Graph Partitioning Revised - a Multiobjective Perspective

Andreas RUMMLER
Department of Electronic Circuits and Systems, Technical University of Ilmenau
Ilmenau, 98684, Germany

and

Adriana APETREI
Department of Computer Science, Al. I. Cuza University
Iasi, 6600, Romania

ABSTRACT

This paper introduces the possibility of solving the graph partitioning problem by use of a hybrid multi-objective evolutionary algorithm. The paper starts with the mathematical formulation of the graph partitioning problem and a short review of conventional approaches. After a short introduction into the basics of multi-objective optimization a new persepective on the task is proposed. The next two sections give an overview over the multi-objective evolutionary algorithm SPEA and the local improvement operator that has been used to solve the problem. In the last part results for several benchmarks are presented and the currently existing deficiencies of the used technique are analyzed. The article concludes with a short outlook on future work that has to be done to make the proposed technique applicable.

Keywords: Genetic/Evolutionary Algorithm, Graph Partitioning, Multi-Objective Optimization

1. INTRODUCTION

Partitioning is still a significant problem in various engineering disciplines. It plays a key role in designing computer systems in general and VLSI chips in particular. In most cases graphs are used for modelling electronic circuits and systems. Therefore, the partitioning of a whole system can be traced back to a graph partitioning problem.

Depending on the desired abstraction level for modelling an electronic circuit several different graph models can be used: simple clique model, bipartite or tripartite graphs. The deficiencies of the clique model have already been shown by Schweikert and Kernighan in [13]. That's why in most cases a bipartite model is used. The implementation of this model can be simplified by the use of hypergraphs. A hypergraph is the generalization of a normal graph with the extension that edges may have more than two incident vertices. Such an edge is called a hyperedge.

The problem of partitioning a hypergraph can be formulated as follows: Let $H = (V, E)$ be a hypergraph with a non-empty set of vertices V and a set of hyperedges E. Each vertex $v_i \in V$ is assigned to one of the subsets (partitions) $V_1 \dots V_k$ such that $V_i \bigcap V_j = \oslash$ for $i \neq j$. As a consequence a set of hyperedges $E_C \in E$ exists that contains all hyperedges that have at least two incident vertices that belong to different vertex subsets. This set of hyperedges is refered to as the *cutset*. The *cutsize* $c(H) = |E_C|$ equals the number of hyperedges belonging to the cutset. The aim of partitioning the hypergraph is to find an assignment of all vertices such that the cutsize becomes minimal. Depending on whether the number of vertex subsets k is known or not, this task is refered to as partitioning of a hypergraph resp. clustering of a hypergraph.

2. CONVENTIONAL APPROACHES

Over the years many algorithms for solving the graph partitioning problem have been developed. Depending on the basic ideas of these algorithms several different classifications can be undertaken. One is the differentiation between *constructive* and *iterative* algorithms.

Constructive algorithms start with empty partitions and fill these partitions with vertices during a run. An example are the algorithms presented by Kodres in [9] which start from one (or more) seed vertices to form partitions. The solution quality of such a clustering algorithm is in most cases highly dependent on the initial choice of the seed vertex.

Iterative techniques start with an initial random solution and try to improve this solution in one or more sequential runs. The first algorithm of this kind has been introduced by Kernighan and Lin in [8] and is specialized in bipartitioning. This algorithm tries to improve an initial solution by swapping the partition assignment of two vertices. The algorithm has been improved later by Fiduccia and Mattheyses in [6]. In this paper the notion of the *gain* of moving a vertex to another partition has been introduced. The gain has been defined as the improvement in cutsize that can be achieved by executing a move. By use of specialized data structures (bucket arrays) a single iteration of the algorithm has the complexity of $O(n)$ instead of $O(n \log n)$ in the KL-algorithm. Krishnamurthy enhanced the FM-algorithm in [10] with a look-ahead strategy using *level gains*. Sanchis generalized these approaches in [12] to k-way partitioning.

