

# Partitionierung von Schaltungen unter Einsatz mehrkriterieller evolutionärer Algorithmen

Andreas Rummler

Technische Universität Ilmenau

Institut für Schaltungstechnik und Elektroniktechnologie

Fachgebiet Elektronische Schaltungen und Systeme

email : arummler@acm.org

www : www.inf-technik.tu-ilmenau.de/~rummler

## Zusammenfassung

Dieser Artikel gibt einen Überblick über die Anwendung mehrkriterieller evolutionärer Algorithmen (Multi-objective Evolutionary Algorithm, MOEA) bei der Partitionierung von Schaltungen. Die ersten beiden Abschnitte enthalten eine kurze Einführung in die Theorie der MOEA sowie die Formulierung des Partitionierungsproblems unter mehrkriteriellen Gesichtspunkten. In den Abschnitten 3 und 4 wird ein MOEA sowie dessen Implementation und Anwendung auf das Partitionierungsproblem konkret vorgestellt. Der Artikel schließt mit einem Abschnitt über den Einsatz des Algorithmus bei derzeit laufenden Forschungsarbeiten.

## 1 Grundlagen mehrkriterieller evolutionärer Algorithmen

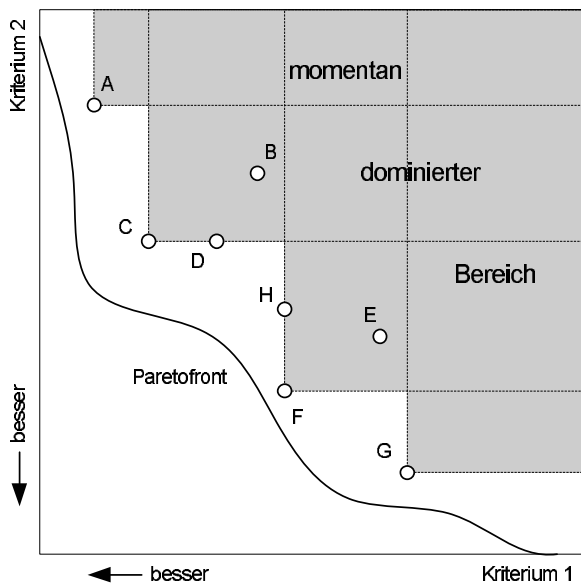
Evolutionäre Algorithmen gehören zur Kategorie stochastischer Suchverfahren. Die algorithmischen Grundlagen wurden bereits in den siebziger Jahren entwickelt ([11], [12] und [6]), eine breitere Anwendung fanden sie allerdings erst in den letzten zehn Jahren. Wie Mietinen in [9] verdeutlicht, werden evolutionäre Algorithmen zur Optimierung von Problemstellungen heute in vielen Gebieten der Ingenieurwissenschaften angewendet. Grundlage dieser Algorithmen ist Darwin's Evolutionstheorie. Es wird mit einer Anzahl von potentiellen Problemlösungen (Individuen) gearbeitet, die einen simulierten Evolutionszyklus durchlaufen. In diesem Zyklus werden Operationen angewendet, die eine Entsprechung biologischer Mechanismen darstellen. Im einzelnen sind das Selektion (Auswahl von "guten" Lösungen), Rekombination (Erzeugen neuer Lösungen) und Mutation (zufällige Veränderung von Lösungen). Durch das Zusammenwirken dieser genetischen Operationen und dem Darwin'schen Prinzip "der Stärkere überlebt" wird während dem Durchlaufen einer Anzahl von Generationen eine Optimierung der Anfangslösungen erzielt. Für eine detailliertere Einführung in die Theorie evolutionärer Algorithmen sei auf [10] verwiesen.

Der Vorgang der Optimierung bedeutet häufig Minimierung (bzw. als Äquivalent Maximierung) unter einem bestimmten Gesichtspunkt. Aus der Menge aller potentiellen Lösungen wird diejenige gesucht, die besser als alle anderen ist. Der Begriff "besser" läßt sich allerdings nur verwenden, wenn ein einzelner Parameter optimiert wird (einkriterielle Optimierung). Bei realen Problemstellungen

müssen aber häufig mehrere Kriterien betrachtet werden, die sich unter Umständen widersprechen (z.B. die Qualität und die Herstellungskosten eines Produkts). Bei einer mehrkriteriellen Optimierung gilt es daher einen guten Kompromiß zu finden.

