Universität Bremen

Fachbereich Informatik
Arbeitsgruppe Rechnerarchitektur

Testmusterkompaktierung mit Genetischen Algorithmen

Diplomarbeit

von

Michael Klemm

Erstprüfer:

Prof. Dr. Rolf Drechsler

Zweitprüfer:

Prof. Dr. Kerstin Schill

Betreuer:

Dipl. Inform. Görschwin Fey

Datum:

26. Oktober 2006

Abstract

The automatic test equipment (ATE) for the manufacturing test of digital integrated circuits requires test pattern sets. These automatically generated test sets are partially specified and highly repetitive, because the test vectors contain don't care (unspecified, X) entries. To reduce storage volume and test application time the objective is to minimize the test sets' size without reducing the fault coverage. In this paper three Genetic Algorithm approaches are introduced to compact the test sets.

Bedanken möchte ich mich bei:

- Prof. Dr. Rolf Drechsler für die gute Zusammenarbeit
- Prof. Dr. Kerstin Schill für die Zweitprüfertätigkeit
- Dipl. Inform. Görschwin Fey für die nette Betreuung
- Dipl.-Inf. André Sülflow für Anregungen und Hilfe
- Dipl.-Ing Uwe Forgber sowie Dipl.-Inf. Daniel Grosse für die technische Unterstützung
- Dr. Nicole Drechsler für die prompte Hilfe bei der Suche nach einem Diplomarbeitsthema
- Wiebke Paczkowski für ihre Geduld und das Korrekturlesen
- Iris Schneider für das Korrekturlesen

Erklärung

Hiermit erkläre ich, dass ich diese Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Die Arbeit wurde bisher keiner anderen Prüfungsbehörde vorgelegt und nicht veröffentlicht.

Bremen, den 26. Oktober 2006 Michael Klemm

Inhaltsverzeichnis

1 Einleitung **1**
1.1 Zielsetzung der Arbeit . . . 2
1.2 Gliederung der Arbeit . . . 3

2 Grundlagen **5**
2.1 Integrierte Schaltungen . . . 5
2.1.1 Testprozess . . . 5
2.1.2 Testmustergenerierung . . . 6
2.1.3 Testmusterkompaktierung . . . 7
2.1.4 Existierende Lösungsansätze . . . 9
2.2 Evolutionäre Algorithmen . . . 10
2.2.1 Biologische Evolutionstheorie . . . 10
2.2.2 Technische Optimierung nach dem Vorbild der biologischen Evolution 12
2.2.3 Genetische Algorithmen . . . 13

3 Umsetzung **17**
3.1 Lösungsansätze . . . 17
3.1.1 Greedy Methode . . . 18
3.1.2 MaxMin Methode . . . 19
3.1.3 Permutationsmethode . . . 20
3.2 Hilfsfunktionen . . . 23
3.3 Genetische Operatoren . . . 24
3.3.1 Selektion . . . 25
3.3.2 Ersetzung . . . 27
3.3.3 Rekombination . . . 28
3.3.4 Mutation . . . 30
3.3.5 2-Opt Mutation . . . 31
3.3.6 Fitnessfunktionen . . . 32
3.3.7 Abbruchkriterien . . . 33
3.4 Implementierung . . . 34
3.4.1 Effiziente Speichernutzung . . . 35
3.4.2 Geschwindigkeitsoptimierung . . . 35

4 Experimentelle Ergebnisse **37**
4.1 Testdaten . . . 37
4.2 Testumgebung . . . 39
4.3 Testverfahren . . . 39
4.4 Greedy Methode . . . 40
4.5 MaxMin Methode . . . 43
4.6 Permutationsmethode . . . 47
4.7 Vergleich der Methoden . . . 49

5 Zusammenfassung **51**

A D-Algorithmus **55**

Abbildungsverzeichnis **59**

Tabellenverzeichnis **61**

Literaturverzeichnis **63**

Stichwortverzeichnis **67**

1 Einleitung

Integrierte Schaltkreise findet man heute in allen Bereichen des täglichen Lebens.

Durch die rasante Entwicklung des Herstellungsprozesses steigt die Anzahl der Transistoren auf einem Chip stetig. Nach dem Mooreschen Gesetz[1] verdoppelt sich die Transistorzahl auf integrierten Schaltungen etwa alle zwölf Monate.

Es werden immer komplexere Schaltungen entworfen, die eine wachsende Funktionalität aufweisen. Steigende Rechenleistung bei gleichzeitig minimalem Leistungsbedarf ermöglicht den Einsatz in einer wachsenden Zahl von Anwendungsgebieten.

Obgleich der Herstellungsprozess stark optimiert ist, weist ein Teil der produzierten Chips Defekte auf. So können beispielsweise kleinste Verunreinigungen des Basismaterials oder Schwankungen in der Zusammensetzung verwendeter Materialien zu Fehlern führen. Spätestens beim Einsatz der Schaltung in sicherheitsrelevanten Bereichen ist es notwendig, fehlerhafte Chips zu erkennen und auszusondern. Das geschieht am Ende des Herstellungsprozesses beim Produktionstest.

Aus betriebswirtschaftlicher Sicht wird der Erfolg einer integrierten Schaltung am erwirtschafteten Gewinn gemessen. Der Gewinn ist die Differenz aus den Mitteln der eingesetzten Ressource und dem Verkaufserlös der verwertbaren, also fehlerfreien Chips. Die eingesetzten Ressourcen umfassen dabei verwendete Materialien, Kosten für die Produktionsstätten, Personalkosten sowie Kosten für den Test. Eine wichtige Kenngröße zur Beurteilung des Fertigungsprozesses und der Wirtschaftlichkeit einer Produktionslinie wird als »Die Yield« bezeichnet. Mit

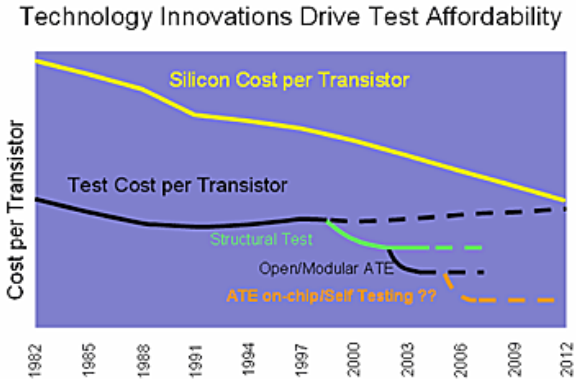


Abb. 1.1: Kostenentwicklung der Produktion und des Tests einer integrierten Schaltung bezogen auf einen Transistor (Quelle: http://www.intel.com/technology/itj/2005/volume09issue03/art03_testequipment/p02_intro.htm)

[1] Gordon E. Moore; amerikanischer Chemiker und Physiker; ∗ 3. Januar 1929 in San Francisco, Kalifornien, USA; Mooresches Gesetz: 1965 Artikel in der Fachzeitschrift Electronics: Verdoppelung der Transistoranzahl auf ICs alle 12 Monate. (Quelle: http://de.wikipedia.org/wiki/Gordon_Moore)

Yield[2] wird das Verhältnis fehlerfreier zur Gesamtzahl aller auf einem Wafer[3] vorhandenen Dice [4] bezeichnet.

Wie Abb. 1.1 zeigt, sinken die Produktionskosten, während die Testkosten annähernd konstant bleiben (bezogen auf einen Transistor einer integrierten Schaltung). Prozentual steigen die Testkosten in Bezug auf die Gesamtkosten. Schon jetzt tragen die Testkosten einen bedeutenden Anteil an den Gesamtkosten, in absehbarer Zeit werden sie 30% der Produktkosten erreichen [Tho96].

Die Gründe dafür liegen im Wesentlichen in der gestiegenen Integrationsdichte sowie den durch die technische Entwicklung des Herstellungsprozesses bedingten gestiegenen Fehlermöglichkeiten. Denn mit wachsender Integrationsdichte und somit steigender Komplexität nimmt der Zeitaufwand zum Testen exponentiell zu. Mit dem Zeitaufwand steigen die Testkosten, da weniger Schaltungen pro Zeiteinheit geprüft werden können.

Der Zeitaufwand entsteht aus der - entsprechend der Komplexität - großen Anzahl zu testender Muster, mit denen nacheinander die integrierte Schaltung geprüft wird. Dieser Zeitaufwand kann auch mit schnellerem Testequipment nicht kompensiert werden. Deshalb verringert man die Anzahl der Testmuster, bei gleichbleibender Anzahl detektierbaren Fehler. Ermöglicht wird das durch zusammenfassen kompatibler Testmuster.

Dieses Verfahren nennt man Testmusterkompaktierung.

1.1 Zielsetzung der Arbeit

Ausgangspunkt dieser Arbeit sind bereits generierte Testmustermengen. Diese liegen in einer logischen Repräsentation, bestehend aus den drei Symbolen »0«, »1« und »X«, vor. Die beiden Symbole »0« und »1« entsprechen den zwei Signalpegeln, »X« steht für einen undefinierten Pegel.

Jedes Testmuster besteht aus einer entsprechend der Anzahl an Datenleitungen langen Folge dieser Symbole. In der Regel wird ein Testmuster benötigt um einen Fehler der integrierten Schalung detektieren zu können.

Das Vorhandensein des undefinierten Symbols »X« ermöglicht es, Testmuster so zusammenzufassen, dass die Schaltung mit einem Testmuster gleichzeitig auf mehrere Fehler getestet werden kann. Dadurch können Zeit und Kosten gespart werden. Die Zielsetzung der Kompaktierung besteht darin, bei gleichbleibender Fehlerabdeckung die Anzahl der Testmuster maximal zu reduzieren.

[2] Yield: Ausbeute
[3] Wafer: Ca. 1 mm dünne Halbleiterscheibe, auf der über Lithographie, Ätz-, Dotier-, und Abscheidungsprozesse die Dice hergestellt werden.
[4] Die, plural Dice: Halbleiterplättchen ohne Gehäuse.

Das Problem der Testmusterkompaktierung ist jedoch nicht trivial! Mit zunehmender Länge und Anzahl der Testmuster steigt exponentiell die (Zeit-) Komplexität. Das Testmusterkompaktierungsproblem ist NP-vollständig[5] [PR96].

Genetische Algorithmen sind Optimierungsverfahren, die erfolgreich bei Problemen der Klasse NP eingesetzt werden. Deshalb wird in dieser Arbeit beschrieben, wie Genetsiche Algorithmen auf das Problem der Testmusterkompaktierung angewendet werden können.

Daraus ergibt sich die Zielsetzung dieser Arbeit:
Die Erstellung und Dokumentation eines Verfahrens, das mittels eines Genetischen Algorithmus eine Testmusterkompaktierung durchführt, d. h. die Testmustermenge so zusammenzufasst, dass die Zahl der kompaktierten Testmuster minimal wird, wobei sämtliche Testfälle erhalten bleiben.

1.2 Gliederung der Arbeit

Das folgende Grundlagenkapitel vermittelt Einblicke in den Test integrierter Schaltungen, eine Einführung in Genetische Algorithmen sowie einen Überblick bereits existierender Arbeiten zum Thema Testmusterkompaktierung. Es wird auf den Testprozess, die Testmustergenerierung und -kompaktierung eingegangen.

Die Umsetzung findet sich Kap. 3. Dort werden die erarbeiteten Lösungsansätze, die speziellen genetischen Operatoren und Details der erstellten Algorithmen beschrieben.

Zur Evaluierung werden die Algorithmen in Kap. 4 einem Leistungsvergleich unterzogen. Die Algorithmen werden untereinander und über standardisierte Testbenchmarks mit bestehenden Arbeiten verglichen.

Schließlich wird die Arbeit in Kap. 5 zusammengefasst.

[5] NP-vollständige Probleme sind solche, für die bisher kein Algorithmus entwickelt werden konnte, der sie in polynomieller Zeit berechnen kann. In der Komplexitätstheorie bezeichnet man ein Problem als in polynomieller Zeit lösbar, wenn die benötigte Rechenzeit einer deterministischen, sequentiellen Rechenmaschine mit der Problemgröße k nicht schneller als $O(n^k)$ wächst. Die besondere Bedeutung der Polynomialzeit besteht darin, dass man sie als eine Grenze zwischen praktisch lösbaren und praktisch nicht lösbaren Problemen betrachtet.

2 Grundlagen

Dieses Kapitel führt in die Thematik der Testmusterkompaktierung mit Genetischen Algorithmen ein.
Dazu wird im ersten Abschnitt der Verifikationsprozess integrierter Schaltungen, die Testmustergenerierung und -kompaktierung beschrieben. Bestehende Arbeiten zur Testmusterkompaktierung werden vorgestellt.
Der zweite Abschnitt vermittelt die Grundlagen Genetischer Algorithmen.

2.1 Integrierte Schaltungen

In diesem Kapitel wird zunächst auf den Testablauf eingegangen. Dabei wird das grundlegende Testverfahren skizziert. Es folgt ein Abschnitt über die Generierung von Testmustern. Danach wird auf die Testmusterkompaktierung, deren Anforderungen und Randbedingungen eingegangen. Der Abschnitt 2.1.4 bietet einen Überblick über existierende Lösungsansätze.

2.1.1 Testprozess

Integrierte Schaltungen, vor allem, wenn sie in sicherheitskritischen Bereichen Einsatz finden, durchlaufen am Ende des Herstellungsprozesses einen Funktionstest.

Ein Verfahren zum Testen integrierter Schaltungen verläuft vereinfacht folgendermaßen (vgl. Abb. 2.1):

An die Eingangspins des zu testenden Chips wird ein Eingabetestmuster angelegt. Ein Testmuster ist auf der physikalischen Ebene eine Folge von Schaltpegeln. Auf der Informationsebene wird er mit 0 (Spannungspegel Low), 1 (Spannungspegel High) und X (Undefiniert) dargestellt.
Daraufhin schalten die Transistoren des Testchips entsprechend den Eingangssignalen durch, so dass an den Ausgangspins ein resultierendes Ausgangsmuster anliegt. Dieses Ausgangsmuster wird mit dem errechneten Soll-Ausgangsmuster des korrespondierenden Eingangsmusters verglichen. Sind die Muster identisch, so ist der Test dieses Musters erfolgreich, der Testprozess wird mit dem nächsten Testmuster durchlaufen. Wenn alle Testmuster

erfolgreich getestet wurden, ist der Chip funktionsfähig. Differiert auch nur ein Ausgangssignal mit einem Soll-Ausgangssignals eines Testmusters, so liegt ein Fehler im getesteten Schaltkreis vor. Der fehlerhafte Schaltkreis wird ausgesondert.

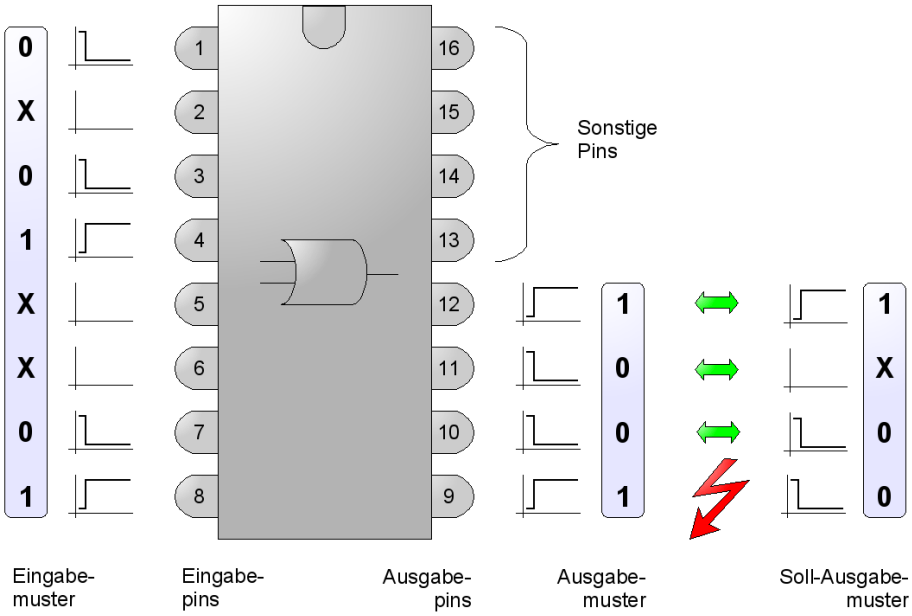


Abb. 2.1: Schematische Darstellung des Testprozesses. Ein Eingabetestmuster wird an die Eingangspins angelegt. Der integrierte Schaltkreis erzeugt daraus ein Ausgabemuster. Dieses wird mit einem berechneten Soll-Ausgabemuster verglichen.

2.1.2 Testmustergenerierung

Alle bekannten, allgemein gültigen Algorithmen zur automatischen Testmustergenerierung sind dem Typ nach NP-vollständig. Das heißt, der Aufwand wächst exponentiell mit der Schaltungsgröße.

Wenn man alle Kombinationen von Testmustern für eine gegebene Schaltung erzeugt, entstehen $2^{\text{Anzahl Signalleitungen}}$ Testmuster. Das ist für heutige integrierte Schaltungen bereits nicht effizient realisierbar.

Ein Rechenbeispiel eines aktuellen Prozessors soll das verdeutlichen:

Um die 100 externen Signalleitungen eines integrierten Schaltkreises zu testen, benötigt man $2^{100} = 1.267.650.600.228.229.401.496.703.205.376$ Testmuster. Wird die Schaltung mit 3 GHz getaktet, können $3.000.000.000 = 3 * 10^9$ Testmuster pro Sekunde getestet werden. Man braucht also $\frac{2^{100}\,\text{Testmuster}}{3{*}10^9\,\frac{\text{Testmuster}}{\text{Sek}} * 60\,\text{Sek.} * 60\,\text{Min} * 24\,\text{Std} * 365\,\text{Tage}} \approx 13.398.978.947.110$ Jahre, um die Schaltung mit allen möglichen Testmustern vollständig zu testen. Zum Vergleich: Das Alter der Erde wird auf 4,5 Milliarden Jahre geschätzt [Dal91]. Ein solcher Test ist also nicht praktikabel!

Da aber in der Regel bei weitem nicht alle möglichen Testmuster benötigt werden, um einen Fehler zu finden, bedient man sich so genannter Testmustergeneratoren. Das sind

Algorithmen, welche die relevanten Testmuster für eine Schaltung generieren, mit dem Ziel, eine maximale Fehlerabdeckung zu erreichen.

Neben der Testmustergenerierung durch Zufallszahlen kommt die algorithmische Generierung zum Einsatz. Voraussetzung für dieses Verfahren ist, dass eine Netzliste der integrierten Schaltung vorliegt.

Testmustergeneratoren sind komplexe Algorithmen, die ein Fehlermodell als Ausgangspunkt zu grunde legen. Es existieren eine Reihe von Fehlermodellen. Dazu gehören Verzögerungs-, Übergangs-, Brücken-, Crosstalk- und Stuck-open-Fehler [JG03, S 498ff]. Das klassische und in der Industrie am häufigsten eingesetzte Modell ist das Haftfehlermodell (stuck-at-Fehlermodell). Es wird hier benutzt, um die in dieser Arbeit zugrunde gelegten Ausgangsdaten zu berechnen. Die Vorteile dieses Modells sind, dass eine große Menge physikalischer Defekte abgedeckt werden, es technologieunabhängig und leicht anzuwenden ist.

Das Prinzip bei der Generierung ist, dass ein Punkt innerhalb der Schaltung fest auf logisch 0 bzw. 1 »geklebt« und so als defekt angenommen wird. Um diesen Defekt erkennen zu können, muss berechnet werden, durch welches Eingangsmuster der Defekt eingestellt werden kann. Ähnlich wird berechnet, ob der Fehler an den Ausgängen beobachtet werden kann.

Die Testmustergeneratoren erzeugen für eine gegebene Schaltung eine Menge von Testmustern wobei jedes Testmuster einen möglichen Fehler abdeckt. Die Testmuster bestehen aus einer Reihe von Symbolen der Länge n, die gleich der Anzahl an Signaldatenleitungen der Schaltung ist. Die Symbole nehmen die Werte 0, 1 und X an, deren Bedeutung bereits weiter oben erklärt wurde.

Heute existiert eine Vielzahl von Testmustergeneratoren, beispielsweise der von Roth entwickelte D-Algorithmus [Rot66], PODEM (Path Oriented DEcision Making) Algorithmus vorgestellt von Goel [Goe81], FAN [FS83] sowie SOKRATES [STS88].

Ein Beispiel zur Testmustergenerierung mit dem D-Algorithmus befindet sich im Anhang A.

2.1.3 Testmusterkompaktierung

Grundsätzlich unterscheidet man zwei Arten der Kompaktierung: Die dynamische und die statische [Mey03, S.86].

Bei der dynamischen Testmusterkompaktierung wird in den Prozess der Testmustergenerierung eingegriffen. Das erste Testmuster wird gemäß des Vorgehens des jeweiligen Testmustergenerators erzeugt. Dabei werden einige interne Leitungen sowie externe Ein- und Ausgänge unberührt bleiben. Nun werden weitere Defekte angenommen, die mit den berechneten Belegungen vereinbar sind. Damit wird das Testmuster so lange schrittweise

vervollständigt, bis alle externen Ein- und Ausgänge definiert sind. Mit diesem Verfahren können schon bei der Erzeugung mehrere Fehler mit einem Testmuster abgedeckt werden.

In dieser Arbeit wird eine statische Testmusterkompaktierung durchgeführt. Bei der statischen Testmusterkompaktierung liegt bereits eine Testmustermenge vor. Es ist dabei gleichgültig, durch welchen Testmustergenerator die Testmustermenge berechnet wurde.

Die zu kompaktierende Testmustermenge steht der Kompaktierung als Datei zur Verfügung. Dabei handelt es sich um eine Textdatei, in der jedes Testmuster eine Zeile belegt. Alle Zeilen und somit jedes einzelne Testmuster weisen die gleiche Länge auf. Das gleiche Format besitzt die von der Kompaktierung erzeugte Ausgabedatei.

