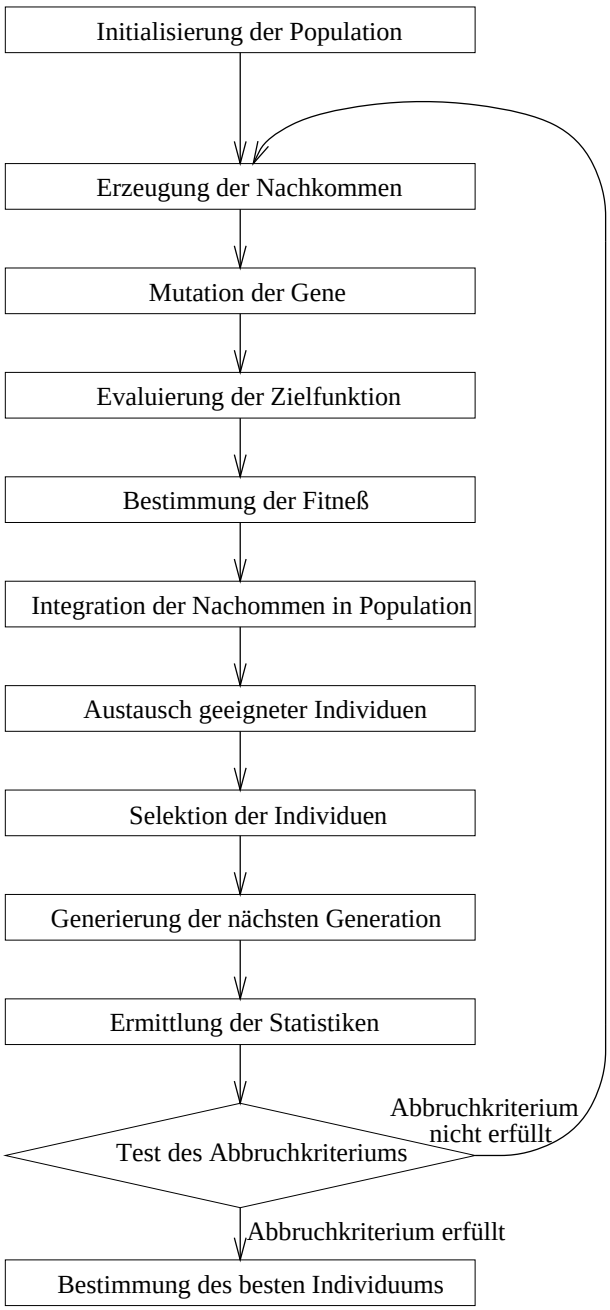

Abbildung 4.1: Der Programmfluß des CNGA

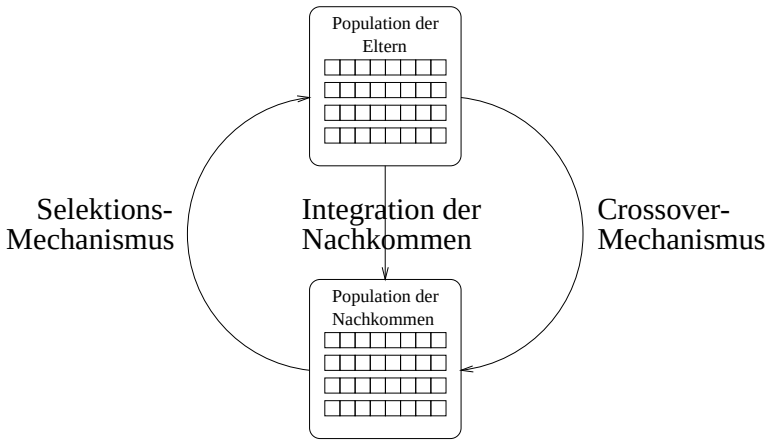

Abbildung 4.2: Datenfluß

- Alter

Die Individuen der PEs werden zu Populationen zusammengefaßt. Es gibt eine Population der Eltern und eine Population der Nachkommen. Beim Crossover wird die Datenstruktur der Nachkommen gefüllt. Dies geschieht in Abhängigkeit von der Crossover-Wahrscheinlichkeit p_{cross}. Anschließend werden die Individuen in die Generation der Eltern durch die Integrationsstrategie integriert. Dazu werden die Eltern der Individuen, die nicht in die Population integriert werden, in die Population der Nachkommen kopiert. Schließlich wird aus der Population der Nachkommen durch den Selektionsmechanismus eine neue Population von Eltern erzeugt und der Kreis schließt sich. Der kreisförmige Datenfluß (Abbildung 4.2) ermöglicht eine effiziente Implementierung, da er unnötiges Kopieren der Individuen vermeidet.

Auf den PEs werden außer den Individuen noch die für den Crossover-Operator und den Mutations-Operator erforderlichen Masken gespeichert. Die Verwendung der Masken ermöglicht ein schnelles Crossover und eine schnelle Mutation. Die i-te ($i \in \{1, \ldots, 16\}$) Crossover-Maske blendet die unteren $16 - i$ Bits eines Wortes aus. Die i-te Mutations-Maske hat das i-te Bit gesetzt und wird zur Selektion einzelner Bits eingesetzt.

Die Datenstruktur des Programms CNGA setzt sich wie in Abbildung 4.3 dargestellt zusammen.

4.6 Zufallszahlen

Zur Implementierung genetischer Algorithmen werden Zufallszahlen benötigt. Diese müssen lokal auf den PEs berechnet werden. CNGA berechnet die Zufallszahlen nach folgendem Verfahren

$$z_{n+1} = (bz_n + c) \mod m \quad \text{mit} \quad n \geq 0$$

Mit $b = 0xE66D$, $c = 0xB$ und $m = 2^{16}$ wobei z_0 das Saatkorn des Zufallszahlengenerators ist. Das Verfahren ist analog zu der Berechnung von `lrand48` wie es meist auf UNIX-Systemen bereitgestellt wird. `lrand48` [Sun 89] berechnet die Zahlen wie folgt:

$$z_{n+1} = (bz_n + c) \mod m \quad \text{mit} \quad n \geq 0$$

Abbildung 4.3: PE-Datenstruktur

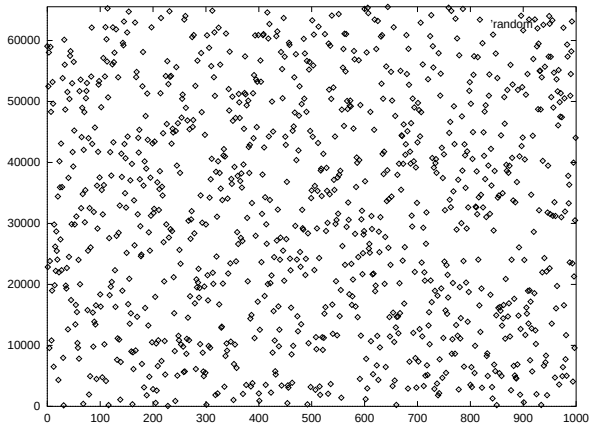

Abbildung 4.4: CNGA-Zufallszahlen-Verteilung

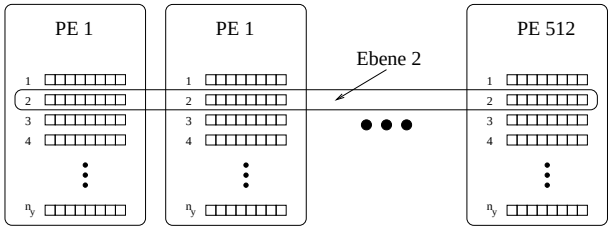

Abbildung 4.5: Ebene aus Individuen

Mit $b = 0x5DEECE66D$, $b = 0xB$ und $m = 2^{48}$. In Abbildung 4.4 ist die Verteilung von 1000 Zufallszahlen angegeben, wie sie von CNGA berechnet werden.

4.7 Partnerwahl

Um den Crossover-Operator anzuwenden, müssen zwei Individuen bestimmt werden, auf die der Operator angewandt wird. Folgende Möglichkeiten der Partnerwahl wurden implementiert. Die angegebene Zeitkomplexität schließt die Zeit c für den Crossover-Operator mit ein. n_y gibt die Anzahl der Individuen je PE und n_x die Zahl der PEs an. Eine Ebene beschreibt im folgenden Text alle Individuen mit gleichem Index (Abbildung 4.5). Die Individuen werden lokal durch `(ind)` und global durch `(pe,ind)` referenziert. Dabei gibt `pe` die Nummer des PEs und `ind` die Nummer innerhalb des PEs an.

- Externe-Zufällige Partnerwahl

  Für jedes Individuum einer Ebene wird ein Partner aus derselben Ebene zum Crossover mittels eines Zufallszahlengenerators ermittelt. Nachdem jedem Individuum der Ebene die Nummer seines Partners bekannt ist, werden nacheinander alle Crossover durchgeführt. Der Reihe nach werden die Individuen durchlaufen, und mit den Individuen

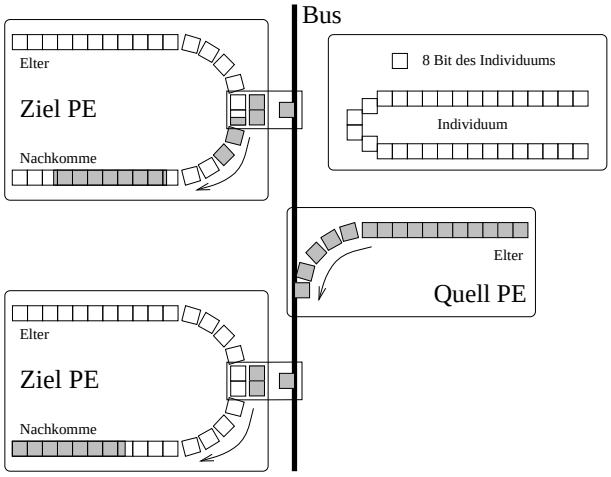

Abbildung 4.6: Externe-Zufällige Partnerwahl

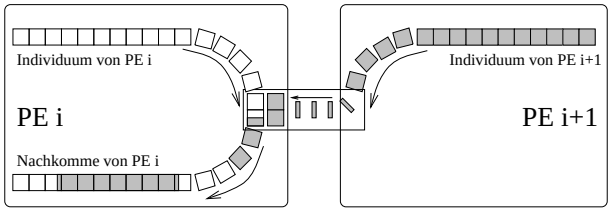

Abbildung 4.7: Externe-Nachbar Partnerwahl

parallel ein Crossover durchgeführt, die denselben Partner gewählt haben. Als Nachkomme wird das Individuum erzeugt, das die meisten Bits des fitteren Individuums erhält (Single-Offspring-Crossover-Heuristik). Es wird also nur ein Nachkomme erzeugt. Die Zeitkomplexität ist $\Theta(n_x n_y c)$.

```
FOR ind := 1 TO n_y DO
  r := Zufallszahl zwischen 1 und n_x
  FOR pe := 1 TO n_x DO
    IF pe = r THEN
      Crossover zwischen Individuum(ind) und
      Individuum(pe,ind) erzeugt Nachkomme(ind)
    END
  END
END
```

- Externe-Nachbar Partnerwahl

  Für jedes Individuum einer Ebene wird ein Crossover mit dem Individuum des nächsten PEs durchgeführt. Auch hier entsteht aus zwei Individuen nur ein neues Individuum.

Durch die Single-Offspring-Crossover-Heuristik wird vor Durchführung des Crossover bestimmt, welches Individuum als Nachkomme erzeugt wird. Die Zeitkomplexität ist $\Theta(n_y c)$.

```
FOR ind := 1 TO n_y DO
  Crossover zwischen Individuum(ind) und
  Individuum(ownPE+1,ind) erzeugt Nachkomme(ind)
END
```

- Interne-Zufällige Partnerwahl

  Für jedes Individuum einer Ebene wird ein Crossover mit einem zufällig ausgewählten Individuum desselben PEs durchgeführt. Die Zeitkomplexität ist $\Theta(n_y c)$.

```
FOR ind := 1 TO n_y DO
  r := Zufallszahl zwischen 1 und n_x
  Crossover zwischen Individuum(ind) und
  Individuum(r) erzeugt Nachkomme(ind)
END
```

- Interne-Nachbar Partnerwahl

  Je zwei Individuen eines PEs werden als Pärchen zusammengefaßt. Individuen mit geradem und ungeradem Index innerhalb eines PEs gelten als ein Paar. Beide Individuen produzieren bei einem Crossover zwei Nachkommen. Die Zeitkomplexität ist $\Theta(n_y c)$.

```
FOR ind := 1 TO n_y STEP 2 DO
  r := Zufallszahl zwischen 1 und n_x
  Crossover zwischen Individuum(ind) und Individuum(ind+1)
  erzeugt Nachkomme(ind) und Nachkomme(ind+1)
END
```

4.8 Nachkommen

Nachdem das Individuum einen Partner bekommen hat, wird mit der Wahrscheinlichkeit p_{cross} ein Crossover-Operator angewandt. Dabei werden je nach Partnerwahl entweder ein oder zwei Individuen als Nachkommen erzeugt. Befinden sich beide Eltern auf demselben PE und werden zwei Nachkommen erzeugt (Interne Partnerwahl), dann arbeitet der Crossover-Operator Wort für Wort des Genotyps ab und produziert zwei Nachkommen.

Wenn ein Elternteil von einem anderen PE geholt werden muß, wird nur ein Nachkomme erzeugt. Um den vorhandenen Speicherplatz möglichst geschickt zu nutzen, und keinen extra Speicherplatz für ein temporäres Individuum zu verwenden, wird beim externen Crossover der Genotyp des Partnerindividuums wortweise übertragen. Mit dem übertragenen Teilstück des Partnerindividuums wird dann das Crossover fortgesetzt.

Je nach Partnerwahl wird einer der folgenden Crossover-Mechanismen angewandt:

- Externes-Globales Crossover

  Bildet aus all den Individuen einer Ebene je PE einen Nachkommen, mit demselben Partner. Welcher Nachkomme erzeugt wird ergibt sich durch die Single-Offspring-Crossover-Heuristik. Zuvor wurden durch die Partnerwahl die PEs inaktiviert, die den augenblicklichen Partner nicht benötigen.

- Externes-Nachbar Crossover

  Jedes Individuum bildet je PE einen Nachkommen mit dem Individuum, das sich auf dem Nachbar-PE befindet. Welcher Nachkomme erzeugt wird, ergibt sich durch die Single-Offspring-Crossover-Heuristik. Alle PEs sind aktiv.

- Internes Crossover

  Bildet aus zwei Individuen, die sich auf demselben PE befinden, zwei Nachkommen. Auch hier sind alle PEs aktiv.

Folgende Crossover-Verfahren [Bäck 92] können durch den Benutzer ausgewählt werden: Dabei gibt `l` die Länge des Genotyps in Worten an.

- Uniformes Crossover

  Das Uniforme Crossover bestimmt für jedes Wort des Genotyps eine Maske und führt so das Crossover durch. Die Zeitkomplexität ist $\Theta(l)$.

- 1-Punkt Crossover

  Das 1-Punkt Crossover arbeitet den Genotyp wortweise ab. An den Schnittstellen wird mittels einer Bitmaske das Crossover durchgeführt. Die Zeitkomplexität ist $\Theta(l)$.

- 2-Punkt Crossover

  Auch das 2-Punkt Crossover arbeitet den Genotyp wortweise ab. Wie beim 1-Punkt Crossover wird ebenfalls an den Schnittstellen mittels einer Bitmaske das Crossover durchgeführt. Die Zeitkomplexität ist $\Theta(l)$.

- n_c-Punkt Crossover

  Das n_c-Punkt Crossover ruft n_c-mal das 1-Punkt-Crossover auf. Alternativ wäre es auch möglich $\lfloor \frac{n_c}{2} \rfloor$-mal 2-Punkt Crossover und $n_c - \lfloor \frac{n}{2} \rfloor$-mal 1-Punkt Crossover aufzurufen. Das n_c-Punkt Crossover könnte auch bitweise mit der Komplexität $\in \theta(n_c \log n_c + l)$ implementiert werden, nachdem zuvor die Crossoverpunkte der Reihe nach sortiert wurden. Diese Lösung wäre aber bei bis zu 16 Crossover-Punkten langsamer. Da die meisten Probleme kaum mehr als 16 Crossover-Punkte erfordern bzw. bei einer großen Anzahl von Crossover-Punkten das Uniforme Crossover statt eines n_c-Punkt-Crossovers einsetzbar ist, wurde die erste Lösung implementiert. Die Zeitkomplexität ist $\Theta(n_c l)$.

4.9 Mutation

Dem Anwender stehen vier verschiedene Mutationsarten zur Auswahl.

- Mutation einzelner Bits

  Für jedes Bit wird einzeln geprüft, ob es zu mutieren ist oder nicht. Dabei gibt p_{mut} die Wahrscheinlichkeit an, mit der ein einzelnes Bit mutiert wird. Ist das Bit gesetzt, wird es bei einer Mutation gelöscht. Wenn das Bit gelöscht ist, wird es bei einer Mutation gesetzt. Dies entspricht dem Standard-Mutations-Operator [Holland 92].

```
FOR i := 1 TO 16l DO
  Mutiere Bit(i) des Genotyps mit Wahrscheinlichkeit p_mut
END
```

- Mutation ganzer Worte

  Bei der Mutation ganzer Worte wird für jedes Wort des Genotyps geprüft, ob das Wort zu mutieren ist. Hier gibt p_{mut} die Wahrscheinlichkeit an, mit der ein ganzes Wort mutiert wird. Wenn ein Wort zu mutieren ist, dann wird mit einem Zufallszahlengenerator ein neues Wort für den Genotyp des Individuums erzeugt.

```
FOR i := 1 TO l DO
  Mutiere Wort(i) des Genotyps mit Wahrscheinlichkeit p_mut
END
```

- Angesammelte Mutation einzelner Bits

  Da meist nur wenige Bits eines Genotyps zu mutieren sind, ist der Aufwand der beiden oben beschriebenen Mutations-Verfahren recht groß. Daher wird bei der angesammelten Mutation die Zahl der Bits, die erwartungsgemäß zu mutieren sind, aufsummiert. Erst wenn diese Zahl größer als 1 ist, wird die zu erwartende Anzahl von Mutationen durchgeführt. Bei einer Mutation wird zuerst die Position, an der die Mutation durchzuführen ist, bestimmt und dort dann die Mutation ausgeführt.

```
Mutationen := p_mut16l
IF Mutationen > 1.0 THEN
  FOR i:=1 TO INT(Mutationen) DO
    Mutiere ein Bit des Genotyps
  END
  Mutationen = Mutationen - INT(Mutationen)
END
```

  Die Variable `Mutationen` wird zu Beginn des genetischen Algorithmus auf einen zufälligen Wert gesetzt.

- Angesammelte Mutation ganzer Worte

  Die angesammelte Mutation ganzer Worte funktioniert analog zu der angesammelten Mutation einzelner Bits. p_{mut} gibt die Wahrscheinlichkeit an, mit der ganze Worte des Genotyps mutiert werden.

```
Mutationen := p_mut l
IF Mutationen > 1.0 THEN
  FOR i:=1 TO INT(Mutationen) DO
    Mutiere ein Wort des Genotyps
  END
  Mutationen = Mutationen - INT(Mutationen)
END
```

4.10 Suchrichtung

Der Wert der Zielfunktion muß zur Bearbeitung durch ein Selektionsverfahren in einen Fitneßwert transformiert werden. Jedes der implementierten Selektionsverfahren ist darauf ausgerichtet, Individuen hoher Fitneß verstärkt auszuwählen. Wird nun nach dem Minimum einer Funktion gesucht, muß der Wert der Zielfunktion negiert werden, um einen Fitneßwert zu bilden. Folgende Transformationsfunktionen wurden implementiert:

- Identität

  Der Wert der Zielfunktion ist gleich dem Fitneßwert.

- Negation

  Der Wert der Zielfunktion wird negiert und ergibt so den Fitneßwert.

Der Fitneßwert kann also durchaus negative Werte annehmen. Selektionsverfahren, wie z.B. Roulette-Rad-Selektion, die positive Fitneßwerte benötigen, müssen also ein weiteres Mal transformiert werden. Diese zweite Transformation wird vom Selektionsverfahren durchgeführt. Die Entscheidung für eine zweistufige Transformation wurde getroffen, um einerseits größte Flexibilität und andererseits kleine Laufzeiten zu erreichen. Flexibilität ist durch die Trennung gegeben. Kleine Laufzeiten sind zu erwarten, da nur die Selektionsverfahren, bei denen eine zweite Transformation nötig ist, diese durchführen. Beim Ranking Selektionsverfahren können die Fitneßwerte direkt für einen Vergleich herangezogen werden.

4.11 Integration

Nachdem der Fitneßwert der Nachkommen bestimmt ist, werden die Nachkommen in die Population integriert. Da die Anzahl der Individuen in der Population konstant bleibt, muß jeder Nachkomme, der in die Population integriert wird, einen Elter ersetzen. Mit einem Integrationsverfahren wird entschieden, wie die Nachkommen in die Population integriert werden. Dem Anwender stehen folgende Verfahren zur Auswahl.

- Alle Nachkommen

  Jeder Nachkomme wird in die Population integriert. Die Zeitkomplexität ist $\Theta(1)$. Sie ergibt sich, da die Nachkommen nicht mehr kopiert werden müssen (siehe Abbildung 4.2).