An alternative classification can be made based on the computation scheme of the algorithm: deterministic and stochastic algorithms. The first group is able to reproduce a result of an algorithm's run. The techniques mentioned in the last section belong to this class (although some random decisions may be made during a run). Stochastic algorithms do not guarantee the finding of a particular solution, instead there is a high propability to find a different solution with each run. An example for stochastic techniques are algorithms based on *simulated annealing*. This kind of algorithms start with a random solution and perform incremental refinements by moving vertices between partitions. A move will be accepted if it introduces an improvement in the cost function and may be accepted with a particular propability if no improvement can be achieved. The propability that a move is accepted is calculated based on the Maxwell-Boltzmann distribution. Another stochastic approach is evolutionary algorithms (EAs). These algorithms borrow their basic idea from Darwin's evolution theory. EAs start with a collection of initial random solutions (the population) and try to improve these solutions by applying *genetic operators*. The authors will dwell on this technique later in this article and some of the operators that can be applied in such algorithms will then be explained further. Stochastic algorithms are able to find near-optimal solutions and are quite insusceptible against being trapped in local optima unlike deterministic algorithms. The major drawback of such techniques is the amount of necessary computational effort that is significantly higher than that of heuristic algorithms. For that reason they only became applicable by the rapid increase of computing power.

3. THE MULTIOBJECTIVE PERSPECTIVE

Conventional partitioning algorithms are specialized in optimizing one single objective (in most cases the cutsize). An additional criterion (partitions of nearly the same size) are normally treated as constraints. In this section the authors introduce a multiobjective perspective which generalizes these approaches.

In real-world optimization tasks it is most likely that there is more than one objective to be handled. These objectives may contradict each other. For example, one may want to buy a new car, which should be as fast and as cheap as possible. In a such scenario it is impossible to find *the best* solution, simply because no such solution exists (nobody will sell a Ferrari for only 1$). Instead the aim of a multiobjective optimization task is finding a good compromise.

To get a better idea have a look at figure 1, which shows a cutout from the search space of a fictitious optimization problem in which the values of two criteria must be minimized. This search space contains a particular number of solutions[1], where it is impossible to improve one criterion any further without deteriorating the other(s). This set of solutions forms a borderline, called the *pareto front*, that cannot be exceeded. So the aim of the optimization task is to find solutions from the search space which are situated as close as possible to this frontier.

[1] In a continuous search space the number of these solutions is infinite.

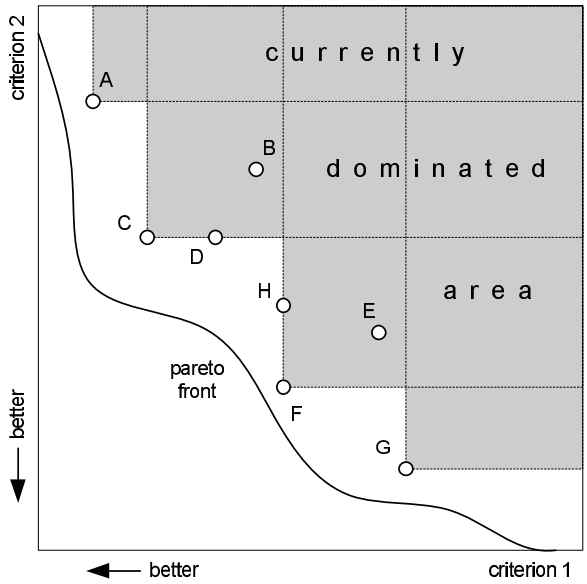

Figure 1: fictitious search space including pareto front, dominated and non-dominated solutions

To be able to evaluate solutions from the search space they must be compared to each other. In the shown example solution C is better in both criteria than solution B. This is called *dominance* of C over B ($C \succ B$) or B is dominated by C. In the case of solutions C and D resp. F and H only one criterion is better, the values of the other one are equal. This case is called *weak dominance* ($C \succeq D$ und $F \succeq H$). Solutions A and G each favour one criterion over another while neglecting the other. Both solutions are *indifferent* to each other ($A \sim G$). The cutout of the search space which is currently dominated is spanned by solutions A, C, F and G and is shaded in figure 1.

As already stated the result of an multiobjective optimization is not a single solution, but a vector of solutions, each representing a pareto optimum. From this vector one solution is picked out as the final solution by a so called *decision maker*. The decision maker (which may be a machine *or* a human) performs the decision by taking existing constraints into account.