Um Zeit und Kosten zu senken ist man bestrebt, die Anzahl der Testmuster zu minimieren. Dabei sollen alle Fehler weiterhin entdeckt werden können. Das kann erreicht werden, indem kompatible Testmuster zusammengefasst werden.

Wert des 1. Testmusters	Wert des 2. Testmusters	Wert des resultierenden Testmusters
0	0	0
0	1	Unvereinbar
0	X	0
1	0	Unvereinbar
1	1	1
1	X	1
X	0	0
X	1	1
X	X	X

Tab. 2.1: Kombinationsmöglichkeiten bei der Kompaktierung. Befindet sich an gleicher Stelle bei zwei Testmustern ein fester, unterschiedlicher Wert (0, 1) so können diese Testmuster nicht zusammengefasst werden.

Zwei Testmuster $X =< x_1, x_2, \ldots, x_n >$ und $Y =< y_1, y_2, \ldots, y_n >$ der Länge n können kombiniert werden, wenn sie zueinander kompatibel sind. Das ist genau dann der Fall, wenn die Werte aller Positionen miteinander kompatibel sind (vgl. Tab. 2.1), d. h. eines der Symbole ist ein X oder beide Werte sind identisch.

$$X \equiv Y \text{ genau dann wenn } \forall_i(x_i = X \text{ oder } y_i = X \text{ oder } x_i = y_i)$$

Beispielsweise sind die beiden Testmuster X und Y miteinander kompatibel.

```
X = 1XXX0X11XXX0XXX11XXX0011XXX
Y = XXX10XXX1110000111111XXXXX01
```

Kompatible Testmuster können zusammengefasst werden. Dazu werden die einzelnen Werte der Testmuster nach Tab. 2.1 kombiniert. Gleiche Werte bleiben erhalten, ein X und ein fester Wert resultieren in diesem festen Wert. Das kombinierte Testmuster Z der Testmuster X und Y aus obigem Beispiel ergibt:

```
Z = 1XX10X11111000011111110011X01
```

So können die beiden Fehler, die von X und Y abgedeckt werden, durch ein einziges Testmuster Z simultan detektiert werden.

Die Schwierigkeit besteht darin, die Testmuster so zusammenzufassen, dass eine minimale Testmustermenge entsteht. Das soll folgendes Beispiel verdeutlichen: Gegeben seien die vier Testmuster

$$T_1 = \texttt{XXX1XX}$$
$$T_2 = \texttt{XX1XXX}$$
$$T_3 = \texttt{XX01XX}$$
$$T_4 = \texttt{XX10XX}$$

Offensichtlich ist, dass die Testmuster T_3 und T_4 miteinander unvereinbar sind. T_1 kann mit T_2 und T_3, T_2 mit T_1 und T_4 zusammengefasst werden. Geht man beim Kompaktieren nach der Reihenfolge vor, so werden T_1 und T_2 kompaktiert. Daraus ergibt sich

$$T_{12} = \texttt{XX11XX}$$

Dieses Muster ist inkompatibel zu T_3 und T_4. Als Resultat ergeben sich also die drei Testmuster T_{12}, T_3 und T_4. Wird hingegen T_1 mit T_3 und T_2 mit T_4 kompaktiert, so ergibt sich die minimale Testmustermenge mit zwei Testmustern T_3 und T_4.

2.1.4 Existierende Lösungsansätze

Auf Grund der hohen praktischen Relevanz und dem großen Testkostensparpotential sind eine Reihe von Kompaktierungsmethoden publiziert worden. Um diese vergleichen zu können, werden standardisierte Vergleichstests zur Berechnung herangezogen. Die bekanntesten sind der ISCAS'85, ISCAS'89 und der ITC'99 Benchmark.

Einige ausgewählte Testmusterkompaktierungsverfahren werden hier in loser Reihenfolge vorgestellt.

Reddy et al. stellten 1992 ROTCO (Reverse Order Test COmpaction) vor [RPR92]. Dieses Verfahren betrachtet die Testmuster in der umgekehrten Generierungsreihenfolge. Dabei werden die X Werte systematisch ersetzt. Wird dabei ein anderes Testmuster abgedeckt, wird dieses aus der Liste gestrichen und so eine Kompaktierung erzielt.

Die Kompaktierungsmethode von Pomeranz und Reddy [PR97] konstruiert die Testmustermenge aus den besten Testmustern, die durch genetische Optimierung generiert wurden. Diese dynamische Methode wird in Verbindung mit einem Testmustergenerator eingesetzt.

2002 veröffentlichten Miyase, Kajihara und Reddy [MKR02] eine Kompaktierungstechnik, die auf dem Lösen des Graphenfärbungsproblems[1] beruht. Damit werden die Testmuster kompaktiert.

[1] Graphenfärbeproblem: Ein Problem aus der Klasse NP, das darin besteht, die Knoten eines Graphen mit einer minimalen Anzahl von Farben so zu färben, dass keinen miteinandern verbundenen Knoten dieselbe Farbe zugeordnet ist.

Sechs auf Heuristiken beruhende Verfahren wurden von Signorini und Remersaro 2005 in [SR05] vorgestellt. Diese ordnen die Testmuster zunächst nach der Anzahl der vorhandenen X Symbole um anschließend die Muster effizient zusammenzufassen. Außerdem nutzen sie einen Graphen, der die Kompatibilität der Testmuster untereinander abbildet. Die Verfahren sind zum Teil sehr schnell, die Lösungen allerdings meist suboptimal.

Ebenfalls relativ neu und auf Heuristiken basierend ist der Ansatz von Coskun und Sarkar [Ays05]. Sie kompaktieren die Testmuster durch drei aufeinanderfolgende Algorithmen. Zunächst wird für jedes Testmuster ein kompatibles Testmuster gesucht. Diese werden zusammengefasst (Merge X) um so eine erste Kompaktierung zu erreichen. Durch den nächsten Algorithmus (Largest Coverage First) werden redundante Testmuster durch die Suche nach am meisten fehlerabdeckenden Mustern gefunden. Diese werden durch den RVE-Algorithmus (Redundant Vector Elimination) zusammengefasst.

Ein interessanter Ansatz wurde von Marculescu, Marculescu und Pedram 1996 beschrieben [MMR96]. Dabei werden dynamische Markov Modelle zur Kompaktierung eingesetzt.

2.2 Evolutionäre Algorithmen

Evolutionäre Algorithmen sind probabilistische[2] Optimierungsverfahren, die an die Darwinsche Evolutionstheorie angelehnt sind. Sie werden erfolgreich bei Problemen der Komplexitätsklasse NP, beispielsweise dem Handelsreisendenproblem[3], angewendet. Da es sich um probalilistische Algorithmen handelt, kann generell nicht garantiert werden, dass die optimale Lösung gefunden wird. Oft ist dies jedoch der Fall. In der Regel wird zumindest eine suboptimale Lösung berechnet.

Da diese Verfahren ein Verständnis der biologischen Evolutionsmechanismen voraussetzen und viele Begriffe aus der Evolutionstheorie übernommen werden, wird zunächst die biologische Evolutionstheorie kurz umrissen. Es folgt eine Beschreibung der technischen Umsetzung und der dazu eingesetzten Mechanismen.

2.2.1 Biologische Evolutionstheorie

Charles Darwin[4] war es, der in seinem 1859 erschienenen Buch »Über den Ursprung der Arten durch das Mittel der natürlichen Auswahl, oder die Erhaltung bevorzugter Rassen im Kampf um das Leben« die vorhandenen Theorien zur Evolution zusammenfasste und

[2] Probabilistisch: Statistische Wahrscheinlichkeit, nicht streng kausal

[3] Das Problem des Handelsreisenden (engl. Traveling Salesman Problem, kurz TSP) ist ein kombinatorisches Optimierungsproblem. Die Aufgabe besteht darin, eine Reihenfolge für den Besuch mehrerer Orte so zu wählen, dass die gesamte Reisestrecke des Handlungsreisenden nach der Rückkehr zum Ausgangsort möglichst kurz ist [Wei02, S. 39ff].

[4] Charles Robert Darwin; britischer Naturforscher; ∗ 12. Februar 1809 in Shrewsbury, England; † 19. April 1882 in London, England; (Quelle: http://de.wikipedia.org/wiki/Charles_Darwin)

durch eigene Beobachtungen ergänzte. Er propagierte die Theorie eines natürlichen Prinzips der Evolution durch graduelle Variation und natürliche Selektion. Sie erklärt die langsame Aufspaltung der Lebewesen in viele verschiedene Arten als Folge von Anpassungen an den Lebensraum. Dieser zentrale Evolutionsmechanismus war der entscheidende Meilenstein zur Evolutionstheorie und wird später bei der technischen Umsetzung aufgegriffen.

Es handelt sich bei der biologischen Evolution also um einen Optimierungsprozess, bei dem sich die Individuen einer Spezies von Generation zu Generation an die jeweiligen Gegenbenheiten der Umwelt anpassen. Wallace[5] nannte das prägnant: »Survival of the fittest«. Die Hauptmechanismen dieses Anpassungsprozesses sind die Selektion, Rekombination und Mutation.

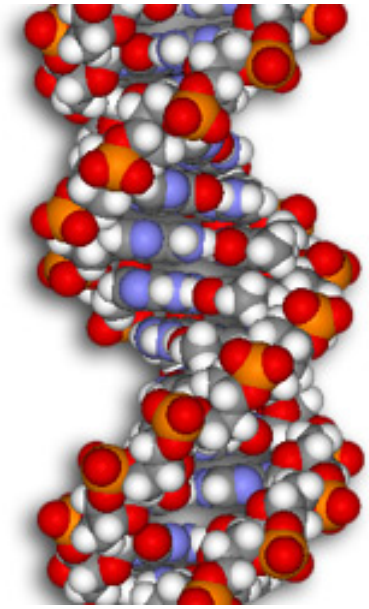


Abb. 2.2: Strukturmodell der Desoxyribonukleinsäure. (Quelle: http://de.wikipedia.org/wiki/Bild:DNA.jpg)

Besser an die Umwelt angepasste (fittere) Individuen haben eine bessere Überlebenschance als schlechter angepasste. Beispielsweise sind gut getarnte, also z. B. an die Farbe des Lebensraums adaptierte Tiere vor Fressfeinden besser geschützt als solche ohne besonderen Anpassung. Bezüglich des Merkmals Farbe findet hier also durch die Fressfeinde eine »Selektion« statt. Aber auch bei der Partnerwahl setzen sich in der Regel fittere Individuen durch. Bei der anschließenden Paarung kommt es zu einer Vermengung des Erbguts. In der Evolutionsterminologie wird dieser Prozess »Rekombination« genannt. Statistisch gesehen besitzt jeder Nachfahre jeweils die Hälfte der Erbinformation der einzelnen Eltern. Bei diesem Prozess entstehen keine neuen Informationen, die bereits vorhandenen Gene werden lediglich gemischt. Neue Gene entstehen durch zufällige Veränderungen während des Verschmelzungsprozesses der DNA, »Mutation« genannt. Durch redundante Gensequenzen, Art der Kodierung und andere Effekte [Bus98, S.74] führt jedoch nicht jede Genänderung zu einer anderen Ausprägung eines Merkmals. Das neue Lebewesen wächst heran, stellt seine Fitness unter Beweis und der Fortpflanzungsprozess beginnt von neuem.

Um den Reproduktions- und Mutationsprozess zu verstehen, folgt ein Einblick in die Genetik (vgl. Abb. 2.2):
Die Erbinformation eines Lebewesens ist im Zellkern lokalisiert. Darin befinden sich die Chromosomen, die aus Desoxyribonukleinsäure (DNA) bestehen. Die DNA wird aus zwei Polynukleotidsträngen gebildet, die eine gegenläufige Polarität besitzen und zu einer plek-

[5] Alfred Russel Wallace; englischer Naturforscher, Philosoph; ∗ 8. Januar 1823 in Usk, Monmouthshire; † 7. November 1913 in Broadstone, Dorset; (Quelle: http://de.wikipedia.org/wiki/Alfred_Russel_Wallace)

tonemischen[6] Doppelhelix umeinander gewunden sind. Daran befinden sich senkrecht zur Halbachse über Wasserstoffbrückenbindungen gepaart die Basen. Es gibt vier verschiedene Basen. Dies sind die Purine Guanin, Adenin sowie die Pyrimidine Cytosin, Thymin. Von diesen paart Adenin stets mit Thymin und Guanin stets mit Cytosin. Über die Reihenfolge dieser Basen sind die Gene und somit die Erbinformationen kodiert. Die spezielle Ausprägung eines Gens wird mit Allel bezeichnet. Durch komplexe biochemische Prozesse entsteht aus diesen genetischen Informationen, in der Gesamtheit als Genotyp bezeichnet, die Ausprägung (Phänotyp) des jeweiligen Individuums.

2.2.2 Technische Optimierung nach dem Vorbild der biologischen Evolution

Technisch betrachtet ist ein Optimierungsproblem die Suche eines Optimums innerhalb eines Suchraums Ω. Bei den Evolutionären Algorithmen wird eine Abbildungsfunktion $f : \Omega \rightarrow \mathbb{R}$ benutzt, um jedem Punkt des Suchraums einen reellen Wert zuzuordnen, so dass die jeweiligen Punkte durch die Relationen »<« und »>« miteinander verglichen werden können.

Die Lösungskandidaten des Optimierungsproblems werden, in Anlehnung an das biologische Vorbild, Individuen genannt. Eine Menge von Individuen wird als Population bezeichnet.

Um eine Optimierung zu erzielen, selektieren, rekombinieren und mutieren Evolutionäre Algorithmen die Individuen einer Population so lange, bis ein Lösungskandidat eine geforderte Qualität aufweist (oder beispielsweise eine festgelegte Anzahl von Generationen berechnet wurde).

Um die Lösungskandidaten miteinander vergleichen zu können, müssen diese bewertet werden. Das geschieht in der sogenannte Fitnessfunktion. Hier werden die im Indiviuum kodierten Informationen, der Genotyp, dekodiert. Daraus ergibt sich der Phänotyp, der im Suchraum Ω liegt. Aus dem Phänotyp wird mittels einer Abbildungsfunktion f der Fitnesswert des jeweiligen Individuums berechnet. Bei einigen Anwendungen sind Genotyp und Phänotyp identisch. In diesen Fällen spricht man von einer »induzierten Fitnessfunktion« [Wei02, S. 50-53].

Mehrere Wissenschaftler griffen das Vorbild der biologischen Evolution unabhängig voneinander auf und entwickelten daraus Optimierungsalgorithmen. Sie werden unter der Bezeichnung »Evolutionäre Algorithmen« zusammengefasst. Dazu gehören die Evolutionsstrategien [Rec73], [Sch68], [Sch75], Evolutionäres Programmieren [FOW66], Genetische Programmierung [Koz92] und die Genetischen Algorithmen [Hol73], [Hol92]. Allen Verfahren gemeinsam ist die Abbildung des Evolutionsprozesses, wobei Unterschiede hinsichtlich

[6] Eine plektonemsiche Doppelhelix entsteht, wenn man zwei Drähte gleichzeitig um einenn Stab windet. Zieht man den Stab heraus, so hängen die Drähte in jeder Windung ineinander und müssen für eine Trennung auseinandergedrillt werden. Im Gegensatz dazu wäre eine paranemische Doppelhelix, die durch aneinanderlegen von zwei getrennt gewickelten Drähten entsteht.

des Einsatzes von Mutation, Selektion, Rekombination, Bewertung und vor allem Art der Kodierung vorliegen.

2.2.3 Genetische Algorithmen

Genetische Algorithmen orientieren sich stark am biologischen Vorbild. Die Chromosomen der Population werden in der Regel als binäre Vektoren kodiert. Deshalb werden hier binäre Vektoren aus der Grundmenge $M = \{0, 1\}$ betrachtet. Ein Chromosom ist ein Element aus $M^n = \{0,1\}^n$ mit $n \in \mathbb{N}, n \geq 1$. Das Chromosom $x =< x_1, x_2, \ldots, x_n >$ weist damit die Länge n auf. Die i-te Position eines Chromosoms $x =< \ldots, x_i, \ldots >\in M^n$ heißt das i-te Gen. Der jeweilige Wert heißt Allel. Je nach Kodierung werden auch Binärsequenzen, also zusammenhängende Chromosomenabschnitte, als Gene bezeichnet. Besondere Aufmerksamkeit wird bei den Genetischen Algorithmen dem Kodierungsproblem, also der Frage der Informationsdarstellung auf dem Chromosom, gewidmet. Nähere Informationen dazu finden sich in [SHF94, S. 187 ff]

Der Ablauf eines Genetischen Algorithmus soll nun vorgestellt werden, wobei die eben eingeführten Definitionen verwendet werden (vgl. Abb. 2.3). Dabei werden die verwendeten genetischen Operatoren erklärt:

Zunächst werden die Individuen der Population initialisiert. Dazu wird jedem Gen x_i mit $1 \leq i \leq n$ der Individuen zufällig ein Wert der Menge M zugewiesen. Bei den meisten Problemen ist es hilfreich, wenn sich die Chromosomen stark voneinander unterscheiden. Durch eine hohe Diversität[7] soll von möglichst weit auseinander liegenden Punkten im Suchraum gestartet werden, um das globale Optimum zu finden.

Nun werden alle Individuen der Population gemäß der Fitnessfunktion $f : \Omega \rightarrow \mathbb{R}$ bewertet. Diese je nach Problemdefinition variierende Funktion dekodiert den Genotyp in den im Suchraum Ω liegenden Phänotyp und berechnet aus diesem den reellwertigen Fitnesswert des jeweiligen Individuums. Dieser Schritt ist nötig, da nun mit Hilfe der Relation $<$ die Individuen verglichen werden können. Eine Bewertung der Güte ist so möglich.

Im nächsten Schritt wird ausgehend von der erzeugten und bewerteten (Eltern-) Population die Nachkommenpopulation erzeugt. Dazu werden zunächst zwei Elternindividuen gemäß einer Selektionsfunktion ausgewählt. Es existiert eine Reihe unterschiedlicher Selektionsfunktionen, auf die wichtigsten wird später noch eingegangen werden (siehe Kap. 3.3.1).

Bei der Selektionsfunktion handelt es sich um ein Verfahren, das unter Berücksichtigung der Fitness zufällig Individuen bestimmt. Bessere (fittere) Individuen haben eine höhere Chance ausgewählt zu werden als schlechtere.

[7] lat. diversitas: Vielfalt, Vielfältigkeit

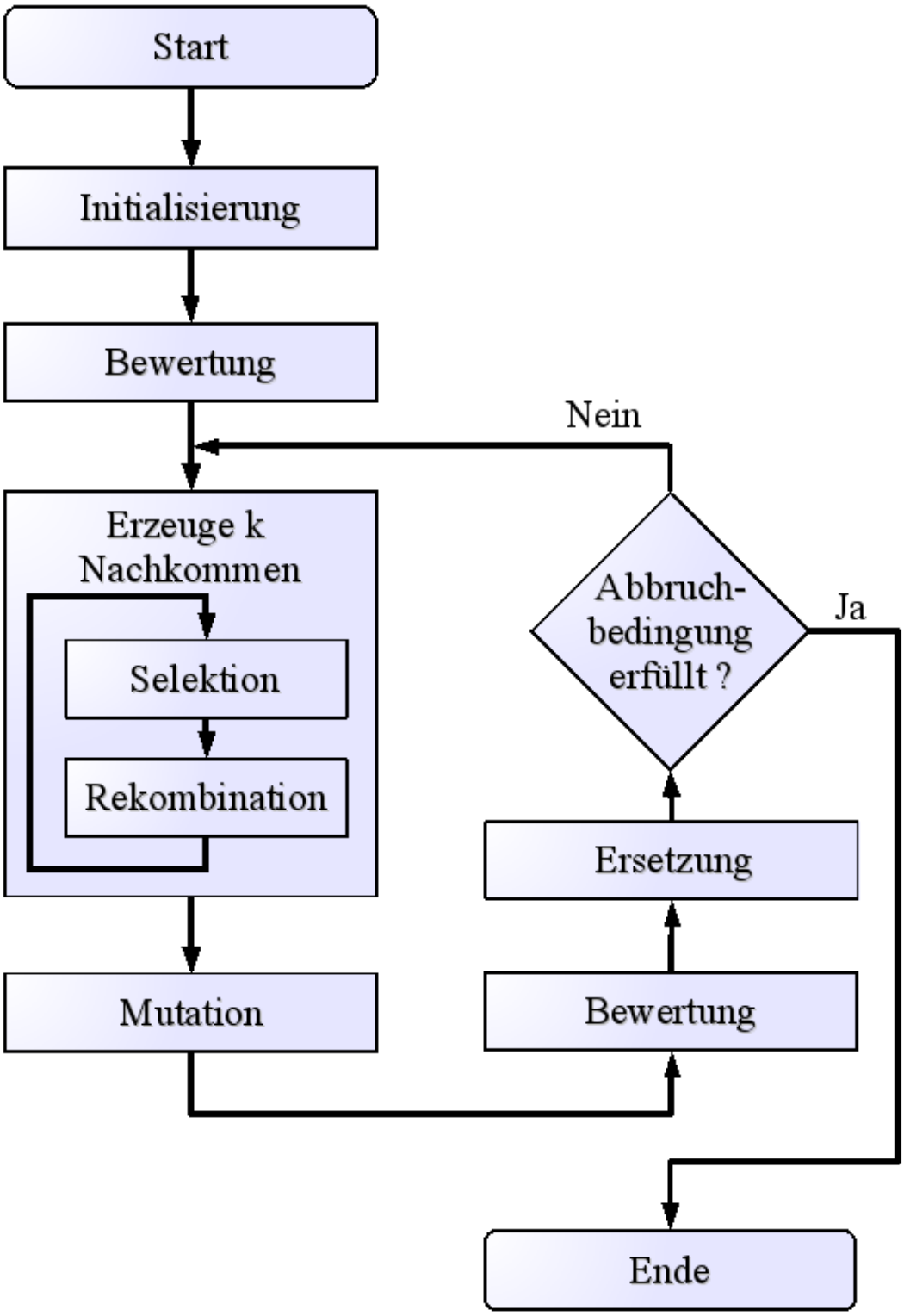


Abb. 2.3: Schematische Darstellung des Zyklus eines Genetischen Algorithmus. Zunächst werden die Individuen einer Population zufällig initialisiert. Auf Grund der während der Bewertung berechneten Fitnesswerte werden Elternindividuen selektiert. Die aus den selektierten Eltern während der Rekombination erzeugten Nachkommenindividuen werden mutiert und anschließend über die Fitnessfunktion bewertet. Die Elternpopulation der nächsten Generation wird je nach Ersetzungsschema durch Individuen der Eltern- und/oder Nachkommenpopulation zusammengesetzt. Dies wiederholt sich so lange, bis ein Abbruchkriterium erfüllt ist (z. B. das Erreichen einer bestimmten Güte).