- Fitte Nachkommen

  Ein Nachkomme mit Index (i) ersetzt nur dann ein Individuum mit dem gleichen Index (i), wenn der Nachkomme eine größere Fitneß besitzt, als das zu ersetztende Individuum. Die Zeitkomplexität ist $\Theta(n_y l)$.

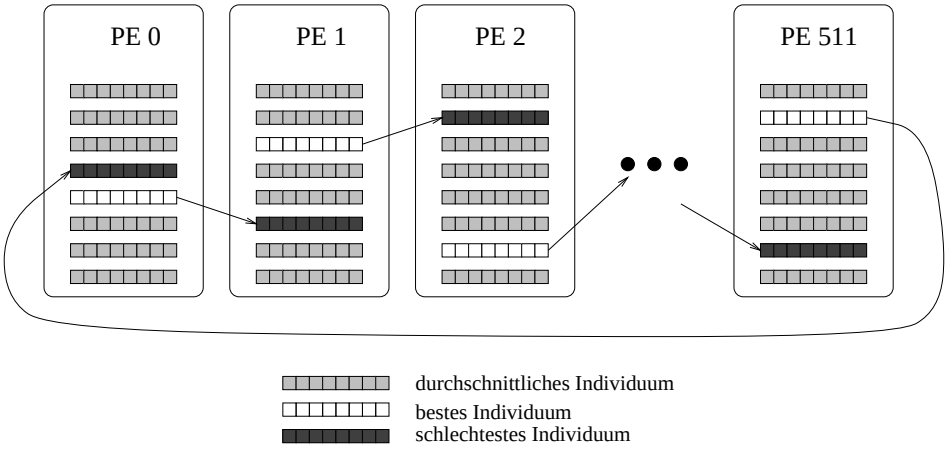

Abbildung 4.8: Verschiebe Bestes auf Schlechtestes

Weitere Integrationsverfahren sind denkbar. In [Goldberg 89] wird z.B. ein Integrationsverfahren beschrieben, bei dem für jedes Nachkommen das zu ihm ähnlichste Individuum in der Population ermittelt wird. Dieses Individuum wird dann von dem ihm ähnlichen Nachkommen ersetzt. Dieses Verfahren ist dazu geeignet, das Problem der vorzeitigen Konvergenz des genetischen Algorithmus zu vermeiden, indem Nischen gebildet werden. Jedes Individuum befindet sich in einer Nische und wird nur von Individuen, die in dieselbe Nische fallen, ersetzt.

4.12 Austausch geeigneter Individuen

Zwischen benachbarten PEs können Individuen ausgetauscht werden. Wie oft dies während einer Generation geschieht, wird durch den Parameter `number_of_exchanges` bestimmt. Dieser Mechanismus ist eigentlich ein weiteres Selektionsverfahren. Aufgrund der speziellen Kommunikationstopologie des Neurocomputers (Bus und 1D) wurden beide Mechanismen voneinander entkoppelt. Der Austausch geeigneter Individuen bedient sich der 1D-Topologie, wobei der Bus genutzt wird, um die 1D-Topologie zu einem Ring zu schließen. Die globale Selektion verwendet die Bus-Topologie und ist zur Implementierung allgemeiner Selektionsverfahren geeignet.

- Verschiebe Bestes auf Schlechtestes

  Je PE wird das beste Individuum ausgewählt. Dies wird über die Ring-Topologie nach rechts geschoben und überschreibt dort auf dem PE das schlechteste Individuum (Abbildung 4.8). Befindet sich auf jedem PE nur ein Individuum, dann hat dieser Mechanismus keine Auswirkungen, da lediglich alle Individuen um einen Platz nach rechts verschoben werden. Dieser Mechanismus sollte nur dann verwendet werden, wenn sich mindestens zwei Individuen auf jedem PE befinden. Die Zeitkomplexität ist $\Theta(n_y + l)$.

- Bedingtes Verschieben

  Je PE wird das beste Individuum ausgewählt. Über die Ring-Topologie wird dieses Individuum nach rechts verschoben und auf dem Nachbar-PE zwischengespeichert. Nur wenn das neue Individuum besser ist, als das schlechteste Individuum auf diesem PE, dann wird

das schlechteste Individuum durch das neue Individuum ersetzt. Die Zeitkomplexität ist $\Theta(n_y + l)$.

- Kopiere das Beste von links und rechts auf das Schlechteste

  Je PE wird das beste Individuum ausgewählt. Dies Individuum wird einmal nach rechts und einmal nach links verschoben. Das Beste der beiden von links und rechts eingetroffenen Individuen ersetzt dann das schlechteste Individuum. Die Zeitkomplexität ist $\Theta(n_y + l)$.

- Bedingtes Kopieren des Besten von links und rechts

  Je PE wird das beste Individuum ausgewählt. Auch beim bedingten Kopieren wird das beste Individuum einmal nach rechts und einmal nach links verschoben. Anschließend wird aus den beiden neu eingetroffenen Individuen das beste Individuum ermittelt. Dieses Individuum ersetzt aber nur dann das schlechteste Individuum auf dem PE, wenn es eine höhere Fitneß als das schlechteste Individuum hat. Die Zeitkomplexität ist $\Theta(n_y + l)$.

4.13 Selektion

Mit dem Selektionsverfahren des CNGA wird für jedes Individuum bestimmt, durch welches andere Individuum es ersetzt wird. Folgende verschiedene Verfahren wurden implementiert:

- Keine Selektion

  Jedes Individuum bleibt auf der gleichen Position. Die Zeitkomplexität ist $\Theta(n_y)$.

- Lokales Roulette-Rad

  Um die Roulette-Rad-Selektion [Holland 92] durchzuführen, müssen zuerst das Individuum mit maximaler, das Individuum mit minimaler Fitneß und die Summe der Fitneßwerte auf jedem PE bestimmt werden. Da auch negative Fitneßwerte auftreten können, werden die Fitneßwerte erst in den positiven Bereich verschoben und anschließend mit dem in [Goldberg 89] beschriebenen Verfahren skaliert. Die Skalierung der Fitneßwerte ist erforderlich, da die Roulette-Rad-Selektion bei nahezu ähnlichen Fitneßwerten sonst zu zufälliger Auswahl der Individuen entartet. Nach der Skalierung werden die Selektionswahrscheinlichkeiten nach der Formel

  $$p_{i_x i_y} = \frac{f_{i_x i_y}}{\sum_{j_y=1}^{\mu} f_{j_y}} \quad 1 \leq i_x \leq n_x \quad \text{und} \quad 1 \leq i_y \leq n_y$$

  bestimmt. Die Bestimmung der Selektionswahrscheinlichkeiten wird in $\Theta(n_y)$ durchgeführt. Schließlich muß noch entsprechend der Selektionswahrscheinlichkeiten die neue Plazierung der Individuen ermittelt werden. Dies geschieht in $\Theta(n_y^2)$.

  Das lokale Roulette-Rad-Selektionsverfahren sollte nur dann angewandt werden, wenn sich eine größere Zahl von Individuen auf den PEs befinden, d.h. wenn sich Subpopulationen auf den PEs befinden.

- Globales Uniformes Ranking mit Kopieren

  Das Uniforme Ranking mit Kopieren ist nach Bäck [Bäck 92] implementiert. Zuerst wird der Rang jedes Individuums ermittelt. Das Individuum mit der höchsten Fitneß

erhält den Rang 1. Individuen mit gleicher Fitneß bekommen ihren Rang entsprechend ihrer Ordnung auf den PEs zugewiesen. Bei gleicher Fitneß hat ein Individuum mit kleinerer PE-Nummer auch einen kleineren Rang. Bei Individuen mit gleicher Fitneß auf demselben PE hat das Individuum mit kleinerem Index den kleineren Rang.

Nachdem der Rang eines jeden Individuums bestimmt worden ist, wird den $n_y\mu$ besten Individuen der Platz zugewiesen, an dem sie sich gerade befinden. Die restlichen Individuen werden zufällig aus den $n_y\mu$ besten ausgewählt. Dies geschieht in $\Theta(n_y^2 n_x)$.

Das Uniforme Ranking ist vor allem dazu geeignet, das Laufzeitverhalten des genetischen Algorithmus zu testen. Mit Uniformem Ranking kann über den Parameter μ genau angegeben werden, wieviele Individuen in die nächste Generation herüberkopiert werden müssen. Da bei den globalen Selektionsverfahren der Bus eingesetzt wird, um die neue Generation zu erzeugen, ist dies eine der teuersten Operationen des genetischen Algorithmus. Mit dem Uniformen-Ranking-Selektionsverfahren kann also genau getestet werden, wie teuer es ist $x\%$ der Individuen je Generation auszuwählen.

- Globales Winston Ranking

  Dieses als "Rank Method" in [Winston 92] beschriebene Verfahren ist eine Mischung aus Ranking-Verfahren und Roulette-Rad. Es weist jedem Individuum eine Selektionswahrscheinlichkeit entsprechend seines Rangs zu. Dem Individuum mit Rang 1 wird eine konstante Selektionswahrscheinlichkeit p_c zugewiesen. Individuen mit Rang i erhalten die Selektionswahrscheinlichkeit $p_i = \left(1 - \sum_{j=1}^{i} p_j\right) \cdot p_c$ mit $i < n_x n_y$. Das letzte Individuum $n = n_x n_y$ bekommt schließlich die Selektionswahrscheinlichkeit $p_n = 1 - \sum_{j=1}^{n-1} p_j$. Die Selektionswahrscheinlichkeiten werden in $\Theta(n_y^2 n_x)$ bestimmt.

  Entsprechend der Selektionswahrscheinlichkeiten wird wie bei der Roulette-Rad-Selektion ermittelt, wie die Individuen verteilt werden. Dazu werden nochmals $\Theta(n_y^2 n_x)$ Schritte benötigt.

- Globales Roulette-Rad

  Wie beim lokalen Roulette-Rad werden wieder zuerst das Individuum mit maximaler, das Individuum mit minimaler Fitneß und die Summe der Fitneßwerte auf jedem PE bestimmt. Anschließend werden die Fitneßwerte auf positive Werte transformiert und mit dem in [Goldberg 89] beschriebenen Verfahren skaliert. Nach der Skalierung werden die Selektionswahrscheinlichkeiten nach der Formel

  $$p_{i_x i_y} = \frac{f_{i_x i_y}}{\sum_{j_x=1}^{\mu} \sum_{j_y=1}^{n_y} f_{j_x j_y}} \quad 1 \le i_x \le n_x \quad \text{und} \quad 1 \le i_y \le n_y$$

  bestimmt. Die Bestimmung der Selektionswahrscheinlichkeiten wird in $\Theta(n_y + n_x)$ durchgeführt. Schließlich muß noch entsprechend der Selektionswahrscheinlichkeiten die neue Plazierung der Individuen ermittelt werden. Dies geschieht in $\Theta(n_y^2 n_x)$.

- Tournament

  Siegmund beschreibt in [Siegmund 92] die Tournament-Selektion. Beim Tournament-Selektionsverfahren wird zuerst entsprechend der eingestellten Topologie für jedes Individuum bestimmt, wo sich seine vier Nachbarindividuen befinden. Aus den Nachbarindividuen wird dann das Individuum mit höchster Fitneß als neues Individuum für die nächste Generation ausgewählt. Tournament-Selektion läuft in Zeit $\theta(n_y^2 n_x)$.

Da zur Kommunikation kein Gitter zur Verfügung steht, ist eine Implementierung in $\Theta(w)$ (dabei gibt w die Anzahl der Nachbarn an) nicht möglich. Trotzdem wurde dieses Verfahren implementiert. Denn der Vergleich der Komplexität mit den anderen Selektionsverfahren, die für CNGA implementiert wurden, zeigt, daß es nicht langsamer ist. Außerdem kann es zu einem guten Konvergenzverhalten führen, da durch eine räumliche Entfernung der Individuen eine Nischenbildung möglich ist.

- Random-Walk

  Die Funktionsweise des Random-Walk ist in Siegmund [Siegmund 92] zusammengefaßt. Auch das Random-Walk-Selektionsverfahren ist wegen eines nicht vorhandenen Gitters nicht effizient zu implementieren. Dennoch wurde das Random-Walk-Selektionsverfahren implementiert, da es ebenfalls Nischenbildung ermöglicht. Da kein Gitter vorhanden ist, kann die notwendige Einschränkung, von allen Individuen aus den gleichen relativen Weg zu wählen, fallengelassen werden.

  Somit wird für jedes Individuum ein Weg entsprechend der eingestellten Topologie bestimmt. Schließlich wird das Individuum aus den auf diesem Weg besuchten Individuen ausgewählt, das die höchste Fitneß besitzt. Random-Walk-Selektion läuft in Zeit $\theta(wn_y^2 n_x)$, wobei w die Länge des Wegs ist.

- Globale deterministische Auswahl

  Nachdem die Selektionswahrscheinlichkeiten $p_{i_x i_y}$ wie beim globalen Roulette-Rad bestimmt sind, wird berechnet, wieviele Nachkommen k jedes Individuum zu erwarten hat. $k_{i_x i_y} = \lfloor p_{i_x i_y} n \rfloor$. Jetzt sind noch $n_r = n - \sum_{i_x=1}^{n_x} \sum_{i_y=1}^{n_y} k_{i_x i_y}$ Nachkommen zu vergeben. Dazu wird der Rang der Wahrscheinlichkeiten $p'_{i_x i_y} = p_{i_x i_y} n - \lfloor p_{i_x i_y} n \rfloor$ bestimmt, und die n_r Individuen mit den größten Wahrscheinlichkeiten $p'_{i_x i_y}$ werden ein weiteres Mal kopiert. Anschließend wird jedes Individuum auf $k_{i_x i_y}$-Plätze verteilt. Dabei werden die Individuen in zwei Schritten verteilt. Zuerst werden die Individuen auf PEs mit gerader PE-Nummer verteilt und dann auf PEs mit ungerader PE-Nummer. Durch dieses Verfahren soll vermieden werden, daß zwei gleiche Individuen mit gleichem Index auf benachbarte PEs plaziert werden. Wäre dies der Fall, dann könnte die Externe-Nachbar Partnerwahl nicht eingesetzt werden. Ebenso soll vermieden werden, daß zwei Individuen auf benachbarte Plätze innerhalb eines PEs gesetzt werden. Interne-Nachbar Partnerwahl wäre dann nicht einsetzbar. Die Zeitkomplexität ist $\Theta(n_y^2 n_x)$.

4.14 Erzeugen der nächsten Generation

Um die nächste Generation für den genetischen Algorithmus zu erzeugen, werden nun die Individuen auf die zuvor durch das Selektionsverfahren ausgewählten Plätze kopiert. Es können zwei Fälle unterschieden werden:

- Globale Verteilung

  Bei der globalen Selektion kann es vorkommen, daß jedes Individuum einen beliebigen Platz zugewiesen bekommen kann. Also muß es für jeden Platz möglich sein, ein beliebiges Individuum über die 1D-Topologie oder über den Bus zu holen. Befindet sich nur ein Individuum auf einem PE, dann muß die globale Verteilung verwendet werden.

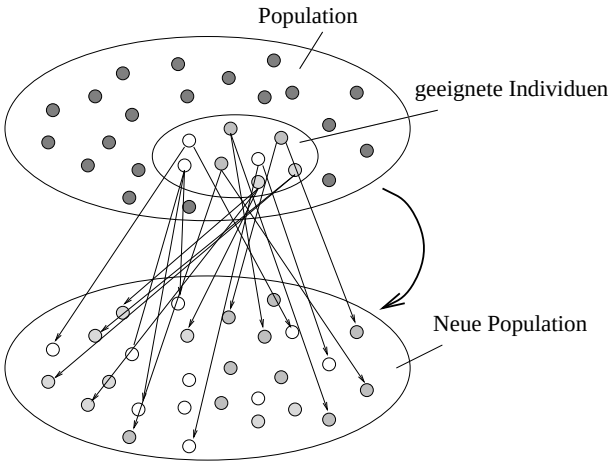

Abbildung 4.9: Selektion guter Individuen

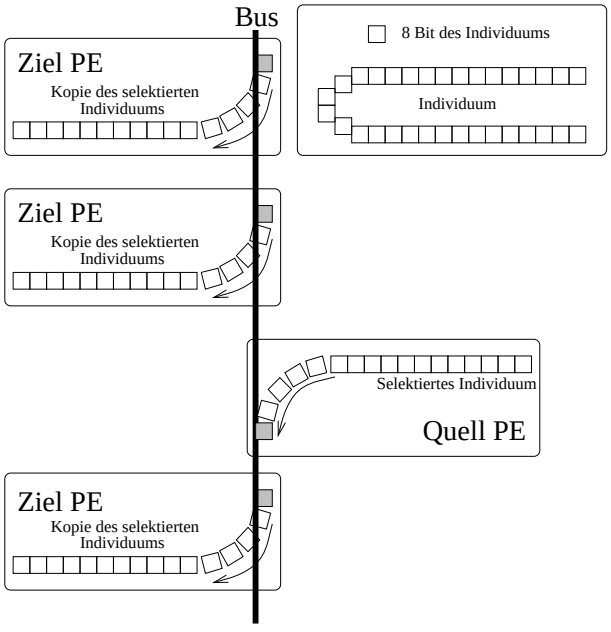

Abbildung 4.10: Globale Selektion

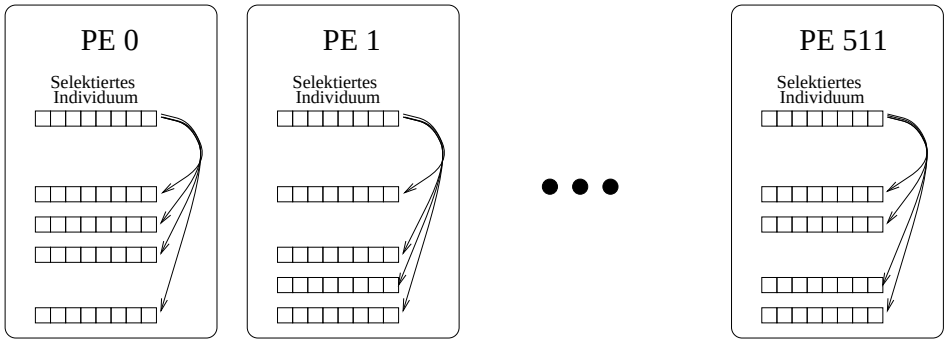

Abbildung 4.11: Lokale Selektion

Es stellt sich die Frage, ob der Bus oder die 1D-Kommunikationstopologie besser geeignet ist, um die Individuen zu verteilen. Beim Neurocomputer CNAPS können über die 1D-Topologie nur 2 Bit in eine Richtung übertragen werden. Der Bus dagegen ist 8 Bit breit. Außerdem ist zu erwarten, daß einige besonders gute Individuen mehrere Nachkommen und besonders schlechte Individuen gar keine Nachkommen erwarten. Es werden also einige wenige Individuen auf viele PEs verteilt (Abbildung 4.9). Aus beiden genannten Gründen wurde in der vorliegenden Arbeit der Bus als Kommunikationsstruktur zur Verteilung der Individuen eingesetzt (Abbildung 4.10). Im folgenden Programmfragment gibt `pe[i]` an, von welchem PE und `ind[i]` von welchem Index innerhalb des PEs der Platz i ein Individuum erhält. Die Zeitkomplexität ist $\Theta(n_y^2 n_x)$.

```
FOR i := 1 TO n_y DO
  FOR x := 1 TO n_x DO
    FOR y := 1 TO n_y DO
      IF x=pe[i] AND y=ind[i] THEN
        Kopiere Individuum (x,y) auf Platz (i)
      END
    END
  END
END
```

- Lokale Verteilung

  Bei der lokalen Verteilung werden die Individuen nur innerhalb desselben PEs kopiert. Dies setzt jedoch voraus, daß ein lokales Selektionsverfahren und mindestens zwei Individuen je PE verwendet werden. Bei der lokalen Verteilung wird keine zusätzliche Kommunikation benötigt (Abbildung 4.11). Die Zeitkomplexität ist $\Theta(n_y)$.

```
FOR i := 1 TO n_y DO
  Kopiere Individuum mit Index (ind[i]) auf Platz (i)
END
```

4.15 Statistiken

Das Programm CNGA berechnet die Statistikdaten wie das von Hummler entwickelte Programm MPGA [Hummler 95]. Die Konvergenzstatistiken werden, wie in [Hummler 95] beschrieben, berechnet.

- Generation
- Maximale Fitneß
- Durchschnittliche Fitneß
- Minimale Fitneß
- Bester erreicher Fitneßwert aller Generationen.
- Generation, in der der bisher beste Fitneßwert aufgetreten ist.
- Diversität D

  Die Diversität D gibt an, wie stark sich die einzelnen Bits voneinander unterscheiden.

  $$D = \frac{2}{16l} \sum_{i=1}^{16l} 2x_i(1 - x_i)$$

  Dabei ist l die Länge des Genotyps in Worten und x_i ist der Prozentsatz von Individuen, deren Bit an Position i gesetzt ist.

- Konvergenz K

  Die Konvergenz K gibt an, wie viele Bits der Genotypen bereits als konvergiert gelten.

  $$K = \frac{1}{16l} \sum_{i=1}^{16l} k_i \quad \text{mit} \quad k_i = \begin{cases} 1 & \text{falls } x_i > c \text{ oder } 1 - x_i > c \\ 0 & \text{sonst} \end{cases}$$

  Der Prozentsatz, ab dem ein Bit als konvergiert gilt, wird durch c bestimmt. Also ist $k_i = 1$, wenn das Bit an Position i als konvergiert gilt und $k_i = 0$ sonst.