After having introduced the basic elements of multiobjective optimization it is possible to apply the graph partitioning problem to this scheme. In graph partitioning most algorithms perform an optimization towards one criterion. The most important one is of course the minimization of the cutsize $c(H)$. In other words the purpose is to minimize the number of hyperedges $|E_C|$ that have incident vertices belonging to different partitions. The incidence of a hyperedge e_i with a vertex v_j is denoted by $e_i \leftrightarrow v_j$. This criterion can be written as

$$\begin{aligned} c(H) = |e_i \in E| \quad &\Rightarrow \text{minimal} \\ \text{with} \quad \exists\, e_i \leftrightarrow v_j, e_i \leftrightarrow v_k & \\ \text{and} \quad v_j \in V_m, v_k \in V_n \quad &\text{and} \quad V_m \cap V_n = \emptyset \end{aligned}$$

An algorithm trying to minimize only the cutsize would produce a particular solution: all vertices assigned to the

same partition. This is a very good solution in terms of cutsize (which is zero), but of course a useless one. That is why most algorithms include a heuristic to preserve or restore the balance of the solution. Balancing a solution means that all partitions have the same size resp. contain the same number of vertices. It is important that the partitions *should* have same sizes but need not. Unlike in conventional approaches where the balance is often treated as a constraint it is possible to turn the balance into a second criterion (according to the multiobjective paradigm introduced). This criterion can be formulated as follows:

$$b(H) = \sum_{i=1}^{k} \left| \frac{n_{opt} - |V_i|}{n_{opt}} \right| \quad \text{with} \quad n_{opt} = \frac{|H|}{k} \quad \Rightarrow \text{minimal} \tag{1}$$

Finding a minimal cutsize implies that most of the hyperedges in a graph should "reside in the same partition", that is, most hyperedges should be incident to vertices assigned to the same partition V_i. A third criterion can be defined for this requirement, that could be used as an alternative to the first one: maximize the number of hyperedges that have incident vertices that belong to the same partition.

Other criteria may be formulated based on the partitioning task originating from the concrete real-world problem. Especially in VLSI design other criteria are conceivable, like for instance the minimization of the signal delay between different partitions.

What is the motivation for this multi-objective approach? Real optimization tasks in which more than one criterion must be considered are not individual cases – they are standard cases. Traditionally a multiobjective optimization problem is led back to one with a single objective. Quite popular is the approach of weighted sums, that assigns a weight to each criterion and adds up all values. But there are many cases where it is difficult to define the weights because of the different range of values of the criteria. Beyond that it can be shown mathematically that in several cases no optimal solution can be found [1].

Evolutionary algorithms work with several solutions simultaneously. For this reason they present themself to be used in multi-objective optimization. Nevertheless have been used for such tasks only recently. The book by Deb [4] gives a very good overview over the current state of research in this field.

4. THE STRENGTH PARETO EVOLUTIONARY ALGORITHM

The *Strength Pareto Evolutionary Algorithm (SPEA)* was introduced in 1999 by Zitzler. It belongs to the class of so-called elitist evolutionary algorithms and was chosen for this work because of its good performance compared to other similiar algorithms [4]. The algorithm is described in detail in [15], therefore only a short overview will be given here.

SPEA incorporates two populations instead of one: a normal population P and a so-called archive A. The archive stores non-dominated solutions, that have been found by the the algorithm during a single evolution cycle. In every generation the individuals of the current population are compared to the ones contained in the archive and the non-dominated are saved in the archive. The individuals from the archive also take part in all genetic operations to steer the algorithm towards areas in the search space with good solutions. An overview of the algorithm is shown in figure 2.