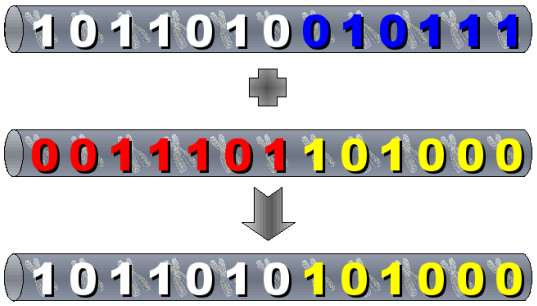


Abb. 2.4: Rekombination am Beispiel der Ein-Punkt Rekombination. Eine Position des Chromosoms wird zufällig bestimmt. An dieser Stelle kreuzen sich die Chromosomen. Das erzeugte Chromosom besteht aus dem ersten Teil des einen und dem zweiten Teil des anderen Elternchromosoms.

Die Chromosomen von jeweils zwei[8] derart ausgewählter Elternindividuen werden nun rekombiniert (vgl. Abb. 2.4). Dabei bekommt das Kindindividuum einen Teil der Erbinformationen vom ersten und den anderen Teil vom zweiten Elternindividuum. Es werden also keine neuen Informationen generiert, vorhandene Gene werden lediglich ausgetauscht. Die Rekombination dient bei den Genetischen Algorithmen dazu, weite Bereiche des Suchraums zu erforschen. Es wurden verschiedene Rekombinationsfunktionen entwickelt, von denen wichtige in Kap. 3.3.3 vorgestellt werden.

Diese beiden Schritte werden wiederholt, bis die Nachkommenpopulation k Nachkommen-Individuen beinhaltet.

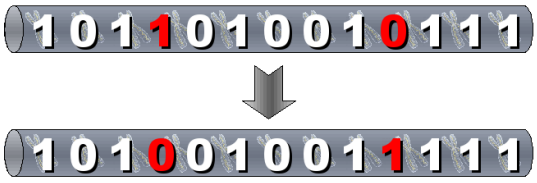


Abb. 2.5: Beispiel einer Mutation. Der Wert zufällig bestimmter Gene wird geändert.

Anschließend werden alle so erzeugten Individuen der Nachkommenpopulation mutiert. Bei den Genetischen Algorithmen hat die Mutation die Aufgabe, Individuen lokal zu optimieren. Im Gegensatz zur Rekombination können bei der Mutation neue Informationen erzeugt werden. Die vielfältigen Mutationsoperatoren, von denen einige in Kap. 3.3.4 erklärt werden, nehmen in der Regel nur eine geringfügige Änderung der Gene vor.
Abb. 2.5 skizziert eine einfache Zufallsmutation. Die zu mutierenden Genpositionen werden zufällig bestimmt und mit zufälligen Werten belegt. Damit ändert sich der Wert des 4. Gens, der des ebenfalls zur Mutation ausgewählte Wert des 9. Gens bleibt jedoch erhalten, da zufällig eine 1 als neuer Wert bestimmt wurde.

Die Individuen der Nachkommenpopulation werden danach bewertet. Dies geschieht durch die oben beschriebene Fitnessfunktion f.

Es folgt ein Generationenwechsel. Dabei werden die Individuen der Elternpopulation durch Individuen der Nachkommenpopulation und/oder der Elternpopulation ersetzt. Je nach Ersetzungsmethode werden die jeweils besten (fittesten) Individuen übernommen. Dies folgt dem biologischen Vorbild, dass »gute« Gene an die nächste Generation weitergegeben werden, während »schlechte« Gene aussterben. In der Literatur werden oft Methoden der Selektion als Ersetzungsoperatoren eingesetzt. Zur Trennung der Vorgänge wird dann von Elternselektion und Umweltselektion gesprochen. Durch Adaptieren der Selektionsparameter läßt sich bei vielen Seletkionsfunktionen der Selektionsdruck einstellen.

[8] Entsprechend einer sexuellen Fortpflanzung.

Zum Schluß wird überprüft, ob ein Abbruchkriterium erfüllt ist. Abbruchkriterien sind beispielsweise das Erreichen einer bestimmten Anzahl durchlaufener Generationen oder einer bestimmten Güte des besten Individuums. Ist das Abbruchkriterium nicht erfüllt, so wird die nächste Generation berechnet. Andernfalls ist die Berechnung zu Ende und somit ist das fitteste Individuum der gesuchte Lösungskandidat.

3 Umsetzung

In diesem Kapitel werden die erarbeiteten Lösungsansätze und Hilfsfunktionen sowie die verwendeten genetischen Operatoren beschrieben. Es wird die grundlegende Idee, der Ablauf und die Besonderheiten der Algorithmen dargestellt.

Begonnen wird mit den Lösungsansätzen (Kap. 3.1). Dabei handelt es sich um drei grundlegende Ansätze, die jeweils spezifische Vor- und Nachteilen aufweisen.

Es folgt in Kap. 3.2 die Beschreibung der Hilfsfunktionen, die dazu dienen, die Berechnung zu beschleunigen.

Danach werden in Kap. 3.3 alle implementierten genetischen Operatoren erläutert. Diese sind größtenteils für alle Lösungsansätze, teilweise spezifisch für jeweils einen Lösungsansatz verwendbar (vgl Tab. 3.1). Hier wird auch auf die Fitnessfunktion und die Abbruchkriterien eingegangen.

Schließlich wird das entwickelte Programm in Kap. 3.4 kurz vorgestellt.

3.1 Lösungsansätze

Zunächst wurde ein Lösungsansatz gewählt, der eine einfache Kompaktierung schnell durchführt. Die Kompaktierung ist in der Regel nicht optimal, allerdings werden die Ausgangstestmuster typischerweise auf ca. 5% der ursprünglichen Größe kompaktiert. Bei dem Ansatz wird ein Chromosom pro Individuum eingesetzt. Die Länge der Chromosomen ist gleich der Testmusterlänge. Die Gene bestehen ausschließlich aus festen Werten (»0« und »1«). Die Kodierung der Chromosomen ist so gewählt, dass das Chromosom direkt ein kompaktiertes Testmuster repräsentiert.
Die Individuen der Population werden eine festgelegt Anzahl von Epochen dahingehend optimiert, dass möglichst viele Testmuster durch den Lösungsvektor abgedeckt werden. Der Lösungsvektor des nach einem Durchlauf besten Individuums, also der mit der maximalen Fehlerabdeckung, wird den Ausgabemustern hinzugefügt. Alle Eingabemuster, die von dieser Lösung abgedeckt sind, werden entfernt und sind so für die weitere Berechnung irrelevant. Der Ablauf wiederholt sich so lange, bis alle Eingabemuster von den Ausgabemustern erfasst wurden. Diese »Greedy[1] Methode« genannte ist die schnellste der drei Lösungsmethoden. Sie wird in Kap. 3.1.1 beschrieben.

[1] Engl. greedy: gierig

Der nächste Lösungsansatz besteht darin, dass ein Individuum mehrere Chromosomen besitzt. Die Chromosomen sind wie bei der ersten Methode kodiert. Die Individuen werden nun so lange optimiert, bis alle Eingabetestmuster durch die Lösungsvektoren eines Individuums abgedeckt werden und die Anzahl der Chromosomen möglichst gering ist. Diese »MaxMin Methode« berechnet in der Regel bessere Lösungen als die erste, allerdings benötigt sie eine deutlich längere Rechenzeit. Die Details sind in Kap. 3.1.2 aufgeführt.

Beim dritten Ansatz, der »Permutationsmethode«, existiert genauso wie beim Ersten pro Individuum genau ein Chromosom. Allerdings wird für die Chromosomen eine andere Kodierung eingesetzt: Jedes Gen repräsentiert einen Indexwert eines Eingabetestmusters. Auf jedem Chromosom befindet sich jeder Indexwert genau einmal. Die Reihenfolge der Indexwerte auf den Chromosomen gibt die Reihenfolge der Kompaktierung der Eingabetestmuster an. Können zwei aufeinanderfolgende Eingabetestmuster zusammengefasst werden, wird das kompaktierte Muster gebildet und mit diesem das nächste Eingabemuster betrachtet, bis keine weitere Kompaktierung möglich ist. Das kompaktierte Muster wird als Ausgabemuster gespeichert. Kann das Eingabemuster nicht mit seinem Nachfolger zusammengefasst werden, wird es unverändert als Ausgabemuster übernommen. So wird garantiert, dass alle Testfälle erhalten bleiben. Diese in Kap. 3.1.3 ausführlich beschriebene Methode liefert gute Ergebnisse bei vergleichsweise geringen Ressourcen.

3.1.1 Greedy Methode

Die Idee der »Greedy Methode« ist, dass Problem zu verkleinern. Dazu werden bei jedem Optimierungsschritt möglichst viele Eingabemuster durch ein Lösungsmuster abgedeckt. Das Lösungsmuster wird den Ausgabetestmustern hinzugefügt und die dadurch abgedeckten Eingabemuster von der weiteren Betrachtung ausgenommen. Dieser Ablauf wird so lange wiederholt, bis alle Eingabemuster abgedeckt sind.

Wegen der einfachen und damit schnelleren Berechnung wird pro Individuum nur ein Chromosom, also ein Lösungsmuster, verwendet. Die Länge des Chromosoms entspricht der Testmusterlänge. Es befinden sich lediglich feste Werte (»0« und »1«) auf den Chromosomen. Der unbestimmte Wert (»X«) wird nicht verwendet.

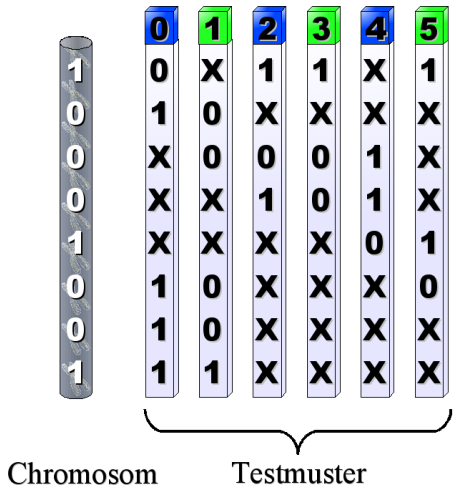


Abb. 3.1: Greedy Methode. Es wird versucht, mit einem Chromosom so viele Testmuster wie möglich abzudecken. Im Bild deckt das Chromosom die grün markierten Testmuster 1, 3 und 5 ab.

Nun wird eine Lösungsmenge (Population) optimiert, wobei die Populationsgröße während der gesamten Berechnung konstant bleibt. Zunächst werden die Individuen, genauer die Werte der Chromosomen, zufällig initialisiert. Dabei wird nach dem in Kap. 2.2.3 erläuterten Algorithmus vorgegangen. Das Optimierungsziel ist ein Individuum, das ein Maximum

an Eingabetestmustern zusammenfasst. Die Berechnung dazu findet in der Fitnessfunktion statt, bei der für jedes Individuum ermittelt wird, wieviele Eingabetestmuster von dessen Chromosom erfasst werden. Der Fitnesswert steigt proportional mit der Anzahl der abgedeckten Eingabetestmuster.

Nachdem eine festgelegte Anzahl von Epochen durchlaufen wurde, wird das beste Individuum verwendet. Aus dem Chromosom wird das Ausgabemuster direkt erzeugt und zur Ausgabetestmustermenge hinzugefügt. Die zugehörigen Eingabetestmuster werden entfernt.

Danach werden die Chromosomen wieder zufällig initialisiert und der Algorithmus startet von neuem. Dies erfolgt so lange, bis alle Eingabetestmuster abgedeckt sind.

Von den in Kap. 3.3 beschriebenen genetischen Operatoren können alle eingesetzt werden, mit Ausnahme der Mutationsoperatoren, welche die Chromosomenanzahl verändern.

3.1.2 MaxMin Methode

MaxMin steht für »Maximale Fehlerabdeckung bei minimaler Anzahl von Chromosomen«. Ähnlich der »Greedy Methode« operiert auch diese mit Genen, deren Allele nur feste Werte annehmen. Auch hier enspricht die statische Chromosomenlänge der Testmusterlänge, genauso wie der Genotyp mit dem Phänotyp (also dem kompaktierten Ausgabetestmuster) identisch ist. Des weiteren sind die Abläufe der Optimierungsalgorithmen identisch. Alle dort benutzten genetischen Operatoren finden hier ohne Änderung Anwendung. Allerdings werden die Mutationsoperatoren um Chromosomenänderungsoperatoren (hinzufügen, löschen, tauschen) erweitert. Dazu mehr bei den einzelnen Operatoren (Kap. 3.3).

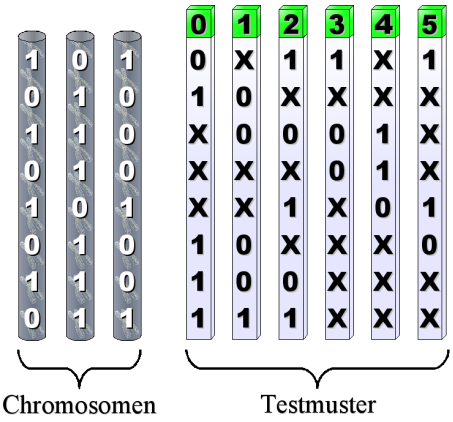


Abb. 3.2: Jedes Individuum besitzt eine veränderbare Anzahl von Chromosomen. Es befinden sich darauf lediglich feste Werte (0, 1). Die Individuen werden so optimiert, dass alle Eingabetestmuster abdeckt und dafür eine minimale Chromosomenanzahl benötigt wird. Hier können die Testmuster von den zwei rechten Chromosomen abgedeckt werden, das linke Chromosom kann entfernt werden.

Obgleich große Gemeinsamkeiten zwischen dieser und der oben vorgestellten »Greedy Methode« bestehen, verfolgt die »MaxMin Methode« ein anderes Ziel: Die Lösung des Testmusterkompaktierungsproblems mit einem einzigen Lösungskandidaten (Individuum). Dabei weist jedes Individuum eine variable und sich während des Optimierungsprozesses dynamisch ändernde Chromosomenanzahl auf. Das Optimierungsziel ist dabei eine möglichst geringe Chromosomenanzahl, wobei die Chromosomen alle Eingabetestmuster abdecken.

Daraus ergibt sich das Problem, dass einerseits, ausgehend von einem Chromosom pro Individuum, die Chromosomenzahl steigen muss, um alle Testmuster kompaktieren zu können,

andererseits soll die Chromosomenzahl und somit die Anzahl kompaktierter Muster ein Minimum erreichen.

Um diesem Problem zu entgehen und so zur Beschleunigung der Berechnung beizutragen, werden die Individuen mit der Chromosomenanzahl initialisiert, die bei der Abschätzung der minimalen Kompaktierbarkeit (siehe Kap. 3.2) ermittelt wurde. Damit ist eine obere maximale Schranke festgelegt. Von dieser ausgehend, ist der Optimierungsprozess bestrebt, unter Beibehaltung aller abgedeckten Testmuster die Chromosomenanzahl zu reduzieren.

Dazu bezieht die Fitnessfunktion mehrere Faktoren ein. Positiv wird die Anzahl der durch das Individuum abgedeckten Eingangstestmuster T bewertet. Negativ geht die Chromosomenanzahl C ein. Außerdem wird noch eine Diversitätsfunktion D berücksichtigt, die angibt, wie verschieden die Chromosomen sind. Die Diversitätsfunktion vergleicht jedes Chromosom mit jedem anderen des Individuums. Weichen zwei Werte an gleicher Genposition voneinander ab, wird der Rückgabewert um eins erhöht. Je unterschiedlicher die Chromosomen sind, desto höher ist der Diversitätswert. Daraus ergibt sich die Fitnessfunktion: $f = T - a * C + b * D$. Hierbei sind a und b Gewichtungsfaktoren, die experimentell bestimmt werden (siehe Kap. 4).

3.1.3 Permutationsmethode

Die »Permutationsmethode« unterscheidet sich von den beiden vorhergehenden Ansätzen deutlich.

Das Chromosom der Individuen kodiert nicht das Ausgabemuster, sondern die Reihenfolge, in der die Eingabetestmuster verarbeitet werden. Dazu kodiert jedes Gen den Indexwert eines Eingabetestmusters. Jeder Indexwert existiert auf jedem Chromosom genau einmal. So ist sichergestellt, dass durch jedes Chromosom alle Eingabetestmuster abgedeckt sind.

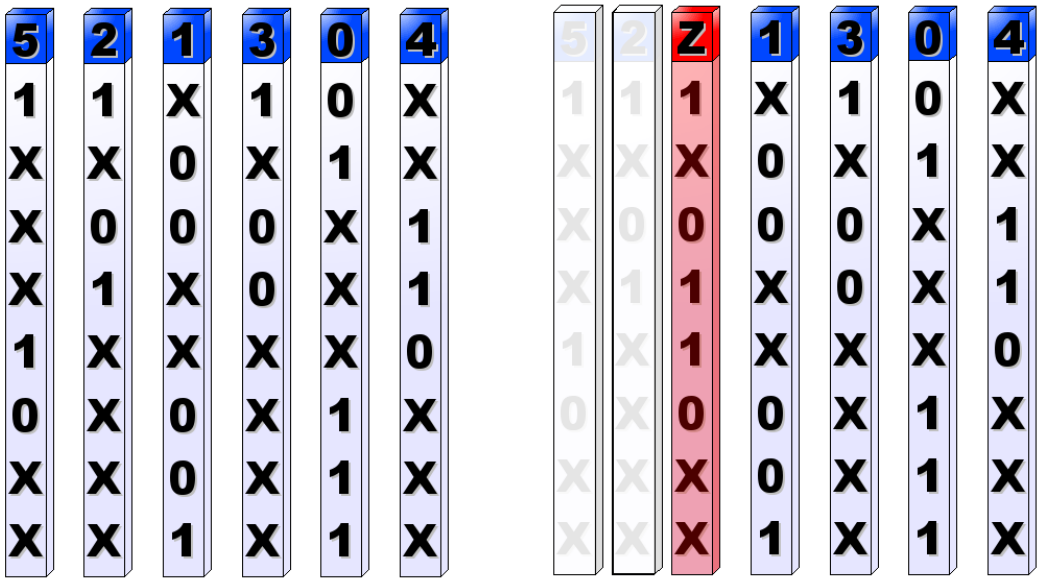
Die Kompaktierung wird durch Zusammenfassen benachbarter Eingabetestmuster erreicht. Dazu werden die Eingabetestmuster nach der auf dem Chromosom kodierten Reihenfolge abgearbeitet. Vergleiche dazu Algorithmus 1 (Die Angaben »erster« und »nächster« beziehen sich auf die durch das Chromosom kodierte Reihenfolge der Eingabetestmuster):

Algorithm 1 Kompaktierung nach der Permutationsmethode

```
1: for Durchlaufe die Eingabetestmuster der angegebenen Reihefolge nach do
2:    if Ist aktuelles Muster kompatibel zu nächstem Eingabetestmuster then
3:       Aktuelles Muster mit nächstem Eingabetestmuster zusammenfassen.
4:       Das resultierende Muster wird zum aktuellen Muster.
5:    else
6:       Aktuelles Muster zu Ausgabetestmustern hinzufügen.
7:       Nächste Eingabetestmuster wird zum aktuellen Muster.
8:    end if
9: end for
10: Falls noch nicht geschehen: Füge aktuelles Muster zu Ausgabetestmustern hinzu.
```

Der Algorithmus arbeitet mit einem temporären Muster. Dieses nimmt am Anfang die Werte des auf dem Chromosom an erster Stelle referenzierten Testmusters an. Nun wird der auf dem Chromosom kodierten Reihenfolge nach jedes Eingabetestmuster verarbeitet. Das aktuelle Muster wird bei jedem Durchgang mit dem jeweils nächsten Eingabetestmuster verglichen. Können die beiden Muster zusammengefasst werden, wird daraus gemäß der Tabelle 2.1 ein neues aktuelles Muster gebildet. Ist keine (weitere) Kompaktierung möglich wird das aktuelle Muster als kompaktiertes Ausgabetestmuster gespeichert, das nächste Eingabetestmuster als aktuelles Muster übertragen und der nächste Durchgang gestartet bis alle Eingabetestmuster bearbeitet wurden.

Ein Beispiel (vgl. Abb. 3.3 und Abb. 3.4) soll verdeutlichen, wie mittels der Eingabemusterreihenfolge eine Kompaktierung berechnet wird:
Es wird davon ausgegangen, dass sechs Eingabetestmuster vorliegen. Diese sind auf der linken Seite von Abb. 3.4(b) (blau) dargestellt. In den Köpfen sind die jeweiligen Indexwerte dargestellt, sie sind nicht Teil des Testmusters.



(a) Zu kompaktierende Eingabetestmuster. Zur Veranschaulichung sind die Indizes über den Testmustern dargestellt. Die Eingabetestmuster werden je nach Kodierung des entsprechenden Chromosoms der Reihe nach verarbeitet. Hier in der Reihenfolge (5-2-1-3-0-4).

(b) Zunächst wird das laufende Muster, das dem von Muster 5 entspricht, mit dem Eingabetestmuster 2 verglichen. Können die Muster kompaktiert werden, so wird das zusammengefasste Muster »Z« (rot) gebildet.