- Inzucht I

  Die Inzucht I gibt an, wie unterschiedlich die Eltern sind, die zur Erzeugung der nächsten Population ausgewählt wurden.

  $$I = 1 - H \quad \text{mit} \quad H = \frac{1}{n_e 16l} \sum_{i=1}^{n_e} d_i$$

  n_e gibt die Anzahl der Eltern und l die Länge des Genotyps an. d_i ist der Hamming-Abstand der Bitstrings der Eltern.

- Inzucht-Koeffizient F

  Der Inzucht-Koeffizient mißt, wie gut die genetische Vielfalt beim Crossover ausgenutzt wird.

  $$F = 1 - \frac{H}{D}$$

- Differenz der maximalen Fitneß zur vermuteten maximalen Fitneß

Kapitel 5

Von CNGA erzeugte Dateien

Das Programm CNGA erzeugt eine Vielzahl von Dateien. Die Bedeutung dieser Dateien soll im folgenden genau beschrieben werden. Dabei entsprechen die Sammel-Datei, die Protokoll-Datei und die Individuen-Datei dem von Baumann [Baumann 95] und Hummler [Hummler 95] in Absprache mit dem Betreuer Zell entwickelten Format.

5.1 Datei mit Einstellungen des Menüs

Dateiendung `.GAset`

In der Einstellungs-Datei werden alle Einstellungen des Menüs gespeichert. Dies dient vor allem dazu, die Arbeit mit dem CNGA-Programm zu erleichtern. Beim Start des CNGA-Programms wird die Datei mit dem Namen `cnga_setf.GAset` geladen. So läßt sich das Programm für jeden Benutzer persönlich konfigurieren. Zudem läßt sich durch den Parameter `settings_file` einstellen, unter welchem Namen die Einstellungsdaten beim Kommando `save` bzw. `load` gespeichert bzw. geladen werden. Dies trägt auch dazu bei, daß die Daten erfolgreicher Läufe gespeichert werden können und so reproduzierbar sind.

Das Datei-Format der Einstellungs-Datei muß mit dem Text "Menu Settings File" (durch ein `<RETURN>` abgeschlossen) beginnen. Anschließend können beliebig viele Kommentarzeilen folgen, die mit dem '`#`'-Zeichen beginnen. Dann folgen in beliebiger Reihenfolge die Belegung der Parameter. In jeder Zeile wird eine Parameterbelegung angegeben. Jede Zeile hat das Format "<Index> : <Wert> : <Kommentar>". <Index> ist der interne Index des Parameters in der Menü-Datenstruktur, <Wert> ist der Wert, auf den der Parameter gesetzt wird und <Kommentar> der Name des Parameters. Der Kommentar dient nur der besseren Lesbarkeit und wird intern nicht verwendet.

```
Menu Settings File
6 : cnaps : server_name
7 : 0 : pipe_results
10 : 512 : number_of_pes_used
11 : 1 : num_individuals/pe
12 : 100 : chromosome_length_in_words
13 : 10 : application
14 : 0 : gray_code
17 : 1 : search_direction
...
```

5.2 Zufallszahlen-Datei

Datei-Endung `.GAseed`

Der genetische Algorithmus liest zu Beginn eines Laufs die Saat für den Zufallszahlengenerator für jedes PE des CNAPS-Servers ein. Jedes Saatelement besteht aus 16 Bits. Diese Saatelemente werden in einer Datei binär gespeichert. Folglich hat die Datei doppelt soviele Bytes wie PEs auf dem CNAPS-Server existieren. Diese Datei wird vom Host bei jedem Lauf auf den CNAPS-Server kopiert. Sie kann mit dem Kommando `generate_seeds` erzeugt werden und mit dem Kommando `display_seeds` angesehen werden. Dabei wird der verwendete Dateiname mit dem Parameter `seed_file` angegeben.

5.3 Sammel-Datei

Datei-Endung `.GAsum`

In der Sammel-Datei werden die wesentlichen Informationen über einen Lauf des genetischen Algorithmus mit dem Kommando `print_statistic` eingetragen. Der Name der Sammel-Datei wird mit dem Parameter `summary_data_file` angegeben. Ein Beispiel einer Sammel-Datei ist in Abbildung 5.1 angegeben.

5.4 Protokoll-Datei

Datei-Endung `.GApro`

In der Protokoll-Datei werden die ermittelten Statistikdaten über den Verlauf des genetischen Algorithmus automatisch festgehalten. Der Name der Protokoll-Datei wird mit dem Parameter `protocol_data_file` angegeben. Eine Protokoll-Datei kann, wie in Abbildung 5.2 angegeben, aussehen.

5.5 Individuen-Datei

Datei-Endung `.GAind`

In der Individuen-Datei werden die Informationen der Individuen ebenfalls automatisch festgehalten. Der Name der Protokoll-Datei wird mit dem Parameter `protocol_data_file` angegeben. Die Individuen-Datei kann wie in Abbildung 5.3 aussehen.

5.6 Gnuplot-Fitneß-Statistikdaten

Für die Visualisierung der Fitneßstatistiken werden drei Dateien mit dem Namen der Protokoll-Datei erzeugt. Lediglich die Endungen unterscheiden sich wie folgt:

- Fitneß-Maximum

  Datei-Endung `.max.GP2D` für die jeweils größte Fitneß.

- Fitneß-Durchschnitt

  Datei-Endung `.avg.GP2D` für die durchschnittliche Fitneß.

```
Genetic Algorithms Summary Protocol File Version 1.0

PARAMETERS                          46

NUMBER_OF_PES                        1
NUMBER_OF_INDIVIDUALS                2
CHROMOSOME_LENGTH                    3
APPLICATION                          4
GRAY_CODING                          5
SEARCH_DIRECTION                     6
REPLACEMENT_METHOD                   7
SELECTION_METHOD                     8
PARTNER_SELECTION                    9
EXCHANGE_METHOD                     10
NUMBER_OF_EXCHANGES                 11
TOPOLOGY                            12
WIDTH_OF_TOPOLOGY                   13
WALK_LENGTH                         14
SELECTION_PARAMETER_MU              15
SCALING_PARAMETER                   16
WINSTON_PROBABILITY                 17
CROSSOVER_POINTS                    18
CROSSOVER_INSIDE_WORDS              19
MUTATION_METHOD                     20
CROSSOVER_RATE                      21
MUTATION_RATE                       22
GENERATION                          23
SUPPOSED_MAX_FITNESS                24
BIT_CONV_LIMIT                      25
DIVERSITY_LIMIT                     26
ABORT_DIFFERENCE                    27
ABORT_CONVERGENCE                   28
ABORT_DIVERGENCE                    29
MAX_FITNESS                         30
AVG_FITNESS                         31
MIN_FITNESS                         32
BEST_FITNESS                        33
FIRST_BEST_GENERATION               34
DIVERSITY                           35
BIT_CONVERGENCE                     36
INBREEDING                          37
INBREEDING_COEFFICIENT              38
DIFF_MAX_FITNESS                    39
SOURCE                              40
SERVER                              41
PIPE_RESULTS                        42
SEED_FILE                           43
PROTOCOL_FILE                       44
RUN_TIME                            45
DATE                                46

01----- 02----- 03----- 04---------------------------- 05----- 06---------------------------- 07------
---------------------- 08---------------------------- 09---------------------------- 10--------------
-------------- 11----- 12---------------------------- 13----- 14----- 15----- 16----- 17----- 18-----
19----- 20---------------------------- 21----- 22----- 23----- 24------------- 25----- 26----- 27-----
28----- 29----- 30------------- 31------------- 32------------- 33------------- 34----- 35----- 36-----
37----- 38----- 39------------- 40----- 41---------------------------- 42----- 43---------------------
------- 44---------------------------- 45----- 46----------------------
512     1       3       f1                          OFF     negation                        children
                        tournament                  external_random_partner         *shift_best_wors
t               0       rectangular                 32      4       128     2.00000 0.20000 1
ON      accumulated_mutation_on_bits    0.99999 0.01000 20      0.000000        0.00000 0.00000 ON
OFF     OFF     0.000000        -0.000536       -0.000891       0.000000        13      0.03476 0.00000
0.89965 0.00000 0.000000        1.0     cnaps                           OFF     cnga_rand
        cnga_prof.f1                    0.00200 Mon Oct  9 03:12:59 1995
```

Abbildung 5.1: Sammel-Datei

```
Genetic Algorithms Session Protocol File Version 1.0

server_name                         cnaps
pipe_results                        OFF
number_of_pes_used                  512
num_individuals/pe                  1
chromosome_length_in_words          3
application                         f1
gray_code                           OFF
search_direction                    negation
replacement_method                  children
selection_method                    tournament
partner_selection                   external_random_partner
exchange_method                     *shift_best_worst
number_of_exchanges                 0
topology                            rectangular
width_of_topology                   32
crossover_points                    1
crossover_between_words             ON
mutation_method                     accumulated_mutation_on_bits
crossover_rate                      0.999990
mutation_rate                       0.010000
verbose                             OFF
report                              normal
protocol_interval                   1
generations                         20
supposed_max_fitness                0.000000
bit_conv_limit                      0.000000
diversity                           0.000000
abort_criteria_maximum              ON
abort_criteria_convergence          OFF
abort_criteria_diversity            OFF
path_of_executable                  /home/ebner/SA/cnga/bin/cnaps/
settings_file                       cnga_setf.f1
seed_file                           cnga_rand
summary_data_file                   cnga_sumf
protocol_data_file                  cnga_prof.f1

PARAMETERS                           11

GENERATION                            1
MAX_FITNESS                           2
AVG_FITNESS                           3
MIN_FITNESS                           4
BEST_FITNESS                          5
BEST_GENERATION                       6
DIVERSITY                             7
BIT_CONVERGENCE                       8
INBREEDING                            9
INBREEDING_COEFFICIENT               10
DIFF_MAX_FITNESS                     11

01----- 02------------ 03------------ 04------------ 05------------ 06-----
07----- 08----- 09----- 10----- 11-------------
1       -0.729675       -12.181152      -38.199219      -0.729675       1
0.06074 0.00000 0.75073 0.00000 0.729675
2       -0.262283       -4.374512       -16.622070      -0.262283       2
0.05943 0.00000 0.75122 0.00000 0.262329
3       -0.202148       -1.371826       -9.279785       -0.202148       3
0.05793 0.00000 0.75537 0.00000 0.202148
```

Abbildung 5.2: Protokoll-Datei

```
Genetic Algorithms Session Individual File Version 1.0

PARAMETERS                          6

PE_NUMBER                           1
INDIVIDUAL_NUMBER_ON_PE             2
AGE                                 3
FITNESS                             4
CHROMOSOME                          5
PHENOTYPE                           6

01----- 02----- 03----- 04------------ 05--------------------------------
-------------------- 06----------------------------
329     0       0       -0.729675       1110111111101110|1111110001101011|
0000110110011010      (-0.642813,-0.143281,0.544063)
388     0       0       -0.262283       1111100101101111|0000100011000010|
0000011010100101      (-0.262656,0.350313,0.265781)
258     0       0       -0.202148       0000000001100100|0000001011110001|
1111010100101000      (0.015625,0.117656,-0.433750)
429     0       0       -0.059193       0000000001100100|0000001010011010|
0000010101111101      (0.015625,0.104063,0.219531)
412     0       0       -0.010381       0000000001101000|1111110110010011|
1111111101010100      (0.016250,-0.097031,-0.026875)
485     0       0       -0.010686       1111111101101000|1111110110010011|
1111111101010100      (-0.023750,-0.097031,-0.026875)
```

Abbildung 5.3: Individuen-Datei

- Fitneß-Minimum

  Datei-Endung `.min.GP2D` für die kleinste Fitneß.

In jeder Zeile der Dateien stehen zwei Werte "<Generation> <Wert>", wobei <Generation> für die Nummer der Generation steht, in der <Wert> ermittelt wurde. Die Daten sind im ASCII-Format und können mit Gnuplot über den Befehl `plot` angezeigt werden.

5.7 Gnuplot-Konvergenz-Statistikdaten

Für die Visualisierung der Konvergenzstatistiken werden die folgenden vier Dateien mit dem Namen der Protokoll-Datei und den folgenden Endungen erzeugt.

- Konvergenz

  Datei-Endung `.con.GP2D` für Konvergenzdaten.

- Diversität

  Datei-Endung `.div.GP2D` für die Diversitätsdaten.

- Inzucht

  Datei-Endung `.inb.GP2D` für die Inzuchtdaten.

- Inzucht-Koeffizient

  Datei-Endung `.ico.GP2D` für die Inzuchtkoeffizientdaten.

Wie bei den Fitneßdaten hat auch hier jede Zeile der Dateien das Format "<Generation> <Wert>", wobei <Generation> für die Nummer der Generation steht, in der <Wert> ermittelt wurde. Die Daten sind ebenfalls im ASCII-Format und können mit Gnuplot über den Befehl `plot` angezeigt werden.

5.8 Gnuplot-Populations-Datei

Datei-Endung `.Gx.pop.GP3D`

Ist die Populations-Berichtsart ausgwählt, dann werden anstatt der normalen Statistikdaten die Fitneßwerte der gesamten Population berichtet. Diese Daten können entsprechend der eingestellten Topologie dreidimensional mit Gnuplot visualisiert werden. Daher werden die Daten in eine Datei mit dem Namen der Protokoll-Datei und der Endung `.Gx.pop.GP3D` geschrieben. Dabei steht `x` für die Nummer der Generation, in der die Daten berechnet wurden. Die Populations-Daten sind für die dreidimensionale Darstellung mit dem Gnuplot-Befehl `splot` geeignet. Eine dreidimensionale Visualisierung der Fitneß für eine Rechtecktopologie ist in Abbildung 5.4 zu sehen.

5.9 Ausgabe-Datei

Datei-Endung `.GAout`

Die Ausgabe-Datei wird von CNGA erstellt, wenn der Parameter `pipe_results` auf `OFF` steht. Die Datei wird zunächst während eines Laufs des CNGA auf dem Server erstellt und

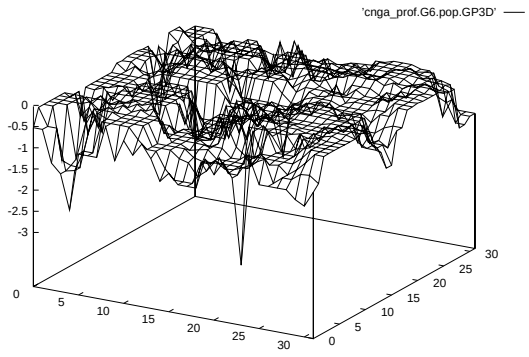

Abbildung 5.4: Visualisierung der Fitneß für eine Rechtecktopologie

nach Beendigung des Laufs auf den Host kopiert. Steht der Parameter `pipe_results` auf `ON`, dann wird zwischen Server und Host eine Pipe aufgebaut, durch die die Daten auf den Host transportiert werden. Dies hat vor allem den Vorteil, daß der Anwender sofort Informationen über den Verlauf des genetischen Algorithmus erhält. Eine Ausgabe-Datei wird in diesem Fall nicht erzeugt. Die Informationen, die durch die Pipe den Host erreichen, haben dasselbe Format, wie die Daten der Ausgabe-Datei.

5.10 Debug-Datei

Datei-Endung `.GAdeb`

Die Debug-Datei wird von der Berichtart "Debug" erzeugt. In der Debug-Datei wird neben den Informationen, die bei der normalen Berichtart erzeugt werden, auch die Belegung nur intern verwendeter Variablen mitgeteilt. Somit ist ein Debuggen des Programms vereinfacht. Die Informationen werden als ASCII-Text gespeichert. Die Berichtart "Debug" sollte jedoch nur mit kleinen Populationen erzeugt werden, da die erzeugte Datenmenge sehr schnell sehr groß wird.

5.11 Fehler-Datei

Datei-Endung `.GAerr`

Die Fehler-Datei wird während eines Laufs des CNGA auf dem Server erzeugt. Sie enthält alle während des Laufs aufgetretenen Fehlermeldungen im ASCII-Format. Nach Beendigung des CNGA wird diese Datei auf den Server kopiert und, falls Fehler aufgetreten sind, angezeigt.

Kapitel 6

Untersuchungen

In den folgenden Tabellen, die die Laufzeiten enthalten, ist die Anzahl der Individuen in der Form n_xxn_y angegeben. Dabei beschreibt n_x die Zahl der PEs und n_y die Zahl der Individuen je PE. Zeit/Individuum ist die Laufzeit pro Generation geteilt durch die Anzahl aller Individuen $(n_x n_y)$.

6.1 Vergleich der Selektionsverfahren

Um die Selektionsverfahren zu vergleichen, wurde eine Reihe von Tests mit der Zielfunktion f_1 für drei Dimensionen durchgeführt. Eine graphische Darstellung der Zielfunktion f_1 ist in Abbildung 6.1 zu sehen. Bei der Zielfunktion f_1 handelt es sich um eine relativ einfach zu berechnende Zielfunktion. Es wurde eine Codierung mit 48 Bit gewählt, wobei jedes Element mit 16 Bit codiert wurde. Somit ist die Zeit für die Evaluierung der Zielfunktion und für das Kopieren der Individuen im Vergleich zur gesamten Laufzeit des genetischen Algorithmus gering. Aufgrund der unterschiedlichen Anzahl von Generationen bis zur Konvergenz der Algorithmen werden im Folgenden die Laufzeiten je Generation miteinander verglichen.

Die Tests wurden bei Erreichen des Minimums abgebrochen, jedoch spätestens nach 100 Generationen. Ein Nachkomme wird nur in die Population integriert, wenn es einen größeren Fitneßwert als sein Elter besitzt. Es wurde eine Crossover-Wahrscheinlichkeit von $p_{\text{cross}} = 0.99998$ und eine Mutations-Wahrscheinlichkeit von $p_{\text{mut}} = 0.005$ verwendet. Als Partnerwahl wurde die Externe-Nachbar-Partnerwahl eingestellt.

f_1	Keine Selektion	Uniformes Ranking	Tournament	Random-Walk
Generationen	100	100	19	15
Individuen	512x1	512x1	512x1	512x1
Laufzeit [s]	0,295	2,937	0,941	0,768
Zeit/Generation [ms]	2,95	29,4	49,5	51,2
Zeit/Individuum [μs]	5,76	57,4	96,7	100

Tabelle 6.1: Laufzeitdaten für die Zielfunktion f_1 bei Selektionsverfahren, die auf Vergleichen der Fitneßwerte basieren.

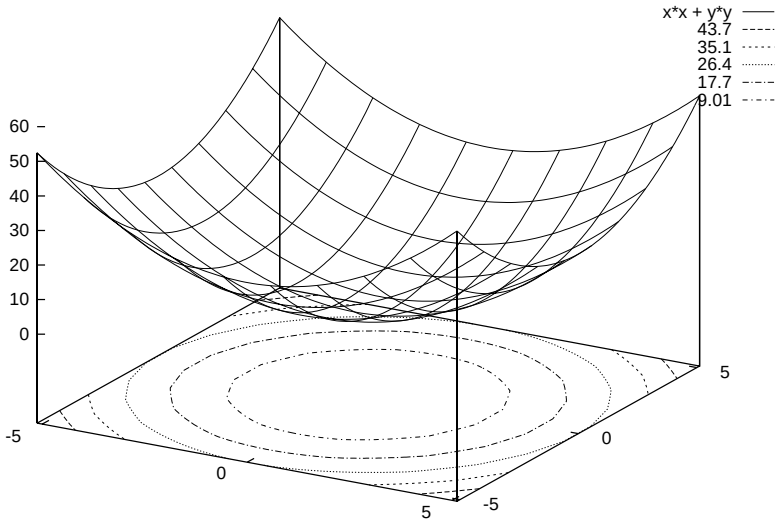

Abbildung 6.1: Zweidimensionale Zielfunktion f_1

Die Selektionsverfahren können in zwei Klassen eingeteilt werden. Eine Klasse besteht aus Selektionsverfahren, die nur auf Vergleichen der Fitneßwerte basieren. Die andere Klasse besteht aus Selektionsverfahren, bei denen zunächst eine Selektionswahrscheinlichkeit berechnet, und anschließend die Individuen entsprechend dieser Wahrscheinlichkeiten verteilt werden.

Zu den Selektionsverfahren, die auf Vergleichen basieren, wurde noch die Laufzeit des genetischen Algorithmus hinzugefügt, wenn keinerlei Selektion vorgenommen wird. Die Individuen verbleiben in diesem Fall auf ihrem PE. Konvergenz ist nur aufgrund der elitären Integrations-Strategie möglich. Ohne Selektion benötigt der genetische Algorithmus 0,295s für 100 Generationen, Danach wurde der Algorithmus, ohne das Minimum zu erreichen, abgebrochen. Als keinster Wert der Zielfunktion wurde 0.001030 erreicht.

Uniformes Ranking ist etwa um den Faktor 10 langsamer als keine Selektion. Es wurden 64 Individuen auf die neue Population verteilt ($\mu = 64$). Auch beim uniformen Ranking wurde das Minimum nicht innerhalb von 100 Generationen gefunden. Dabei wurde der kleinste Wert der Zielfunktion 0.000777 bereits in Generation 7 erreicht.