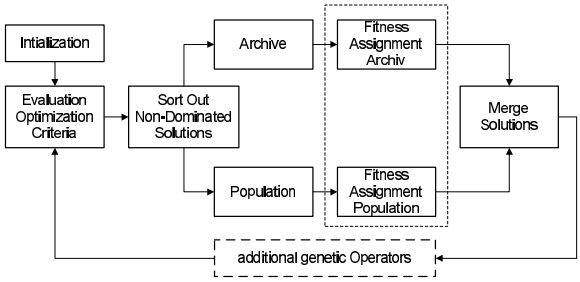

Figure 2: schematic overview over SPEA

SPEA starts with an initial number of N_p randomly created individuals and an empty archive of maximum size N_a. For all individuals the objective function(s) are evaluated and the non-dominated solutions are copied into the archive. Solutions that are already contained in the archive but are now dominated by newer individuals are deleted. After this step the archive contains the non-dominated individuals of both population and old archive contents. During the algorithm's runtime there is the danger of an overcrowded archive. SPEA solves this problem by applying a clustering algorithm if the number of solutions contained in the archive is greater than its maximum size. The clustering algorithm reduces the solutions that represent the pareto front. For a detailed description of the clustering algorithm one may refer to [15].

The fitness assignment is divided into two parts. First is the assignment of the fitness values to the individuals contained in the archive. This value is called *strength* S and is defined as:

$$S_i = \frac{n_i}{N+1} \tag{2}$$

The value of n_i represents the number of solutions in the current population, that are dominated by solution i. N is the total number of individuals in the current population. It applies that $S_i < 1$.

In the second step each individual j in the population is assigned a fitness value F in a similiar way:

$$F_j = 1 + \sum_{i \in A \wedge i \succ j} S_i \tag{3}$$

The fitness F_j is equal to the sum of all strength values of the individuals in the archive that dominate the solution j. It applies that $F_j > 1$. With a comparison of both inequations it can be seen that individuals contained in the population can never have a better fitness than the ones in the archive.[2]

After assigning the fitness values the solutions of both the archive and the population are merged and processed with suitable genetic operators that are problem-specific. The operators that have been used in this work for graph

[2] Unlike in other evolutionary algorithms in SPEA a smaller fitness value is defined to be the better one.

partitioning are explained in more detail in the next section.

5. GENETIC OPERATORS & LOCAL IMPROVEMENT

The genetic operators used for offspring creation (selection, recombination and mutation) in an evolutionary algorithm are always problem-specific. They must be tailored to the genetic representation that has been chosen to represent a potential solution of the particular optimization problem.

For the graph partitioning problem a vector-like genetic representation presents itself. Each vertex can be assigned an integer number indicating the index of the partition the vertex belongs to. Therefore a suitable genetic representation is a vector of integer numbers with the length equal to the number of vertices in the hypergraph.

All genetic operators used in the evolutionary algorithm must be tailored to the chosen vector representation. For recombination a biased uniform crossover has been used. The basic form of this operator has been described in detail in [14]. The modification that was made here concerns the contribution propability. This propability was calculated dependent on the fitness of both parents. The parent with the better fitness contributes more to the offspring than the worse one. Offspring solutions are more alike to the better parent than to the worse. For mutation a simple scrambling of the integer vector was used. The operator is applied with a propability of 0.05. Tournament selection was chosen as selection operator. As termination condition the average cutsize of all solutions representing the pareto front was calculated. The algorithm stops when the change of the average cutsize over the last 20 generations drops below 5%.

For support of the multiobjective evolutionary algorithm described above a local improver has been developed. Due to the fact that the improver must be applied to a number of individuals, it must have a small runtime. Because of this reason the multi-way partitioning algorithm developed by Sanchis has been used and modified for use as a local improver. The algorithm is explained in detail below. The reasons for incorporating a local improver into the evolutionary algorithm are explained in section 6.

The local improvement algorithm that has been used here is a move-based algorithm. The basic idea is to analyze the possible moves of all vertices for an improvement in cutsize. Only moves which improve the cutsize are executed - a typical hill-climbing strategy.

Every vertex i is assigned a gain vector Γ_i of the size equal to the number of partitions. Each element γ_j with $0 < j < |\Gamma_i|$ in the vector Γ_i represents the gain that is achieved by moving the vertex to partition j. The gain is defined as the improvement (the decrease) in cutsize that would be achieved if that move was executed. The gain can be both positive and negative.