Nachfolgend sollen einige Begriffe aus dem Bereich der mehrkriteriellen Optimierung erläutert werden, die zum weiteren Verständnis dieses Artikels notwendig sind. Alle nachfolgenden Erläuterungen beziehen sich auf das schematische Beispiel aus Abbildung 1. In diesem Beispiel sind Lösungen für ein Problem gesucht bei dem beide Optimierungskriterien minimiert werden sollen. Im Suchraum existieren Lösungen, bei denen es nicht möglich ist, ein Kriterium weiter zu verbessern, ohne das andere zu verschlechtern. Diese Lösungen liegen auf einer Grenze im Suchraum, die nicht überschritten werden kann, der sogenannten *Paretofront* oder *Paretogrenze*. Das Ziel der Optimierungsprozesses besteht darin, Lösungen zu finden, die möglichst nah an dieser Paretofront liegen.

Um unterschiedliche Lösungen aus dem Suchraum bewerten zu können, müssen diese miteinander verglichen werden. Im abgebildeten Beispiel ist Lösung  $C$  in beiden Kriterien besser als Lösung  $B$ . Man spricht in diesem Fall auch von einer Dominanz von  $C$  über  $B$  ( $C \succ B$ ) oder auch davon, daß  $B$  von  $C$  dominiert wird. Im Falle der Lösungen  $C$  und  $D$  bzw.  $F$  und  $H$  ist lediglich ein Kriterium besser, das andere jedoch gleich. In diesem Falle wird von schwacher Dominanz gesprochen ( $C \succeq D$  und  $F \succeq H$ ). Beim Vergleich der Lösungen  $A$  und  $G$  fällt auf, daß bei beiden Lösungen ein Kriterium jeweils besser ist. Beide Lösungen sind zueinander indifferent ( $A \sim G$ ). Der momentan dominier-



**Abbildung 1:** Paretofront mit dominierten und nichtdominierten Lösungen

te Ausschnitt aus dem Suchraum wird von den (nichtdominierten) Lösungen *A*, *C*, *F*, und *G* aufgespannt und ist in der Abbildung grau gekennzeichnet.

Als Resultat einer mehrkriteriellen Optimierung läßt sich keine einzelne Lösung finden, schlicht aus dem Grund weil es keine einzelne Lösung gibt. Das Ergebnis ist ein Vektor verschiedener Lösungen, die jede für sich ein sogenanntes Pareto-Optimum darstellt. Aus diesem Vektor wird durch einen sogenannten *Decision Maker*<sup>1</sup> (u.U. mit Berücksichtigung verschiedener Nebenbedingungen) eine Lösung als Endlösung ausgewählt.

Auch evolutionäre Algorithmen arbeiten mit einer Anzahl möglicher Lösungen gleichzeitig, was sie für den Einsatz bei mehrkriterieller Optimierung anbietet. Trotzdem werden sie erst seit kurzem auch auf derartige Problemstellungen angewendet. Für eine Übersicht über den aktuellen Stand der Entwicklung von Algorithmen auf diesem Gebiet sei auf [4] verwiesen.

Wo liegt die Motivation für den Einsatz mehrkriterieller Optimierungsverfahren? Reale Problemstellungen, bei denen mehr als ein Optimierungskriterium betrachtet werden muß, sind keine Ausnahme sondern eher ein Regelfall. Traditionelle Verfahren führen eine mehrkriterielle Aufgabe meist auf eine einkriterielle zurück. Weit verbreitet und (scheinbar) einfach anzuwenden ist z.B. das Verfahren der gewichteten Summen, bei dem alle Kriterien mit einem Wichtungsfaktor versehen und summiert werden. Genauer betrachtet erweist es sich als schwierig die Wichtungsfaktoren festzulegen, da die einzelnen Summanden in der Regel verschiedene Wertebereiche haben und die Wichtungen der einzelnen Kriterien sich bei Probleminstanzen unterschiedlicher Größe gegeneinander verschieben. Darüberhinaus läßt sich zeigen, daß

<sup>1</sup>Es ist eine durchaus übliche Praxis, den Anwender als Decision Maker einzusetzen.

sich bei einer Paretofront mit einer bestimmten Form nicht alle Pareto-Optima finden läßt [2].

## 2 Problemformulierung

Das Problem der Partitionierung von Schaltungen findet sich im VLSI-Designprozeß recht häufig wieder. Als Einsatzgebiete sind in erster Linie Layoutentwurf, FPGA-Mapping und auch Schaltungssimulation zu nennen [8]. Ziel einer Partitionierung ist es, eine Schaltung in mehrere kleinere Teile zu zerlegen. Die Anzahl der Verbindungen zwischen diesen Teilen sollte möglichst gering sein.