Die Laufzeit bei Tournament-Selektion und Random-Walk-Selektion bei einer 16x32 Rechtecktopologie liegen sehr nahe beieinander. Als Weglänge wurde 4 eingestellt. Die Tournament-Selektion berücksichtigt nur die nächsten Nachbarn in horizonaler und vertikaler Richtung. Daher ist zu erwarten, daß bei der Tournament-Selektion etwa gleichviel Individuen wie bei der Random-Walk-Selektion zu verteilen sind. Dies zeigt sich in Tabelle 6.1. Die Laufzeit pro Generation der Tournament-Selektion und der Random-Walk-Selektion ist jedoch deutlich langsamer als die uniforme Selektion. Im Gegensatz zum uniformen Ranking ist die Zahl der zu verteilenden Individuen bei der Tournament-Selektion und bei der Random-Walk-Selektion variabel.

In Tabelle 6.2 werden die Laufzeitdaten bei Selektionsverfahren, die auf Wahrscheinlich-

f_1	Winston Ranking	Roulette-Rad	Det. Auswahl	Lokales Roulette
Generationen	10	16	20	15
Individuen	512x1	512x1	512x1	512x16
Laufzeit [s]	0,556	1,201	1,759	0.541
Zeit/Generation [ms]	55,6	75,1	88,0	36,1
Zeit/Individuum [μs]	109	147	172	4,40

Tabelle 6.2: Laufzeitdaten für die Zielfunktion f_1 bei Selektionsverfahren, die auf Wahrscheinlichkeiten basieren.

f_{12}	Internal Adjacent Partner	Internal Random Partner	External Adjacent Partner	External Random Partner
Generationen	100	100	100	100
Individuen	512x2	512x2	512x2	512x2
Laufzeit [s]	1,688	1,825	2,216	124,7
Zeit/Generation [ms]	16,9	18,3	22,2	1247
Zeit/Individuum [μs]	16,5	17,8	21,6	1218

Tabelle 6.3: Laufzeitdaten für die Zielfunktion f_{12} bei unterschiedlicher Art der Partnerwahl.

keiten basieren einander gegenübergestellt. Die Roulette-Rad-Selektion, die deterministische Auswahl und das von Winston beschriebene Ranking-Selektionsverfahren, zeigen Laufzeiten je Generation, die etwa in der gleichen Größenordnung liegen. Sie liegen im Bereich der Laufzeiten von Tournament-Selektion und Random-Walk-Selektion.

Das lokale Roulette-Selektionsverfahren sticht deutlich aus Tabelle 6.2 heraus. Es werden 16-mal soviel Individuen wie bei den anderen Verfahren auf einem PE plaziert. Um eine lokale Selektionsstrategie anzuwenden, müssen ausreichend Individuen auf jedem PE plaziert werden. Im Falle der deterministischen Auswahl waren dies 16 Individuen. Diese Individuen werden nach der Selektion nur lokal innerhalb des PEs verteilt. Da die Externe-Nachbar-Partnerwahl für das Crossover verwendet wurde, gelangt bei jedem Crossover ein Teil des Genotyps der Nachbar-Individuen auf den eigenen PE. Dies hat zur Folge, daß einerseits ein Austausch des Genotyps erfolgt, und zugleich das Crossover äußerst effektiv ist. Ein Crossover zwischen zwei identischen Individuen ist höchst unwahrscheinlich.

6.2 Vergleich der Art der Partnerwahl und der Crossover-Mechanismen

Zum Vergleich des Einflusses der Partnerwahl und somit der verschiedenen Crossover-Mechanismen wurde die Zielfunktion f_{12} verwendet. Als Genotyp wurde ein Bitstring der Länge 1600 Bits verwendet. Durch die große Länge des Genotyps ist gesichert, daß ausreichend Zeit des genetischen Algorithmus für das Crossover verwendet wird. Um einen Vergleich aller vier Arten

der Partnerwahl zu ermöglichen, wurden auf jedes PE zwei Individuen plaziert. Die Interne-Nachbar-Partnerwahl ist erst ab zwei Individuen je PE einsetzbar, da jeweils zwei benachbarte Individuen für das Crossover zusammengefaßt werden. Die Interne-Zufällige-Partnerwahl ist erst ab zwei Individuen je PE sinnvoll, da der Crossover-Partner innerhalb des PEs zufällig bestimmt wird. Als Crossover-Wahrscheinlichkeit wurde $p_{\text{cross}} = 0.99998$ gesetzt. Also wird für fast jedes Individuum ein Nachkomme erzeugt. Für den Vergleich wurde kein Selektionsverfahren gewählt. Somit bleiben die Individuen auf ihren PEs und werden nicht durch die Selektion verteilt. Die Tests wurden für jeweils 100 Generationen durchgeführt.

Der Vergleich der Crossover-Mechanismen in Tabelle 6.3 zeigt, daß die Interne-Nachbar-Partnerwahl die geringste Laufzeit liefert. Sie kommt durch die Art der Partnerwahl zustande, da bei einem Crossover zwei Nachkommen erzeugt werden, und sich beide Partner auf demselben PE befinden.

Bei der Internen-Zufälligen-Partnerwahl wird bei jedem Crossover zu jedem Individuum ein Partner zufällig aus den Individuen desselben PEs bestimmt. Weil aber nur ein Nachkomme erzeugt wird, benötigt die Interne-Zufällige-Partnerwahl doppelt soviel Zeit für das Crossover. Jedoch resultiert dies in einer geringfügig längeren Gesamtlaufzeit.

Die Externe-Nachbar-Partnerwahl führt ein Crossover zwischen jeweils zwei Individuen derselben Ebene, die sich auf benachbarten PEs befinden, durch. Dabei wird das Individuum des rechten Nachbarn stückweise auf den eigenen PE unter Verwendung des Rings (1D-Kommunikationstopologie zum Ring geschlossen) kopiert und so ein Nachkomme erzeugt. Durch die zusätzliche Kommunikation der PEs ist diese Art der Partnerwahl langsamer als die Interne-Zufällige-Partnerwahl.

Tabelle 6.3 zeigt deutlich, daß die Externe-Zufällige-Partnerwahl mit großem Abstand am meisten Zeit benötigt. Bei dieser Art der Partnerwahl wird ein Crossover zwischen zwei beliebigen Individuen einer Ebene durchgeführt. Auch hier wird nur ein Nachkomme erzeugt. Da sich die Individuen aber auf verschiedenen PEs befinden, ist zusätzlich Zeit für die Kommunikation der PEs nötig. Der Genotyp wird stückweise über den Bus an die PEs verteilt, die diesen Genotyp für ein Crossover benötigen. Wegen $p_{\text{cross}} = 0.99998$ ist aber eine fast vollständige Verteilung der Individuen notwendig, die natürlich viel Zeit beansprucht.

Die Externe-Zufällige-Partnerwahl sollte nur dann eingesetzt werden, wenn der genetische Algorithmus bei den drei anderen Arten der Partnerwahl nicht konvergierte. Dieser Fall könnte eintreten, wenn ein Selektionsverfahren verwendet wird, das häufig identische Individuen einer Ebene auf benachbarte PEs oder identische Individuen auf benachbarte Ebenen innerhalb der PEs plaziert. Denn dann würden die Externe-Nachbar-Partnerwahl bzw. die Interne-Partnerwahl häufig ein Crossover zwischen zwei identischen Individuen durchführen. Bei den bereits implementierten Selektionsverfahren wurde allerdings darauf geachtet, daß die Individuen so plaziert werden, daß zwischen identischen Individuen keine Nachbarschaft für ein Crossover exisiert.

6.3 Vergleich der Kommunikationstopologien

Für den Vergleich der Kommunikationstopologien wurde die Zielfunktion f_{12} ausgewählt. Die Länge des Genotyps wurde auf 1600 Bits gesetzt, damit der für die Kommunikation erforderliche Zeitanteil hervorgehoben wird. In Tabelle 6.4 sind die Laufzeiten der Zielfunktion unter Verwendung der verschiedenen Kommunikationstopologien gegenübergestellt. Der genetische Algorithmus wurde jeweils nach 100 Generationen abgebrochen. Auf jedem PE wurden

f_{12}	Keine Selektion	Uniformes Ranking	Shift-Best-Worst	Cond.-Best-Worst	Left-Right-Best-Worst	Left-Right-Cond.-Best
Ergebnis	697	333	335	451	371	448
Generat.	100	100	100	100	100	100
Individuen	512x2	512x2	512x2	512x2	512x2	512x2
Laufzeit [s]	1,667	25,58	18,61	21,27	19,21	19,28
Zeit/Generation [ms]	16,7	256	186	213	192	193
Zeit/Individuum [μs]	16,3	250	182	208	188	188

Tabelle 6.4: Laufzeitdaten der Zielfunktion f_{12} zum Vergleich der Kommunikationstopologien.

2 Individuen plaziert. Ein Crossover wurde jeweils zwischen den Individuen desselben PEs durchgeführt, wobei zwei Nachkommen erzeugt wurden.

Ohne jegliche Kommunikation benötigt der genetische Algorithmus 1,667s für 100 Generationen. Beim Uniformen-Ranking-Selektionsverfahren werden für 100 Generationen 25,58s benötigt. Das uniforme Ranking verwendet den Bus zur Verteilung der Individuen. Der Parameter μ wurde auf 64 gesetzt. Es wurden also 128 Individuen auf die Population verteilt.

Bei den Austauschverfahren werden die Individuen über den Kommunikationsring zwischen benachbarten PEs ausgetauscht. Um einen Vergleich zum uniformen Ranking zu ermöglichen, wurde der Parameter `number_of_exchanges` auf 128 bei den Austauschverfahren `shift_best_worst` und `shift_conditional_best` gesetzt. Bei diesen Austauschverfahren werden somit ebenfalls 128 Individuen bewegt.

Da bei den Austauschverfahren `left_right_best_worst` und `left_right_conditional_best` jeweils die Individuen nach links und rechts kopiert werden, wurde der Parameter `number_of_exchanges` hier auf den Wert 64 gesetzt. Die Laufzeit des genetischen Algorithmus bei Verwendung der Austauschverfahren liegen alle bei etwa 20s.

Beim uniformen Ranking wird jedoch im Vergleich zu den Austauschverfahren der Rang der Individuen global bestimmt. Bei den Austauschverfahren wird lediglich das beste und das schlechteste Individuum innerhalb der PEs bestimmt. Daher ergibt sich für das uniforme Ranking eine etwas längere Laufzeit.

6.4 Einfluß der Genauigkeit

Um zu testen, welchen Einfluß die Genauigkeit auf Performance des genetischen Algorithmus hat, wurde eine Reihe von Experimenten durchgeführt. Als Zielfunktion wurde die Funktion f_{10} ausgewählt. Bei der Zielfunktion handelt es sich um das Problem des Handlungsreisenden (Traveling-Salesman-Problem, kurz TSP). Die Aufgabe des Handlungsreisenden ist, eine vorgegebene Anzahl von Städten in einer geeigneten Reihenfolge zu besuchen, wobei die Kosten der Reise so klein wie möglich sein sollen. Das TSP-Problem ist NP-vollständig [Hopcroft 90] [Cormen et al. 90], d.h. es sind derzeit nur Algorithmen bekannt, die das Problem mit exponentieller Laufzeit lösen.

Die Zielfunktion des TSP-Problems wurde von Bäck [Bäck 92] wie folgt für 100 Städte

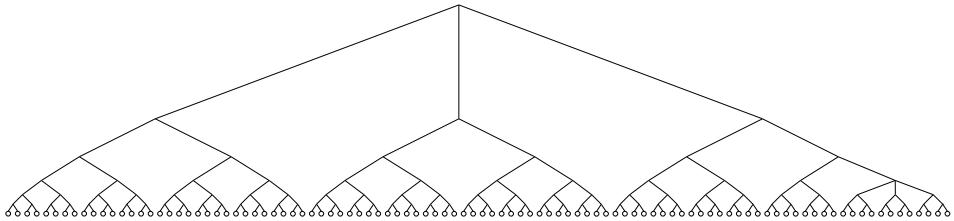

Abbildung 6.2: Addition der Distanzen beim TSP Problem

beschrieben:

$$f_0(\vec{x}) = \sum_{i=1}^{n} d(c_{\pi(i \mod n)}, c_{\pi((i+1) \mod n)})$$

Dabei ist π eine Permutation der Städte. Als Reisekosten wird die Distanz der Städte verwendet, also

$$d(c_i, c_j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}.$$

Da auf dem Neurocomputer die Berechnung der Wurzel aufgrund der Ganzzahlenarithmetik schwierig und zeitintensiv ist, wurde stattdessen folgende Abstandsfunktion verwendet:

$$d(c_i, c_j) = (x_i - x_j)^2 + (y_i - y_j)^2.$$

Als Koordinaten der Städte wurden die von Bäck [Bäck 92] angegebenen verwendet. Dies erlaubt einen Vergleich mit dem genetischen Algorithmus von Bäck [Bäck 92] und den von Baumann [Baumann 95] und Hummler [Hummler 95] entwickelten Programmen, die ebenfalls dieselben Daten verwenden.

Zunächst wurde die Summe der Zielfunktion linear gebildet. Dabei zeigte das Programm CNGA ein äußerst schlechtes Verhalten. Der genetische Algorithmus gelangte schon nach wenigen Generationen in ein lokales Optimum, das er dann nicht mehr verlassen konnte. Dies ist auf die geringe Ganauigkeit der verwendeten Fließkommazahlen zurückzuführen. Bevor zwei Fließkommazahlen z_1 und z_2 addiert werden können, müssen zuerst die beiden Exponenten angeglichen werden. Es sei $z_1 = m_1 2^{e_1}$ und $z_2 = m_1 2^{e_1}$ mit $z_1 \leq z_2$ und beide Zahlen seien normalisiert, d.h. das erste Bit der Mantisse ist eine 1, falls die Mantisse nicht Null ist. Ab einer Differenz der Exponenten von 15 ($e_2 - e_1 \geq 15$) ist $z_1 + z_2 = z_2$, da die Bits der Mantisse z_1 beim Angleichen der Exponenten verlorengehen. Auf dieses Problem der Rechengenauigkeit weisen Appelrath et al. [Appelrath et al. 91] hin. Je mehr Werte (bei etwa gleicher Größenordnung) addiert werden, desto stärker tritt dieser Effekt in Erscheinung.

Um das beschriebene Problem zu Lösen, wurden die einzelnen Werte nicht mehr linear, sondern baumartig addiert, wie in Abbildung 6.2 zu sehen ist. Mit dieser Änderung war der genetische Algorithmus erstmals in der Lage, nach einem geeigneten Individuum zu suchen. Als bestes Ergebnis ergab sich ein Individuum mit der Weglänge 38852. Nun stellte sich die Frage, ob das relativ schlechte Ergebnis im Vergleich zu den Ergebnissen anderer Programme (siehe den Vergleich mit anderen Implementierungen weiter unten) auf die geringe Genauigkeit oder die relativ kleine Anzahl von Individuen zurückzuführen ist.

Ob die Genauigkeit einen Einfluß auf die Performance des Algorithmus hat, wurde mit

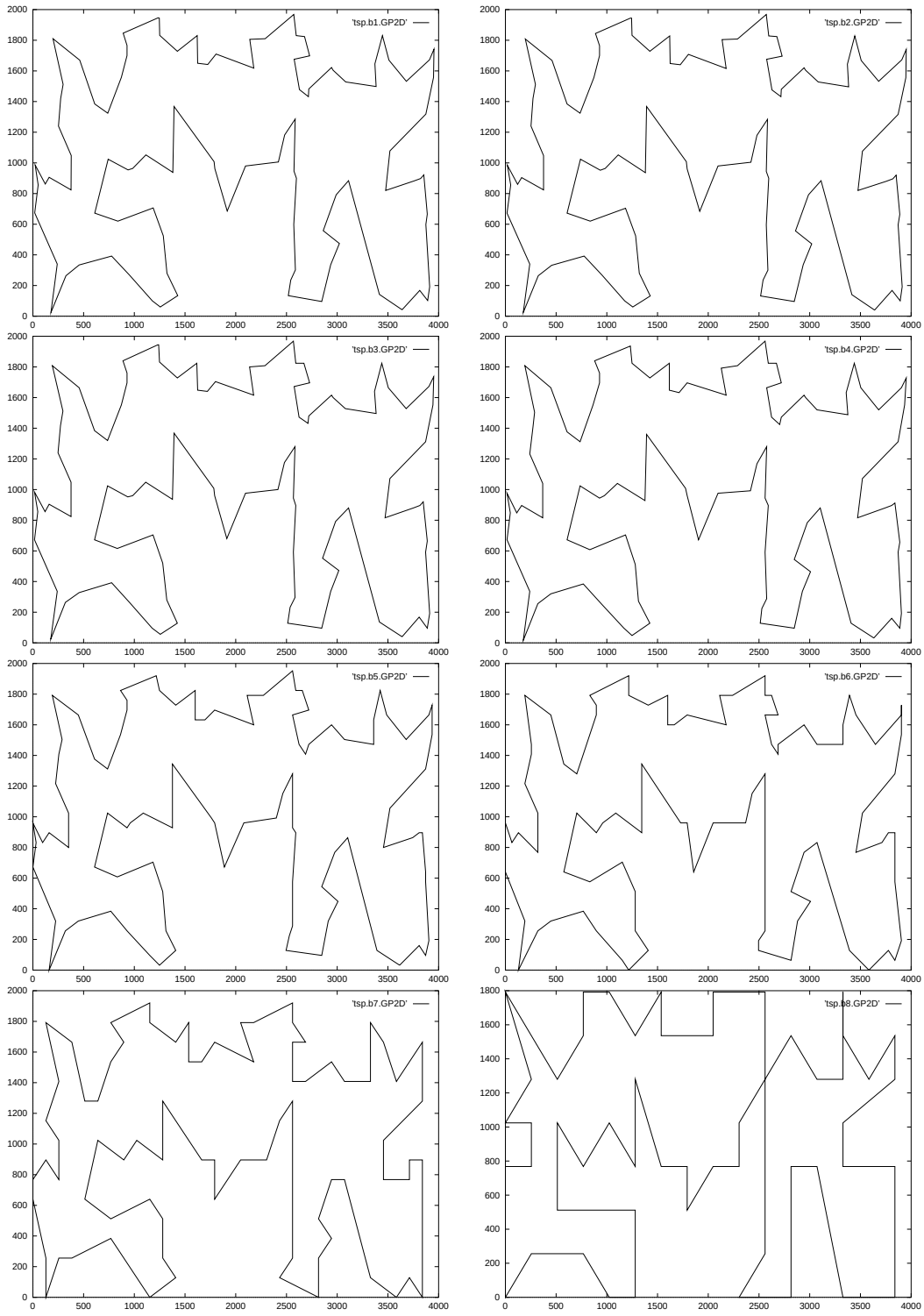

Abbildung 6.3: Vergröberung des TSP Problems

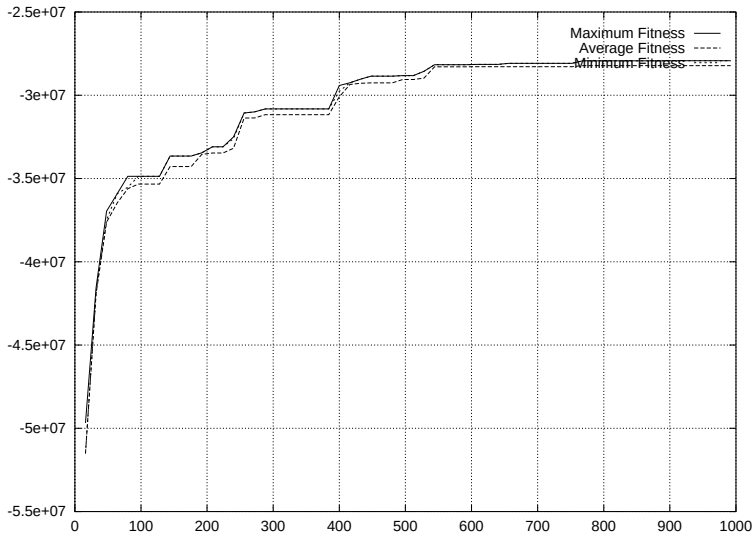

Abbildung 6.4: Veränderung der Fitneß bei variabler Distanzfunktion

einer veränderlichen Distanzfunktion getestet

$$d(c_i, c_j) = (\lfloor \frac{x_i}{2^s} \rfloor 2^s - \lfloor \frac{x_j}{2^s} \rfloor 2^s)^2 + (\lfloor \frac{y_i}{2^s} \rfloor 2^s - \lfloor \frac{y_j}{2^s} \rfloor 2^s)^2$$

wobei $s = 8 - \min\{8, \lfloor \frac{\text{Generation}}{128} \rfloor\}$. Diese Distanzfunktion blendet einfach die unteren s Bits der Koordinaten aus. Die optimale Route durch alle 100 Städte verändert sich dadurch wie in Abbildung 6.3 dargestellt.

In Abbildung 6.4 ist deutlich zu sehen, daß alle 128 Generationen eine kleinere Verbesserung der maximalen Fitneß eintritt. In genau diesen Abständen wird aber ein weiteres Bit der Koordinaten zur Berechnung der Zielfunktion verwendet. Somit zeigt sich, daß diese Bits ebenfalls zum Wert der Zielfunktion beitragen.