In most cases a number of possible moves have the same gain, so it is difficult to find the move which is to be executed. For that reason each vertex was assigned a second gain vector Θ_i with the same size as Γ_i. The gain θ_j with $0 < j < |\Theta_i|$ represents the incidence gain that is achieved by moving the vertex to partition j. The incidence gain is defined as the improvement (the decrease) in the incidence degree of a vertex that would be achieved if that move was executed. The incidence degree is defined as the number of different partitions the neighbours of a vertex belong to. Hyperedges incident to a vertex can only be removed from the cutset if the incidence degree is equal to 1 (the vertex has only neighbours in the same partition), therefore the incidence degree must be lowered if possible.

Similiary two vectors B_i and B'_i are assigned to each hyperedge, both with the size of the number of partitions. Each element β_j in vector B_i represents the number of neighbouring vertices belonging to partition j. The elements in vector B'_i are defined as the sum of the β_j values that do not belong to the same partition.

All possible moves are held in a hash table, which guarantees storage and removal operations to be executed in *nlogn* time. Each executed move and the associated value of the cutsize are recorded in a list. At the end of a run of the improver this list is searched for the move where the cutsize has been minimal. Up to that move the recorded moves are replayed. The resulting configuration is used as genetic information in the next evolution cycle.

Algorithm 1 local improvement

```
initialize B and B' in all hyperedges
initialize Γ and Θ vectors in all vertices
calculate all β values in B vectors of all hyperedges
calculate all β' values in B' vectors of all hyperedges
calculate all γ and θ values of all vertices
create set of possible moves
while more moves are possible do
  get best move from move record
  execute move of vertex
  delete moves for this vertex from move record
  update β and β' values of incident hyperedges
  update γ and θ values of neighbouring vertices
end while
find move with best cutsize
replay stored moves
```

The whole improvement algorithm is shown in algorithm 1. By inspecting the algorithm it stands out that the balance of the created partitions was not taken into account. This is not necessary because the balance has been defined as an optimization criterion in SPEA – the multiobjective evaluation mechanism handles this criterion automatically.

6. ANALYSIS & RESULTS

The performance of the partitioning algoritm has been tested on several graphs from the benchmark suite available from [2]. The characteristics of the graphs are itemized in table 1.

The performance of evolutionary algorithms in terms of quality have been reported as very good in the literature. For our application the quality was very poor and quite disappointing. For this reason the local improvement operator has been incorporated. The difference between the multiobjective EA with and without local improvement is shown in figure 3. It seems that the evolutionary algorithm without the help of the improver nearly does not work at all. The reasons for that are analyzed later in this section.

Graph	# Vertices	# Hyperedges
fract	149	147
balu	801	735
primary1	833	902
bm1	882	903
test04	1515	1658
test03	1607	1618
struct	1952	1920

Table 1: characteristics of the test graphs including the number of vertices and hyperedges

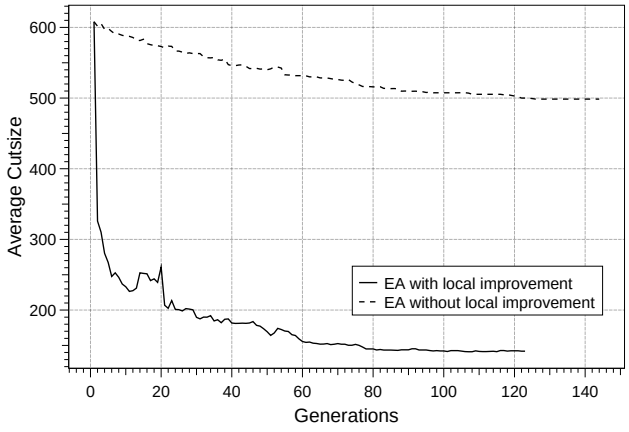

Figure 3: progress of the average cutsize in graph *balu* with and without local improvement

Figure 4 shows the progress of the pareto front in graph *balu* partitioned into four parts. The progress is quite fast during the first generations but slows down quite fast. After 80 generations the algorithm converges and the pareto front contracts towards one single point. This behaviour has been observed in almost all of the experiments.