Mathematisch läßt sich das Partitionierungsproblem als die Zerlegung eines Graphen darstellen, wobei die Komponenten der Schaltung als Knoten angesehen werden. In der Literatur finden sich unterschiedliche Modelle für die Beschreibung einer Schaltung als Graph. Dabei ist die Darstellung des Graphen in einer Adjazenzmatrix ungeeignet [13], so daß je nach erforderlicher Abstraktionsebene Modelle basierend auf bipartiten, tripartiten oder Hyper-Graphen [8] zur Anwendung kommen. Für die vorliegende Arbeit wurden bipartite Graphen ausgewählt.

Ein bipartiter Graph  $G = (V, E)$  enthält eine Menge Knoten  $V$  und eine Menge Kanten  $E$ . Jeder Knoten  $v_i \in V$  läßt sich einer von zwei Untermengen  $V_p$  (primäre Knoten) oder  $V_s$  (sekundäre Knoten) zuordnen. Ein beliebiger Graph  $G$  ist genau dann bipartit, wenn er ausschließlich Kanten zwischen Knoten der beiden Mengen  $V_p$  und  $V_s$  besitzt. Elektronische Schaltungen lassen sich mit bipartiten Graphen darstellen, indem alle Schaltungselemente primären Knoten und alle Netze sekundären Knoten zugeordnet werden (siehe dazu den oberen Teil von Abbildung 4).

Bei der Partitionierung eines Graphen werden die Knoten der Menge  $V_p$  einzelnen Untermengen  $V_{p1} \dots V_{pk}$  zugeordnet. Die Anzahl der Verbindungen zwischen Knoten der Untermengen soll dabei möglichst minimal werden. Diese Anzahl wird als Schnittgröße oder häufig auch als *NetCut* bezeichnet. Da ein bipartiter Graph verwendet wird, existieren keine direkten Kanten zwischen den Knoten der Untermengen; jede beliebige Verbindung läuft über einen Knoten der Menge  $V_s$ . Der *NetCut* ist also gleich der Anzahl der sekundären Knoten, die Kanten zu primären Knoten in mehr als einer Untermenge  $V_p$  besitzen.

Für das Partitionierungsproblem existieren mehrere verschiedene Optimierungskriterien. Da diese in einem Lösungsalgorithmus eine tragende Rolle spielen, sollen die wichtigsten nachfolgend erläutert werden.

Das primäre Kriterium, der *NetCut*  $c(G)$ , wurde bereits angesprochen. Das Kriterium läßt sich wie folgt definieren:

$$c(G) = |v_{si} \in V_s| : \exists e\{v_{si}, v_{pl}\} \wedge e\{v_{si}, v_{pm}\} \Rightarrow \text{minimal}$$

$$\text{mit } v_{pl} \in V_{pn}, v_{pm} \in V_{po} \text{ und } V_{pn} \neq V_{po} \quad (1)$$

Der Ausdruck  $e\{v_{si}, v_{pl}\}$  bezeichnet dabei eine Kante  $e$  zwischen dem sekundären Knoten  $v_{si}$  und dem primären Knoten  $v_{pl}$ , der der Knotenmenge  $V_{pn}$  zugeordnet ist.

Neben der Minimierung des *NetCuts* wird in der Regel ein Balance-Kriterium eingeführt. Dies ist notwendig, da ein stochastischer Suchalgorithmus eine Lösung finden würde, bei der eine Partition einen Großteil der Schaltungselemente enthält, während alle anderen Partitionen (nahezu) leer sind. Dies wäre eine gute Lösung, wenn nur der *NetCut* minimiert werden soll, ist in der praktischen Anwendung aber nutzlos. Ein Maß für die Balance  $b(G)$  eines partitionierten Graphen  $G$  mit  $k$  Partitionen läßt sich wie folgt angeben:

$$b(G) = \sum_{i=1}^k \left| \frac{n_{opt} - |V_{pi}|}{n_{opt}} \right| \quad \text{mit} \quad n_{opt} = \frac{|V_p|}{k} \quad (2)$$

Der Wert  $|V_p|$  ist dabei die Anzahl der primären Knoten im Graphen,  $|V_{pi}|$  ist die Anzahl der Knoten in einer Partition,  $n_{opt}$  die optimale Größe einer Partition. Es ist leicht zu sehen, daß ein perfekt ausbalancierter Graph, bei dem alle Partitionen die gleiche Anzahl von Knoten besitzen, eine Balance von  $b(G) = 0$  haben muß, abweichende Graphen jedoch Werte größer 0 für  $b(G)$  aufweisen. Der Wert  $b(G)$  für eine beliebige Partitionierung muß also minimiert werden.