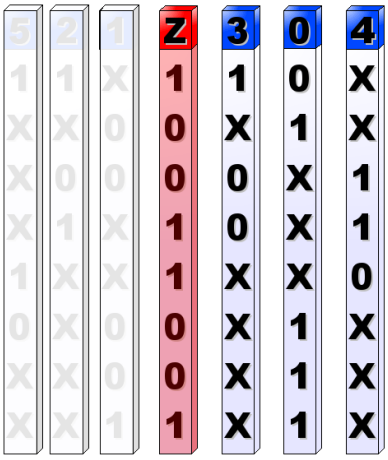
Abb. 3.3: Auf den Chromosomen sind die Indexwerte der Eingabetestmuster abgelegt. Nach dieser Reihenfolge werden die Testmuster kompaktiert.

Die Berechnung wird nun für eine angenommene Gensequenz (5-2-1-3-0-4) durchgeführt. Diese Zahlen der Gensequenzreihenfolge definieren die Reihenfolge, in der die Eingabetestmuster durchlaufen werden, wobei die Zahlen mit den Indizes der Eingabetestmuster

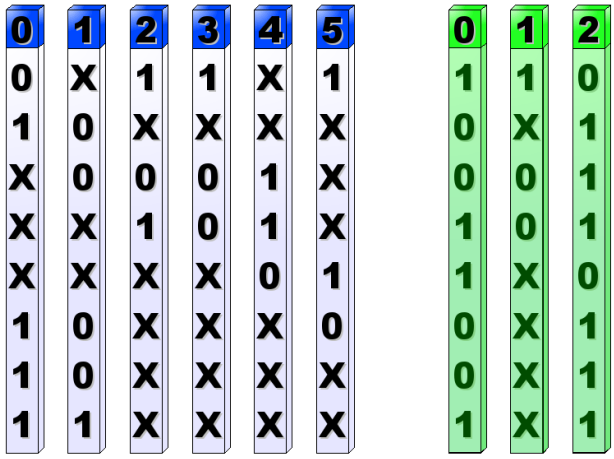
korrespondieren. Die Eingabemuster werden also wie in Abb. 3.3(a) abgebildet von links nach rechts verarbeitet.

Das erste Eingabetestmuster (Index 5) wird als aktuelles Muster übernommen. Nun durchläuft der Algorithmus nacheinander in der angegebenen Reihenfolge die folgenden Eingabetestmuster. Dazu wird das aktuelle Muster zunächst mit Muster 2 verglichen. Wie man sehen kann, lassen sich die Muster kompaktieren und zu einem neuen aktuellen Muster zusammenfassen (Abb. 3.3(b)).

Jetzt beginnt der nächste Durchgang, in dem das nächste Eingabetestmuster (Index 2) betrachtet wird. Dieses lässt sich mit dem aktuellen Muster zusammenfassen (Abb. 3.4(a)). Das so entstandene aktuelle Muster kann im nächsten Durchlauf nicht mit Muster 3 kompaktiert werden. Deshalb wird es als erstes Ausgabetestmuster abgespeichert und anschließend durch das Muster 3 ersetzt. So setzt sich der Algorithmus Durchgang für Durchgang fort, bis alle Eingabetestmuster bearbeitet wurden. Daraus ergibt sich die in Abb. 3.4(b) grün dargestellte kompaktierte Ausgabetestmustermenge.



(a) Mit dem in Abb. 3.3(b) zusammengefassten Muster Z wird nun das nächste Eingabemuster (1) verglichen. Auch diese können kompaktiert werden. Wieder wird das zusammengefasste Muster Z (rot) berechnet.



(b) Kann ein Eingabetestmuster nicht mit dem anktuell kompaktierten Muster Z zusammengefasst werden, wird das Muster Z als Ausgabetestmuster abgespeichert und mit dem Algorithmus von vorne begonnen. Daraus ergibt sich für dieses Beispiel die grün gefärbte kompaktierte Ausgabemustermenge.

Abb. 3.4: Berechnung der Ausgabetestmuster (grün) aus den Eingabetestmustern (blau) nach dem Permutationsansatz an einem Beispiel.

Das Optimierungsproblem besteht also darin, die Testmusterindexwerte in eine optimale Reihenfolge zu bringen. Die Problemstellung ist mit dem Handelsreisendenproblem (vgl. 2.2, Seite 10) nahe verwandt.

Es ergibt sich aber auch eine Konsequenz für die genetischen Operatoren. Diese müssen sämtlich so ausgelegt sein, dass auf jedem Chromosom jeder Index immer genau einmal vorhanden ist. So gibt es spezielle Operatoren, die in Kap. 3.3 beschrieben werden.

3.2 Hilfsfunktionen

Neben diesen Lösungsansätzen wurden vier Funktionen entwickelt, um den Zeitaufwand zu verringern.

Idee der ersten drei Funktionen ist es, die bei jedem Durchlauf des genetischen Algorithmus notwendigen Vergleiche zu verringern. Das geschieht in der Fitnessfunktion. Hier werden bei allen Lösungsansätzen alle Eingabetestmuster durchsucht. Da diese Funktion sehr oft durchlaufen wird, liegt es nahe, die Anzahl der Eingabetestmuster zu verkleinern.

Die vierte Funktion ermittelt eine maximalen Anzahl kompaktierter Testmuster. Mit dieser oberen Grenze wird die Initialchromosomenanzahl für die MaxMin-Methode bestimmt.

Entfernen verarbeiteter Testmuster

Bei der ersten Funktion handelt es sich um das Verringern der Eingabetestmustermenge durch sukzessives Entfernen bereits erfasster Testmuster. In folgenden Verarbeitungsschritten werden nur noch die restlichen Testmuster betrachtet. Die Testmusteranzahl nimmt also mit fortschreitenden Berechnungsdurchläufen ab, entsprechend schneller können die Algorithmen zum Ende hin die Durchläufe berechnen. Hierbei liegt es auf der Hand, dass die Algorithmen bei weniger Eingabedaten und damit weniger Vergleichen schneller Ergebnisse liefern. Allerdings kann damit eine optimale Lösung in der Regel nicht gefunden werden.

Bereichsmethode

Bei der zweiten Funktion handelt es sich um eine Bereichsmethode. Mit dieser Technik wird immer nur ein Ausschnitt aus der Eingabetestmustermenge betrachtet. Ist der aktuelle Bereich vollständig berechnet worden, wird der nächste Bereich eingeblendet. Dies wird so lange wiederholt, bis alle Eingabetestmuster bearbeitet worden sind. Auch hierbei ist das Ziel eine Beschleunigung der Berechnung durch Verringerung der Vergleiche. Wie bei der ersten Hilfsfunktion kann auch hier im Allgemeinen keine optimale Lösung gefunden werden.

Irrelevante Testmuster entfernen

Die dritte Hilfsfunktion wird vor einer Berechnung eingesetzt. Hier werden der Reihe nach alle Eingabetestmuster durchlaufen und mit den noch nicht verarbeiteten verglichen. Somit beträgt die Laufzeit im ungünstigsten Fall $O(\frac{1}{2}n^2)$.

Ist ein Testmuster inkompatibel zu allen anderen Testmustern, so wird es direkt zur Ausgabetestmustermenge hinzugefügt sowie aus der Eingabemenge entfernt.
Aus der Eingabemenge entfernt wird es auch, wenn es irrelavant ist. Das ist dann der Fall, wenn das Testmuster komplett durch ein anderes abgedeckt wird. Mit anderen Worten dürfen feste Symbole (0, 1) in dem zu entfernenden Testmuster maximal an den Stellen auftreten, an denen das Vergleichsmuster ebenfalls feste Symbole aufweist.

Dazu ein Beispiel. Gegeben seien die Testmuster

```
X = X1OXX1
A = XXOXX1
B = X1XXOX
C = X1XXXX
```

Alle Fehler, die durch Testmuster A und C detektiert werden können, lassen sich auch durch das Testmuster X feststellen. Auf sie kann also ohne Informationsverlust verzichtet werden. Hingegen befindet sich an der fünften Position des Testmusters B eine 0. Da sich bei Testmuster X an dieser Stelle ein undefiniertes Symbol (X) befindet, kann B nicht entfernt werden.

Abschätzung einer oberen Grenze

Die letzte Hilfsfunktion wird zur Abschätzung einer maximalen Anzahl kompaktierter Testmuster benutzt. Dazu werden alle Testmuster nach dem Vorbild der Permutationsmethode, allerdings mit der beim Laden vorgegebenen statischen Reihenfolge, zusammengefasst. Dabei wird die Anzahl der entstehenden kompaktierten Testmuster gezählt. Diese Anzahl wird dann beispielsweise bei der »MaxMin Methode« eingesetzt, um die Individuen mit dieser Anzahl an Chromosomen zu initialisieren. Im schlechtesten Fall hat die Methode eine Laufzeit von $O(\frac{1}{2}n^2)$.

3.3 Genetische Operatoren

In diesem Abschnitt werden die implementierten genetischen Operatoren vorgestellt. Dazu gehören die Selektion (Kap. 3.3.1), Rekombination (Kap. 3.3.3), Mutation (Kap. 3.3.4) und die Ersetzung (Kap. 3.3.2). Die Selektions-, Rekombinations- und Ersetzungsmethoden können für alle Lösungsansätze eingesetzt werden. Die Kombinationsmöglichkeiten zwischen Mutations- und Lösungsmethoden sind in Tab. 3.1 aufgeführt.

	Greedy Methode	MaxMin Methode	Permutationsmethode
Bit-Flip Mutation	X	X	
Gentausch Mutation	X	X	X
Gensequenz verschieben	X	X	X
Gensequenz invertieren	X	X	X
Chromosom hinzufügen		X	
Chromosom entfernen		X	
Chromosomen tauschen		X	
2-Opt Mutation			X

Tab. 3.1: Kombinationsmöglichkeiten der Mutationsmethoden mit den Lösungsansätzen.

Nach den genetischen Operatoren werden in den letzten Abschnitten die Fitnessfunktionen (Kap. 3.3.6) und die Abbruchkriterien (Kap. 3.3.7) behandelt.

3.3.1 Selektion

Das Prinzip der Selektion besteht darin, dass bessere (fittere) Individuen eine größere Wahrscheinlichkeitt besitzen Nachkommen zu erzeugen, als schlechtere. Durch die Selektion wird die durchschnittliche Güte von Generation zu Generation gesteigert.

Alle Selektionsmethoden beruhen darauf, einen Selektionsdruck zu erzeugen. Unter Selektionsdruck versteht man die Stärke der Bevorzugung besserer Individuen. Bei der Wahl des Selektionsdrucks werden zwei gegensätzliche Ziele verfolgt:

Zum einen die Erkundung (Exploration) des Suchraums. Damit ist gemeint, dass die Individuen möglichst breit über den Suchraum verteilt sein sollen, damit die Chance, das globale Optimum zu finden, möglichst groß ist. Dazu ist ein geringer Selektionsdruck erforderlich, der vor allem zu Beginn der Berechnung sinvoll ist.

Zum anderen wird das Ziel angestrebt, die Ausbeutung (Exploitation) guter Individuen voranzutreiben. Es soll also das (im Allgemeinen lokale) Optimum in der Nähe von guten Individuen gefunden werden. Denn bei einem dieser lokalen Optima könnte es sich um das gesuchte globale Optimum handeln. Dazu ist ein hoher Selektionsdruck von nöten. Ein großer Selektionsdrucks ist sinnvoll, nachdem bereits ein großer Suchraum erkundet wurde und das Optimum der guten Individuen gefunden werden soll.

In den folgenden Abschnitten werden die drei implementierten Selektionsalgortithmen beschrieben. Das sind die »Fitnessproportionale Selektion«, die »Rangbasierte Selektion« und die »Turnierselektion«.

Fitnessproportionale Selektion

Die grundlegende Idee dieses auch als »Rouletteradselektion« bezeichneten Verfahrens besteht darin, dass Eltern mit hoher Fitness eine größere Wahrscheinlichkeit haben, »gute« Nachkommen zu erzeugen, als solche mit niedrigen Fitnesswerten. Bei dieser Selektionsmethode werden daher gute Individuen bevorzugt, allerdings haben auch schlechtere Individuen die Chance ausgewählt zu werden. Die Selektionswahrscheinlichkeit ist proportional zur Höhe der Fitness.

Zur Vorgehensweise des Verfahrens stelle man sich ein Rouletterad vor. Für jedes Individuum der Population sei ein Fach (Sektor) vorgesehen, dessen Größe dem jeweiligen Fitnesswert entspricht. Addiert man alle Fitnesswerte zusammen erhält man die Gesamtfitness der Population. Wird nun eine Kugel geworfen ist die Wahrscheinlichkeit größer, ein Individuum mit höherer Fitness auszuwählen, als eines mit niedrigerer.

Genauso arbeitet der Algorithmus: Die Gesamtfitness der Population wird berechnet. Aus diesem Bereich wird eine Zufallszahl ermittelt.

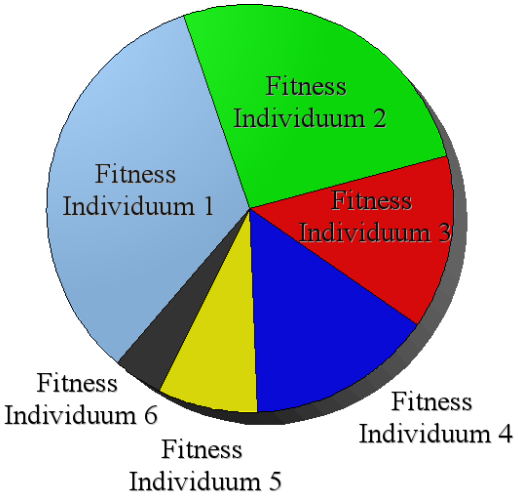


Abb. 3.5: Fitnessproportionale- / Rouletteradselektion. Durch jedes Segment wird ein Individuum repräsentiert. Die Größe der jeweiligen Segmente entspricht der Fitness des zugeordneten Individuums. Bessere Individuen haben eine höhere Selektionswahrscheinlichkeit als schlechtere.

Die Fitnesswerte der Individuen werden nacheinander aufsummiert, bis die Zufallszahl erreicht ist. Entsprechend haben Individuen mit höheren Fitnesswerten eine größere Selektionschance als solche mit niedrigeren Werten.

Eine große Gefahr besteht bei diesem Verfahren: Das Dominanzproblem. Hat ein Individuum einen sehr viel größeren Fitnesswert als alle anderen Individuen, wird fast ausschließlich dieses Individuum ausgewählt. Entsprechend dominieren die Nachkommen dieses Individuums die Folgegenerationen (Crowding). Einher geht ein Verlust der Diversität. Das kann zum Ende des Optimierungsprozesses durchaus erwünscht sein, da hierdurch schnell ein (lokales) Optimum gefunden werden kann. Allerdings führt es zu Beginn der Berechnung mit hoher Wahrscheinlichkeit zum erstbesten lokalen Optimum und damit in der Regel zu einer suboptimalen Lösung.

Rangbasierte Selektion

Bei der rangbasierten Selektion werden die Individuen zunächst nach ihrer Fitness absteigend sortiert. So erhält jedes Individuum einen Rang in der Population. Über die Rangskala wird eine Wahrscheinlichkeitsverteilung definiert: Je höher der Rang (also je kleiner die Rangnummer), desto größer ist die Wahrscheinlichkeit ausgewählt zu werden. Mit dieser Verteilung wird anschließend eine Rouletteradauswahl durchgeführt.

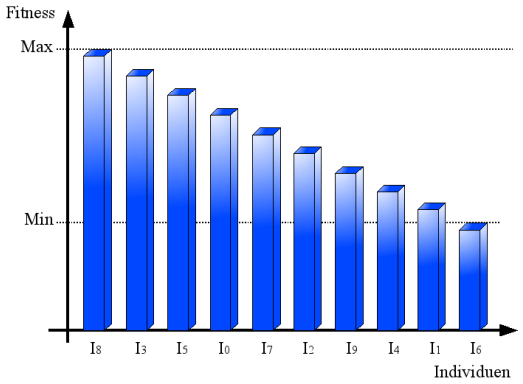


Abb. 3.6: Rangbasierte Selektion. Die Individuen werden nach absteigender Fitness sortiert. Die Auswahl geschieht über den Rang - nicht direkt über den Fitnesswert.

Dieses Verfahren besitzt den Vorteil, dass das Dominanzproblem weitgehend vermieden wird, da der Wert der Fitnessfunktion nicht direkt die Auswahlwahrscheinlichkeit beeinflußt. Des weiteren kann über die Wahrscheinlichkeitsverteilung auf der Rangskala der Selektionsdruck sehr einfach eingestellt werden. Nachteilig ist das zwingende Sortieren der Individuen, das zusätzlich Zeit kostet.

Turnierselektion

Bei diesem Verfahren wird eine bestimmte Anzahl von Individuen zufällig aus der Population ausgewählt. Diese Individuen werden miteinander verglichen, das mit dem höchsten Fitnesswert gewinnt und wird zur Reproduktion herangezogen. Über die Anzahl der ausgewählen Individuen (Turniergröße) kann der Selektionsdruck gesteuert werden. Das Dominanzproblem wird weitgehend vermieden, da die Fitnesswerte nur indirekt eingehen. Jedoch hat das schlechteste Individuum bei diesem Verfahren keine Chance ausgewählt zu werden.

3.3.2 Ersetzung

Die Ersetzung wird in der Literatur oft mit Selektion bezeichnet, da häufig die gleichen Algorithmen eingesetzt werden. Dabei wird zwischen Eltern- und Umweltselektion unterschieden. Zur besseren Unterscheidung wird in dieser Arbeit die Selektion von der Ersetzung getrennt. Letztere wird hier vorgestellt.

Die Ersetzung (engl. replacement) hat die Aufgabe, aus der Elternpopulation und der Nachkommenpopulation eine neue Population zu erstellen. Diese Folgegeneration fungiert im nächsten Durchlauf als Elternpopulation. Es wurden folgende Ersetzungsmethoden implementiert:

Komplette Nachkommenpopulation mit optionalem Elitismus

Eine implementierte Methode übernimmt die komplette Nachkommenpopulation. Da davon ausgegangen wird, dass die durchschnittliche Güte der Population von Generation zu Generation durch die anderen genetischen Operatoren steigt, findet hier keine weitere Selektion statt.

Dabei besteht jedoch die Gefahr, dass die Gene eines guten Lösungskandidaten zufällig oder durch den Einfluss von Rekombination oder Mutation nicht weitervererbt werden. Die beste Fitness könnte somit auch fallen. Um dem zu begegnen, wird eine festgelegte kleine Anzahl der besten Elternindividuen in die nächste Generation überführt. Die restliche Population wird wie gehabt durch die Nachkommenindividuen aufgefüllt. Dieses »Elitismus« genannte Verfahren gerantiert eine stetig wachsende Fitness, wobei die Gefahr besteht, dass die Diversität zurückgeht, da viele Nachkommen von diesen sehr fitten Individuen entstehen.

Beste Individuen aus Eltern- und Nachkommenpopulation

Diese Ersetzungmethode wählt immer die besten Individuen der Eltern- und Nachkommenpopulation aus. Damit ist sichergestellt, dass die beste Fitness monoton wächst. Auch hier besteht die Gefahr, dass sich die Gene der fitten Individuen, die ja schon vermehrt zur

Reproduktion herangezogen wurden, und vermutlich mit deren Genen gute Nachkommen erzeugt haben, überproportional vermehren.

3.3.3 Rekombination

Generell wird bei der Rekombination aus mehreren Elternindiviuden ein oder mehrere Kindindividuen erzeugt. Die in dieser Arbeit implementierten Rekombinationsoperatoren erzeugen stehts aus zwei Elternindividuen zwei Kindindividuen.

Die Rekombination der genetischen Informationen orientiert sich stark am biologischen Vorbild. Die verschiedenen Methoden kombinieren die Eigenschaften von unterschiedlichen Individuen neu, sodass im Optimalfall die vorteilhaften Bestandteile der Elternindividuen zu besseren Kindindividuen führen. Dieser Operator sorgt normalerweise dafür, dass weite Bereiche des Suchraums durchforstet werden. Man spricht dann von einem »expandierenden Operator«. Allerdings läßt sich bei mangelnder bzw. fehlender Diversität innerhalb der Population beobachten, dass dieser Operator auch zur lokalen Konvergenz führen kann. Es ist dann ein »kontrahierender Operator« [Wei02, S.82].

Ein-Punkt Rekombination

Bei der »Ein-Punkt Rekombination« (engl. one point crossover) wird zufällig eine Position innerhalb der Elternchromosomen bestimmt (siehe Abb. 3.7). An dieser Stelle werden die Chromosomen auseinandergeschnitten und über Kreuz miteinander verbunden. Die erzeugten Chromosomen bestehen aus dem ersten Teil des einen und dem zweiten Teil des anderen Elternchromosoms.

Dieses Vorgehen ist bei einer bitweisen Kodierung, wie sie bei der Greedy- und Max-Min Methode verwendet wird, sehr einfach durchzuführen. Bei der Permutationsmethode dagegen muss darauf geachtet werden, dass nach der Rekombination jeder Indexwert genau einmal vorhanden ist.

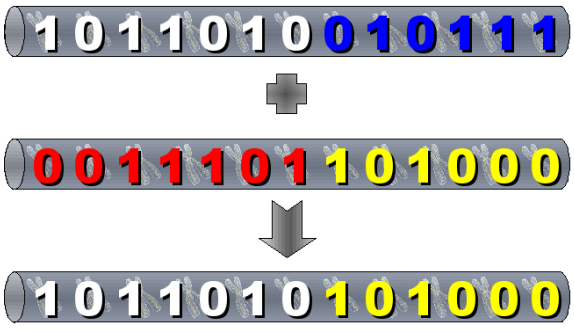


Abb. 3.7: Ein-Punkt Rekombination. Eine Position des Chromosoms wird zufällig bestimmt. An dieser Stelle kreuzen sich die Chromosomen. Das erzeugte Chromosom besteht aus dem ersten Teil des einen und dem zweiten Teil des anderen Elternchromosoms.