By looking at figure 3 it might be assumed that graph partitioning with hybrid evolutionary algorithms incorporating local improvers works well. But indeed this is not the case if the results presented here are compared to other techniques. The partitioning results are summarized in tables 2 and 3. The first two columns show results from the hybrid evolutionary algorithm introduced here (best cutsize bHEA and average cutsize aHEA). For each graph the algorithm has been run 15 times. As final solution the solution with the lowest cutsize and a partition imbalance of not more than 10% has been chosen. For comparison the results from a heuristic (PFM3) and a simulated annealing algorithm (SA) reported in [3] and results generated with the hMETIS (MT) partitioning software available from [7] have been included.

Such heuristics are available for instance in the software package hMETIS [7]. The algorithm described here has been implemented by use of a generic software framework for evolutionary algorithms written in Java ([11], [5]) while hMETIS has been written in C. hMETIS has been highly optimized in the task of recursive bipartitioning while generic toolkits naturally include quite a lot of computational overhead. Therefore both implementations are too different to be compared in terms of runtime. But a

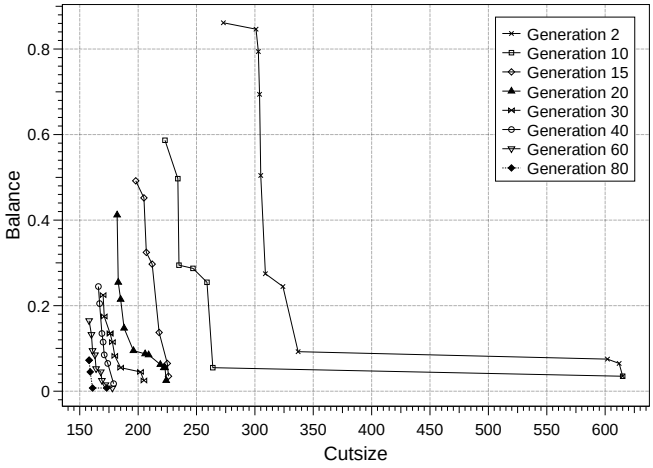

Figure 4: progress of the pareto front over 80 generations in graph *balu*

comparison of the quality of the results can be made and reveals that repetitive bipartitioning is able to produce results that are better by factor 2-3!

Graph	bHEA	aHEA	PFM3	SA	MT
fract	22	34	n/a	n/a	22
balu	124	141	75	76	45
primary1	155	186	112	114	92
bm1	158	186	n/a	n/a	91
test03	329	396	246	157	102
test04	316	363	n/a	n/a	97
test06	294	343	138	151	84
struct	284	308	111	130	64

Table 2: benchmark results, 4 partitions

Graph	bHEA	aHEA	PFM3	SA	MT
balu	178	227	159	146	74
primary1	230	303	149	144	115
bm1	224	247	n/a	n/a	127
test03	432	507	348	251	158
test04	440	488	n/a	n/a	130
test06	400	441	203	192	98
struct	360	400	295	180	93

Table 3: benchmark results, 8 partitions

Although the feasibility of evolutionary algorithms for optimization tasks in many engineering disciplines has been reported, this approach does not seem to work very well in our case. This raises the question for possible reasons which are analyzed in the rest of this section.

This first weak point is the genetic representation that has been used. In the vector representation that was used here there are p^v possible "individuals" with v being the number of vertices in the graph and p the number of partitions. A closer look reveals that there are a number of variants for the representation vector possible, that are redundant. For better explanation imagine a very simple graph with three vertices A, B and C that should be splitted in

two pieces. For this case there are $p^v = 2^3 = 8$ possible variants. But by counting all possiblilities by hand there are less: namely the possibilities $\{ABC\}\{\}$, $\{A\}\{BC\}$, $\{AB\}\{C\}$ and $\{AC\}\{B\}$. This is because there are variants that describe the same case. The two representation variants {000} and {111} describe that vertices A, B and C are assigned to partition 0 resp. to partition 1 – but the real proposition is another: all vertices are assigned to the same partition. Therefore one of both variants is redundant. Unfortunately 'relative' propositions like *"vertices A and B are assigned to another partition than vertex C"* cannot be expressed in this form of genetic representation.