Ein weiteres Kriterium kann die Signalverzögerung zwischen Elementen in verschiedenen Partitionen sein. Diese Signalverzögerung  $d$  soll während der Optimierung minimiert werden:

$$d(V_{pi}, V_{pj}) \quad \text{mit} \quad (i \neq j) \quad \Rightarrow \quad \text{minimal} \quad (3)$$

Bei der Partitionierung von Schaltungen finden sich allerdings nicht nur Optimierungskriterien, sondern auch Nebenbedingungen die zwingend eingehalten werden müssen. Diese sind von der konkreten Problemstellung abhängig und werden im Abschnitt 5 näher behandelt.

### 3 Der Strength Pareto Evolutionary Algorithm

Der Strength Pareto Evolutionary Algorithm (SPEA) wurde 1999 von Zitzler in [15] vorgeschlagen. Er gehört zur Familie der sogenannten elitären evolutionären Algorithmen und zeichnet sich durch gute Performance aus [4], weshalb er unter mehreren verschiedenen Algorithmen für diese Arbeiten ausgewählt wurde. Der Algorithmus soll im folgenden näher erläutert werden.

SPEA arbeitet mit zwei Populationen, einer normalen  $P$  und einem sogenannten *Archiv*  $A$ . Im Archiv werden nicht-dominierte Lösungen gespeichert, die mit fortschreitendem Algorithmus gefunden werden. In jeder Generation werden die aktuellen Lösungen mit denen im Archiv verglichen und die nicht-dominierten in das Archiv übernommen. Die Lösungen im Archiv nehmen ebenfalls an den genetischen Operationen teil, in der

Hoffnung den Suchprozeß in Bereiche mit guten Lösungen zu lenken. Ein Schema des Algorithmus ist in Abbildung 2 dargestellt.

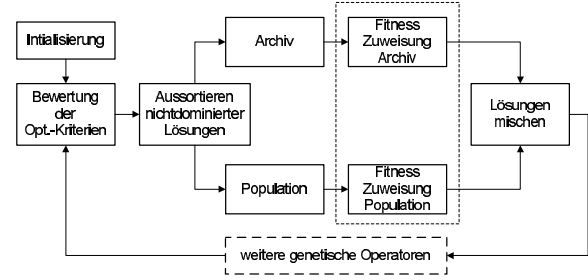


Abbildung 2: schematische Darstellung von SPEA

Der Algorithmus beginnt mit  $N_p$  zufällig erzeugten Individuen und einem (zunächst leeren) Archiv der maximalen Größe  $N_a$ . Für diese Individuen werden danach die Zielfunktionswerte berechnet. Anhand dieser ist es möglich die nicht-dominierten Lösungen auszusortieren und in das Archiv zu kopieren, das sich im Verlauf des Algorithmus mit nicht-dominierten Lösungen füllt. Individuen, die sich bereits im Archiv befinden, nun aber von neuen Lösungen aus der aktuellen Population dominiert werden, werden aus dem Archiv gelöscht. Damit bleiben im Archiv die nicht-dominierten Lösungen beider Populationen zurück. Mit fortschreitendem Algorithmus besteht die Gefahr der Überfüllung des Archivs mit nicht-dominierten Lösungen. SPEA begegnet dieser Gefahr mit der Limitierung der Archivgröße und einem Clustering-Algorithmus, der die Anzahl der Lösungen, die die Paretofront kennzeichnen, reduziert. Dieser soll an dieser Stelle nicht erläutert werden, für weitere Informationen sei auf [15] verwiesen.

Der nächste Schritt des Algorithmus ist die Fitneßzuweisung<sup>2</sup> aller Individuen. Diese unterteilt sich in zwei Phasen: zunächst Fitneßzuweisung der Individuen im Archiv und danach Fitneßzuweisung der Individuen in der aktuellen Population. Der Fitneßwert der Individuen wird *Strength*  $S$  genannt. Jeder Lösung  $i$  im Archiv wird der Wert  $S_i$  zugewiesen:

$$S_i = \frac{n_i}{N + 1} \quad (4)$$

Der Wert  $n_i$  steht für die Anzahl der Lösungen in der aktuellen Population, die von Lösung  $i$  dominiert werden.  $N$  ist die Gesamtanzahl der Lösungen in der aktuellen Population. Mit der Addition mit 1 wird sichergestellt, daß in jedem Fall  $S_i < 1$  gilt.