Als weiteren Schritt wurde versucht, das Problem durch folgende Abstandsfunktion weiter zu vereinfachen:

$$d(c_i, c_j) = (\lfloor \frac{x_i}{2^s} \rfloor - \lfloor \frac{x_j}{2^s} \rfloor)^2 + (\lfloor \frac{y_i}{2^s} \rfloor - \lfloor \frac{y_j}{2^s} \rfloor)^2$$

wobei $1 \leq s \leq 8$. Da die von Bäck verwendeten Abstandsdaten in x-Richtung zwischen 0 und 4000 liegen und in y-Richtung im Bereich zwischen 0 und 2000 liegen ergibt sich als Maximum der Zielfunktion mit $s = 8$ der Wert $100(15^2 + 7^2) = 27400$. Dieser Wert ist mit 15 Bits darstellbar. Doch auch mit diesem Ansatz wurde kein besseres Ergebnis erzielt.

6.5 Regionenbildung

Zur Untersuchung der Regionenbildung wurde die Zielfunktion f_1 mit 10 Elementen x_i herangezogen. Jedes Element wurde mit 16 Bits codiert. Als Parameter wurden gewählt: 512x2 Individuen, elitäre Integrationsstrategie, Random-Walk-Selektion, Externe-Nachbar-Partnerwahl, kein Austausch fitter Individuen, 32x32 Rechteck-Topologie, Länge des Random-Walks 4, 1-Punkt-Crossover, Crossover innerhalb eines Wortes erlaubt, angesammelte Mutation auf Bits,

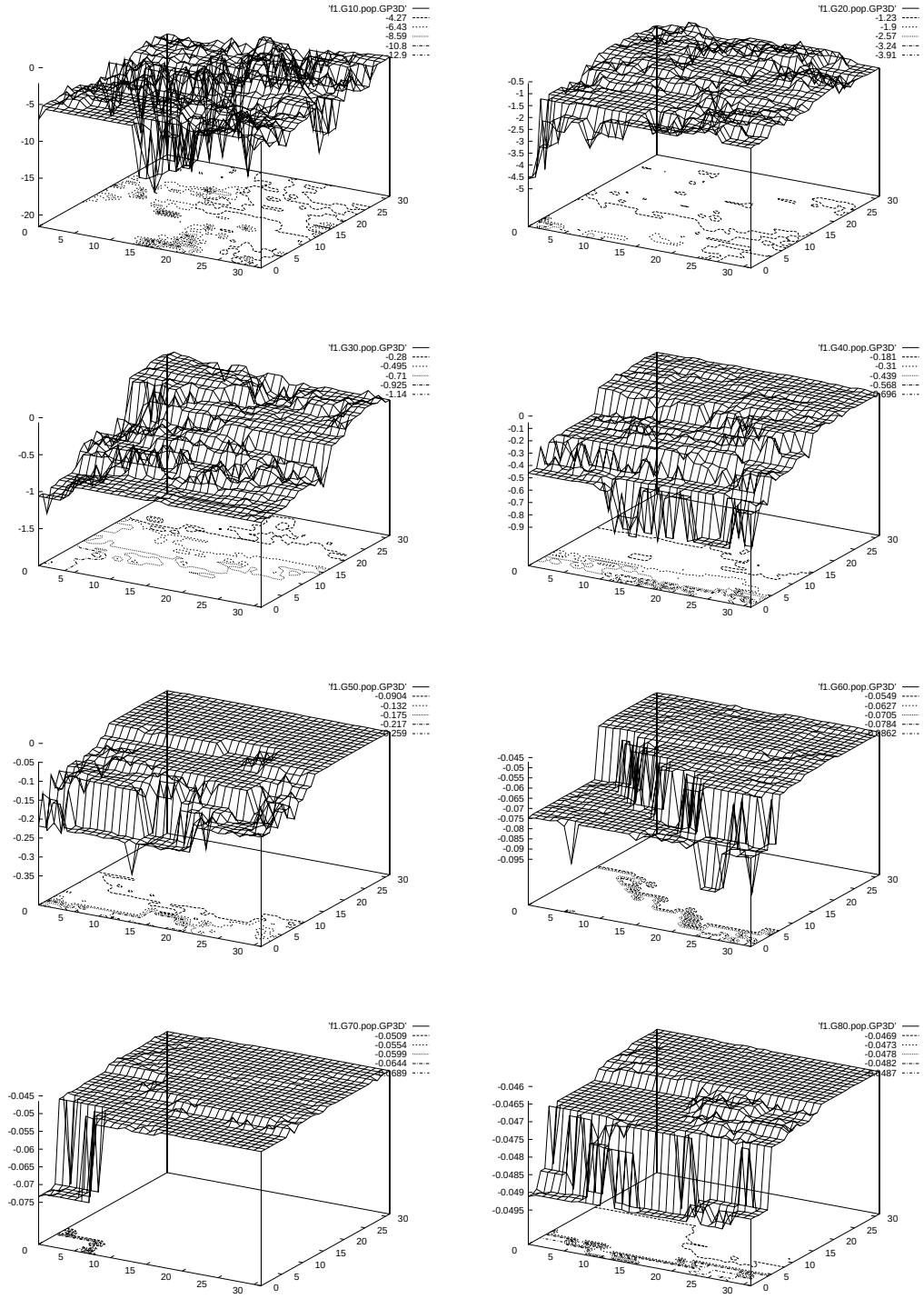

Abbildung 6.5: Regionenbildung bei der Zielfunktion $f_1(10)$

Crossover-Wahrscheinlichkeit $p_{\text{cross}} = 0.99998$, Mutations-Wahrscheinlichkeit $p_{\text{mut}} = 0.005$. Alle 10 Generationen wurden die Fitneßwerte jedes Individuums gespeichert. Der Algorithmus wurde nach 80 Generationen abgebrochen. In Abbildung 6.5 ist die dreidimensionale Visualisierung der Population durch Gnuplot zu sehen.

Es ist deutlich ein stetiges Ansteigen der Fitneßwerte zu erkennen. In der dreidimensionalen Visualisierung Abbildung 6.5 ist eine Wellenbewegung der Fitneßwerte von hinten nach vorne zu erkennen. Diese Welle kommt durch die Externe-Nachbar-Partnerwahl mit elitärer Integrationsstrategie zustande. Durch die Externe-Nachbar-Partnerwahl wird bei jedem Crossover ein Teil des Genotyps von PE $i+1$ auf PE i kopiert. Bei der elitären Integrationsstrategie überlebt das Individuum mit dem neuen Teilstück des Genotyps nur, wenn es eine höhere Fitneß als sein Elter hat. Daher ist der normalen, durch die Random-Walk-Selektion gegebene Regionenbildung, noch eine Welle überlagert.

Kapitel 7

Vergleich mit anderen Implementierungen massiv paralleler genetischer Algorithmen

Zum Vergleich mit anderen Implementierungen wurden folgende Programme herangezogen:

- GENEsYs Version 1.0

  Das Programm GENEsYs wurde von Bäck [Bäck 92] an der Universität Dortmund entwickelt. Es ist in ANSI-C geschrieben und somit auf vielen Architekturen lauffähig. Als Plattform für das Programm GENEsYs wurde im folgenden Vergleich eine Sparc 10 verwendet.

- VEGA (Verteilte genetische Algorithmen)

  VEGA wurde von Roland Baumann [Baumann 95] innerhalb des Projektes EvA am IPVR an der Universität Stuttgart entwickelt. Das Programm VEGA ist für den Einsatz auf dem Parallelrechner Intel Paragon konzipiert aber auch auf einer normalen Workstation einsetzbar. Es ist in ANSI-C geschrieben. Für die parallelen Sprachelemente wird eine Erweiterungsbibliothek verwendet.

- MPGA (Massiv parallele genetische Algorithmen)

  Das Programm MPGA wurde von Alex Hummler innerhalb des Projektes EvA am IPVR an der Universität Stuttgart entwickelt. MPGA ist für den Computer MasPar MP-1 entwickelt und in der Sprache MPL implementiert.

7.1 Zielfunktion f_1

Die Zielfunktion f_1 wurde mit anderen Implementierungen massiv paralleler genetischer Algorithmen verglichen. Die Zeit zur Berechnung der Zielfunktion f_1 ist klein im Vergleich zur Laufzeit des genetischen Algorithmus. Die Zielfunktion wurde für drei Dimensionen berechnet ($f_1(\vec{x}) = \sum_{i=1}^{3} x_i^2$). Bei nur drei Dimensionen ist außerdem die Zeit, die zum Kopieren der Individuen innerhalb eines PEs und zum Austausch der Individuen zwischen den PEs benötigt wird, gering. Somit ist die Laufzeit des Programms bei der Zielfunktion f_1 ein Maß für den

f_1	GENEsYs (Sparc 10)	VEGA (16+1 nodes)	MPGA	CNGA
Generationen	63	92	25	15
Individuen	1x350	16x22	16384x1	512x1
Laufzeit [s]	6,5	1,6	10,4	0,768
Zeit/Generation [ms]	102	17,4	416	51,2
Zeit/Individuum [μs]	295	49,4	25	100

Tabelle 7.1: Laufzeitdaten für die Zielfunktion f_1. Die Daten der Programme GENEsYs, VEGA und MPGA stammen von Zell et al. [Zell et al. 95a].

eigentlichen genetischen Algorithmus. Für die Codierung der Zielfunktion wurden insgesamt 48 Bits verwendet, daher paßt jedes Element x_i in ein Wort (16 Bits).

Bei der Zielfunktion f_1 handelt es sich um eines der einfacheren Benchmark-Probleme, die von Back [Bäck 92] implementiert wurden. Wie in Tabelle 7.1 zu sehen ist, konvergiert der genetische Algorithmus schon nach 15 Generationen. Dies ist vor allem darauf zurückzuführen, daß die Fließkommazahlenberechnungen nur mit einer Genauigkeit von 16 Bits durchgeführt werden. Da die Elemente x_i vor der Summation quadriert werden, sind nur 8 Bit je Element für das Ergebnis der Zielfunktion relevant. Somit braucht sich der genetische Algorithmus um eine korrekte Belegung der unteren 8 Bit nicht zu kümmern. Er kann also schneller das gewünschte Ergebnis $f_1(\vec{x}) = 0$ erreichen.

In Tabelle 7.1 sticht die geringe Laufzeit des Programms CNGA deutlich heraus. CNGA ist insgesamt 2,08-mal schneller als das Programm VEGA. Aufgrund der unterschiedlichen Genauigkeit der Fließkommazahlenarithmetik, der in Tabelle 7.1 angegebenen Programme, ist jedoch die benötigte Zeit je Generation die am aussagekräftigste Größe. Bei Vergleich dieser Größe stellt sich das Programm VEGA als eindeutiger Sieger mit nur 17,4s je Generation heraus. Allerdings ist diese Zahl von der relativ geringen Zahl von 22 Individuen je Population abhängig. Das Programm CNGA ist fast doppelt so schnell, wie das von Bäck [Bäck 92] entwickelte Programm GENEsYs. Erstaunlich langsam ist das Programm MPGA mit 416s je Generation. Dies ist vermutlich auf den 4-Bit Prozessor zurückzuführen. Da die verwendete MasPar jedoch 16384 PEs besitzt, ergibt sich für MPGA schließlich die geringste Zeit je Individuum.

7.2 Zielfunktion f_{10}

In Tabelle 7.2 sind die Laufzeiten der einzelnen Programme für das Problem des Handlungsreisenden mit 100 Städten angegeben. Für das Programm CNGA wurde eine Codierung von 16 Bits je Element x_i des Genotyps gewählt. Somit ergibt sich eine Gesamtlänge von 1600 Bits. Die Parameter wurden wie folgt eingestellt: 512x1 Individuen, elitäre Integrationsstrategie, uniformes Ranking, $\mu = 64$, Externe-Nachbar-Partnerwahl, kein Austausch fitter Individuen, 2-Punkt-Crossover, Crossover innerhalb eines Wortes erlaubt, angesammelte Mutation auf Bits, Crossover-Wahrscheinlichkeit $p_{\text{cross}} = 0.01$, Mutations-Wahrscheinlichkeit $p_{\text{mut}} = 0.01$. Der Algorithmus wurde nach 400 Generationen abgebrochen. Als kürzeste Route wurde ein Weg der Länge 38852 gefunden.

f_{10}	GENEsYs (Sparc 10)	VEGA (64+1 nodes)	MPGA	CNGA
Generationen	1000	1000	1000	400
Individuen	1x7200	64x256	16384x1	512x1
Kürzester Weg	30932	29498	24312	38852
Laufzeit [s]	40617	1320	1831	48.63
Zeit/Generation [s]	40,6	1,32	1,831	0,122
Zeit/Individuum [μs]	5600	80,56	111	237

Tabelle 7.2: Laufzeitdaten für die Zielfunktion f_{10}. Die Daten der Programme GENEsYs, VEGA und MPGA stammen von Zell et al. [Zell et al. 95a].

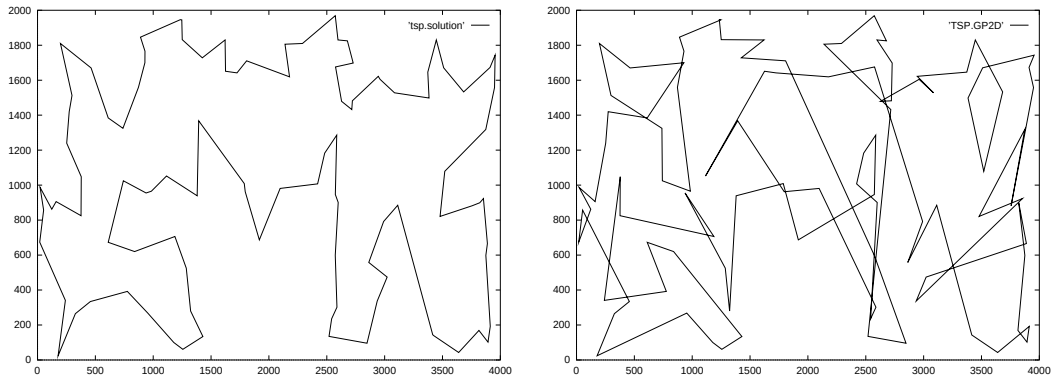

Abbildung 7.1: Lösung des TSP Problems und beste Route des Programms CNGA

Die Programme GENEsYs, VEGA und vor allem MPGA lieferten wesentlich bessere Ergebnisse. Jedoch hat keines von ihnen die in Abbildung 7.1 angegebene kürzeste Route mit einer Weglänge von 21285 gefunden. Das eindeutig beste Ergebnis lieferte das Programm MPGA mit einer Weglänge von 24312. Hummler beschreibt in [Hummler 95], daß MPGA bei kleinen Crossover-Wahrscheinlichkeiten ein besseres Ergebnis aufgrund der langsameren Konvergenz erreicht. Auch CNGA zeigt dieses Verhalten. Mit einer Crossover-Wahrscheinlichkeit von 0.99998 wurde nur ein Weg der Länge 42639 gefunden.

Für einen Vergleich der unterschiedlichen Programme ist wiederum die Zeit je Generation wesentlich. GENEsYs zeigt das mit Abstand langsamste Verhalten (40,6s), VEGA (1,32s) und MPGA (1,831s) zeigen gegenüber GENEsYs eine deutlich geringere Laufzeit je Generation. CNGA mit nur 0,122s je Generation ist jedoch nochmals deutlich schneller als VEGA und MPGA.

Zum Vergleich seien noch die Ergebnisse von Engst [Schwehm et al. 93] mit einem an der Friedrich-Alexander-Universität in Erlangen-Nürnberg entwickelten massiv parallelen Genetischen Algorithmus genannt. Engst versucht das Problem des Handlungsreisenden mit einem, an das Problem angepaßten, Crossover-Operator zu lösen. Damit erreicht er in 125 Sekunden ein Ergebnis, das nur 1% von der optimalen Lösung abweicht.

f_{12}	GENEsYs (Sparc 10)	VEGA (9+1 nodes)	MPGA	CNGA
Generationen	350	200	191	296
Individuen	512	9x100	16384x1	512x1
Laufzeit [s]	803	85,1	457,7	22,38
Zeit/Generation [ms]	2294	425,5	2396	75,61
Zeit/Individuum [μs]	4481	472,8	146,3	147,7

Tabelle 7.3: Laufzeitdaten der Zielfunktion f_{12} für typische Parametereinstellungen. Die Daten für das Programm VEGA stammen aus [Wakunda 95b].

7.3 Zielfunktion f_{12}

Die Zielfunktion f_{12} (Bitsumme) wurde für einen weiteren Vergleich mit anderen Implementierungen ausgewählt. Diese Zielfunktion kann mit Ganzzahlenarithmetik berechnet werden. Daher ergibt diese Testreihe einen Anhaltspunkt für die Laufzeiten bei Problemen, die mit Ganzzahlenarithmetik berechnet werden können. Als Genotyp wurde ein String aus 1600 Bits gewählt. Dadurch ist gewährleistet, daß die Zeit für die Bewegung der Individuen in die Gesamtlaufzeit eingeht.

Die Laufzeiten der Tabelle 7.3 wurden mit für die jeweiligen Programme typischen Einstellungen ermittelt. Daher sind die ermittelten Laufzeiten vermutlich nicht optimal. Da die einzelnen Programme nach einer unterschiedlichen Anzahl von Generationen gestoppt wurden, ist wieder die Laufzeit pro Generation wesentlich. Die Programme GENEsYs und MPGA benötigen beide etwa die gleiche Zeit pro Generation. MPGA ist jedoch in der Gesamtlaufzeit fast doppelt so schnell wie GENEsYs. VEGA wurde nach 200 Generationen vorzeitig abgebrochen (Fitneßwert 1468). Die Generationen der anderen Programme geben die Anzahl der Generationen bis zum Erreichen des Optimums an. Auffallend ist wieder die geringe Laufzeit von CNGA mit nur 75,61s je Generation und 22,38s Gesamtlaufzeit.

Kapitel 8

Erweiterungen

Um eine neue Zielfunktion in das Programm CNGA einzubinden, müssen folgende Schritte durchgeführt werden. Aufgrund der durch die CNAPS-Architektur bedingten Trennung in CNAPS-Programmcode und ANSI-C Programmcode, sollten diese Schritte genau befolgt werden. Damit wird erreicht, daß der auf dem Host angezeigte Phänotyp mit dem auf der CNAPS berechneten Genotyp korrespondiert.

8.1 Menüeintrag

In der Datei `cnga_objf.c` muß ein Eintrag in die Struktur `cnga_objData` gemacht werden. Jede Zeile der Struktur erzeugt einen Eintrag im CNGA-Menü. Durch diesen Eintrag wird die Zielfunktion dem Anwender zugänglich gemacht. Der Eintrag muß genau denselben Index (beginnend bei Null) haben, wie der Parameter `cnga_fNum`, mit dem in dem CNAPS-spezifischen Teil die Zielfunktion ausgewählt wird.

```
struct objCngaStruct {
       void (*f)();            /* function to convert geno to pheno */
       char *name;             /* name of objective function        */
       int  min;               /* minimum length of genotype        */
       int  max; };            /* maximum length of genotype        */
typedef struct objCngaStruct objCngaType;
```

Das erste Element der Struktur ist ein Zeiger auf eine Routine, die als Eingabe den Genotyp eines Individuums erhält und den Phänotyp als String ausgibt. Diese Routine hat folgende Struktur:

```
void cnga_geno2pheno(pheno,geno,gray)
char *pheno;                   /* destination of phenotype          */
char *geno;                    /* source of genotype                */
int  gray;                     /* gray code flag                    */
  {

...

  }
```

Dabei ist `pheno` ein Zeiger auf den zu erzeugenden Phänotyp-String. `geno` ist ein Zeiger auf den Genotyp des Individuums in ASCII-Darstellung, wobei '|' als Separator in den String integriert sein kann. `gray` ist ein Flag, das angibt, ob Gray-Codierung verwendet wurde.

Das zweite Element des Eintrags ist ein Zeiger auf den Text, der im Menü erscheint. Das erste Zeichen dieses Textes gibt an, ob die Zielfunktion installiert ist. Der Text sollte mit einem '*' beginnen. Dies zeigt an, daß der Eintrag erst freigegeben werden muß.

Das dritte Element des Eintrags ist die kleinste gültige Chromosomenlänge für die Zielfunktion. Das vierte Element gibt die größte erlaubte Chromosomenlänge für die Zielfunktion an. Dabei gibt -1 an, daß keine spezielle Beschränkung gewünscht wird. Die Länge des Genotyps auf $1 \leq l \leq$ CNGA_MAX_N ist trotzdem gegeben.

```
objCngaType cnga_objData[]={
    { cnga_gtpNone,   "*Random",  -1,-1 },
    { cnga_gtpF1,     "*f1",      -1,-1 },
    { cnga_gtpF1,     "*f2",      -1,-1 },
....
    { cnga_gtpNone,   "(empty)",  -1.-1 },
    { 0,              0          }  };
```

Die Funktion `cnga_gtpNone` kann angegeben werden, wenn keine Konvertierung des Genotyps gewünscht wird.