The second weak point is the recombination operator. The idea of recombination in evolutionary algorithms is to inherit good properties or attributes from parents in hope for an improvement of these attributes in the offspring. This does not really happen in biased uniform crossover that has been used here. Namely, the operator permits more influence of better parents, but does not care about information about the current partitioning structure contained in the parent individuals. In This case the directed search through the search space that should be performed by the recombination operator is implemented in a very weak way.

A third point can be observed when examining figure 4. The pareto front contracts over the runtime of the algorithm towards a single point. This means the diversity of points in the search space that is used for the search decreases with progressing algorithm. With this decrease the chance of exploring new areas in search space is naturally reduced and the algorithm converges too early. This effect can also be seen when examining the benchmark results. The cutsizes achieved with the best runs of the algorithm are significantly lower than the average ones.

7. CONCLUSION

In this article an advanced approach for partitioning hypergraphs based on multi-objective evolutionary optimization coupled with local improvement support has been presented. Although the shown mechanism works, it is not able to compete with conventional heuristics. According to existing literature evolutionary algorithms are able to outperform other techniques in terms of result quality. In contrast the authors have shown that the application of such algorithms does not automatically imply good results. The application of standard operators for solving particular problems is no guarantee for the creation of a well-performing algorithm. All operators must be tailored carefully to the optimization task, otherwise the resulting algorithm works only with low performance or even not at all.

The weak points in the presented algorithm have been analyzed and are now subject of further investigations. It may not be possible to solve all open problems. Especially eliminating the redundant variants included in the genetic representation may be hard or even impossible. On the other hand it should be viable to incorporate more intelligence into the recombination to perform a much more effective search. The authors are convinced that with the development of such new operators similiar approaches like the one presented in this paper become applicable in the future.

References

[1] Y. Censor. Pareto Optimality in Multiobjective Problems. *Applied Mathematics and Optimization*, (4):41–59, 1977.

[2] The Circuit Partitioning Page. `http://vlsicad.cs.ucla.edu/~cheese/benchmarks.html`. URL time: January 20th, 2002, 10^{00}.

[3] Ali Dasdan and Cevdet Aykanat. Two Novel Multiway Circuit Partitioning Algorithms Using Relaxed Locking. Technical report, Computer Engineering Department, Bilkent University, Turkey, 1996.

[4] Kalyanmoy Deb. *Multi-Objective Optimization using Evolutionary Algorithms*. John Wiley & Sons, Chichester, 2001.

[5] eaLib. `http://www.inf-technik.tu-ilmenau.de/~rummler/eng/ealib.html`. URL time: August 28th, 2001, 14^{30}.

[6] Charles M. Fiduccia and R. M. Mattheyses. A Linear-Time Heuristic for Improving Network Partitions. In *Proceedings of the 19th IEEE Design Automation Conference*, pages 175–181. 1982.

[7] hMetis. `http://www-users.cs.umn.edu/~karypis/metis/hmetis`. URL time: April 4th, 2002, 14^{10}.

[8] B. W. Kernighan and S. Lin. An Efficient Heuristic Procedure for Partitioning Graphs. *The Bell System Technical Journal*, 29(2):291–307, 1970.

[9] U. R. Kodres. Partitioning and Card Selection. In Melvin A. Breuer, editor, *Design Automation of Digital Systems: Theory and Techniques*, volume 1, pages 172–212. Prentice Hall, 1972.

[10] Balakrishnan Krishnamurthy. An Improved Min-Cut Algorithm for Partitioning VLSI Networks. *IEEE Transactions on Computers*, 33(5):438–446, 1984.

[11] Andreas Rummler and Gerd Scarbata. eaLib - a Java Framework for Implementation of Evolutionary Algorithms. In *Computational Intelligence: Theory and Applications*, volume 2206 of *Lecture Notes in Computer Science*, pages 92–103. Springer-Verlag, 2001.

[12] Laura A. Sanchis. Multi-way Network Partitioning. *IEEE Transactions on Computers*, 38(1):1384–1397, 1989.

[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] Gilbert Syswerda. Uniform Crossover in Genetic Algorithms. In D. Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*. Morgan Kaufmann, 1989.

[15] Eckart Zitzler. *Evolutionary Algorithms for Multiobjective Optimization: Methods and Applications*. Ph.D. thesis, Eidgenössische Technische Hochschule Zürich, 1999.