In ähnlicher Weise wird jedem Individuum  $j$  in der aktuellen Population ein Fitneßwert  $F$  zugewiesen:

$$F_j = 1 + \sum_{i \in A \wedge i \succ j} S_i \quad (5)$$

<sup>2</sup>In der Regel wird eine bessere Fitneß durch einen höheren Zahlenwert dargestellt. Diese Definition wird in SPEA umgekehrt; ein niedrigerer Zahlenwert stellt also eine bessere Fitneß dar.

Die Fitness  $F_j$  berechnet sich aus der Summe aller Strength-Werte der Individuen im Archiv, die die Lösung  $j$  dominieren. Die Addition mit eins stellt sicher, daß für alle Fitnesswerte  $F_j > 1$  gilt. Aus beiden genannten Ungleichungen läßt sich leicht ersehen, daß Individuen in der momentanen Population niemals eine bessere Fitness haben können als die Lösungen im Archiv.

Nach der Fitnesszuweisung werden die Lösungen aus der aktuellen Population und aus dem Archiv vermischt und mit genetischen Operatoren weiter verarbeitet. Zusammen mit diesen Operatoren bildet der beschriebene Mechanismus den gesamten Evolutionszyklus. Die Operatoren, die für die weitere Verarbeitung eingesetzt werden, sind problemspezifisch und werden im nächsten Abschnitt beschrieben.

#### 4 Genetische Repräsentation und Erzeugung neuer Lösungen

Und einen vollständigen Evolutionszyklus zu bilden, wurden die in Bild 3 dargestellten Operatoren eingesetzt. Bevor diese allerdings näher erläutert werden können, muß zunächst die Art der genetischen Repräsentation<sup>3</sup> diskutiert werden.

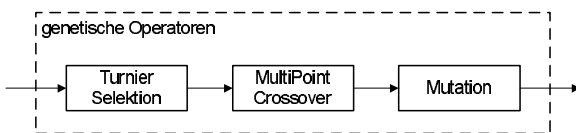


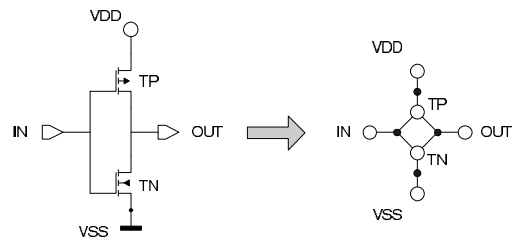
Abbildung 3: verwendete genetische Operatoren

Wie bereits angesprochen, wird bei der Partitionierung von Schaltungen jedem Schaltungselement (also jedem primären Knoten im Graphen) eine Partitionsnummer zugeordnet. Es bietet sich daher ein Vektor mit Partitionsnummern als genetische Repräsentation an. Abbildung 4 zeigt das Prinzip anhand eines einfachen CMOS-Inverters. In diesem Beispiel wird das Element  $TP$  der Partition 1 und das Element  $TN$  der Partition 2 zugeordnet. Die Länge des Vektors ist gleich der Anzahl der Schaltungselemente (zwei beim Inverter), falls I/O- und Versorgungspins nicht als Schaltungselemente gerechnet werden.

Nach der Festlegung der genetischen Repräsentation können geeignete Rekombinations- und Mutationsoperatoren ausgewählt werden. Da jedes Individuum einen Vektor aus Zahlen als 'Erbgut' trägt, müssen auch die verwendeten Operatoren vektorbasiert sein. Da der Zahlenvektor (zunächst) ohne Einschränkungen jede beliebige Form annehmen kann, bietet sich als Rekombinationsoperator einfaches Multipointcrossover (MPX) an. Dieses ist in Abbildung 5 dargestellt.

Beim MPX wird eine (alternativ auch mehrere) zufällige Position im Zahlenvektor ausgewählt. Elemente aus Elternteil  $A$  werden zunächst in den Nachkommen

<sup>3</sup>die genetische Repräsentation ist die Datenstruktur auf der alle genetischen Operatoren aufsetzen



Repräsentation durch Vektor :

Netzlistenelemente	TP	TN	...
zugeordnete Partitionen	1	2	...