Deshalb wird in dem Fall wie gehabt zunächst der erste Teil des einen Chromosoms übernommen. Danach werden die restlichen Gene des anderen Elternchromosoms übertragen, falls diese auf dem Nachkommenchromosom noch nicht vorhanden sind. Die evtl. noch fehlenden Gene werden schließlich der Reihe nach in die unbesetzten Positionen eingefügt. So ist sichergestellt, dass jeder Eingabetestmusterindex genau einmal auf jedem Chromosom existiert. Es hat sich lediglich die Reihenfolge geändert.

Zwei-Punkt Rekombination

Der Ablauf der »Zwei-Punkt Rekombination« (engl. two point crossover) ist der »Ein-Punkt Rekombination« (vgl. Abb. 3.8) sehr ähnlich. Die Verfahren unterscheiden sich sich einzig darin, dass nicht eine, sondern zwei Schnittstellen zufällig bestimmt werden. An diesen Positionen werden die Elternchromosomen getrennt und über Kreuz zusammengefügt.

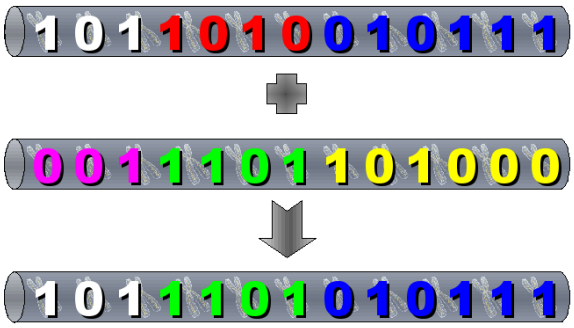


Abb. 3.8: Zwei-Punkt Rekombination. Zwei Positionen des Chromosoms werden zufällig bestimmt. An diesen Stellen kreuzen sich die Chromosomen. Das erzeugte Chromosom besteht aus dem ersten und letzten Teil des einen und dem mittleren Teil des anderen Elternchromosoms.

Es entstehen Chromosomen, die aus dem ersten und letzten Teil des einen und dem mittleren Teil des anderen Elternchromosoms bestehen (siehe Abb. 3.8).

Auch hier muss beim Einsatz während der Permutationsmethode darauf geachtet werden, dass jeder Indexwert genau einmal erhalten bleibt. Das geschieht analog zu dem beim »Ein-Punkt Rekombination« beschriebenen Verfahren.

Uniforme Rekombination

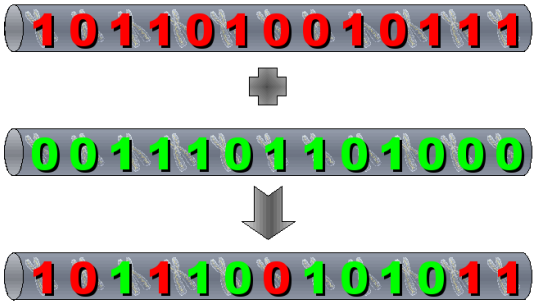


Abb. 3.9: Uniforme Rekombination. Die Genpositionen werden der Reihe nach durchlaufen. Für jede Position wird entschieden, ob das Gen vom einen oder anderen Elternchromosom übernommen wird.

Bei diesem Rekombinationsverfahren werden die Positionen der Chromosomen der Reihe nach durchlaufen. Für jedes Gen wird zufällig entschieden, ob es vom Chromosom des einen oder anderen Elternindividuums übernommen wird (siehe Abb. 3.9).

Dieses für ein bitkodiertes Chromosom einfach zu implementierende Verfahren muss für die Permutationsmethode angepasst werden. Dort werden zunächst die Gene eines Elternindividuum der Reihe nach durchlaufen und gegebenenfalls übernommen. Die freien Plätze werden durch die noch nicht verwendeten Gene des anderen Elternindividuums in der auftretenden Reihenfolge belegt. So ist gewährleistet, dass der Genotyp stets gültig ist.

3.3.4 Mutation

Mutatationsoperatoren ändern den Genotyp der Individuen. Je nach Stärke der Änderung kommen den Mutationsoperatoren verschiedene Rollen zu. Oft wird durch die Mutation nur eine kleine Modifikation vorgenommen. Es ergibt sich daraus in der Regel eine relativ kleine Abweichung des Gütewertes. In solchen Fällen wird in der Nähe des Individuums eine Feinabstimmung (exploitation) vorgenommen. Die Hauptaufgabe des genetischen Operators liegt dann darin, die Erreichbarkeit aller Punkte im Suchraum zu gewährleisten.

Mutationsoperatoren, die größere Modifikationen am Genotyp vornehmen, haben dagegen oft die Aufgabe, weite Bereiche des Suchraums zu erkunden (exploration) [Wei02, S. 77].

Bit-Flip Mutation

Die »Bit-Flip Mutation« eignet sich besonders für bitkodierte Chromosomen. Einige Genpositionen werden zufällig ausgewählt. An diesen Stellen wird der Wert des Gens »umgedreht« (engl. umdrehen: to flip). Aus einer 1 wird eine 0 und umgekehrt.

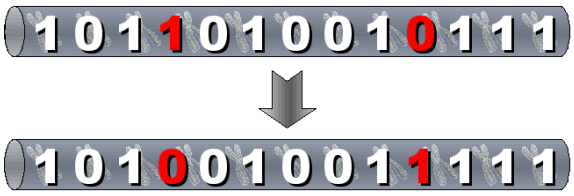


Abb. 3.10: Bit-Flip Mutation. Der Wert zufällig bestimmter Gene wird umgedreht.

Daher und durch die interne Kodierung der Gene durch Bits leitet sich der Name dieses Mutationsoperators ab.

Die »Bit-Flip Mutation« wird ausschließlich für die Bitkodierten Lösungsmethoden eingesetzt.

Genaustausch Mutation

Bei dieser Methode werden zwei zufällig ermittelte Gene ausgetauscht. In Abb. 3.11 sind das die grün und rot gefärbten Werte.

Hierbei ändert sich das Chromosom nicht zwangsläufig! Sind die Werte der Gene gleich, ist die Mutation von außen nicht zu beobachten.

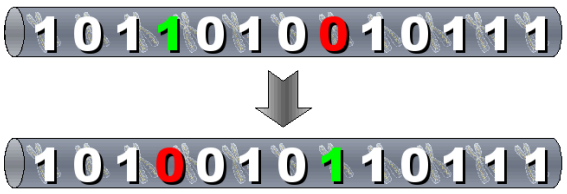


Abb. 3.11: Gen-Austausch Mutation. Zwei zufällige ausgewählte Gene werden ausgetauscht.

Dieser Mutationsoperator ist sowohl für die bitkodierten Lösungsmethoden als auch für die Permutationsmethode verwendbar, denn es ist sichergestellt, dass durch diesen Operator alle Indexwerte genau einmal erhalten bleiben.

Gensequenz verschieben

Hierbei wird zunächst eine zufällige Gensequenz bestimmt. Diese wird aus dem betreffenden Chromosom ausgeschnitten und an einer beliebigen anderen Position eingefügt. Die Gensequenz wird also verschoben.

Inverse Mutation

Dieser Mutationsoperator kann für bitkodierte wie auch für nummernkodierte Chromosomen gleichermaßen eingesetzt werden.

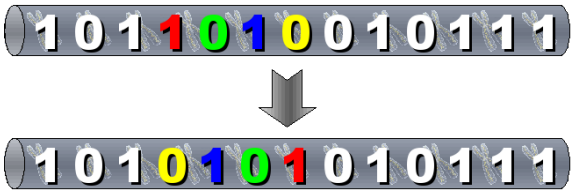


Abb. 3.12: Inverse Mutation. Zwei zufällige ausgewählte Gene werden ausgetauscht.

Der Algorithmus arbeitet folgendermaßen: Zunächst wird die Anfangsposition a und die Länge n einer Gensequenz des Chromosoms $C =< x_1, x_2, \ldots, x_k >$ zufällig bestimmt (bunter Abschnitt in Abb. 3.12 oben). Nun werden die Gene x_{a+i} mit $x_{a+(n-1)-i}$ für $i = 0 \ldots (n \div 2)$ getauscht. Dadurch wird die Gensequenz umgekehrt (siehe Abb. 3.12 unten).

Chromosomen Mutationen

Speziell für die MaxMin Methode wurden drei Mutationsoperatoren implementiert. Diese wirken jeweils auf ein vollständiges Chromosom.

Der erste Operator fügt einem Individuum ein Chromosom hinzu. Dieses wird nur dann eingefügt, wenn noch nicht abgedeckte Testmuster vorhanden sind. Das eingefügte Chromosom wird kompatibel zu diesem Testmuster initialisiert. Das Gegenstück liefert der zweite Operator: Er entfernt ein Chromosom, wenn dieses nicht benötigt wird. Die dritte Methode tauscht zufällig die Positionen zweier Chromosomen innerhalb eines Individuums.

Mit Hilfe dieser Verfahren ist es der MaxMin Methode möglich, die Anzahl der Chromosomen und damit der Lösungstestmuster zu beeinflussen.

3.3.5 2-Opt Mutation

Der 2-Opt Algorithmus beruht auf der r-Optimalität. Dieses Verbesserungsverfahren wurde speziell für das Problem des Handelsreisenden entwickelt. Es wird hier für die Permutationsmethode angepasst (vgl. Kap. 3.1.3).

Der beste Algorithmus zur Lösung des Problems ist zur Zeit der von Keld Helsgaun[2] [Hel00]. Der dänische Wissenschaftler setzt dazu eine modifizierte Version der Lin-Kernighan-Heuristik [LK73] ein, die eine variable r-Optimierung durchführt.

Per Definition wird eine Rundreise als r-optimal bezeichnet, wenn es unmöglich ist, die Rundreise durch den Austausch von r in ihr enthaltenen Kanten gegen r nicht enthaltene Kanten zu verkürzen. Im Allgemeinen steigt die Wahrscheinlichkeit, eine optimale Rundreise zu berechnen, je größer r gewählt wird. Allerdings steigt mit zunehmendem r der Zeitaufwand rapide an, weshalb hier r = 2 gewählt wurde.

2-Opt erzeugt 2-optimale Reihenfolgen. Der angepasste Algotithmus arbeitet nach diesem Schema: Ersetze zwei Verbindungen einer gegebenen Abarbeitungsreihenfolge mit zwei anderen Verbindungen in der Form, dass die neue Abarbeitungsreihenfolge besser ist. Führe das so lange aus, bis keine weitere Verbesserung möglich ist.

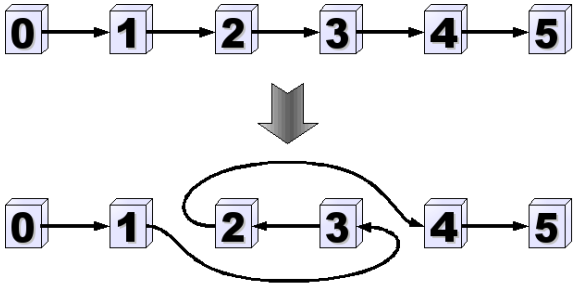


Abb. 3.13: 2-Opt Mutation. Wenn sich eine Verbesserung ergibt wird die Abarbeitungsreihenfolge geändert, indem jeweils 2 Verbindungen getauscht werden. Das wird so lange wiederholt, bis keine weitere Verbesserung erzielt werden kann.

Am Beispiel in Abb. 3.13 wird das Verfahren verdeutlicht: Gegeben sei die auf einem Chromosom kodierte Reihenfolge 0-1-2-3-4-5. Nun werden die Verbindungen 1-2 und 3-4 entfernt und durch 1-3 sowie 2-4 ersetzt. Die Güte des Individuums wird mit dieser Permutation überprüft. Ist sie besser als die vorherige, bleibt die Änderung bestehen, ansonsten wird sie revidiert. Dieser Vorgang wird so lange durchgeführt, bis die Reihenfolge 2-optimal ist.

Ein Problem bei diesem Verfahren stellt die Berechnung der Fitness dar. Sie ist relativ zeitintensiv und wird sehr oft durchgeführt. Daher wird mit einer Heuristik gearbeitet, welche die Ähnlichkeit zweier Testmuster zu grunde legt.

Zwei Testmuster werden dabei verglichen. Sind sie nicht kompatibel zueinander ergibt sich ein Wert von 0. Ansonsten werden sie der Reihe nach verglichen. Zwei identische Werte erhöhen das Ähnlichkeitsmaß stark, ein fester Wert und ein X führen zu einer kleinen Erhöhung.

3.3.6 Fitnessfunktionen

In der Fitnessfunktion muss zunächst der Genotyp in den Phänotyp überführt werden, da letzterer im Suchraums Ω liegt. Dann kann mit Hilfe der Abbildungsfunktion $f : \Omega \rightarrow \mathbb{R}$ die Güte des jeweiligen Individuums berechnet werden.

[2] Stand: 9. August 2006; Bester Weg für 1.904.711 Städte der Welt. (Quelle: http://www.tsp.gatech.edu/world/)

Je nach Lösungsmethode werden verschiedene Fitnessfunktionen benutzt. Allen gemeinsam ist, dass der Fitnesswert umso höher ist, je mehr Testmuster kompaktiert werden können.

Bei der MaxMin Methode geht darüber hinaus die Chromosomenanzahl negativ ein. Das heißt also, je mehr Chromosomen ein Individuum hat, je größer also die kompaktierte Testmusteranzahl ist, desto kleiner ist der Fitnesswert. Außerdem wird die Diversität der Chromosomen innerhalb eines Individuums berechnet. Eine größere Diversität geht positiv in den Fitnesswert ein.

Bei der Permutationsmethode wird entsprechend der auf dem Chromosom kodierten Reihenfolge ein Ähnlichkeitswert bestimmt. Dieser ist die Summe aus den Ähnlichkeitswerten jeweils zweier benachbarter Testmuster. Je höher dieser Wert, desto höher der Fitnesswert.

3.3.7 Abbruchkriterien

Während des Optimierungsprozesses werden die Individuen einer Population von Generation zu Generation verbessert. Da bei den zu lösenden Suchproblemen in der Regel nicht bekannt ist, welchen Wert das Optimum besitzt, der Algorithmus also terminieren muss, existieren eine Reihe von Abbruchkriterien. Vier davon wurden implementiert. Diese können auch parallel angegeben werden. Es wird das Abbruchkriterium benutzt, welches als erstes erfüllt ist.

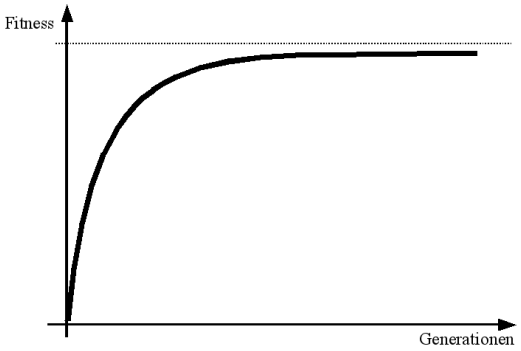


Abb. 3.14: Typische Fitnesswertentwicklung. Am Anfang der Berechnung ergeben sich schnellere und höhere Fitnesszugewinne. Die Fitnesssteigerung nimmt mit zunehmender Berechnung ab.

Zunächst kann eine maximale Berechnungszeit angegeben werden nach der die Berechnung stoppt. Eine zweite Methode ist das Festlegen einer bestimmten Güte. Beim Erreichen eines Fitnesswerts terminiert der Algorithmus. Die letzten zwei Verfahren beziehen die Anzahl der Generationen ein. Entweder kann das statisch sein, d. h. es wird nach einer festgelegten Anzahl von Generationen aufgehört, oder der Algorithmus kann nach einer bestimmten Anzahl von Generationsdurchläufen ohne Fitnessverbesserung beendet werden. Letzteres beruht auf der Beobachtung, dass die Güte (mit Elitismus) typischerweise wie in Abb. 3.14 dargestellt gesteigert wird. Am Anfang der Berechnung steigt der Fitnesswert schnell, im Laufe der Optimierung nimmt die Fitnesssteigerung beständig ab. Man kann also davon ausgehen, dass der Genetische Algorithmus ein Optimum gefunden hat wenn über mehrere Generationen keine Steigerung der Fitness stattfindet.

3.4 Implementierung

Das Programm wurde unter Linux in C++ mit Qt 3.0 entwickelt. Es kann über ensprechende Startparameter entweder als Konsolenapplikation oder als Programm mit grafischer Benutzeroberfläche gestartet werden (siehe dazu Abb. 3.15). Alle Programmfunktionen wurden komplett selbst geschrieben, es wurde auf den Einsatz von Bibliotheken (z. B. GAlib[3]) verzichtet, da so größtmögliche Entwicklungsfreiheit bestand.

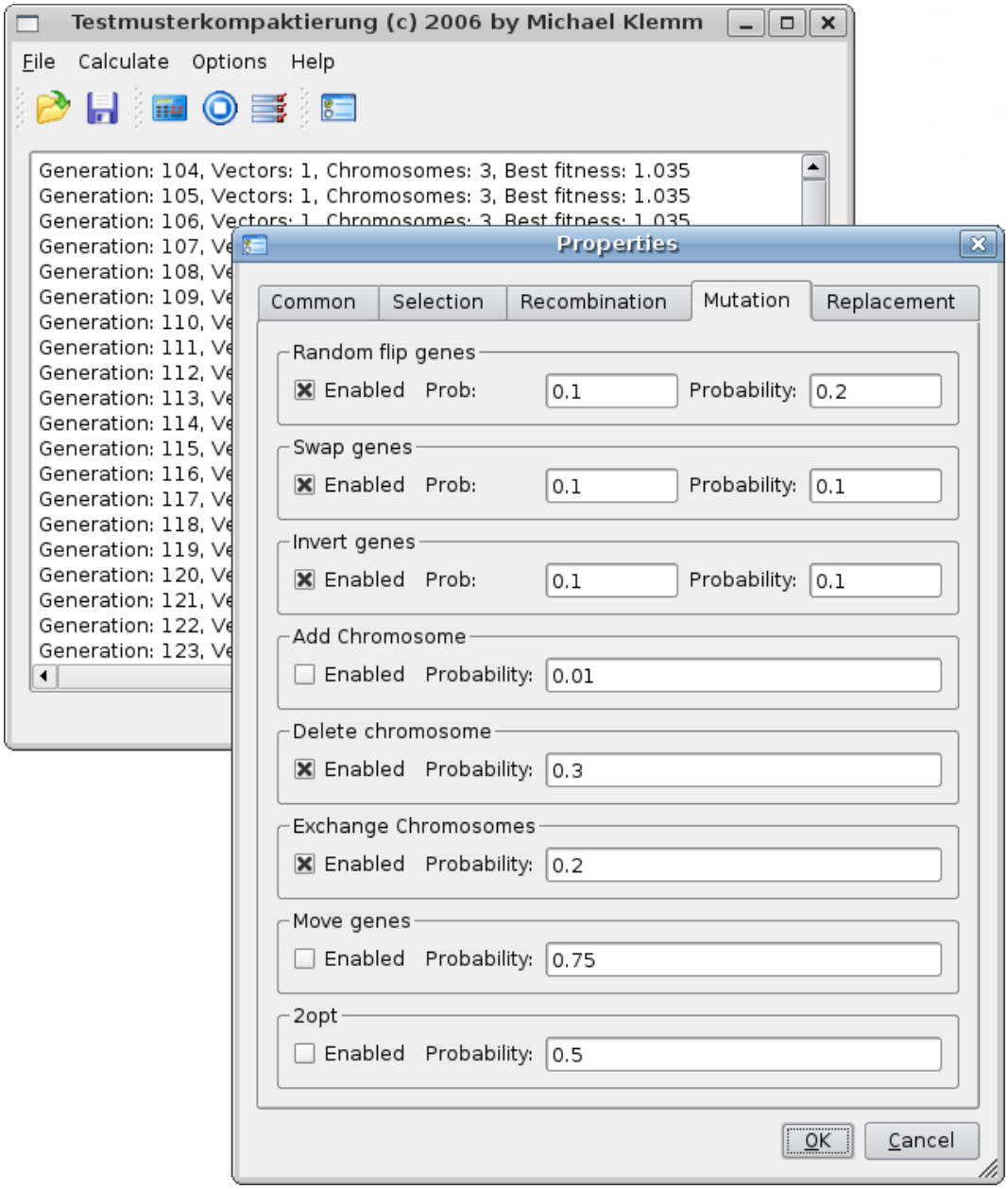


Abb. 3.15: Screenshot des Programms. Das Hauptfenster im Hintergrund gibt Zwischenergebnisse aus. Im Einstellungsdialog im Vordergrund können die wichtigsten Parameter eingegeben werden.

[3]GAlib: http://lancet.mit.edu/ga/

Auf zwei Implementierungsdetails soll hier näher eingegangen werden: Die effiziente Speichernutzung und die Geschwindigkeitsoptimierung an zentraler Stelle.

3.4.1 Effiziente Speichernutzung

Symbol	Hi Bit	Lo Bit
0	0	1
1	1	0
X	1	1
Ende	0	0

Tab. 3.2: Kodierung der Symbole durch zwei Bit.

Um eine hohe Rechengeschwindigkeit zu erreichen, ist es wichtig, die Eingabetestmuster im Hauptspeicher zu halten. Da die Datenmengen sehr groß werden können und die Speicherressourcen beschränkt sind, wurde eine effiziente Speichernutzung implementiert:
Die Eingabetestmuster werden intern als »Strings« gehalten, wobei ein Zeichen (also ein Byte) vier Symbole kodiert. Die Kodierung eines Symbols ist der Tab. 3.2 zu entnehmen. Da für die Repräsentation eines Symbols zwei Bit benötigt werden, können genau vier Symbole in einem Byte untergebracht werden. Bei Testmusterlängen, die größer als vier sind, werden mehrere kodierte Bytes hintereinander betrachtet.

Beispielsweise wird das Testmuster	`T = XX10 1001 X1X`
als Bitfolge	`11111001 10010110 11101100`
kodiert, was hexadezimal der Bytefolge	`$F9 $96 $EC`
entspricht.	

Derart kodiert wird lediglich ein viertel des ursprünglich benötigten Speichers belegt.