8.2 Freigabe des Menüeintrags

In die Funktion `cnga_adaptMenu` müssen folgende Zeilen integriert werden, wobei statt ? der Index der Zielfunktion angegeben wird:

```
#ifdef CNGA_F?
  cnga_enableChoice(m,&cnga_fNum,CNGA_F_F?);
#endif
```

8.3 Eintrag in der Datei `cnga.h`

In die Datei `cnga.h` muß folgende Zeile eingefügt werden, wobei statt ? der Index der Zielfunktion angegeben wird:

```
#define CNGA_F_F?           ?
```

8.4 Eintrag in der Datei `cnga_defs.h`

Soll die Zielfunktion in das zu übersetzende Programm integriert werden, so ist folgender Eintrag in der Datei `cnga_defs.h` zu machen. Auch hier steht ? wieder für den Index der Zielfunktion.

```
#define CNGA_F?
```

8.5 Implementierung der Zielfunktion

Schließlich ist die eigentliche Zielfunktion zu implementieren. Dazu ist der entsprechende Programmcode in die Datei `cnga_app.cn` zu schreiben, wenn die Domain vom Typ "Generic" ist und in die Datei `cnga.cn` zu integrieren, wenn die Domain "indCngaDomain" benötigt werden sollte.

8.6 Auswahl der Zielfunktion

Damit die richtige Zielfunktion zur Berechnung ausgewählt wird, muß in die Routine `cnga_evaluateObjective` ein dem folgenden Programmstück entsprechender Code eingefügt werden.

```
#ifdef CNGA_F?
          case  CNGA_F_F?:
                cnga_f?(cnga_nDim,kid[i].geno,&kid[i].obj);
                break;
#endif
```

Dabei steht `?` wieder für den Index der Zielfunktion. `cnga_nDim` gibt die Dimension der Zielfunktion an, `kid[i].geno` ist ein Zeiger auf den Genotyp und `&kid[i].obj` ist ein Zeiger auf den Typ `realCngaType`, in den der Fitneßwert zu schreiben ist.

8.7 Besondere Initialisierung

Falls nötig, kann der Genotyp mit einem Eintrag in der Routine `cnga_initGeno` initialisiert werden. Auch hier steht `?` wieder für den Index der Zielfunktion.

```
    CNGA_F_F?: cnga_initGenoF?(cnga_nDim,geno);
               break;
```

Kapitel 9

Aufbau des Programms

Das CNGA-Programm ist aufgrund der CNAPS-Architektur zweigeteilt. Der eine Teil ist in C [Kernighan et al. 83] geschrieben und läuft auf dem Host. Der andere Teil ist in CNAPS-C geschrieben und läuft auf dem CNAPS-Server. Jedes, der im folgenden angegebenen Module, besteht aus einem Implementierungsmodul mit der Endung ".c" und einem Definitionsmodul mit der Endung ".h". Es gibt hierzu nur eine Ausnahme. Das Definitionsmodul `cnga.h` wird sowohl von `cnga.c` als auch von `cnga.cn` verwendet. Diese Lösung wurde gewählt, um Übereinstimmung zwischen Host-spezifischem Teil und Server-spezifischem Teil zu erreichen. Ferner existiert noch eine Datei mit dem Namen `cnga_defs.h`. Sie enthält alle Flags zur bedingten Übersetzung. Der CNAPS-Teil des CNGA ist so umfangreich geworden, daß der CNAPS-C-Compiler nicht mehr in der Lage ist, das vollständige Programm zu übersetzen. In der Datei `cnga_defs.h` wird angegeben, welche Teile des Programms übersetzt werden. Die Zusammenarbeit der einzelnen Programmkomponenten wird in Abbildung 9.1 graphisch dargestellt.

9.1 ANSI-C-Module

1. CNGA-Modul: `cnga.c`

   Dieses Modul ist das eigentliche CNGA-Programm. Hier laufen alle Module zusammen. Es führt die erforderliche Kommunikation mit dem CNAPS-Server durch, um den genetischen Algorithmus zu starten.

2. CNGA-Gnuplot-Ausgabe-Modul: `cnga_gnup.c`

   Enthält alle Routinen, um die Gnuplot-Informationen anzuzeigen.

3. CNGA-Fehlermeldungs-Modul: `cnga_errm.c`

   Enthält alle Routinen für die Ausgabe der Fehlermeldungen.

4. CNGA-UIEA-Modul: `cnga_init.c`

   Enthält alle Routinen, mit der die von Hasel [Hasel 95] und Görzig [Görzig 95] entwickelte Datenstruktur gefüllt wird. Die Datenstruktur wird von Hasel für die Erzeugung der innerhalb des Projekts einheitlichen graphischen Benutzeroberfläche verwendet. Görzig verwendet die Datenstruktur, um eine einheitliche Benutzeroberfläche mit TCL auf Textbasis zu generieren.

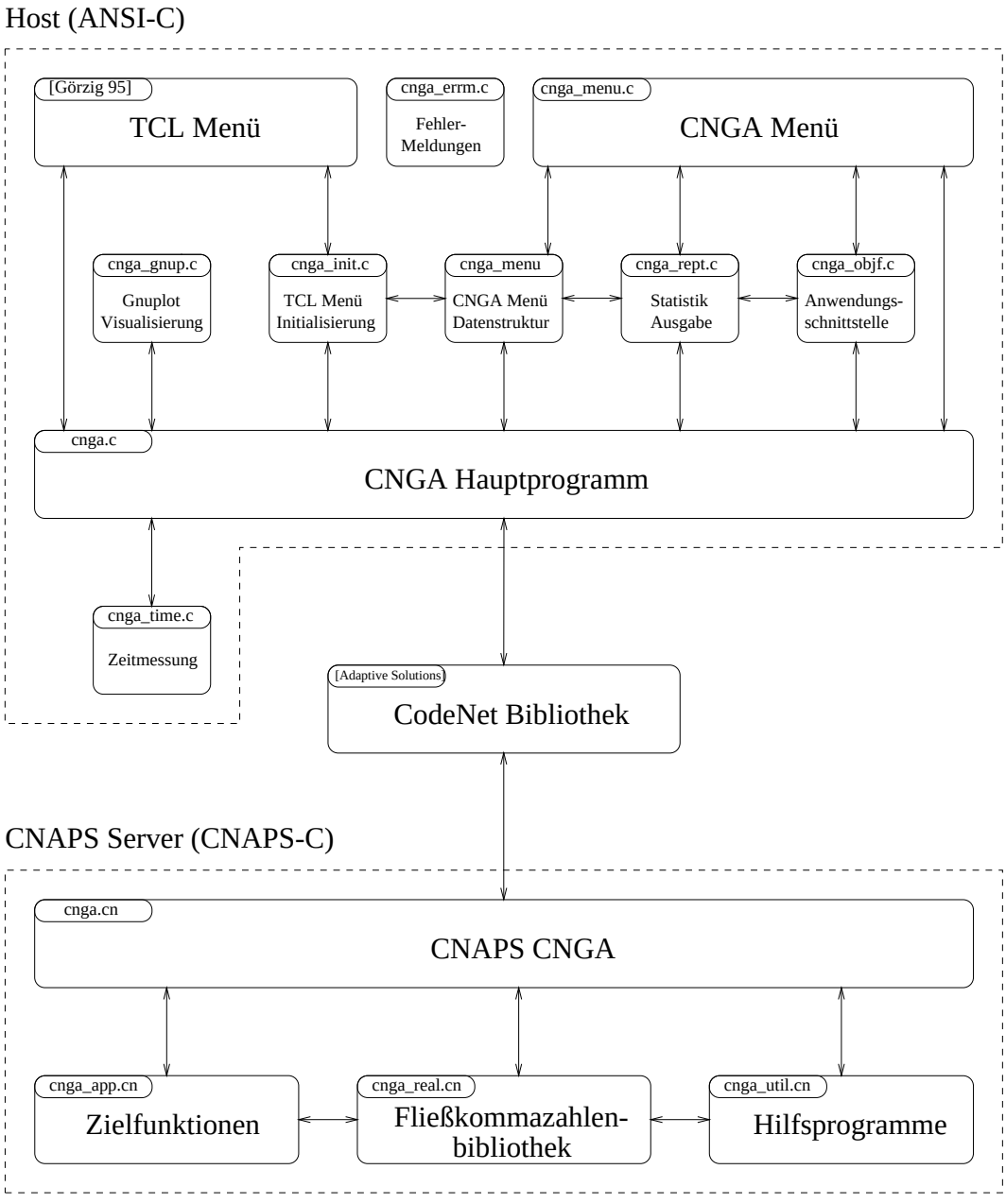

Abbildung 9.1: Zusammenarbeit und Aufteilung der CNGA-Module

5. CNGA-Menü-Modul: `cnga_menu.c`

   Enthält alle Routinen für die Ausgabe des CNGA-Menüs, falls TCL nicht installiert ist. Es besitzt das gleiche "Look and Feel", wie das von Baumann [Baumann 95] und Hummler [Hummler 95] entwickelte Eingabemenü.

6. CNGA-Zielfunktionen-Modul: `cnga_objf.c`

   Enthält alle Daten und Routinen für die Einträge der Zielfunktionen im Eingabemenü und die Konvertierung des Genotyps in den korrespondierenden Phänotyp.

7. CNGA-Zielfunktionen-Modul: `cnga_objf.c`

   Enthält alle Routinen für die Generierung der Berichte. Damit werden alle Dateien in dem von Baumann [Baumann 95] und Hummler [Hummler 95] in Absprache mit dem Betreuer Zell entwickelten Format ausgegeben.

8. CNGA-Zeitmessungs-Modul: `cnga_time.c`

   Dieses Modul enthält die Routinen, mit denen die Zeitmessung eines CNGA-Laufs durchgeführt wird.

9.2 CNAPS-C-Module

1. CNGA-Modul: `cnga.cn`

   Dies ist der Hauptteil des CNAPS-spezifischen CNGA-Programms. Es enthält die "Entry-Points" `main` und `cnga_cont` und den eigentlichen genetischen Algorithmus.

2. CNGA-Anwendungs-Modul: `cnga_app.cn`

   Enthält den Programmcode zur Berechnung der Zielfunktionen. Lediglich Funktionen, die Daten wie z.B. Crossover oder Mutations-Masken benötigen, sind im CNGA-Modul `cnga.cn` enthalten.

3. CNGA-Fließkommazahlen-Modul `cnga_real.cn`

   Enthält den Programmcode zur Benutzung von Fließkommazahlen.

4. CNGA-Hilfsprogramm-Modul: `cnga_util.cn`

   Enthält einige Hilfsprogramme, die allgemein einsetzbar sind.

9.3 Make-Dateien

Zur Übersetzung als Einzelprogramm ohne TCL-Unterstützung kann die Datei `makefile` verwendet werden. In der von Hasel [Hasel 95] entwickelten Dateien makefile.cnga und depend.CNGA sind die Informationen zur Übersetzung mit der von Hasel [Hasel 95] entwickelten graphischen Oberfläche zusammengefaßt.

Kapitel 10

Das Programm CNGA - CNAPS Genetic Algorithms

10.1 Starten des Programms

Das Programm CNGA wird durch die Eingabe `cnga` auf dem Host Computer gestartet. Nach dem Start des Programms wird das Titelbild angezeigt und nach der Einstellungs-Datei mit dem Namen `cnga_setf.GAset` im aktuellen Verzeichnis gesucht. Wird diese Datei gefunden, so wird das CNGA-Menü mit den Einstellungen aus der Datei initialisiert. Ansonsten werden die vorgegebenen Einstellungen auf dem Programm CNGA verwendet. Nach der ersten Benutzung des Programms sollte eine eigene Einstellungs-Datei erzeugt werden.

Als nächstes wird die Verbindung zum angegebenen CNAPS-Server aufgebaut und der CNAPS-Programmcode `cnga.x` aus dem durch den Parameter `path_of_executable` angegebenen Pfad auf den CNAPS-Server kopiert. Um mit dem CNAPS-Server zu kommunizieren wird das von Adaptive Solutions bereitgestellte Interface CodeNet ([Adaptive Solutions 93b], [Adaptive Solutions 94d]) verwendet. Es wird über die sogenannte cnlib C-Bibliothek angesprochen.

```
+-------------------------------------------------------------------------+
|                      CNGA - CNAPS Genetic Algorithms                    |
|                                                                         |
|                                  Version:                               |
|                                    1.0                                  |
|                                                                         |
|                                     by                                  |
|                                 Marc Ebner                              |
|                                Karlstrasse 50                           |
|                               70839 Gerlingen                           |
|                                  Germany                                |
|                                                                         |
|     Copyright (C) 1995 EA-Group, IPVR, Universitaet Stuttgart, Germany  |
+-------------------------------------------------------------------------+
Settings successfully loaded.
INFO   : CNLIB successfully initialized
INFO   : cnConnectN() successful: cnaps
INFO   : cnCd() successful: /home/ebner/SA/cnga/bin/cnaps/
INFO   : cnLoadExe() successful: cnga.x
```

```
INFO   : cnCpDn() successfully completed: cnga.init.data to cnga.init.data
INFO   : cnCd() successful: /tmp_mnt/home/ebner/SA/cnga/source
```

Nach den Statusmeldungen der Initialisierung wird das CNGA-Menü angezeigt. Jetzt können die Parameter des CNGA Programms verändert werden. Erst wenn der genetischen Algorithmus mit dem Kommando `run` gestartet wird, wird der genetische Algorithmus auf dem CNAPS-Server ausgeführt.

Das Programm CNGA kann auch mit `cnga <DATEI>` gestartet werden. Dabei steht `<DATEI>` für einen Dateinamen. Die angegebene Datei wird als Datei mit Batch-Informationen interpretiert. Dazu werden die einzelnen Befehle, wie sie auch im interaktiven Betrieb benutzt werden können, in ASCII-Form in die Datei geschrieben. Die Datei wird vom Programm CNGA Zeile für Zeile interpretiert und ausgeführt.

10.2 Menü

Das Menü zeigt die aktuellen Parameter an. Das Programm CNGA enthält zwei verschiedene Arten von Menüs. Welches von beiden verwendet wird, hängt von der Compilierung des Programms ab. Ist das Flag `__TCL__` nicht gesetzt, so wird das CNGA-Menü compiliert. Es ist ohne TCL lauffähig. Das CNGA-Menü ist vollständig vom Programm entkoppelt. Es Menü wird durch eine einfach zu benutzende Datenstruktur initialisiert.

```
outputCngaType cnga_menu[]={
  { CNGA_LINE,    78, 78,        "-",              cnga_depend, CNGA_TRUE },
  { CNGA_TEXT,    78, CNGA_CENTER, "CNGA - CNAPS Genetic Algorithms",
                                                   cnga_depend, CNGA_TRUE },
  { CNGA_TEXT,    78, CNGA_CENTER,
    "by Marc Ebner, Karlstrasse 50, 70839 Gerlingen, Germany",
                                                   cnga_depend, CNGA_TRUE },
  { CNGA_TEXT,    78, CNGA_CENTER,
    "Copyright (C) 1995 EA-Group, IPVR, Universitaet Stuttgart, Germany",
                                                   cnga_depend, CNGA_TRUE },
  { CNGA_LINE,    78, 0,         "-",              cnga_depend, CNGA_TRUE },
  { CNGA_CLASS,   78, 0,     "H|Server|Server...", cnga_depend, CNGA_TRUE },
  { CNGA_STRING,  78, CNGA_LEFT, &cnga_servData,   cnga_depend, CNGA_TRUE },
...
  { CNGA_LINE,    26, 0,         "-",              cnga_depend, CNGA_TRUE },
  { CNGA_ENDOM,   0,  0,         0,                0,            0 },
  };
```

Diese Datenstruktur wird auch für die Erstellung der Berichte verwendet.

```
+-------------------------------------------------------------------------+
|                      CNGA - CNAPS Genetic Algorithms                    |
|          by Marc Ebner, Karlstrasse 50, 70839 Gerlingen, Germany        |
|     Copyright (C) 1995 EA-Group, IPVR, Universitaet Stuttgart, Germany  |
+-------------------------------------------------------------------------+
| serv   server_name                  =                           cnaps |
| pipe   pipe_results                 =                             OFF |
+-------------------------------------------------------------------------+
| pe     number_of_pes_used           =                             512 |
```

```
| nip    num_individuals/pe            =                                1 |
| cl     chromosome_length_in_words    =                                3 |
| a      application                   =                              f12 |
| gray   gray_code                     =                              OFF |
+-----------------------------------------------------------------------+
| dir    search_direction              =                         negation |
| rm     replacement_method            =                         children |
| sm     selection_method              =                      uniform_rank |
| part   partner_selection             =        external_adjacent_partner |
| em     exchange_method               =                *shift_best_worst |
| ne     number_of_exchanges           =                                0 |
| wot    width_of_topology             =                               32 |
| mu     cutoff_parameter_mu           =                              128 |
+-----------------------------------------------------------------------+
| cp     crossover_points              =                                1 |
| cbw    crossover_between_words       =                               ON |
| mm     mutation_method               =     accumulated_mutation_on_bits |
| cr     crossover_rate                =                         0.999990 |
| mr     mutation_rate                 =                         0.010000 |
+-----------------------------------------------------------------------+
| v      verbose                       =                              OFF |
| rtyp   report                        =                           normal |
| pi     protocol_interval             =                                1 |
+-----------------------------------------------------------------------+
| g      generations                   =                              100 |
| smf    supposed_max_fitness          =                         0.000000 |
| bcl    bit_conv_limit                =                         0.900000 |
| div    diversity                     =                         0.000000 |
| acm    abort_criteria_maximum        =                              OFF |
| acc    abort_criteria_convergence    =                              OFF |
| acd    abort_criteria_diversity      =                              OFF |
+-----------------------------------------------------------------------+
| poe    path_of_executable            =   /home/ebner/SA/cnga/bin/cnaps/ |
| setf   settings_file                 =                       cnga_setf |
| seedf  seed_file                     =                       cnga_rand |
| sdf    summary_data_file             =                       cnga_sumf |
| pdf    protocol_data_file            =                       cnga_prof |
+----------------------++-----------------------++----------------------+
| q      quit           || m      menu           || h      help           |
| r      run            || s      step           || c      continue       |
| cts    connect_to_serv || dfs    disconnect_from || ps     print_statistic |
| save   save_settings  || gs     generate_seeds  || pfit   plot_fitness   |
| load   load_settings  || ds     display_seeds   || pcon   plot_convergenc |
+----------------------++-----------------------++----------------------+
```

Ist TCL auf dem Host Computer installiert, so kann das CNGA Programm mit dem Flag `__TCL__` übersetzt werden. Dann wird obige Datenstruktur ausgelesen, und die von Hasel [Hasel 95] und Görzig [Görzig 95] entwickelte Datenstruktur gefüllt. Die Ausgabe des TCL Menüs wurde von Görzig [Görzig 95] programmiert. Beide Menüs haben das gleiche "Look and Feel" wie das von Baumann [Baumann 95] und Hummler [Hummler 95] entwickelte Menü.

10.3 Die Kommandozeile

Das Programm CNGA beginnt die Kommandozeile mit dem Prompt `CNGA>` und wartet auf eine Eingabe des Benutzers. In der Eingabezeile können mehrere Befehle hintereinander eingegeben werden. Es gibt folgende Befehle.

- Kommandos

  Ein Kommando besteht aus einem einzigen Wort und ist im unteren Teil des Menüs zusammengefaßt. Jedes Kommando bewirkt eine bestimmte Aktion.

- Parameteränderungen

  Parameteränderungen bestehen aus zwei aufeinander folgende, durch Zwischenraum getrennte Wörter. Dabei bestimmt das erste Wort den Namen des Parameters und das zweite den neuen Wert des Parameters.

- Hilfe

  Hilfe kann durch Eingabe des Kommandos `help` oder die Abkürzung `h` gefolgt von dem Parameter, Kommando oder der Einstellung über den Hilfe erwünscht wird, erhalten werden. Als Hilfe wird der Name des Parameters bzw. des Kommandos, dessen Abkürzung und ein kurzer Text zur Erklärung des Parameters. Bei Parametern werden zusätzlich noch die für den Parameter gültigen Werte ausgegeben.

10.4 Parameter

Der genetische Algorithmus wird mit folgenden Parametern aufgerufen:

10.4.1 Servereinstellungen

- `server_name`

  Über diesen Parameter kann eingestellt werden, mit welchem CNAPS-Server das Programm CNGA bei einem `connect`-Befehl eine Verbindung herstellt.

- `pipe_results`

  Mit diesem Schalter kann eingestellt werden, wie die Ergebnisse des CNGA-Programms, das auf dem CNAPS-Server abläuft, übermittelt werden. Steht der Schalter auf `ON`, dann wird eine Pipe-Verbindung zwischen Host und Server aufgebaut. Durch diese Pipe werden dann die Statistikdaten und das Endergebnis transportiert, sobald diese berechnet wurden. Daher erhält der Anwender Ergebnisse über den Verlauf des genetischen Algorithmus noch während dieser läuft. Wenn der Schalter auf `OFF` steht, werden die Statistikdaten und das Endergebnis in einer Datei auf dem CNAPS-Server gespeichert. Erst wenn der genetische Algorithmus terminiert hat, wird diese Datei als Ganzes auf den Host-Computer transportiert. Somit wird der Anwender erst nach der Terminierung des Algorithmus über den Verlauf informiert.

10.4.2 Einstellungen der Anwendung

- `number_of_pes_used` (n_x)

  Definiert die Anzahl der Prozessoren, die zur Berechnung des genetischen Algorithmus herangezogen werden ($1 \leq n_x \leq n_{x_{\max}}$). Wobei $n_{x_{\max}}$ die Anzahl der Prozessoren auf dem CNAPS-Server angibt.