Abbildung 4: genetische Repräsentation

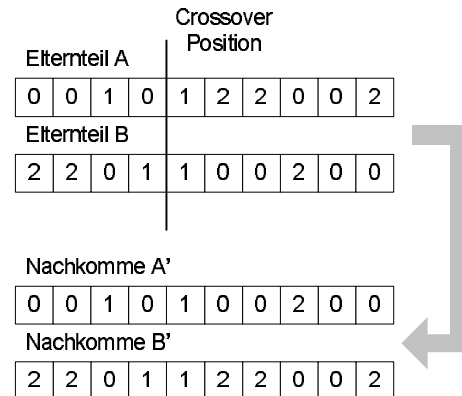


Abbildung 5: Multipointcrossover

$A'$  und Elemente aus Elternteil  $B$  in Nachkommen  $B'$  übernommen. Wenn die vorher gewählte Position erreicht ist wird das Schema vertauscht, d.h. Elemente aus Elternteil  $B$  werden nach  $A'$  übernommen und umgekehrt. Bei mehreren anfänglich gewählten Positionen wird dieser Vorgang alternierend wiederholt.

Als Mutationsoperatoren kommen mehrere verschiedene zur Anwendung. Zwei davon sind stellvertretend in den Abbildungen 6 und 7 dargestellt. Ersterer ist die sogenannte *Swap Mutation*. Dabei werden zwei Elemente im Vektor zufällig ausgewählt und vertauscht. Dieser Vorgang läßt sich auch mehrmals wiederholen, abhängig davon welcher Grad der Durchmischung erreicht werden soll.

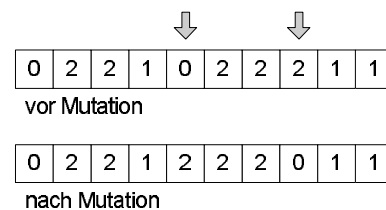
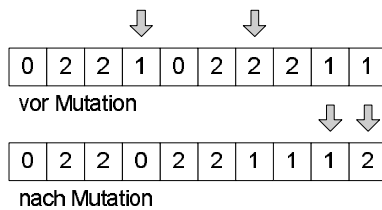


Abbildung 6: Swap Mutation

Ein zweiter Mutationsoperator ist die *Shift Mutation*. Dabei wird eine Anzahl von Elementen zufällig ausgewählt und aus dem Elementvektor entfernt (in der Dar-

stellung mit Pfeilen markiert). Alle Elemente, die zwischen dem ersten entfernten Element und dem Ende des Vektors liegen, werden nach vorn verschoben (*Shift*). In einem weiteren Schritt werden die vorher entfernten Elemente an den Vektor in der gleichen Reihenfolge angehängt.



**Abbildung 7:** Shift Mutation

Es ist leicht zu sehen, daß beide Mutationsoperatoren unterschiedlich starke Wirkung auf den Elementvektor haben. *Swap Mutation* verändert nur einige einzelne Elemente. Es wird jeweils ein Knotenpaar im Graphen ausgewählt und die Partitionszuordnungen der Knoten vertauscht. Die Partitionen werden also lediglich wenig verändert. Im Gegensatz dazu verändert *Shift Mutation* die Partitionszuordnung der meisten Knoten des Graphen (je nach Startposition für die Shift-Operation). Lediglich die Zuordnung der Knoten links der Startposition bleibt unverändert.

Zusätzlich zu den beschriebenen Mutationsoperatoren, bei denen die Größen der Partitionen konstant bleiben, existieren noch andere Operatoren, die die Größe der Partitionen verändern, indem Knoten aus einer beliebigen Partition in eine (oder mehrere) andere übergehen. Allen ist allerdings ihr Zweck gemein: sie haben die Aufgabe die Suche im Suchraum in verschiedene Richtungen und unterschiedlich stark auszudehnen.

Als Selektionsoperator (siehe Abbildung 3) kommt die sogenannte Turnirselektion zur Anwendung. Dabei werden aus beiden Population zwei Individuen zufällig ausgewählt. Mit diesen wird ein Turnier durchgeführt, das das Individuum mit dem besseren Fitnesswert gewinnt. Dieser Vorgang wird so oft wiederholt bis genügend Individuen für den darauf folgenden Rekombinationsoperator selektiert wurden.