3.4.2 Geschwindigkeitsoptimierung

Ohne Geschwindigkeitsoptimierung dauerten die Berechnungen sehr lange. Einen großen Anteil daran hatte die Fitnessfunktion. Bei allen Lösungsmethoden wird während ihres Ablaufs eine erhebliche Rechenzeit benötigt um festzustellen, ob zwei Testmuster zueinander kompatibel sind. Neben diesem wesentlichen wurden drei weitere Punkte identifiziert, bei denen eine Geschwindigkeitsoptimierung messbare wesentliche Verbesserungen brachte. Dies sind die Berechnungen der

1. kompaktierten Muster von zwei Eingabetestmustern.
2. Ähnlichkeitswerte zweier Muster.
3. Relevanz eines Testmusters bezüglich eines anderen.

Naheliegend wäre, vor dem Optimierungsprozess vier Matrizen aus den Testmustern zu berechnen. Während der Optimierung könnte dann sehr schnell auf die entsprechenden Einträge zugegriffen werden. Allerdings hätte jede Matrix eine Größe von n × n (mit n

Testmustern). Da die Größe schon bei mittelgroßen Testinstanzen den Speicherplatz sprengen würde, wurde ein modifizierte Variante implementiert.

Dabei wurde die im letzten Abschnitt beschriebene Kodierung ausgenutzt. Da bis zu vier Symbole in einem Byte gespeichert werden und ein Byte 256 Werte annehmen kann, wurden die Matrizen mit einer Größe von jeweils 256×256 erzeugt und die entsprechend berechneten Werte in ihnen abgelegt.
Bei der späteren Berechnung muss dann nur noch der entsprechende Eintrag nachgeschlagen werden. Das muss entsprechend der Testmusterlänge gegebenenfalls mehrmals durchgeführt werden, wobei die einzelnen Werte summiert bzw. logisch verknüpft werden. Durch diese vier Matrizen konnten die Rechenzeiten wesentlich verkürzt werden.

4 Experimentelle Ergebnisse

In diesem Kapitel werden die experimentell ermittelten Ergebnisse präsentiert. Zunächst werden die verwendeten Testdaten, die Testumgebung und das Testverfahren beschrieben. Im Anschluss daran folgen die Testergebnisse der einzelnen Lösungsansätze, die abschließend miteinander verglichen werden.

4.1 Testdaten

Es standen mehrere Testmusterdateien zur Verfügung, um die Geschwindigkeit und Güte der Lösungsansätze zu testen. Dabei handelte es sich um Testmuster, die aus den Iscas'89-Schaltungen generiert wurden. Zum einen wurden diese mit einem Testmustergenerator von Philips Semiconductors erzeugt und zur Unterscheidung mit dem Postfix »_S« gekennzeichnet; die charakteristischen Werte sind in Tab. 4.1 aufgeführt. Zum anderen wurden aus der gleichen Quelle Testmuster mit dem PODEM Algorithmus generiert. Diese beinhalten im Vergleich zu ersteren eine größere Testmusterzahl. Von den mit PODEM generierten Testdaten sind optimalen Lösungen bekannt (Spalte Min). Als Postfix wird »_P« verwendet, die Werte finden sich in Tab. 4.2.

In den Tabellen ist die Schaltungsbezeichnung (Spalte »Schaltung«), die Anzahl der Testmuster (Spalte »Anzahl«), die Testmusterlänge (Spalte »Länge«) sowie der prozentuale Anteil an undefinierten Symbolen (Spalte »X%«) aufgeführt. Die Werte der Spalte »Relevant« gibt die Anzahl relevanter Testmuster an. Diese wurden mit dem Verfahren aus Kap. 3.2 ermittelt. Der Algorithmus entfernt irrelevante Testmuster. Von den zwei Testmustern `A = XX1OX` und `B = XXXOX` ist das zweite irrelevant, da es bereits vollständig durch das Testmuster A abgedeckt ist und keine festen Werte an Positionen enthält, an denen sich in A undefinierte Symbole (X) befinden. In der Spalte »Max« sind die Werte enthalten, welche die Hilfsfunktion aus Kap. 3.2 als maximale Anzahl an Testmustern ermittelt hat. Die Tabelle mit den PODEM Testmustern (Tab. 4.2) enhält zusätzlich die Angabe der bekannten besten Lösungen (Spalte Min).

Auffällig ist, dass die PODEM-Testdaten viele irrelevante Testmuster enthielten. Dadurch konnte die Datenmenge teilweise auf bis zu 7 % (s1488_P) der Ausgangsgröße reduziert werden. Die mit dem Philips Semiconductors Testgenerator erzeugten Daten wiesen dagegen nur wenige irrelevanten Testmuster auf.

Schaltung	Anzahl	Länge	X %	Relevant	Max
s27_S	21	7	53 %	15	7
s298_S	163	17	72 %	94	29
s344_S	124	24	79 %	90	25
s349_S	127	24	79 %	90	24
s382_S	138	24	79 %	106	30
s386_S	158	13	39 %	83	76
s400_S	127	24	80 %	97	27
s444_S	134	24	80 %	97	28
s510_S	159	25	73 %	100	74
s526_S	241	24	74 %	168	67
s641_S	251	54	88 %	211	44
s713_S	230	54	88 %	197	36
s820_S	344	23	63 %	196	117
s832_S	340	23	63 %	197	118
s1423_S	490	91	92 %	405	68
s1488_S	393	14	44 %	241	137
s1494_S	393	14	44 %	240	139
s5378_S	1569	199	95 %	1216	187
s9234_S	2105	247	94 %	1573	228
s13207_S	3351	700	99 %	2616	280
s15850_S	3858	611	99 %	3009	200

Tab. 4.1: Mit einem Testmustergenerator von Philips Semiconductors erzeugte Iscas'89 Testmuster. Die Testdaten weisen nur einen sehr geringen Anteil irrelevanter Testmuster auf.

Schaltung	Anzahl	Länge	X %	Relevant	Max	Min
s298_P	1800	17	74 %	313	62	38
s526_P	3537	24	78 %	531	96	74
s820_P	6955	23	67 %	956	238	186
s832_P	7040	23	67 %	981	264	183
s1196_P	15553	32	75 %	1967	551	204
s1238_P	16288	32	75 %	2029	558	306
s1423_P	9554	91	92 %	2474	243	205
s1488_P	15745	14	48 %	1127	390	205
s1494_P	15865	14	48 %	1192	366	327
s5378_P	82517	214	95 %	23173	892	721

Tab. 4.2: Mittels PODEM erzeugte Iscas'89 Testmuster. Der Anteil irrelevanter Testmuster ist sehr groß.

Bemerkenswert ist außerdem, dass mit dem Verfahren zur Abschätzung der maximalen Testmusteranzahl, also mit einem sehr einfachen Algorithmus, bereits gute Kompaktierungsergebnisse erreicht werden konnten. Verglichen mit den bekannt optimalen Ergebnissen wurden lediglich bei s1196_P und s1238_P deutlich schlechtere Kompaktierungen berechnet.

4.2 Testumgebung

Die Tests wurden auf mehreren Computern durchgeführt. Diese besaßen mindestens 1 GB Hauptspeicher, sodass die Eingabetestmuster komplett im Speicher gehalten werden konnten. Da es sich um Rechner mit unterschiedlichen CPUs und Taktfrequenzen handelte, auf denen teilweise weitere rechenintensive Prozesse liefen, wurden die Prozesszeiten ermittelt und normiert. Dazu wurden die Zeiten auf den schnellsten Computer, einen AMD Athlon 64 3500+, umgerechnet.

4.3 Testverfahren

Zunächst wurden für alle Lösungsmethoden die besten Parameter experimentell ermittelt. Dazu wurden umfangreiche Tests mit kleinen Populationen (50 Individuen) und das Abbruchkriterium »keine Verbesserung nach 100 Generationen« gewählt. Dadurch konnten viele Tests in kurzer Zeit absolviert werden. Nacheinander wurden systematisch die Parameter variiert, wobei nicht betreffende Funktionen möglichst abgeschaltet wurden. Da es sich um ein probabilistisches Verfahren handelt, wurden bei jedem Versuch 5 Durchläufe berechnet.

Ein-Punkt Rekomb.	Zwei-Punkt Rekomb.	Uniforme Rekomb.	Anzahl Testmuster
-	-	X	217 - 220
-	X	-	183 - 195
X	-	-	190 - 193
-	X	X	186 - 200
X	-	X	185 - 192
X	X	-	**133 - 156**
X	X	X	141 - 186

Tab. 4.3: Parametereinstellung der Rekombinationsmethoden am Beispiel der Permutationsmethode mit den Tetdaten s298_P. Das beste Ergebnis erzielten die Ein-Punkt und Zwei-Punkt-Rekombination zusammen.

Beispielhaft sei hier ein Teil der Parametereinstellungen für die Permutationsmethode beschrieben. Zunächst wurden alle genetischen Operatoren abgeschaltet. Es wurde die fitnessproportionale Selektion und die Ersetzung der besten Eltern und Kinder gewählt. Nun wurden die Rekombinationsparameter ermittelt. Dazu wurden alle Kombinationen durchgetestet. In den Fällen mit mehreren Operatoren wurden diese gleichwahrscheinlich zur Rekombination herangezogen. Wie Tab. 4.3 zeigt wurden die besten Ergebnisse mit der Ein- und Zwei-Punkt-Rekombination erzielt.

Nachdem die beiden favorisierten Methoden ermittelt waren, folgte die Paramterermittlung der Auftrittswahrscheinlichkeiten (vgl. dazu Tab. 4.4), bei der sich schließlich ergab, dass die der Ein-Punkt-Rekombination 80 % und die der Zwei-Punkt-Rekombination bei 20 % lag.

Ein-Punkt Rekomb.	Zwei-Punkt Rekomb.	Anzahl Testmuster
0.1	0.9	151 - 187
0.2	0.8	139 - 172
0.3	0.7	138 - 173
0.4	0.6	135 - 163
0.5	0.5	133 - 156
0.6	0.4	127 - 153
0.7	0.3	122 - 154
0.8	0.2	**116 - 150**
0.9	0.1	137 - 179

Tab. 4.4: Auswahlwahrscheinlichkeiten der Rekombinationsmethoden. Das beste Ergebnis ist fett hervorgehoben.

Mit diesen Werten wurden danach in gleicher Weise die Parameter für die Mutations-, die Selektions- und die Ersetzungsoperatoren ermittelt. Schließlich wurden die Populationsgröße und die Anzahl der Generationen ohne Verbesserungen getestet. Letzteres stellte einen Kompromiss zwischen Rechenzeit und Qualität dar: Je größer die Individuenanzahl und die Generationen als Abbruchkriterium gewählt werden, desto besser werden Ergebnisse und desto länger fällt die Berechnungszeit aus.

Die so ermittelten Werte wurden verwendet, um die Berechnung der Testdaten durchzuführen. Für alle Testdaten der Philips-Datensätze wurden drei Durchläufe berechnet.

Auch bei der Greedy Methode konnten die PODEM-Datensätze mit drei Durchläufen berechnet werden. Die anderen Methoden mussten auf einen Durchlauf reduziert werden, da die Rechenzeiten zu lang wurden. Bei mehreren Berechnungszyklen wurde das jeweils beste Ergebnis in die Ergebnisliste übernommen. Die angegebenen Zeiten sind die Durchschnittswerte der Berechnungen. In der Regel waren die Abweichungen sowohl bei den Ergebnissen als auch bei den Zeiten minimal.

4.4 Greedy Methode

Zunächst wurden Tests durchgeführt, um geeignete Parameter für diese Lösungsmethode zu ermitteln. Die experimentell erhaltenen Parameter stellten einen Kompromiss zwischen guten Ergebnissen und geringer Rechenzeit dar. Sie wurden folgendermaßen festgelegt:

- Populationsgröße: 100 Individuen
- Abbruchkriterium: nach 30 Generationen ohne Verbesserung
- Selektion: fitnessproportional
- Rekombination: uniforme Rekombination
- Replacement: beste Individuen aus Eltern- und Nachkommenpopulation (0.6), alle Kinder mit Elitismus (0.4)
- Mutation: Zufallsmutation (0.5), Bittausch (0.1) und Invert (0.1)

Der typische Verlauf abgedeckter Testmuster über die Generationen hinweg ist in Abb. 4.1 am Beispiel der Testdaten s298_P dargestellt. Die Methode optimiert die Individuen dahingehend, dass möglichst viele Testmuster abgedeckt werden. Dies ist gut am Diagrammanfang zu erkennen. Dort sieht man beim ersten Bereich, dass die Lösungsmuster optimiert werden, bis keine weitere Verbesserung über die festgelegte Generationenanzahl erfolgt. Das Muster des besten Individuums wird in die Ausgabetestmustermenge übernommen und die von ihm abgedeckten Eingabetestmuster aus der Eingabemenge entfernt. Danach startet der nächste Optimierungsdurchlauf im reduzierten Suchraum. So ergeben sich die Sprünge im Diagramm.

Am Anfang existieren viele Testmuster, die sich leicht zusammenfassen lassen. Deshalb steigt die Anzahl der kompaktierten Testmuster zunächst sehr stark. Je mehr Optimierungsdurchläufe erfolgen, desto weniger Testmuster lassen sich miteinander kompaktieren. Entsprechend flacher verläuft die Kurve zum Ende hin.

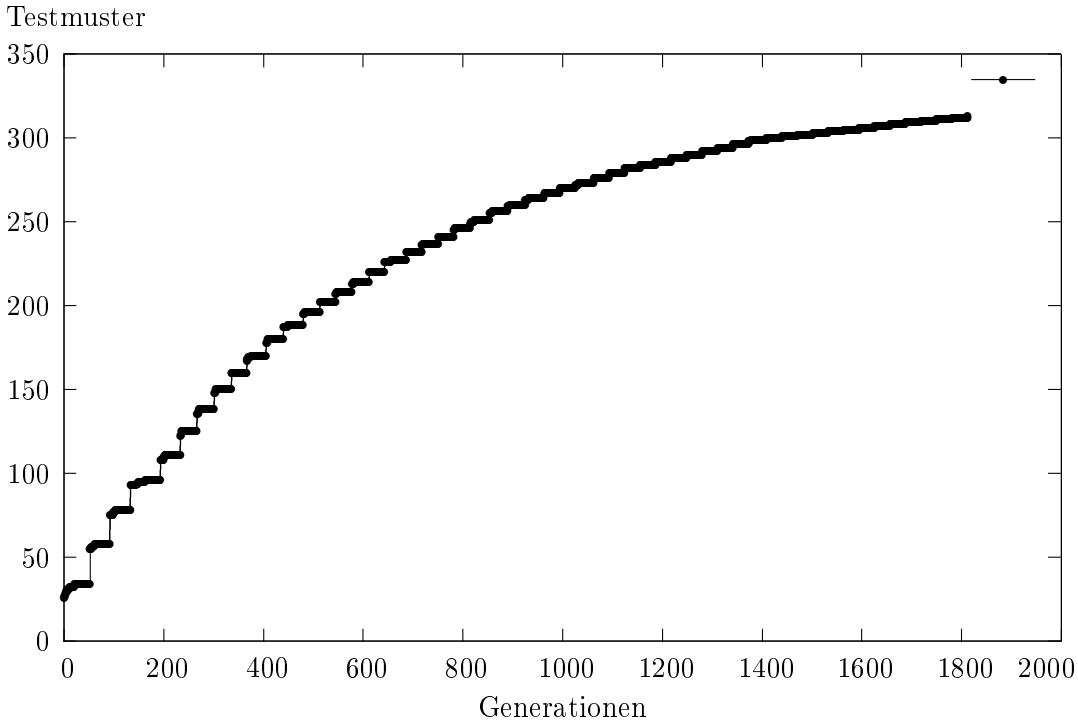


Abb. 4.1: Verlauf der Greedy Methode am Beispiel der Testdaten 298_P. Aufgezeichnet ist die Anzahl abgedeckter Testmuster über die Generationen. Typisch ist der schnelle Anstieg am Anfang. Hier werden sehr viele Testmuster zusammengefasst. Zum Ende können immer weniger Muster kompaktiert werden, sodass die Kurve entsprechend abflacht. Die Sprünge ergeben sich dadurch, dass der Algorithmus nach der festgelegten Anzahl Generationen ohne Verbesserung das aktuell beste Muster übernimmt, die abgedeckten Eingabetestmuster entfernt und mit einer neuen Suche beginnt.

Die Ergebnisse der Berechnungen sind in Tab. 4.5 aufgeführt. Hier sind für alle Testdaten die Eingabetestmusteranzahl (Spalte »Anzahl«), die Zahl der relevanten Testmuster (»Relevant«), die aus der Literatur bekannt besten Werte (»Min«) sowie die von der Greedy Methode erreichte Kompaktierung (Spalte »Kompakt«), diese prozentual im Vergleich

zur Gesamttestmusterzahl (»Prozent«) und die für die Berechnung benötigte Zeit (Spalte »Zeit«) aufgelistet.

Schaltung	Anzahl	Relevant	Min	Kompakt	Prozent	Zeit
s27_S	21	15		7	33 %	5 Sek.
s298_S	163	94		26	16 %	5 Sek.
s344_S	124	90		25	20 %	5 Sek.
s349_S	127	90		25	20 %	10 Sek.
s382_S	138	106		30	22 %	10 Sek.
s386_S	158	83		76	48 %	10 Sek.
s400_S	127	97		27	21 %	10 Sek.
s444_S	134	97		31	23 %	10 Sek.
s510_S	159	100		72	45 %	20 Sek.
s526_S	241	168		68	28 %	20 Sek.
s641_S	251	211		45	18 %	30 Sek.
s713_S	230	197		40	17 %	30 Sek.
s820_S	344	196		113	33 %	30 Sek.
s832_S	340	197		111	33 %	30 Sek.
s1423_S	490	405		72	15 %	2 Min.
s1488_S	393	241		129	33 %	30 Sek.
s1494_S	393	240		128	33 %	30 Sek.
s5378_S	1569	1216		241	15 %	15 Min.
s9234_S	2105	1573		284	14 %	22 Min.
s13207_S	3351	2616		308	9 %	1:20 Std.
s15850_S	3858	3009		244	6 %	52 Min.
s298_P	1800	313	38	54	3 %	15 Sek.
s526_P	3537	531	74	90	3 %	30 Sek.
s820_P	6955	956	186	232	3 %	1:30 Min.
s832_P	7040	981	183	247	4 %	1:45 Min.
s1196_P	15553	1967	204	520	3 %	5:30 Min.
s1238_P	16288	2029	306	531	3 %	5:30 Min.
s1423_P	9554	2474	205	276	3 %	9 Min.
s1488_P	15745	1127	205	338	2 %	1:45 Min.
s1494_P	15865	1192	327	**325**	2 %	1:45 Min.
s5378_P	82517	23173	892	1014	1 %	7:40 Std.

Tab. 4.5: Ergebnisse der Greedy Methode. Schnelle Berechnungszeiten und akzeptable Kompaktierungsergebnisse kennzeichnen diesen Ansatz. In der Tabelle sind die Eingabetestmusteranzahl (Anzahl), relevanten Testmuster (Relevante), bekannt besten Lösungen (Min), mit der Greedy Methode berechnete kompaktierte Anzahl (Kompakt), in Bezug auf die Gesamtanzahl prozentuale Verringerung (Prozent) sowie die für die Berechnung benötigte normalisierte Zeit (Spalte Zeit) aufgelistet.

Es zeigt sich, dass die Rechenzeiten bei den kleineren Testdaten sehr kurz waren. Das ist auf das Abbruchkriterium mit den gewählten 30 Generationen ohne Verbesserung zurückzuführen. Wählt man hier einen höheren Wert, so werden einzelne Ergebnisse geringfügig besser, andere dafür geringfügig schlechter. In jedem Fall aber dauern die Berechnungen wesentlich länger.

Die Berechnungszeiten nehmen mit zunehmender Testmusteranzahl und vor allem Testmusterlänge zu. Dazu vergleiche man die Testmusterdaten s13207_S und s15850_S. s13207_S weist eine um 393 geringere relevante Testmusterzahl, dafür aber um 89 Symbole längere Testmuster auf. Dennoch ist die benötigte Rechenzeit für s13207_S um etwa eine halbe Stunde länger als die für s15850_S. Der Grund hierfür ist, dass der Suchraum mit zunehmender Testmusterlänge exponentiell steigt. Das hat einen größeren Einfluß, als die größere Testmusteranzahl, welche die Rechenenzeit in etwa linear steigen läßt (vgl. s382_S und s820_S).

Die erreichten Kompaktierungswerte sind sehr unterschiedlich. Bei den Philips-Testdaten lagen die prozentualen Verbesserungen in Bezug zu den Ausgangsdaten im Bereich von 10 bis 50 Prozent, bei den PODEM-Testmustern lagen die Kompaktierungsraten bei unter fünf Prozent. Zu den letztgenannten Daten existieren aus der Literatur [SR05] bekannt minimale Werte. Im Vergleich zu diesen liegen die erreichten Kompaktierungsraten teilweise nur unwesentlich darüber (s298_P, s526_P, s820_P, s832_P, s1423_P), teilweise weit darüber (s1196_P, s1238_P, s1488_P) und einmal darunter (s1494_P). Hier konnte der Algorithmus in 1:45 Minuten eine neue beste Kompaktierung finden.

4.5 MaxMin Methode

Bei dieser Methode wurden drei Verfahren ausprobiert:

Als erstes startete jedes Individuum mit einem Chromosom und wurde daraufhin optimiert, dass alle Testmuster abgedeckt wurden. Dies wurde durch Hinzufügen von zufällig initialisierten Chromosomen erreicht. Waren alle Eingabetestmuster erfasst, galt es, die Chromosomenanzahl zu minimieren.
Dieser Ansatz wurde jedoch nach den ersten größeren Tests wieder verworfen, da die Rechenzeit bis zur Erfassung aller Testmuster sehr lang war. Außerdem waren im Vergleich zur bekannt besten Lösung sehr viele Chromosomen erzeugt worden, so dass absehbar war, dass die Rechenzeit unverhältnismäßig hoch werden würde.