- `num_individuals/pe` (n_y)

  Definiert die Anzahl der Individuen je Prozessor ($1 \leq n_y \leq n_{x_{\min}}$ und n_y muß eine gerade Zahl sein). Dabei gibt $n_{x_{\min}}$ die Zahl der Individuen an, die sich auf jedem PE befinden können. Somit ergibt sich die Anzahl aller Individuen n in der Population aus $n = n_x n_y$.

- `chromosome_length_in_words` (l)

  Gibt die Länge des Genotyps in Worten an. Ein Wort hat 16 Bits. Somit besteht der Genotyp eines Individuums aus $16l$ Bits. Dieser Parameter gibt an, wieviele Bits des Genotyps bei Crossover, Mutation und Kopieroperationen bearbeitet werden. Bei einigen Zielfunktionen steuert dieser Parameter auch, wieviele Werte im Genotyp codiert sind. Bei der Zielfunktion f_1 wird z.B. über l Werte summiert $f_1(\vec{x}) = \sum_{i=1}^{l} x_i^2$.

- `application`

  Nummer der Zielfunktion. Für den genetischen Algorithmus wurden eine Reihe verschiedener Zielfunktionen implementiert. Mit dem Parameter `application` wird die Zielfunktion ausgewählt, deren Optimum der genetische Algorithmus zu ermitteln versucht. Mit dem Parameter `application` wird ebenfalls bestimmt, wie der Genotyp eines jeden Individuums interpretiert wird.

- `gray_code`

  Mit diesem Schalter wird angegeben, ob die Elemente des Genotyps mit der Gray-Codierung codiert sind. Dieser Parameter wird nur von den Zielfunktionen berücksichtigt, die Ganzzahlen oder Fließkommazahlen im Genotyp codieren. Zielfunktionen, die den Genotyp als Bitstring betrachten, berücksichtigen den Parameter `gray_code` nicht.

10.4.3 Strategien

- `search_direction`

  Gibt an, ob der genetische Algorithmus das Maximum oder das Minimum der Zielfunktion ermitteln soll. Der Wert der Zielfunktion wird mit der angegebenen Funktion transformiert, um den Fitneßwert zu bestimmen. Folgende Einstellungen sind möglich:

  - `identity`

    Der genetische Algorithmus verwendet direkt den Wert der evaluierten Zielfunktion als Fitneß des Individuums.

  - `negation`

    Der genetische Algorithmus verwendet den negierten Wert der evaluierten Zielfunktion als Fitneß des Individuums.

- `replacement_method`

  Definiert das Verfahren mit dem die Nachkommen in die Population der Eltern integriert werden. Es stehen folgende Verfahren zur Auswahl:

  - `children`
    Die Nachkommen ersetzen völlig ihre Eltern.
  - `no_children`
    Keines der Elter wird durch ein Kind ersetzt.
  - `fitter`
    Ein Individuum ersetzt nur seinen Elter, wenn es eine höhere Fitneß als dieser hat.

- `selection_method`

  Gibt das Selektionsverfahren an, mit dem aus einer alten Generation eine neue erzeugt wird. Folgende Selektionsverfahren stehen zur Auswahl:

  - `no_selection`
    Kein Selektionsverfahren wird angewandt. Es ist nur noch möglich durch ein geeignetes Integrationsverfahren oder Austauschverfahren ein Individuum mit optimaler Fitneß zu finden.
  - `uniform_rank_with_copy`
    Uniformes Ranking mit Kopieren. Die $n_y\mu$ besten Indivduen werden für die nächste Generation ausgewählt. Die restlichen Individuen der Population werden aus den $n_y\mu$ besten Individuen zufällig aufgefüllt.
  - `winston_rank`
    Das als Rank Method in [Winston 92] beschriebene Verfahren weist jedem Individuum eine Selektionswahrscheinlichkeit zu. Dabei erhält das erste Individuum die Selektionswahrscheinlichkeit $p_1 = p_c$, alle anderen (bis auf das letzte) $p_i = \left(1 - \sum_{j=1}^{i} p_j\right) \cdot p_c$ mit $i < n$. Das letzte Individuum schließlich erhält die Wahrscheinlichkeit $p_n = 1 - \sum_{j=1}^{n-1} p_j$. Anschließend wird wie beim Roulette-Verfahren ermittelt, welche Nachkommen für die nächste Generation ausgewählt werden.
  - `roulette_wheel`
    Beim Roulette-Rad-Verfahren werden zuerst die Selektionswahrscheinlichkeiten $p_{i_x i_y} = \frac{f_{i_x i_y}}{\sum_{j_x=1}^{\mu} \sum_{j_y=1}^{n_y} f_{j_y}}$ $1 \leq i_x \leq n_x$ und $1 \leq i_y \leq n_y$ und $p_{i_x i_y} = 0$ für $1 \leq i_x \leq n_x$ bestimmt. Dann wird mit n Zufallsexperimenten ermittelt, welche Individuen für die nächste Generation ausgewählt werden.
  - `tournament`
    Bei der Tournament-Selektion werden für jedes Individuum anhand der eingestellten Topologie dessen Nachbarn bestimmt. Das Beste der Nachbar-Individuen ersetzt dann in der nächsten Generation das Individuum.
  - `random_walk`
    Von jedem Individuum wird zufällig ein Weg der Länge l_w anhand der eingestellten Topologie gewählt. Anschließend wird dem aktuellen Platz das Beste der Individuen zugewiesen, die auf dem Weg angetroffen wurden.

- `deterministic_sampling`

  Bei der deterministischen Auswahl werden wie beim Roulette-Rad die Selektionswahrscheinlichkeiten $p_{i_x i_y} = \frac{f_{i_x i_y}}{\sum_{j_y=1}^{\mu} f_{j_y}}$ mit $1 \leq i_x \leq n_x$ und $1 \leq i_y \leq n_y$ berechnet. Jedes Individuum erzeugt $\lfloor np_{i_x i_y} \rfloor$ Nachkommen. Jetzt sind noch $n_r = n - \lfloor np_{i_x i_y} Nachkommen \rfloor$ Individuen für die nächste Generation zu vergeben. Dazu werden die Individuen nach den Wahrscheinlichkeiten $p'_{i_x i_y} = p_{i_x i_y} - \lfloor np_{i_x i_y} \rfloor$ sortiert und die n_r Individuen mit der größten Wahrscheinlichkeit $p'_{i_x i_y}$ werden ein weiteres Mal für die nächste Generation kopiert.

- `local_roulette_wheel`

  Beim lokalen Roulette-Rad-Verfahren werden die Selektionswahrscheinlichkeiten $p_{i_x i_y} = \frac{f_{i_x i_y}}{\sum_{j_y=1}^{\mu} f_{j_y}}$ mit $1 \leq i_x \leq n_x$ und $1 \leq i_y \leq n_y$ lokal bestimmt. Anschließend werden mit n_y Zufallsexperimenten je PE die Individuen für die nächste Generation ausgewählt.

- `partner_selection`

  Um ein Crossover durchzuführen, werden zwei Individuen benötigt. Folgende Partner-Auswahlverfahren wurden implementiert.

  - `external_random_partner`

    Für jedes Individuum mit Index i wird ein Partner zufällig aus den Individuen auf anderen PEs, ebenfalls mit Index i ausgewählt.

  - `external_adjacent_partner`

    Für jedes Individuum mit Index i wird als Partner das Individuum auf dem nächsten PE, ebenfalls mit Index i ausgewählt.

  - `internal_random_partner`

    Für jedes Individuum mit Index i wird als Partner ein Individuum zufällig aus den Individuen ausgewählt, die sich auf demselben PE befinden.

  - `internal_adjacent_partner`

    Für das Individuum mit Index i wird als Partner das Individuum mit Index $i+1$ desselben PEs bestimmt, falls i ungerade ist, sonst wird als Partner das Individuum mit Index $i-1$ bestimmt. Nur bei dieser Art der Partnerwahl werden beim Crossover zwei Nachkommen erzeugt.

- `exchange_method`

  Über den Parameter `exchange_method` kann bestimmt werden, wie Individuen zwischen benachbarten PEs über den Kommunikationsring ausgetauscht werden. Dabei sind folgende Einstellungen möglich:

  - `shift_best_worst`

    Auf jedem PE wird das beste Individuum ermittelt. Dieses wird auf das folgende PE transportiert und ersetzt dort das schlechteste Individuum.

  - `shift_conditional_best`

Auch hier wird auf jedem PE das beste Individuum ermittelt und auf den Nachbar-PE transportiert. Jedoch ersetzt es nur dann das schlechteste Individuum auf dem Nachbar-PE, wenn es einen höheren Fitneßwert als dieses hat.

- `left_right_best_worst`

  Zuerst wird das beste Individuum auf jedem PE bestimmt. Dann werden die jeweils besten Individuen mit rechtem und linkem Nachbar-PE ausgetauscht. Auf jedem PE wird schließlich das schlechteste Individuum durch das beste Indivduum des linken und rechten Nachbarn ersetzt.

- `left_right_conditional_best`

  Auf jedem PE wird das beste Individuum bestimmt und mit rechtem und linkem Nachbar ausgetauscht. Jedoch ersetzt das beste Individuum des linken und rechten Nachbarn nur dann das schlechteste Individuum, wenn es eine höhere Fitneß als dieses hat.

- `number_of_exchanges`

  Der Parameter `number_of_exchanges` gibt an, wie oft der Austausch guter Individuen innerhalb einer Generation vorgenommen werden. Steht der Parameter auf Null, dann wird kein Austausch guter Individuen vorgenommen. Mit einem Wert größer als Null können gute Individuen über größere Entfernungen ausgetauscht werden.

- `topology`

  Gibt die Topologie für Random-Walk- und Tournament-Selektion an.

  - `rectangular`

    Bei der Rechteck-Topologie sind die einzelnen Prozessoren im Rechteck angeordnet. Die Breite des Rechtecks wird durch den Parameter `width_of_topology` bestimmt. Die Höhe des Rechtecks ergibt sich durch die Anzahl der benutzten Prozessoren. Also kann bei 512 PEs ein Rechteck aus 32x16 Prozessoren erzeugt werden, wenn der Parameter `width_of_topology` auf 16 eingestellt wird. Mit zwei Individuen je PE kann ein 32x32 Quadrat erzeugt werden. Die Individuen je Prozessor sind vertikal angeordnet.

- `width_of_topology`

  Gibt die Breite der Topologie `rectangular` an.

- `length_of_random_walk`

  Gibt die Weglänge des Random-Walks an.

- `cutoff_parameter_mu`

  Bestimmt den Parameter μ für proportionale Selektion und uniformes Ranking mit Kopieren.

- `scaling_constant_for_roulette`

  Gibt die Anzahl der zu erwartenden Nachkommen eines Individuums mit maximaler Fitneß bei der proportionalen Selektion an.

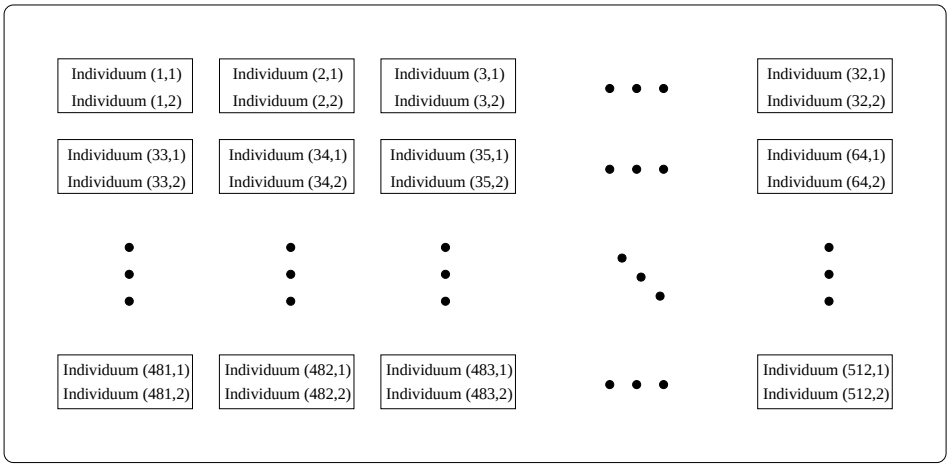

Abbildung 10.1: Verteilung der Individuen bei einer 32x32 Rechteck-Topologie

- `winston_ranking_probability`

  Gibt die Wahrscheinlichkeit an, die von der Einstellung `winston_rank` verwendet wird, um die Selektionswahrscheinlichkeiten zu bestimmen.

10.4.4 Operatoren

- `crossover_points`

  Mit dem Parameter `crossover_points` wird angegeben, an wievielen Crossoverpunkten ein Individuum aufgetrennt wird, wenn ein Crossover durchgeführt wird. Wird der Parameter `crossover_points` auf Null gesetzt, dann wird ein Uniformes Crossover durchgeführt.

- `crossover_between_words`

  Gibt an, ob Crossover innerhalb eines Wortes durchgeführt werden darf oder nicht. Steht der Schalter auf `OFF`, dann befindet sich keine Crossover-Position innerhalb eines Wortes. Diese Crossover-Art ist für Codierungen geeignet, bei denen ein Crossover innerhalb eines Wortes einen ungültigen Genotyp erzeugen könnte.

- `mutation_method`

  Definiert das Mutations-Verfahren. Es stehen folgende Mutations-Verfahren zur Auswahl:

  - `mutation_on_bits`

    Mutation einzelner Bits. Die Mutations-Wahrscheinlichkeit p_{mut} wird durch den Parameter `mutation_rate` angegeben. Sie definiert, wie wahrscheinlich es ist, daß ein Bit des Individuums mutiert.

  - `mutation_on_words`

    Mutation von ganzen Worten. Für jeden Bitblock wird mit der Wahrscheinlichkeit p_{mut} entschieden, ob der Bitblock mit einer neuen Bitfolge belegt wird. Dieser Mutations-Operator ist für Codierungen der Individuen gedacht, bei denen ein Allel

drei oder mehr Werte annehmen kann, und die Anzahl der Allelen keine Zweierpotenz ist. In diesem Fall würde eine "gepackte" Codierung mit einem Mutations-Operator, der auf einzelnen Bits operiert, ungültige Individuen erzeugen. Der Erwartungswert der mutierten Bits beträgt hier $16np_{\text{mut}}$, da in jedem Block durchschnittlich 16 Bits mutiert werden.

- `accumulated_mutation_on_bits`

  Bei sehr kleinen Mutations-Wahrscheinlichkeiten $p_{\text{mut}} \ll 1$ ist es effizienter, die Anzahl der zu mutierenden Bits zu akkumulieren, dann anschließend die Mutationen direkt durchzuführen. Es sei $m_i < 0$ die Anzahl der akkumulierten Mutationen aus der Generation i. In Generation $i+1$ wird dann zunächst die Anzahl der zu mutierenden Bits akkumuliert $m_{i+1} = m_i + 16np_{\text{mut}}$. Falls $m_i > 0$ ist, Dann werden $\lfloor m_{i+1} \rfloor$ Mutationen auf den Bits des Individuums durchgeführt und $m_{i+1} = m_{i+1} - \lfloor m_{i+1} \rfloor$ gesetzt.

- `accumulated_mutation_on_words`

  Bei sehr kleinen Mutations-Wahrscheinlichkeiten $p_{\text{mut}} \ll 1$ ist es effizienter, die Anzahl der zu mutierenden Bitblöcke zu akkumulieren, dann anschließend die Mutationen direkt durchzuführen. Es sei $m_i < 0$ die Anzahl der akkumulierten Mutationen aus der Generation i. In Generation $i+1$ wird dann zunächst die Anzahl der zu mutierenden Bitblöcke akkumuliert $m_{i+1} = m_i + np_{\text{mut}}$. Falls $m_i > 0$ ist, dann werden $\lfloor m_{i+1} \rfloor$ Mutationen auf den Bitblo̊cken des Individuums durchgeführt und $m_{i+1} = m_{i+1} - \lfloor m_{i+1} \rfloor$ gesetzt.

- `crossover_rate`

  Gibt die Wahrscheinlichkeit an, mit der der Crossover-Operator auf ein Individuum angewandt wird.

- `mutation_rate`

  Gibt die Mutations-Wahrscheinlichkeit für den Mutations-Operator an. Je nach Parametereinstellung `mutation_method` wird die Wahrscheinlichkeit der Mutation eines einzelnen Bits oder der eines ganzen Wortes angegeben.

10.4.5 Bericht

- `verbose`

  Steht der Parameter `verbose` auf `ON` und der Parameter `report` auf `normal`, dann werden die ermittelten Statistikdaten auch direkt ausgegeben. Nach jeder Ausgabe kann mit `quit` die Ausgabe beendet werden, oder mit `<RETURN>` fortgesetzt werden.

- `report`

  Mit dem Parameter `report` wird die Art der Statistikdaten eingestellt, die vom genetischen Algorithmus ermittelt werden.

  - `no_report`

    Es werden keinerlei Statistikdaten ermittelt. Lediglich das Individuum mit der höchsten Fitneß wird bei Beendigung des genetischen Algorithmus ausgegeben. Diese Einstellung sollte für Laufzeitmessungen verwendet werden.

- `normal`

  Erzeugt Statistikdaten und generiert die Individuen- und Protokolldatei `Protokoll-Datei.GAind` und `Protokoll-Datei.GApro` in der Form, wie sie von Baumann [Baumann 95] und Hummler [Hummler 95] in Absprache mit dem Betreuer Zell entwickelt wurde. Dabei steht Protokoll-Datei für den mit dem Parameter `protocol_data_file` eingestellten Dateinamen.

- `population`

  Mit der Einstellung `population` kann die Regionenbildung der Fitneßwerte bei der eingestellten Topologie verfolgt werden. Dazu werden dreidimensionale Gnuplot-Daten unter dem Namen `Protokoll-Datei.Gx.pop.GP3D` erzeugt. Dabei steht Protokoll-Datei für den mit dem Parameter `protocol_data_file` eingestellten Dateinamen und `x` steht für die Generation, in der die Daten ermittelt wurden.

- `debug`

  Zur Entwicklung weiterer Module für den CNGA ist es wichtig, sich auch über nur intern verwendete Variablen zu informieren. Ein umfaßendes Bild der internen Variablen erhält man mit der Einstellung `debug`. Die Debug-Informationen werden als ASCII-Datei unter dem Namen `Protokoll-Datei.GAdeb` erzeugt. Dabei steht Protokoll-Datei für den mit dem Parameter `protocol_data_file` eingestellten Dateinamen.

- `protocol_interval`

  Der Parameter `protocol_interval` gibt die Anzahl der Generationen an, die verstreichen, bis die Statistikdaten erneut ermittelt und ausgegeben werden.

10.4.6 Abbruchkriterien

- `generations`

  Gibt die Anzahl der durchzuführenden Generationen an.

- `supposed_max_fitness`

  Mit dem Parameter `supposed_max_fitness` wird die vermutlich größte zu erreichende Fitneß angegeben. Dieser Wert wird von der Statistik "Differenz zum Maximum" verwendet. Außerdem wird mit diesem Parameter der Wert bestimmt, bei dessen Überschreiten der genetische Algorithmus abgebrochen wird, wenn der Parameter `abort_criteria_maximum` auf `ON` steht.

- `bit_convergence`

  Der Parameter `convergence` gibt an, ab wieviel Prozent ein Bit als konvergiert gilt.

- `diversity`

  Der Parameter `diversity` gibt den kleinsten Wert der Diversität für das Abbruchkriterium Diversität an.

- `abort_criteria_maximum`

  Steht der Schalter auf `ON`, dann wird der genetische Algorithmus beendet, wenn der durch den Parameter `supposed_max_fitness` angegebene Fitneßwert überschritten wird.

- `abort_criteria_convergence`

  Wenn der Schalter auf `ON` steht, dann wird der genetische Algorithmus beendet, wenn alle Bits eines jeden Individuums konvergiert sind. Ab wann ein Bit als konvergiert gilt, wird durch den Parameter `bit_convergence` bestimmt.

- `abort_criteria_diversity`

  Steht der Schalter `abort_criteria_diversity` auf `ON`, dann wird beim Unterschreiten der durch den Parameter `diversity` angegebenen Diversität der genetische Algorithmus beendet.

10.4.7 Dateinamen

- `path_of_executable`

  Mit diesem Parameter muß der Pfad angegeben werden, unter dem das ausführbare Programm für den CNAPS-Server gespeichert ist.

- `settings_file`

  Der Parameter `settings_file` gibt an, von welcher Datei die Menüeinstellungen mit dem Kommando `load_settings` geladen und in welcher Datei die Menüeinstellungen mit dem Kommando `save_settings` gespeichert werden.

- `seed_file`

  Mit dem Parameter `seed_file` wird angegeben, von welcher Datei die Werte zur Initialisierung der Zufallszahlengeneratoren auf den PEs geladen werden, wenn der genetische Algorithmus mit dem Kommando `run` gestartet wird. Außerdem gibt dieser Parameter den Dateinamen für die Kommandos `generate_seeds` und `display_seeds` an.

- `summary_data_file`

  Der Parameter `summary_data_file` gibt den Dateinamen an, unter dem die Übersichtsdatei gespeichert wird. Dazu wird an den angegebenen Dateinamen noch `.GAsum` angehängt.