## 5 Anwendung des Algorithmus in laufenden Arbeiten

Laufende Arbeiten am Fachgebiet ESS befassen sich mit der Abbildung von analogen Schaltungen auf programmierbare analoge Bausteine (siehe dazu [7]). Der beschriebene Algorithmus soll in eine Software zur automatischen Partitionierung, Platzierung und Routing von Schaltkreisen integriert werden und wird derzeit anhand von verschiedenen Benchmarks evaluiert. Die Anwendung des Algorithmus ist für den Entwurf analoger Schaltungen vorgesehen, die beschriebene Methodik läßt sich allerdings prinzipiell auch für digitale Schaltungen verwenden.

Der Algorithmus wurde zunächst auf sein Verhalten bei  $k$ -facher Partitionierung (d.h. Zerlegung von Graphen in  $k$  Partitionen) untersucht. Als Benchmark lagen Graphen aus [3] zugrunde, um Ergebnisse mit bereits publizierten Resultaten vergleichen zu können. Als Optimierungskriterien kamen wie bereits beschrieben die Größe des NetCuts und die Balance zum Einsatz. Es zeigte sich, daß der Algorithmus in seiner momentanen Form z.T. in der Lage ist, mit anderen gängigen Algorithmen zu konkurrieren, allerdings auf Kosten von höherer Rechenzeit.

Die Parameter der verwendeten genetische Operatoren wurden wie folgt gesetzt:

- Anzahl der Individuen : 120
- Größe des Archivs : 30
- Mutationswahrscheinlichkeit :  $0.1^4$
- Anzahl der Generationen : 100

Die Größe des zu partitionierenden Graphen beeinflußt in wesentlichem Maße die Laufzeit eines (beliebigen) Lösungsalgorithmus. Das feste Abbruchkriterium von 100 Generationen, das für die ersten Versuche verwendet wurde, ist für den realen Einsatz des Algorithmus nahezu nutzlos. Mit diesem Abbruchkriterium läßt sich das z.T. gute (geringer NetCut bei hoher Laufzeit) bzw. sehr schlechte (hoher NetCut bei großen Graphen) Verhalten des Algorithmus erklären. Für die praktische Anwendung muß also noch ein geeignetes Abbruchkriterium gefunden werden.

Zur softwaretechnischen Umsetzung des Algorithmus muß angemerkt werden, daß die Implementation mit Hilfe eines am Fachgebiet ESS entwickelten Frameworks [5] zum Fast-Prototyping von evolutionären Algorithmen in Java erfolgt ist. Der Code der bisherigen Umsetzung ist in keinsten Weise optimiert. Dazu kommt ein Performanceverlust durch die Verwendung der Java-Technologie, sowie Overhead durch Verwendung des generischen Toolkits. Bei einer optimierten Umsetzung des Algorithmus in C sind daher wesentlich kürzere Laufzeiten zu erwarten.

Bei einer Partitionierung einer Schaltung in  $k$  Teile werden in der Regel Heuristiken zur Bipartitionierung rekursiv angewendet [14]. Bei realen Partitionierungsaufgaben ist die Zahl  $k$  allerdings in der Regel unbekannt und wird von einer oder mehreren Nebenbedingungen (Constraints) bestimmt. Bei der Abbildung von Schaltungen auf programmierbare Bausteine darf beispielsweise die Größe einer Partition (also die Anzahl der Elemente, eventuell jeweils multipliziert mit einem Wichtungsfaktor) eine gewisse Maximalgröße nicht überschreiten. Die genannte Aufgabenstellung aus dem Analogbereich verschärft diese Bedingungen sogar: es dürfen jeweils nur eine maximale Anzahl von Elementen (Widerstände,

<sup>4</sup>Die Mutationswahrscheinlichkeit ist die Wahrscheinlichkeit, mit der ein Mutationsoperator ein Individuum verändert. Diese darf nicht zu hoch gewählt werden, da andernfalls das Verhalten des Algorithmus einer zufälligen Suche ähneln würde.

Kondensatoren, Transistoren) in einer Partition vorhanden sein, da ansonsten eine ungültige Partition entsteht, die keiner Zelle auf dem programmierbaren Baustein bei der Platzierung zugewiesen werden kann.

Dem Problem der Behandlung von Nebenbedingungen kann durch verschiedene Techniken begegnet werden. Eine Variante ist, in den Algorithmus einen Operator zur 'Reparatur' von Individuen einzuführen. Damit würde sichergestellt, daß alle Lösungen zu jedem Zeitpunkt gültig sind. Allerdings kann diese Reparatur sehr aufwendig werden, je komplexer die jeweiligen Nebenbedingungen aussehen. Eine andere Variante ist das Zulassen ungültiger Lösungen. Diese Lösungen könnten nach Abbruch des Algorithmus herausgefiltert werden. Jedoch kann an dieser Stelle nicht sichergestellt werden, daß überhaupt eine gültige Lösung gefunden wird.