Im zweiten Verfahren wurde versucht, die Individuen mit der Anzahl an Chromosomen zu initialisieren, welche die Abschätzungsfunktion (vgl. Kap. 3.2) ermittelte. Dabei wurden die Chromosomen so initialisiert, dass jedes Individuum alle Eingabetestmuster abdeckte. Bei der Mutation wurden die in Tab. 3.1 aufgeführten Methoden mit Ausnahme der Mutation »Chromosom hinzufügen« benutzt. Das hatte zur Folge, dass die Chromosomenanzahl lediglich reduziert werden konnte.
Hier zeigte sich allerdings, dass sich die Chromosomenanzahl wegen der mangelnden Diversität nur sehr geringfügig verringerte.

Schließlich wurde eine dritte Variante untersucht: Die Anfangschromosomenzahl wurde wie beim letzten Versuch festgelegt. Allerdings wurden die Chromosomen jetzt komplett

zufällig initialisiert, sodass zwar nicht alle Eingabetestmuster abgedeckt wurden, jedoch eine sehr hohe Diversität vorhanden war.
Für die Faktoren a und b der Fitnessfunktion f = T−a∗C+b∗D mit der Anzahl abgedeckter Testmuster T, der Chromosomenzahl C (gleich der Anzahl kompaktierter Testmuster) und der Diversitätsfunktion D wurden die Werte a = 0.01 und b = 0 experimentell ermittelt. Es zeigte sich, dass die Diversitätsfunktion nicht zur gewünschten Optimierung beitrug.

Die nun folgende Optimierung verlief typischerweise wie in Abb. 4.2 und 4.3 dargestellt. Durch die Wahl des niedrigen Faktorenwerts a wurde sichergestellt, dass das wichtigste Optimierungsziel die Abdeckung aller Testmuster war (impliziter Faktor 1.0). Entsprechend ergab sich anfangs eine rasche Zunahme der abgedeckten Testmuster (Abb. 4.2). Die Chromosomenanzahl (Abb. 4.3) wurde nur sekundär optimiert, weshalb die Werte hier zunächst springen. War das Maximum der abgedeckten Testmuster erreicht (in diesem Beispiel nach ca. 200 Generationen), konnte eine weitere Erhöhung der Fitness nur noch durch eine niedrigere Chromosomenanzahl erreicht werden. Von da an sank die Chromosomenzahl, die der Anzahl kompaktierter Testmuster entspricht.

Aus dieser Wirkungsweise ergab sich der Name »MaxMin«: Zunächst wurde auf das *Max*imum abgedeckter Testmuster optimiert, dann auf ein *Min*imum dafür benötigter Chromosomen.

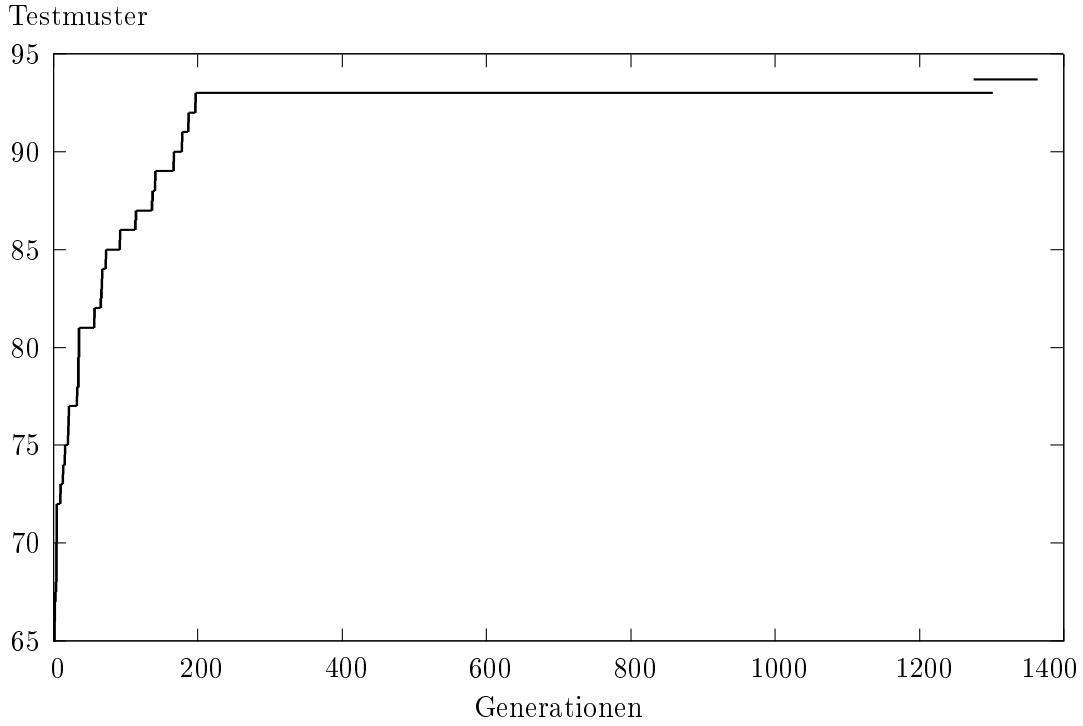


Abb. 4.2: Typischer Verlauf der MaxMin Berechnung am Beispiel der Testdaten 298_ S. Zunächst wird auf die Anzahl abgedeckter Testmuster optimiert. Das Maximum ist in diesem Fall nach ca. 200 Generationen erreicht.

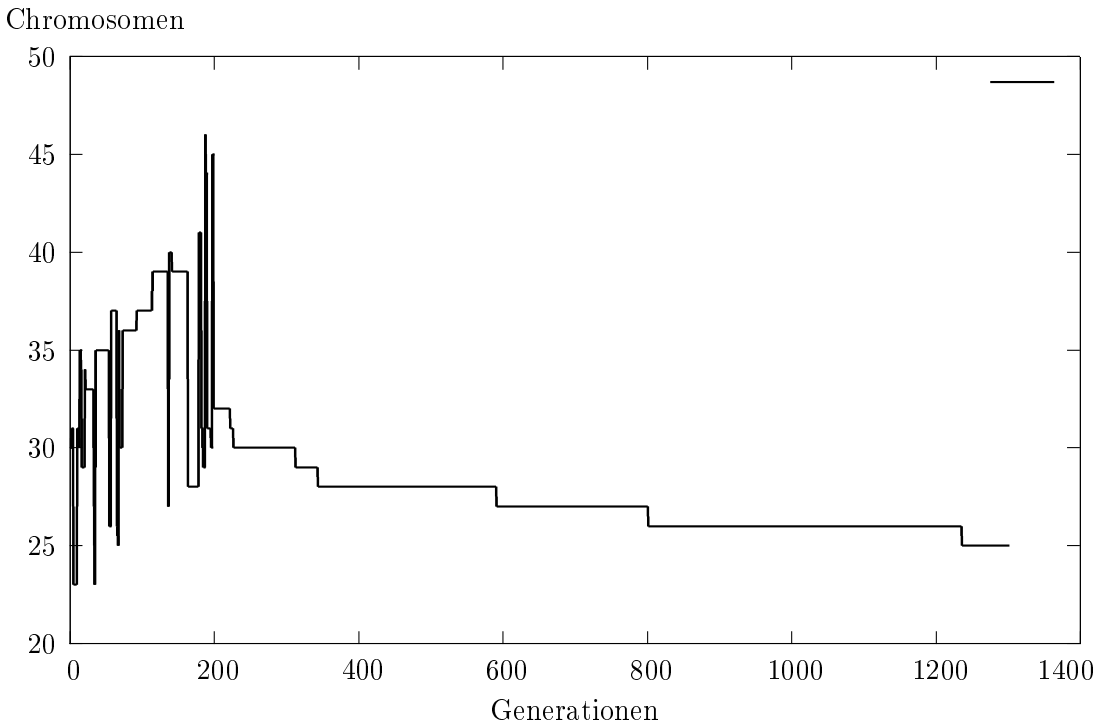


Abb. 4.3: Bis zum Erreichen des Maximums abgedeckter Testmuster (ca. 200 Generationen) werden die Chromosomenzahlen nur sekundär minimiert. Daher springen die Werte bis zu diesem Punkt. Danach findet eine weitere Fitnesssteigerung lediglich über die Minimierung der Chromosomenzahl statt.

Auch hier wurden vor den Berechnungen folgende Parameter experimentell ermittelt, wobei ein Kompromiss zwischen Geschwindigkeit und Güte eingegangen werden musste:

- Populationsgröße: 100 Individuen
- Abbruchkriterium: nach 300 Generationen ohne Verbesserung
- Selektion: fitnessproportional
- Rekombination: uniforme Rekombination
- Replacement: beste Individuen aus Eltern- und Nachkommenpopulation
- Mutation: Zufallsmutation (0.3), Bittausch (0.1), Invert (0.1), Chromosom hinzufügen (0.2), Chromosom entfernen (0.3) und Chromosomen tauschen (0.1)

Wie erwartet waren die Rechenzeiten sehr lang, da die Berechnung der Anzahl abgedeckter Testmuster während der Fitnessfunktion aufgrund der vielen Chromosomen sehr zeitintensiv war. Deshalb wurden einige Testdaten auch nicht bis zum Ende berechnet. Nach einer maximalen Dauer von fünf Stunden wurden die Berechnungen beendet.

Außerdem wurde ein erhöhter Speicherbedarf erwartet. In der Tat wurde mehr Speicher als bei den anderen Methoden benötigt, die Werte lagen jedoch nur um wenige Prozent höher, sodass hier keine Probleme entstanden.

Denn durch Einsatz mehrerer Chromosomen bestand die Vermutung, dass sich (oft) eine optimale Lösung finden läßt. Dies ist aber nicht der Fall. Bei allen Versuchen landete der Algorithmus in einem Suboptimum, das allerdings in der Regel nicht weit vom optimalen Wert entfernt war.

Die ausführlichen Ergebnisse sind in der folgenden Tabelle aufgeführt:

Schaltung	Anzahl	Relevant	Min	Kompakt	Prozent	Zeit
s27_S	21	15		8	38 %	5 Sek.
s298_S	163	94		26	16 %	13 Min.
s344_S	124	90		29	23 %	8 Min.
s349_S	127	90		24	19 %	15 Min.
s382_S	138	106		33	24 %	19 Min.
s386_S	158	83		76	48 %	25 Min.
s400_S	127	97		30	24 %	13 Min.
s444_S	134	97		31	23 %	15 Min.
s510_S	159	100		75	47 %	37 Min.
s526_S	241	168		69	29 %	1:15 Std.
s641_S	251	211		61	24 %	1:25 Std.
s713_S	230	197		50	22 %	1:25 Std.
s820_S	344	196		118	34 %	2:40 Std.
s832_S	340	197		117	34 %	2:40 Std.
s1423_S	490	405		109	22 %	3:30 Std.
s1488_S	393	241		138	35 %	2:35 Std.
s1494_S	393	240		136	35 %	4:50 Std.
s5378_S	1569	1216				$>$ 5 Std.
s9234_S	2105	1573				$>$ 5 Std.
s13207_S	3351	2616				$>$ 5 Std.
s15850_S	3858	3009				$>$ 5 Std.
s298_P	1800	313	38	62	3 %	50 Min.
s526_P	3537	531	74	102	3 %	3:15 Std.
s820_P	6955	956	186			$>$ 5 Std.
s832_P	7040	981	183			$>$ 5 Std.
s1196_P	15553	1967	204			$>$ 5 Std.
s1238_P	16288	2029	306			$>$ 5 Std.
s1423_P	9554	2474	205			$>$ 5 Std.
s1488_P	15745	1127	205			$>$ 5 Std.
s1494_P	15865	1192	327			$>$ 5 Std.
s5378_P	82517	23173	892			$>$ 5 Std.

Tab. 4.6: Ergebnisse der MaxMin Methode. Die erzielten Ergebnisse sind gut, die Rechenzeiten sehr lang. Aus Zeitgründen konnten einige Testdaten nicht bis zum Ende berechnet werden.

4.6 Permutationsmethode

Durch Versuche wurden zunächst geeignete Parameter ermittelt. Dabei ergaben sich folgende Ergebnisse:

- Populationsgröße: 100 Individuen
- Abbruchkriterium: nach 300 Generationen ohne Verbesserung
- Selektion: fitnessproportional (0.5), rangbasiert (0.5)
- Rekombination: Ein-Punkt- (0.8), Zwei-Punkt (Order Crossover) (0.2)
- Replacement: beste Individuen aus Eltern- und Nachkommenpopulation
- Mutation: Gentausch (0.1), Inversion (0.8), Verschieben (0.9), 2-Opt (0.001)

Mit diesen Einstellungen wurden die Versuche gestartet. Generell ergab sich ein Verlauf wie in Abb. 4.4 dargestellt. Am Anfang der Berechnungen konnten schnell bessere Permutationen gefunden werden. Entsprechend schnell verbesserten sich die Ergebnisse. Je mehr Generationen berechnet wurden, desto geringer fielen die Verbesserungen aus.

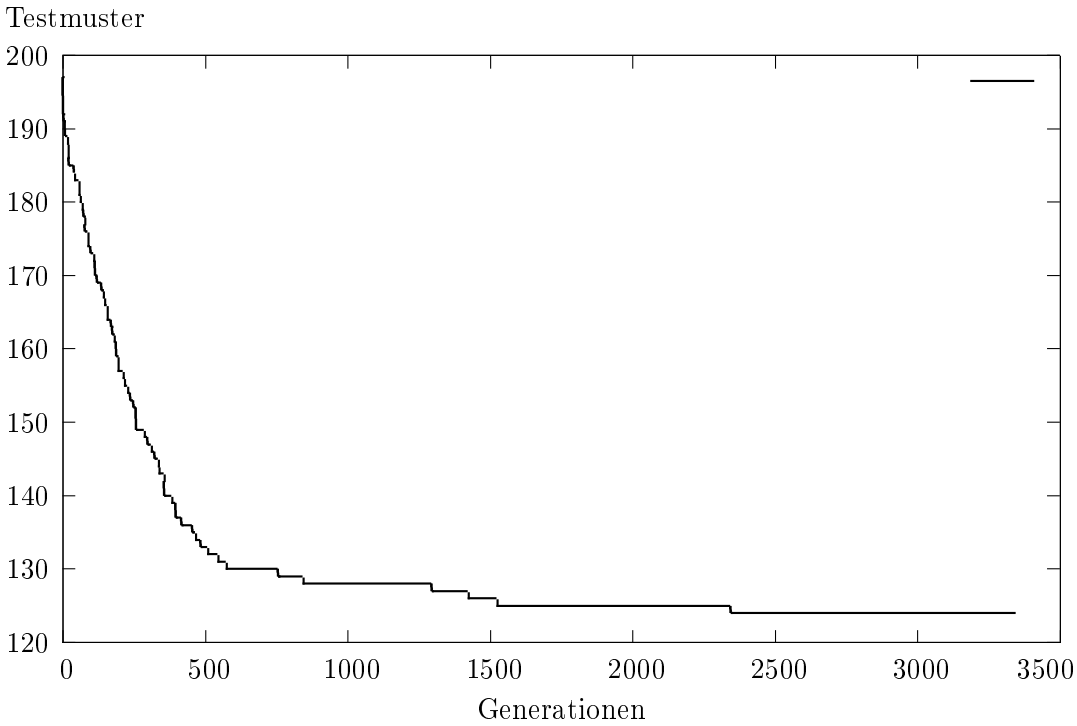


Abb. 4.4: Typischer Verlauf der Permutationsmethode am Beispiel der Testdaten 1488_S. Anfangs können leicht bessere Permutationen gefunden werden, weshalb die Anzahl kompaktierter Testmuster schnell sinkt. Zum Ende hin wird es immer schwieriger, eine bessere Reihenfolge zu finden.

Die Ergebnisse der Berechnungen sind in Tab. 4.7 aufgelistet.

Schaltung	Anzahl	Relevant	Min	Kompakt	Prozent	Zeit
s27_S	21	15		7	33 %	20 Sek.
s298_S	163	94		25	15 %	50 Sek.
s344_S	124	90		24	19 %	45 Sek.
s349_S	127	90		23	18 %	1 Min.
s382_S	138	106		29	21 %	1 Min.
s386_S	158	83		74	47 %	30 Sek.
s400_S	127	97		25	20 %	1 Min.
s444_S	134	97		27	20 %	1 Min.
s510_S	159	100		70	44 %	45 Sek.
s526_S	241	168		67	28 %	2 Min.
s641_S	251	211		42	17 %	9 Min.
s713_S	230	197		36	16 %	11 Min.
s820_S	344	196		109	32 %	4:15 Min.
s832_S	340	197		108	32 %	4:30 Min.
s1423_S	490	405		69	14 %	40 Min.
s1488_S	393	241		125	32 %	5 Min.
s1494_S	393	240		125	32 %	4 Min.
s5378_S	1569	1216		232	15 %	1 Std.
s9234_S	2105	1573		295	14 %	4:30 Std.
s13207_S	3351	2616				$>$ 5 Std.
s298_P	1800	313	38	54	3 %	35 Min.
s526_P	3537	531	74	115	3 %	1:10 Std.
s820_P	6955	956	186	268	4 %	1 Std.
s832_P	7040	981	183	286	4 %	1 Std.
s1196_P	15553	1967	204			$>$ 5 Std.
s1238_P	16288	2029	306			$>$ 5 Std.
s1423_P	9554	2474	205			$>$ 5 Std.
s1488_P	15745	1127	205	402	3 %	1:20 Std.
s1494_P	15865	1192	327	395	2 %	1:20 Std.

Tab. 4.7: Ergebnisse der Permutationsmethode. Viele Ergebnisse der Philips Testdaten werden mit dieser Methode sehr gut berechnet. Der Zeitaufwand bei kleinen Testinstanzen ist moderat. Bei größeren Testdaten steigt die Rechenzeit stark an. Deshalb wurden einige Berechnungen nach 5 Stunden abgebrochen. Die PODEM Ergebnisse kommen nicht an die minimalen Kompaktierungen heran.

Erwartet war, dass diese Methode (nahezu) optimale Ergebnisse bei moderater Rechenzeit erzielt. Es zeigte sich, dass die Ergbenisse bei kleinen Testinstanzen wie erwartet sehr gut sind. Auch die Rechenzeiten sind hier kurz. Bei größeren Datenmengen stieg der Rechenaufwand jedoch deutlich an. Deshalb wurden einige Berechnungen nach einer Dauer von 5 Stunden abgebrochen. Die Ergebnisse der PODEM-Testdaten liegen im Bereich der minimalen Kompaktierungen, kommen jedoch nicht an sie heran. Dies ist auf die Größe des Suchraums zurückzuführen. Sie nimmt mit der Fakultät der Testmusteranzahl zu.

4.7 Vergleich der Methoden

In diesem Abschnitt werden die Ergebnisse der Lösungsmethoden verglichen. Zur besseren Übersicht sind sie in Tab. 4.8 zusammengefasst. Fett hervorgehoben sind die besten Kompaktierungswerte.

Schaltung	#	Min	Relevant	Abschätzung	Greedy	MaxMin	Permutation
s27_S	21		15	**7**	**7**	8	**7**
s298_S	163		94	29	26	26	**25**
s344_S	124		90	25	25	29	**24**
s349_S	127		90	24	25	24	**23**
s382_S	138		106	30	30	33	**29**
s386_S	158		83	76	76	76	**74**
s400_S	127		97	27	27	30	**25**
s444_S	134		97	28	31	31	**27**
s510_S	159		100	74	72	75	**70**
s526_S	241		168	**67**	68	69	**67**
s641_S	251		211	44	45	61	**42**
s713_S	230		197	**36**	40	50	**36**
s820_S	344		196	117	113	118	**109**
s832_S	340		197	118	111	117	**108**
s1423_S	490		405	**68**	72	109	69
s1488_S	393		241	137	129	138	**125**
s1494_S	393		240	139	128	136	**125**
s5378_S	1569		1216	**187**	241	—	232
s9234_S	2105		1573	**228**	284	—	295
s13207_S	3351		2616	**280**	308	—	—
s15850_S	3857		3009	**200**	244	—	—
s298_P	1800	38	313	62	**54**	62	**54**
s526_P	3537	74	531	96	**90**	102	115
s820_P	6955	186	956	238	**232**	—	268
s832_P	7040	183	981	264	**247**	—	286
s1196_P	15553	204	1967	551	**520**	—	—
s1238_P	16288	306	2029	558	**531**	—	—
s1423_P	9554	205	2474	**243**	276	—	—
s1488_P	15745	205	1127	390	**338**	—	402
s1494_P	15865	327	1192	366	**325**	—	395
s5378_P	82517	721	23173	**892**	1014	—	—

Tab. 4.8: Ergebnisse der drei Lösungsmethoden im Überblick. Die besten Ergebnisse sind fett hervorgehoben. Die Greedy Methode ist die schnellste. Sie liefert gute bis sehr gute Ergebnisse. Die maximalen Kompaktierungszahlen der MaxMin Methode stehen hinter den Erwartungen zurück. Bei kleineren Testinstanzen findet die Permutationsmethode die besten Werte der drei Lösungsmethoden. Die Zeiten sind geringfügig schlechter als die der Greedy Methode.

Durch das Entfernen irrelevanter Testmuster konnte die Testmusteranzahl besonders bei den PODEM-Daten deutlich verringert werden. Dadurch reduziert sich die Rechenzeit aller Methoden.

Die Abschätzungsfunktion ermittelt Werte, die im Bereich der bestmöglichen Kompaktierung liegen. Bei den Schaltungen s27_S, s526_S, s713_S und s1423_S konnten keine besseren Werte berechnet werden.

Die Greedy Methode liefert sehr schnell gute Ergebnisse. Hinter den besten Kompaktierungen der Philips-Testdaten lagen die Werte oft nur geringfügig. Aufgrund der geringen Rechenzeiten konnten mit diesem Verfahren alle Testdaten berechnet werden. Hier zeigte sich, dass die ermittelten Werte im Bereich der besten Werte lagen. Bei der Schaltung s1494_P konnte eine Kompaktierung gefunden werden, die besser als die bisher beste ist.

Auch die berechneten Werte der MaxMin Methode lagen im Bereich der besten bekannten Werte. Allerdings waren die Rechenzeiten sehr lang, sodass nicht alle Testdaten berechnet werden konnten.