- `protocol_data_file`

  Der Parameter `protocol_data_file` gibt den Dateinamen an, unter der die Informationen über einen Lauf des genetischen Algorithmus gespeichert werden. Dabei wird die Endung `.GApro` für die Protokolldatei und die Endung `.GAind` für die Individuen-Datei verwendet, wenn der Parameter `report` auf `normal` eingestellt ist. Außerdem werden unter demselben Dateinamen die Gnuplotdaten zur Darstellung der maximalen, durchschnittlichen und minimalen Fitneß und die Gnuplotdaten zur Darstellung der Konvergenz, der Diversität, der Inzucht und des Inzuchtkoeffizienten gespeichert. Dazu werden jeweils die Endungen `.max.GP2D`, `.avg.GP2D`, `.min.GP2D`, `.con.GP2D`, `.div.GP2D`, `.inb.GP2D`, `.ico.GP2D` angehängt. Steht der Parameter `report` auf `debug`, dann werden unter dem angegebenen Dateinamen mit der Endung `.GAdeb` umfangreiche Debuginformationen gespeichert. Ist mit dem Parameter `report` die Funktion `population` ausgewählt, dann werden mit der Endung `.Gx.pop.GP3D` die dreidimensionalen Gnuplotdaten abgespeichert, wobei `x` für die Generationsnummer steht.

10.5 Kommandos

- `quit`

  Mit dem Kommando `quit` wird das Programm CNGA verlassen. Offene Fenster werden geschlossen und eine eventuell zum CNAPS-Server bestehende Verbindung gelöst.

- `menu`

  Das Menü mit allen Einstellungen und Kommandos wird angezeigt.

- `help`

  Eine kurze Information, wie Hilfe zu einzelnen Kommandos zu erhalten ist, wird angezeigt. Durch Eingabe von `help <Parameter>` oder `help <Kommando>` wird der Hilfetext zum entsprechenden Parameter oder Kommando ausgegeben. Der Hilfetext für Parameter enthält auch die für die Eingabe zulässigen Werte.

- `run`

  Startet den genetischen Algorithmus mit den derzeitigen Parametereinstellungen auf dem CNAPS-Server.

- `step`

  Nachdem zuvor der genetische Algorithmus mit dem Kommando `run` gestartet wurde, kann mit `step` eine weitere Generation des genetischen Algorithmus berechnet werden.

- `continue`

  Mit `continue` kann der genetische Algorithmus fortgesetzt werden. Dabei sollte vorher das Abbruchkriterium neu eingestellt werden. Mit `continue` werden nochmals soviel Generationen, wie durch den Parameter `generations` vorgegeben, berechnet bzw. bis das durch `abort_criteria_maximum`, `abort_criteria_convergence` oder `abort_criteria_diversity` vorgegebene Abbruchkriterium erfüllt ist.

- `connect_to_server`

  Mit `connect_to_server` wird eine Verbindung zu dem CNAPS-Server aufgebaut, der über den Parameter `server_name` eingestellt ist.

- `disconnect_from_server`

  Durch das `disconnect_from_server` kann die Verbindung zum CNAPS-Server gelöst werden. Jetzt kann ein anderer Anwender den Server benutzen. Verläßt der Anwender das Programm, so wird automatisch ein `disconnect_from_server` durchgeführt.

- `print_statistics`

  Mit dem Kommando `print_statistics` wird ein Eintrag für den zuletzt ausgeführten Lauf des CNGA in der Sammel-Datei erzeugt. Der genetische Algorithmus muß also mit dem Kommando `run` und der Berichtart `normal` mindestens einmal gestartet worden sein.

- `save_settings`

  Speichert die Einstellungen des Menüs in der durch den Parameter `settings_file` angegebenen Datei mit der Endung `.GAset` .

- `load_settings`

  Lädt die Einstellungen des Menüs in der durch den Parameter `settings_file` angegebenen Datei mit der Endung `.GAset` .

- `generate_seeds`

  Erzeugt eine durch den Parameter `seed_file` angegebene Datei mit der Endung `.GAseed`, in der die Saatkörner je PE für den Zufallszahlengenerator gespeichert sind.

- `display_seeds`

  Zeigt die in der durch den Parameter `seed_file` angegebenen Datei mit der Endung `.GAseed` enthaltenen Saatkörner an.

- `plot_fitness`

  Öffnet ein Fenster mit der Gnuplot-Ausgabe, falls zuvor Fitneßstatistikdaten durch CNGA erzeugt wurden. Als Dateinamen wird der Parameter `protocol_data_file` mit den Endungen `.max.GP2D`, `.avg.GP2D` und `.min.GP2D` verwendet. In diesem Fenster werden auch bei zukünftigen Läufen die Fitneßstatistikdaten angezeigt.

- `plot_convergence`

  Öffnet ein Fenster mit der Gnuplot-Ausgabe, falls zuvor Konvergenzstatistikdaten durch CNGA erzeugt wurden. Als Dateinamen wird der Parameter `protocol_data_file` mit den Endungen `.con.GP2D`, `.div.GP2D`, `.inb.GP2D` und `.ico.GP2D` verwendet. In diesem Fenster werden auch bei zukünftigen Läufen die Konvergenzstatistikdaten angezeigt.

Kapitel 11

Zusammenfassung und Ausblick

Für den Neurocomputer CNAPS wurde ein genetischer Algorithmus, genannt CNGA, entwickelt. Dazu wurde zunächst geprüft, inwieweit sich genetische Algorithmen zur Implemenierung auf einem Computer mit Ganzzahlenarithmetik eignen. Es wurde der Kompromiß, die Fitneßwerte durch Fließkommazahlen zu realisieren, getroffen. Dazu wurde eine Bibliothek entwickelt, die die erforderlichen Routinen für die Fließkommazahlen bereitstellt.

Als nächstes wurde ein Ablaufschema des genetischen Algorithmus entwickelt, der sowohl den Bus als auch die 1D-Topologie zur Kommunikation nutzt. Diese beiden Mechanismen wurden separiert und sind somit gleichzeitig einsetzbar. Bei der Implementierung des genetischen Algorithmus wurde auf einen effizienten Datenfluß geachtet, der unnötiges Kopieren der Individuen vermeidet. Die 16-Bit Architektur des Neurocomputers wurde durch ein 16-Bit Crossover und einen kompakten Genotyp voll ausgenutzt. Auf eine maximale Parallelisierung des Algorithmus wurde geachtet.

Das Programm CNGA könnte mit weiteren Operatoren und Mechanismen erweitert werden. Das Programm CNGA ist aber bereits in der jetzigen Form so umfangreich geworden, daß nur ausgewählte Programmteile übersetzt werden können. Somit können nicht alle Teile des Quellcodes gleichzeitig verwendet werden. Eine Erweiterung des Codes ist also nur eingeschränkt möglich.

Schließlich wurden die implementierten Mechanismen mit Benchmark-Problemen analysiert. Die Laufzeiten des Programms CNGA wurden mit anderen parallelen Implementierungen genetischer Algorithmen verglichen. Die Ergebnisse zeigten, daß Neurocomputer trotz ihrer Ganzzahlenarithmetik hervorragend zur Optimierung mittels genetischen Algorithmen eingesetzt werden können. Besonders deutlich wurde dies durch die geringen Laufzeiten von CNGA im Vergleich zu den anderen Implementierungen.

Die Schwierigkeiten beim Problem des Handlungsreisenden zeigten jedoch die Grenzen des Algorithmus auf. Daher könnte in weiteren Arbeiten untersucht werden, für welche realen Applikationen (z.B. zur Bestimmung molekularer Ähnlichkeit [Wagener 93]) der genetische Algorithmus auf dem Neurocomputer CNGA eingesetzt werden kann. Die Voraussetzung um weitere Zielfunktionen zu integrieren wurden durch die vorliegende Arbeit geschaffen.

Anhang A

Zielfunktionen

Folgende Zielfunktionen [Bäck 92] wurden für das Programm CNGA implementiert.

1. Sphere Model:

$$f_1(\vec{x}) = \sum_{i=1}^{n} x_i^2$$

$$-5.12 < x_i < 5.12$$

$$\min(f_1) = f_1(0, \ldots, 0) = 0$$

2. Generalized Rosenbrock's Function:

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

$$-5.12 < x_i < 5.12$$

$$\min(f_2) = f_2(1, \ldots, 1) = 0$$

3. Step Function:

$$f_3(\vec{x}) = 6n + \sum_{i=1}^{n} \lfloor x_i \rfloor$$

$$-5.12 < x_i < 5.12$$

$$\min(f_3) = f_3([-5.12, -5], \ldots, [-5.12, -5]) = 0$$

4. Quartic Function with Noise:

$$f_4(\vec{x}) = \sum_{i=1}^{n} i x_i^4 + \text{gauss}(0, 1)$$

$$-1.28 < x_i < 1.28$$

$$\min(f_4) = f_4(0, \ldots, 0) = 0$$

5. Shekel's Foxholes:

$$\frac{1}{f_5(\vec{x})} = \frac{1}{500} + \sum_{j=1}^{25} \frac{1}{j + \sum_{i=1}^{2}(x_i - a_{ij})^6}$$

$$(a_{ij}) = \begin{pmatrix} -32 & -16 & 0 & 16 & 32 & -32 & \cdots & 0 & 16 & 32 \\ -32 & -32 & -32 & -32 & -32 & -16 & \cdots & 32 & 32 & 32 \end{pmatrix}$$

$$-65.536 < x_i < 65.536$$

$$\min(f_5) = f_5(-32, -32) \approx 1$$

6. Schwefel's Function 1.2:

$$f_6(\vec{x}) = \sum_{i=1}^{n} \left(\sum_{j=1}^{i} x_j \right)^2$$

$$-65.536 \leq x_i < 65.536$$

$$\min(f_6) = f_6(0, \ldots, 0) = 0$$

7. Generalized Rastrigin's Function:

   Nicht implementiert [cos].

8. Sphere Model, Changing Environment:

$$f_8(\vec{x}(t)) = \begin{cases} \sum_i^n x_i^2(t) & t \text{ gerade} \\ \sum_i^n (x_i - 4)^2 & t \text{ ungerade} \end{cases}$$

$$-5.12 < x_i < 5.12$$

$$\min(f_8) = \begin{cases} f_8(0, \ldots, 0) = 0 & t \text{ gerade} \\ f_8(4, \ldots, 4) = 0 & t \text{ ungerade} \end{cases}$$

9. Ackley's Function:

   Nicht implementiert [$\sqrt{\ }$, e].

10. Krolak's 100 City TSP:

$$f_{10}(\vec{x}) = \sum_{i=1}^{n} d(c_{\pi(i \mod n)}, c_{\pi((i+1) \mod n)})$$

$$0 \leq x_i < 65536$$

$$\min(f_{10}) = 21285$$

11. Low Autocorrelation Binary Sequences:

$$f_{11} = (\vec{a}) = \sum_{k=1}^{l-1} \left(\sum_{i=1}^{l-k} \beta_i \beta_{i+k} \right)^2 ; \beta_i = \begin{cases} -1 & , \alpha_i = 0 \\ 1 & , \alpha_i = 1 \end{cases}$$

12. Hamming Distance to $\vec{0}$:

$$f_{12}(\vec{a}) = \sum_{i=1}^{l} \alpha_i$$

$$\min(f_{12}) = f_{12}(0, \ldots, 0) = 0$$

13. Weierstrass-Mandelbrot Fractal Function:

    Nicht implementiert [cos, x^y].

14. Fully Deceptive Function:

$$f_{14}(\vec{a}) = \begin{cases} 2^{-l} & , f_{12}(\vec{a}) = 0 \\ \frac{1+f_{12}(\vec{a})}{l} & , 0 < f_{12}(\vec{a}) < l \\ 0 & , f_{12}(\vec{a}) = l \end{cases}$$

$$\min(f_{14}) = f_{14}(1, \ldots, 1) = 0$$

15. Weighted Sphere Model:

$$f_{15}(\vec{x}) = \sum_{i=1}^{n} i x_i^2$$

$$-5.12 < x_i < 5.12$$

$$\min(f_{15}) = f_{15}(0, \ldots, 0) = 0$$

16. Fletcher and Powell:

    Nicht implementiert [sin, cos, Genauigkeit]

17. Fletcher and Powell:

    Nicht implementiert [sin, cos, Genauigkeit]

18. Shekel-5:

$$f_{18}(\vec{x}) = -\sum_{i=1}^{m} \frac{100}{(x - A(i))(x - A(i))^T + c_i}$$

$$n = 4; m = 5; 0.0 \leq x_i < 100.0$$

i	A(i)				c_i
1	40	40	40	40	10
2	10	10	10	10	20
3	80	80	80	80	20
4	60	60	60	60	40
5	30	70	30	70	40

$$\min(f_{18}) = f_{18}(0, \ldots, 0) = 0$$

19. Shekel-7:

$$f_{19}(\vec{x}) = -\sum_{i=1}^{m} \frac{100}{(x - A(i))(x - A(i))^T + c_i}$$

$$n = 4; m = 7; 0.0 \leq x_i < 100.0$$

i	A(i)				c_i
1	40	40	40	40	10
2	10	10	10	10	20
3	80	80	80	80	20
4	60	60	60	60	40
5	30	70	30	70	40
6	20	90	20	90	60
7	50	50	30	30	30

$$\min(f_{19}) = f_{19}(0, \ldots, 0) = 0$$

20. Shekel-10:

$$f_{20}(\vec{x}) = -\sum_{i=1}^{m} \frac{100}{(x - A(i))(x - A(i))^T + c_i}$$

$$n = 4; m = 10; 0.0 \leq x_i < 100.0$$

i	A(i)				c_i
1	40	40	40	40	10
2	10	10	10	10	20
3	80	80	80	80	20
4	60	60	60	60	40
5	30	70	30	70	40
6	20	90	20	90	60
7	50	50	30	30	30
8	80	10	80	10	70
9	60	20	60	20	50
10	70	36	70	36	50

$$\min(f_{20}) = f_{20}(0, \ldots, 0) = 0$$

21. Griewank:

    Nicht implementiert [cos, $\sqrt{\ }$, Genauigkeit].

22. Griewank:

    Nicht implementiert [cos, $\sqrt{\ }$, Genauigkeit].

23. Galar:

    Nicht implementiert [e^x].

24. Kowalik:

    Nicht implementiert [Genauigkeit].

Literaturverzeichnis

[Adaptive Solutions 93a] Adaptive Solutions. Getting Acquainted with CNAPS, Oct 1993.

[Adaptive Solutions 93b] Adaptive Solutions. Getting Started with CodeNet, Aug 1993.

[Adaptive Solutions 93c] Adaptive Solutions. Learning CPL, Oct 1993.

[Adaptive Solutions 94a] Adaptive Solutions. CNAPS-C Language Reference, Jul 1994.

[Adaptive Solutions 94b] Adaptive Solutions. CNAPS-C Programming Guide, Jul 1994.

[Adaptive Solutions 94c] Adaptive Solutions. CPL Programming Guide, Oct 1994.

[Adaptive Solutions 94d] Adaptive Solutions. Writing CodeNet Applications in C, Jan 1994.

[Almasi et al. 94] George S. Almasi and Allan Gottlieb. Highly Parallel Computing. The Benjamin/Cummings Publishing Company, Inc., Redwood City, California, 1994.

[Appelrath et al. 91] Hans Jürgen Appelrath und Jochen Ludewig. Skriptum Informatik - eine konventionelle Einführung. B. G. Teubner, Stuttgart, 1991.

[Bäck 92] Thomas Bäck. A User's Guide to GENEsYs 1.0. Technical report, University of Dortmund, Department of Computer Science, Jul 1992.

[Baumann 95] Roland Baumann. Verteilte genetische Algorithmen auf dem MIMD-Parallelrechner Intel Paragon. Diplomarbeit Nr. 1227, Universität Stuttgart, Fakultät Informatik, IPVR, Mai 1995.

[Bräunl 94] Thomas Bräunl. Parallele Programmierung: eine Einführung. Vieweg, Braunschweig, 1994.

[Bronstein et al. 89] Il'ja N. Bronštein und Konstantin A. Semendjajew. Taschenbuch der Mathematik. Verlag Harri Deutsch, Thun und Frankfurt/Main, 24. Edition, 1989.

[Cormen et al. 90] Charles E. Leiserson und Ronald L. Rivest Thomas H. Cormen. Introduction to Algorithms. The MIT Press, Cambridge, Massachusetts, 1990.

[Davis et al. 83] Martin D. Davis and Elaine J. Weyuker. Computability, Complexity, and Languages. Academic Press Limited, San Diego, California, 1983.

[Dorigo et al. 93] Marco Dorigo and Vittorio Maniezzo. Parallel Genetic Algorithms: Introduction and Overview of Current Research, p.5-42, in: Joachim Stender (eds.): Parallel Genetic Algorithms: Theory and application IOS Press, Amsterdam, 1993.

[Görzig 95] Steffen Görzig. Massiv parallele Evolutionsstrategien auf dem Parallelrechner MasPar MP-1. Diplomarbeit Nr. 1291, Universität Stuttgart, Fakultät Informatik, IPVR, 1995.

[Goldberg 89] David E. Goldberg. Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley Publishing Company, Reading, Massachusetts, 1989.

[Goldberg 90] David Goldberg. Computer Arithmetic; p. A-1 - A-66. in: John L. Hennessy and David A. Patterson. Computer Architecture: A Quantitative Approach. Morgan Kaufmann Publishers, Inc., San Mateo, California, 1990.

[Hasel 95] Alexander Hasel. Eine graphische Oberfläche für Parallele Genetische Algorithmen und Evolutionsstrategien. Diplomarbeit, Universität Stuttgart, Fakultät Informatik, IPVR, 1995.

[Hennessy et al. 90] John L. Hennessy and David A. Patterson. Computer Architecture: A Quantitative Approach. Morgan Kaufmann Publishers, Inc., San Mateo, California, 1990.

[Holland 92] John H. Holland. Adaptation in natural and artifical systems: an inroductory analysis with applications to biology, control, and artificial intelligence. The MIT Press, Cambridge, Massachusetts, 1992.

[Hopcroft 90] John E. Hopcroft und Jeffrey D. Ullman. Einführung in die Automatentheorie, formale Sprachen und Komplexitätstheorie. Addison-Wesley (Deutschland) GmbH, Bonn, 1990.

[Hummler 95] Alex Hummler. Massiv parallele genetische Algorithmen auf dem Parallelrechner MasPar MP-1. Diplomarbeit Nr. 1240, Universität Stuttgart, Fakultät Informatik, IPVR, Jun 1995.

[Kernighan et al. 83] Brian W. Kernighan and Dennis M. Ritchie. Programmieren in C mit dem C-Reference Manual in deutscher Sprache. Carl Hanser Verlag, München Wien, 1983.

[Rechenberg 73] Ingo Rechenberg. Evolutionsstrategie. Friedrich Frommann Verlag, Stuttgart, 1973.

[Rechenberg 94] Ingo Rechenberg. Evolutionsstrategie '94. Friedrich Frommann Verlag, Stuttgart, 1994.

[Schöneburg et al. 94] Eberhard Schöneburg, Frank Heinzmann und Sven Feddersen. Genetische Algorithmen und Evolutionsstrategien. Addison-Wesley (Deutschland) GmbH, Bonn, 1994.

[Schwehm et al. 93] Markus Schwehm, Karl Dieter Reinartz, Thomas Walter, Sönke-Sonnich Gold, Christoph Schäftner, Thilo Opaterny, Alexander Ost und Norbert Engst. Massiv Parallele Genetische Algorithmen. Interner Bericht, Friedrich-Alexander-Universität Erlangen-Nürnberg, 1993.

[Siegmund 92] Frederik Siegmund. Einfluß der Kommunikationstopologie auf massiv paralle Genetische Algorithmen. Diplomarbeit, Friedrich-Alexander-Universität Erlangen-Nürnberg, Aug 1992.

[Tanenbaum 89] Andrew S. Tanenbaum. Computer Networks. Prentice-Hall International (UK) Limited, London, second edition, 1989.

[Sun 89] lrand48 Manual Page. C Library Functions. Sun Release 4.1, Feb 1989.

[Wagener 93] Markus Heinrich Wagener. Bestimmung molekularer Ähnlichkeit mit Hilfe eines Genetischen Algorithmus. Doktorarbeit, Technische Universität München, Fakultät für Chemie, Biologie und Geowissenschaften, 1993.

[Wakunda 95a] Jürgen Wakunda. Verteilte Evolutionsstrategien auf dem Supercomputer Intel Paragon. Diplomarbeit, Universität Stuttgart, Fakultät Informatik, IPVR, 1995.

[Wakunda 95b] Jürgen Wakunda. Statistikdaten des Programms VEGA. Private E-Mail an Marc Ebner, Oct 1995

[Winston 92] Patrick Henry Winston. Artificial Intelligence. Addison-Wesley Publishing Company, Reading, Massachusetts, third edition, 1992.

[Zell et al. 95a] Andreas Zell and Roland Baumann. Distributed Genetic Algorithms on the Intel Paragon, a large MIMD-Computer. Universität Stuttgart, IPVR, 1995.

[Zell et al. 95b] Andreas Zell and Alex Hummler. Massively Parallel Genetic Algorithms on the SIMD-Computer MasPar MP-1. Universität Stuttgart, IPVR, 1995.

[Zell 94] Andreas Zell. Simulation Neuronaler Netze. Addison-Wesley (Deutschland) GmbH, Bonn, 1994.