Aus Sicht des Autors erscheint es günstig, einen neuen Rekombinationsoperator zu entwickeln, der automatisch die vorhandenen Schaltungselemente auf mehrere Partitionen verteilt, so daß immer gültige Individuen entstehen. Dazu könnte die notwendige Anzahl von Partitionen vor Beginn der Optimierung abgeschätzt und zur Sicherheit vergrößert werden (z.B. +100%). Die Anzahl der verwendeten Partitionen könnte in ein Optimierungskriterium verwandelt werden und schrittweise verkleinert werden. Dazu könnten neue Mutationsoperatoren zum Einsatz kommen, die in [1] vorgeschlagen wurden und momentan vom Autor und der Autorin von [1] weiterentwickelt werden.

## 6 Zusammenfassung

In diesem Artikel wurde das Problem der Partitionierung von Schaltungen unter mehrkriteriellen Gesichtspunkten betrachtet. Für die Lösung der Partitionierungsaufgabe wurde der mehrkriterielle evolutionäre Algorithmus SPEA beschrieben. Der Algorithmus wird derzeit auf sein Verhalten bei verschiedenen Benchmarks untersucht. Die Experimente sind noch nicht abgeschlossen, zeigen aber vorläufig vielversprechende Ergebnisse. Sollte sich der Algorithmus im Laufe dieser Arbeiten als robust und zuverlässig erweisen, steht seiner Anwendung bei anderen Problemstellungen, wie Platzierung und Routing, nichts im Weg.

## Literatur

- [1] Adriana Apetrei, Ovidiu Gheorghies, Henri Luchian and Rolf Drechsler. An Evolutionary Approach to Graph Partitioning. In *Evolutionary Methods for Design Optimisation and Control*. 2002.
- [2] Y. Censor. Pareto Optimality in Multiobjective Problems. *Applied Mathematics and Optimization*, (4):41–59, 1977.
- [3] The Circuit Partitioning Page. <http://vlsicad.cs.ucla.edu/~cheese/benchmarks.html>. URL time: January 20th, 2002, 10<sup>00</sup>.
- [4] Kalyanmoy Deb. *Multi-Objective Optimization using Evolutionary Algorithms*. John Wiley & Sons, Chichester, 2001.
- [5] eaLib - A Java Evolutionary Computation Toolkit. <http://www.inf-technik.tu-ilmeneau.de/~rummler/eng/ealib.html>. URL time: August 28th, 2001, 14<sup>30</sup>.
- [6] John Henry Holland. *Adaption in Natural and Artificial Systems*. MIT Press, Cambridge, Massachusetts, 1975.
- [7] Jürgen Kampe, Christiane Wisser and Gerd Scarbata. Module Generators for a Regular Analog Layout. In *Proceedings 1. Workshop System Design Automation*. 1998.
- [8] Pinaki Mazumder and Elizabeth M. Rudnick. *Genetic Algorithms for VLSI Design, Layout & Test Automation*. Prentice Hall PTR, 1999.
- [9] K. Miettinen, P. Neittaanmäki, M. M. Mäkelä and J. Periaux, editors. *Evolutionary Algorithms in Engineering and Computer Science*. John Wiley and Sons, Chichester, Weinheim, New York, 1999.
- [10] Hartmut Pohlheim. *Evolutionäre Algorithmen*. Springer-Verlag Berlin, Heidelberg, New York, 2000.
- [11] Ingo Rechenberg. *Evolutionsstrategie - Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Fromman-Holzboog, Stuttgart, 1973.
- [12] Hans-Paul Schwefel. *Evolutionsstrategie und numerische Optimierung*. Ph.D. thesis, Technische Universität Berlin, 1975.
- [13] D.G. Schweikert and B.W. Kernighan. A Proper Model for the Partitioning of Electrical Circuits. In *Proceedings of the Ninth Design Automation Workshop on Design Automation*, pages 57–62. 1972.
- [14] Naveed Sherwani. *Algorithms for VLSI Physical Design Automation*. Kluwer Academic Publishers, Boston, Dordrecht, London, third edition, 1998.
- [15] Eckart Zitzler. *Evolutionary Algorithms for Multi-objective Optimization: Methods and Applications*. Ph.D. thesis, Eidgenössische Technische Hochschule Zürich, 1999.