Bei kleinen Testinstanzen lieferte die Permutationsmethode sehr gute Ergbenisse in kurzer Zeit. Die Berechnungszeiten wurden bei einer größer werdenden Anzahl an Testmustern immer länger. Erreicht wurden auch hier Kompaktierungen im Bereich der besten bekannten Werte, allerdings konnten diese nicht erreicht bzw. verbessert werden.

5 Zusammenfassung

In dieser Arbeit wurden drei Methoden vorgestellt, um Testmuster mit Genetischen Algorithmen zu kompaktieren.

Die erste Methode besteht darin, ein Muster dahingehend zu optimieren, dass es möglichst viele Testmuster zusammenfasst. Dabei sind die Chromosomen bitkodiert, der Genotyp entspricht dem Phänotyp, welcher direkt einem kompaktierten Muster entspricht. Nach dem Optimierungsprozess wird das kompaktierte Muster übernommen und die erfassten Testmuster von der weiteren Berechnung ausgenommen. Dieser Prozess wiederholt sich, bis alle Testmuster abgedeckt sind. Nach dem Wirkungsprinzip wurde dieses Verfahren »Greedy Methode« genannt.
Sie zeichnet sich durch eine sehr schnelle Berechnung aus. Besonders bei großen Testinstanzen (mit kleiner Testmusterlänge) berechnet diese Methode gute Ergebnisse bei gleichzeitig schnellster Ausführungszeit. Sie fand für eine Testinstanz (s1494_P) eine neue beste Kompaktierung.

Die Verarbeitung der zweiten Methode verläuft in zwei Phasen: Zuerst wird mittels Individuen mit dynamischer Chromosomenzahl eine maximale Testmusterabdeckung berechnet. Ist diese erreicht, wird die Chromosomenanzahl minimiert. Daraus leitet sich der Name ab: »MaxMin Methode«. Die Chromosomen sind ebenso wie bei der Greedy Methode kodiert. Leider blieben die Ergebnisse hinter den Erwartungen zurück. Es konnte kein Optimum gefunden werden, die Werte waren jedoch nur unwesentlich schlechter als die der Greedy Methode. Diese Methode ist jedoch sehr rechenintensiv. Der Speicherbedarf liegt gering über dem der anderen Methoden.

Anders kodiert sind die Chromosomen bei der dritten, der Permutationsmethode. Jedes Gen entspricht einem Index der Testmustermenge. Die Testmuster werden nach der auf den Chromosomen vorgegebenen Reihenfolge kompaktiert. Sind zwei aufeinanderfolgende Muster kompatibel, werden sie kompaktiert und die Kompaktierung wird mit dem entstandenen Muster weitergeführt. Ist das aktuelle Muster mit dem nächsten Testmuster inkompatibel, wird das aktuelle zur Ausgabemenge hinzugefügt. Da jedes Eingabetestmuster auf dem Chromosom genau einmal vorhanden ist, wird sichergestellt, dass alle Eingabetestmuster abgedeckt werden. Das Optimierungsziel ist eine Testmusterreihenfolge, die eine minimale Zahl kompaktierter Testmuster ergibt. Dadurch wird die Testmusterkompaktierung auf das sehr gut untersuchte Handelsreisendenproblem zurückgeführt, mit dem Unterschied, dass es sich beim Testmusterkompaktierungsproblem nicht um eine zyklische Permutation han-

delt.
Die experimentell ermittelten Ergebnisse zeigen bei kleinen Testinstanzen eine sehr gute Qualität bei geringen Rechenzeiten. Bei steigender Testmusteranzahl dauerten die Berechnungszeiten länger und die Ergebnisse blieben hinter denen der Greedy Methode zurück.

Des Weiteren wurde eine Methode vorgestellt, um die irrelevanten Testmuster zu entfernen. Allein hierdurch ergab sich bei den PODEM-Testdaten eine wesentliche Kompaktierung.

Die eingesetzte Methode zur Abschätzung der maximalen Testmuster arbeitete ebenfalls sehr gut. Bei den Philips-Testdaten waren die Abschätzungen oft nur geringfügig schlechter als die Werte, welche die jeweils beste Methode berechnete. Von den bekannt besten Werten der PODEM-Testdaten waren die Werte prozentual zur Gesamttestmusterzahl geringfügig schlechter.

Ausblick

Die Permutationsmethode bietet ein hohes Potential, was sich daran zeigt, dass sie bei den Philips-Testdaten beste Ergebnisse lieferte. Durch erweiterte Rekombinations- und Mutationsoperatoren könnte diese Methode nach Meinung des Authors auch gute bis sehr gute Ergebnisse in annehmbarer Zeit für größere Testmusterinstanzen liefern.

Anhang

A D-Algorithmus

Diese Idee zur Pfadverfolgung in Form des »D-Algorithmus« wurde 1966 von Roth erstmals publiziert [Rot66] und wird im Folgenden vorgestellt. Weiterführende Informationen zum D-Algorithmus sind in [Rot66] und [JG03, S.197 ff] zu finden.

Dazu wird das »Stuck-At-Fehlermodell« benutzt. Die grundlegende Annahme hierbei ist, dass ein Schaltungsknoten auf logisch-0 oder logisch-1 »festgeklebt« wird. Dazu kommt es z. B. bei einer offenen Verbindung bzw. einem Kurzschluss. Weiterhin wird davon ausgegangen, dass immer nur ein Fehler vorliegt.

Der D-Algorithmus berechnet für alle relevanten Fehler ein Testmuster. Dabei wird aus der Schaltung ein Gatter bestimmt, das als fehlerhaft angenommen wird. Dann wird die Art des Fehlers (»Stuck-At-0« oder »Stuck-At-1«) am Ausgang festgelegt. Anschließend wird ermittelt, welche Bit-Information am Eingang des betreffenden Gatters den Fehler an dessen Ausgang erkennbar macht.

Danach wird ermittelt, unter welchen Belegungen der Fehler zu den Ausgängen durchgeschaltet werden kann und so beobachtbar wird. Die betrachteten Knoten der Schaltung werden entsprechend belegt.

Zum Schluss wird eine entsprechende Belegung bis zu den Eingängen so zurückpropagiert, dass beim Anlegen der berechneten Eingangsbelegung der betrachtete Knoten den gewünschten Wert annimmt. Der Knoten wird durch das entsprechende Muster einstellbar. Dieser Schritt kann sehr komplex ausfallen. So kann es passieren, dass ein Knoten über unterschiedliche Pfade verschiedene Werte annimmt. Dann sind die Schritte so weit zurückzunehmen, bis sich wieder konsistente Werte ergeben. Dieses Verfahren wird »Backtracking« genannt.

Symbol	fehlerfrei	fehlerhaft
0	0	0
1	1	1
X	X	X
D	1	0
$\overline{\text{D}}$	0	1

Tab. A.1: Symbole der 5-wertigen Logik des D-Algorithmus für den fehlerfreien und fehlerhaften Zustand.

Um diesen Algorithmus mathematisch exakt beschreiben zu können, führte Roth das fünfwertige (D)iscrepancy-Kalkül ein (siehe Tab. A.1). Daher auch der abgeleitete Name D-Algorithmus. Es existiert dabei für jedes Symbol ein Wert für den fehlerfreien und den fehlerhaften Schaltkreis. Wobei die Belegungen für 0, 1 und den unbestimmten Wert (X) jeweils für beide Zustände gleich sind. Die Symbole D und $\overline{\text{D}}$ repräsentieren im fehlerfreien und -haften Fall unterschiedliche Logikwerte.

Das Verhalten der Gatter wird durch so genannte »D-Kuben« beschrieben, die sich in Fortpflanzungs- und Fehlerkuben gliedern. Sie beschreiben das Verhalten des Elements bei der Weiterleitung des Fehlers respektive Auftreten des Fehlers im Gatter.

NAND	0	1	X	D	$\overline{D}$
0	1	1	1	1	1
1	1	0	X	$\overline{D}$	D
X	1	X	X	X	X
D	1	$\overline{D}$	X	$\overline{D}$	1
$\overline{D}$	1	D	X	1	D

Tab. A.2: Fortpflanzungskubus für ein NAND Gatter

Zur Veranschaulichung des eben beschriebenen ist in Tab. A.2 der Fortpflanzungskubus beispielhaft für ein NAND-Gatter aufgeführt. Befindet sich z. B. an den Eingängen des NAND-Gatters eine 1 und ein D, so folgt logisch daraus, dass sich am Ausgang ein $\overline{D}$ befinden muss. Denn im fehlerfreien Fall führt eine Eingangsbelegung von zwei Einsen zu einem Ausgangswert von 0. Im fehlerhaften Fall lautet die Eingangsbelegung 1 und 0. Am Ausgang muss sich dann eine 1 ergeben. Laut Tab. A.1 ist das Symbol für 0/1 ein $\overline{D}$. Mit der gleichen logischen Überlegung gelangt man auch zu allen anderen Werten anderer Gatter.

Das oben beschriebene Verfahren des D-Algorithmus soll hier zur Verdeutlichung an einem Beispiel erläutert werden (vgl. Abb. A.1).

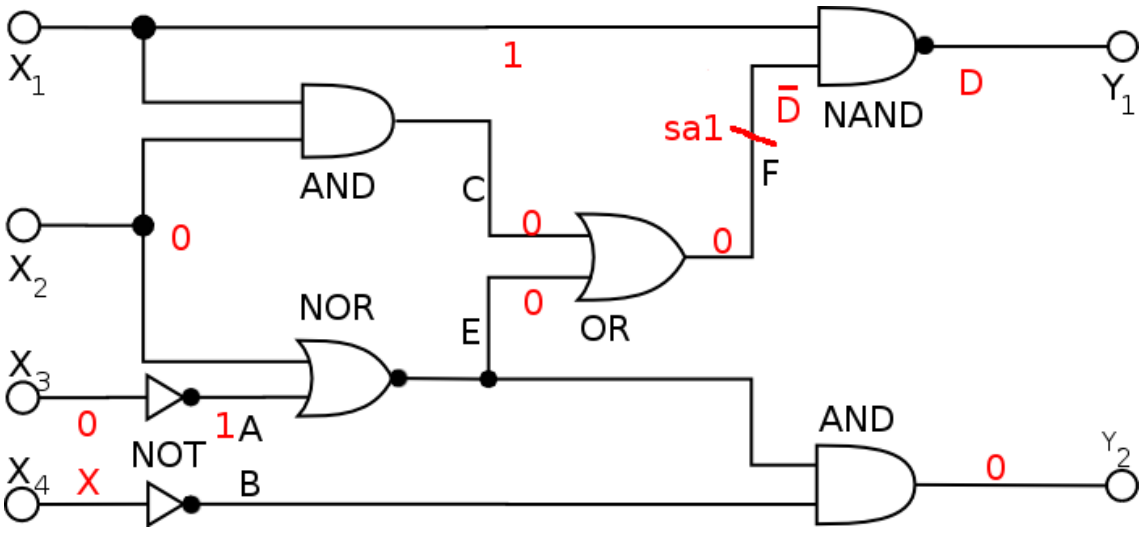


Abb. A.1: Beispielschaltung. Hier wird am Knoten F ein Stuck-At-1 Fehler festgelegt. Dieser wird an die Ein- und Ausgänge propagiert. Rot eingezeichnet sind die entsprechenden Belegungen der Knoten.

Beispielhaft wird am Knoten F ein »Stuck-At-1-Fehler« berechnet. Um einen möglichen Fehler aufzudecken muss der Ausgang des OR-Gatters also auf 0 eingestellt werden. Zunächst wird der Eingang des folgenden NAND-Gatters mit $\overline{D} = 0/1$ markiert, denn im fehlerfreien Zustand soll eine logische 0 anliegen, im fehlerhaften Fall eine logische 1. Nach den Propagierungsregeln werden alle folgenden und vorhergehenden Gatter durchlaufen und die Ein- und Ausgänge entsprechend berechnet. Es ergibt sich die mit rot markierte Belegung. Daher kommt es zu einer Eingangsbelegung von $(x_1 = 1, x_2 = 0, x_3 = 0, x_4 = X)$ und einer Ausgangsbelegung im fehlerfreien Zustand von $(y_1 = 0, y_2 = 0)$. Ein evtl. vor-

handener »Stuck-At-1-Fehler« an der Stelle F würde das Ausgangsmuster ($y_1 = 1, y_2 = 0$) verursachen, womit der Fehler detektierbar wäre.

Abbildungsverzeichnis

1.1 Entwicklung der Chipkosten 1

2.1 Schematischer Testprozess 6
2.2 Strukturmodell der Desoxyribonukleinsäure 11
2.3 Schematische Darstellung des Zyklus eines Genetischen Algorithmus 14
2.4 Rekombination 15
2.5 Mutationsbeispiel 15

3.1 Greedy Methode 18
3.2 MaxMin Methode 19
3.3 Kompaktierung nach der Permutationsmethode I 21
3.4 Kompaktierung nach der Permutationsmethode II 22
3.5 Fitnessproportionale Selektion 26
3.6 Rangbasierte Selektion 26
3.7 Ein-Punkt Rekombination 28
3.8 Zwei-Punkt Rekombination 29
3.9 Uniforme Rekombination 29
3.10 Bit-Flip Mutation 30
3.11 Gen-Austausch Mutation 30
3.12 Inverse Mutation 31
3.13 2-Opt Mutation 32
3.14 Typische Fitnesswertentwicklung 33
3.15 Programm Testmusterkompaktierung 34

4.1 Diagramm Greedy Methode 41
4.2 Diagramm MaxMin Methode - Testmuster 44
4.3 Diagramm MaxMin Methode - Chromosomen 45
4.4 Diagramm Permutationsmethode 47

A.1 Beispielschaltung zum D-Algorithmus 56

Tabellenverzeichnis

2.1 Kombinationsmöglichkeiten bei der Kompaktierung 8

3.1 Kombion der Mutations- und Lösungsmethoden 24
3.2 Kodierung der Symbole 35

4.1 Iscas'89 Testmuster (Philips Semicondurctors) 38
4.2 Iscas'89 Testmuster (PODEM) 38
4.3 Parametereinstellung der Rekombinationsmethoden 39
4.4 Parametereinstellung II 40
4.5 Ergebnisse Greedy Methode 42
4.6 Ergebnisse der MaxMin Methode 46
4.7 Ergebnisse der Permutationsmethode 48
4.8 Gesamtergebnisse 49

A.1 Symbole des D-Algorithmus 55
A.2 Fortpflanzungskubus für ein NAND Gatter 56

Literaturverzeichnis

[Ays05] Ayse Kivilcim Coskun and Subhradyuti Sarkar: Test Pattern Compaction. 2005. – Forschungsbericht

[Bus98] Buselmeier, Werner: *Biologie für Mediziner.* 8. Auflage. Springer, 1998

[Dal91] Dalrymple, G. B.: *The Age of the Earth.* California : Stanford University Press, 1991

[FOW66] Fogel, L. J. ; Owens, A. J. ; Walsh, M. J.: *Artificial Intelligence through Simulated Evolution.* New York, USA : John Wiley, 1966

[FS83] Fujiwara, Hideo ; Shimono, Takeshi: On the Acceleration of Test Generation Algorithms. In: *IEEE Trans. Computers* 32 (1983), Nr. 12, S. 1137–1144

[Goe81] Goel, P.: An Implicit Enumeration Algorithm to Generate Tests for Combinational Logic Circuits. In: *IEEE Transactions on Computers* (1981), März, Nr. 3, S. 215–222

[Hel00] Helsgaun, Keld: An Effective Implementation of the Lin-Kernighan Traveling Salesman Heuristic. In: *European Journal of Operational Research* 1 (2000), Nr. 126, S. 106–130

[Hol73] Holland, John: Genetic algorithms and the optimal allocation of trials. In: *SIAM Journal on Computing* Bd. 2, 1973, S. 88–105

[Hol92] Holland, John: *Adaptation in natural and artificial systems.* Cambridge : MIT Press, 1992

[JG03] Jha, Niraj ; Gupta, Sandeep: *Testing of Digital Systems.* Cambridge University Press, 2003

[Koz92] Koza, John: *Genetic Programming.* Cambridge : MIT Press, 1992

[LK73] Lin, Shen ; Kernighan, Brian W.: An Effective Heuristic Algorithm for the Travelling-Salesman Problem. In: *Operations Research* 21 (1973), S. 498–516

[Mey03] Meyer, Volker Hans-Walther: *Test produktionsbedingter Laufzeitfehler in hochintegrierten, digitalen Schaltungen.* Bremen, Universität Bremen, Diss., 4 November 2003

[MKR02] Miyase, Kohei ; Kajihara, Seiji ; Reddy, Sudhakar M.: A Method of Static Test Compaction Based on Don't Care Identification. In: *DELTA '02: Proceedings of the The First IEEE International Workshop on Electronic Design, Test and Applications (DELTA '02)*. Washington, DC, USA : IEEE Computer Society, 2002. – ISBN 0–7695–1453–7, S. 392

[MMR96] Marculescu, Radu ; Marculescu, Diana ; Redram, Massoud: Vector Compaction Using Dynamic Markov Models / Department of Electrical Engineering - Systems. Los Angeles, California, Februar 1996. – Forschungsbericht

[PR96] Pomeranz, Irith ; Reddy, Sudhakar M.: On static compaction of test sequences for synchronous sequential circuits. In: *DAC '96: Proceedings of the 33rd annual conference on Design automation*. New York, NY, USA : ACM Press, 1996. – ISBN 0–89791–779–0, S. 215–220

[PR97] Pomeranz, Irith ; Reddy, Sudhakar M.: On the Compaction of Test Sets Produced by Genetic Optimization. In: *ATS '97: Proceedings of the 6th Asian Test Symposium*. Washington, DC, USA : IEEE Computer Society, 1997. – ISBN 0–8186–8209–4, S. 4

[Rec73] Rechenberg, Ingo: *Evolutionsstrategie. Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Stuttgart : Friedrich Frommann Verlag, 1973. – ISBN 3–7728–0373–3

[Rot66] Roth, J. P.: Diagnosis of automata failures: A calculus and a method. In: *IBM Journal of Research and Development* (1966), Nr. 10, S. 278 – 292

[RPR92] Reddy, Lakshmi N. ; Pomeranz, Irith ; Reddy, Sudhakar M.: ROTCO: A Reverse Order Test Compaction Technique. In: *In Proceeding of EURO-ASIC*, 1992, S. 189–194

[Sch68] Schwefel, Hans P.: Experimentelle Optimierung einer Zweiphasendüse / Bericht 35 des AEG Forschungsinstituts Berlin zum Projekt Staustahlrohr. 1968. – Forschungsbericht

[Sch75] Schwefel, Hans P.: *Evolutionsstrategie und numerische Optimierung*, Technische Universität Berlin, Diss., Mai 1975

[SHF94] Schöneburg, Eberhard ; Heinzmann, Frank ; Feddersen, Sven: *Genetische Algorithmen und Evolutionsstrategien*. 1. Auflage. Addison-Wesley, 1994. – ISBN 3–89319–493–2

[SR05] Signorini, Alessio ; Remersaro, Santiago: Test-cubes compaction for digital circuits. (2005), 3 Mai

[STS88] Schulz, M. H. ; Trischler, E. ; Sarfert, T. M.: SOKRATES: A highly efficient automatic test pattern generation system. In: *IEEE Transaction on Computer-Aided Design* (1988), S. 126–137

[Tho96] Thompson, Kenneth M.: Intel and the Myths of Test. In: *IEEE Des. Test* 13 (1996), Nr. 1, S. 79–81. – ISSN 0740–7475

[Wei02] Weicker, Karsten: *Evolutionäre Algorithmen*. 1. Auflage. Teubner, März 2002. – ISBN 3–519–00362–7

Stichwortverzeichnis

A

Abbruchkriterien 33
Allel 12, 13
Ausbeutung 25
Ausblick 52

B

Backtracking 55
Bewertung 13
Biologische Evolutionstheorie 10

C

Chromosom 13
Crowding 26

D

D-Algorithmus 7, 55
Darwin, Charles 10
Desoxyribonukleinsäure 11
Dominanzproblem 26

E

Elitismus 27
Ergebnisse 37
  Greedy Methode 40
  MaxMin Methode 43
  Permutations Methode 47
Erkundung 25
Ersetzung 27
Evolutionäre Algorithmen 10
Evolutionäres Programmieren 12
Evolutionsstrategien 12
Existierende Lösungsansätze 9
Exploitation 25
Exploration 25

F

FAN 7
Fitnessfunktion 13, 32

G

Genetische Algorithmen 13
Genetische Operatoren 24
Genetsiche Programmierung 12
Genotyp 12
Geschwindigkeitsoptimierung 35
Greedy Methode 18

H

Hilfsfunktionen 23

I

Implementierung 34
induzierte Fitnessfunktion 12
Initialisierung 13
Integrierte Schaltung 5

K

Kodierung 13
Kodierungsproblem 13
Kompatibel 8

L

Lösungsansätze 17

M

MaxMin Methode 19
Mooresches Gesetz 1
Mutation 11, 15, 30
  2-Opt 31
  Bit-Flip 30
  Chromosom entfernen 31
  Chromosom hinzufügen 31
  Chromosomen tauschen 31
  Genaustausch 30
  Gene verschieben 31
  Inverse 31

P

Permutationsmethode 20

Phänotyp 12
PODEM 7
Population 13

R

Rekombination 11, 15, 28
  Ein-Punkt 28
  One point crossover 28
  Two point crossover 29
  uniforme 29
  Zwei-Punkt 29
Replacement 27
Rouletteradselektion 25

S

Selektion 11, 13, 25
  fitnessproportional 25
  rangbasiert 26
  Turnierselektion 27
Selektionsdruck 25
SOKRATES 7
Speichernutzung 35
Stuck-At-Fehlermodell 55

T

Testdaten 37
Testmuster 5
Testmustergenerator 6
Testmustergeneratoren 6
Testmusterkompaktierung 7
Testprozess 5
Testumgebung 39
Testverfahren 39

Y

Yield 1

Z

Zielsetzung 2
Zusammenfassung 51