Multi-Objective Optimization in Evolutionary Algorithms Using Satisfiability Classes

Nicole Drechsler, Rolf Drechsler, Bernd Becker

Institute of Computer Science
Albert-Ludwigs-University
79110 Freiburg im Breisgau, Germany
email: {ndrechsler,drechsler,becker}@informatik.uni-freiburg.de

Abstract. *Many optimization problems consist of several mutually dependent subproblems, where the resulting solutions must satisfy all requirements.*
*We propose a new model for Multi-Objective Optimization (MOO) in Evolutionary Algorithms (EAs). The search space is partitioned into so-called Satisfiability Classes (*SC*), where each region represents the quality of the optimization criteria. Applying the* SCs *to individuals in a population a fitness can be assigned during the EA run. The model also allows the handling of infeasible regions and restrictions in the search space. Additionally, different priorities for optimization objectives can be modeled. Advantages of the model over previous approaches are discussed and an application is given that shows the superiority of the method for modeling MOO problems.*

1 Introduction

Evolutionary Algorithms (EAs) become more and more important as a tool for search and optimization. Especially for hard combinatorial problems they often have successfully been applied (see e.g. [10, 3]). Multiple, competing criteria have to be optimized subject to a large number of non-trivial constraints. One strategy is to artificially divide a problem into a number of subproblems, which are then solved in sequence. Obviously, this is not a promising strategy if the objectives are conflicting. EAs are well suited for solving this kind of problems, when mutually dependent subproblems are considered in parallel. One problem arises when using EAs is to rank a pool of solutions.

Traditionally, the evaluation is done by an objective function which maps a solution of multiple objectives to a single value. A classical aggregative method is the *linear combination by weighted sum* where the value of each objective is weighted by a constant coefficient. The values of the weights determine how strong the specific objective influences the value of the single fitness value. Disadvantages are that the weights have to be known in advance to find good solutions or have to be determined by experiments. Obviously, this is time consuming and not desirable, since the parameters resulting from different runs may vary ending in “in-stable” algorithms [6].

1.1 Previous Work

Advanced methods for ranking solutions with multiple objectives have been developed over the years. If priorities exist between the objectives, a simple *lexicographic* order can be used[1].

In [8] a method is described where solutions with multiple objectives without preferences can be compared. This is realized by a relation, called *dominate*. A solution x *dominates* y, if x is equal or better for each objective than y, and x is for at least one component strongly better than y. Thus, the solutions in the search space can be ranked by the relation *dominate*. Implementations of this approach are presented in [9, 11].

In [7] another approach is proposed, where the search space is divided in a *satisfiable*, *acceptable*, and *invalid* range. This model has successfully been applied to a specific problem in the area of *chip design*, but it requires a user interaction. The designer has to specify the limits between *satisfiable*, *acceptable*, and *invalid* solutions.

1.2 Results

In this paper we propose a new model for ranking solutions of *Multi-Objective Optimization* (MOO) problems. A relation *favor* is defined analogously to *dominate* in [8], but this relation has several advantages, e.g. *favor* is able to compare solutions which are not comparable using relation *dominate*. The search space is divided into several *Satisfiability Classes* (*SCs*). Thus our approach can be seen as a generalization of the approach in [7] using a finer granularity. By this technique no user interaction is required any more. Furthermore, handling of priorities is also supported. Infeasible solutions are assigned to their own *SCs* depending on the objectives which are in an infeasible range. The *SCs* can be efficiently manipulated using operations on graphs. The model has been implemented and application to an example from the domain of *heuristic learning* [5] demonstrate the efficiency of our approach.

2 Multi-objective Optimization Problems

In general, many optimization problems consist of several mutually dependent subproblems. MOO problems can be defined as follows: Let $\Pi \subset \mathbf{R}^n$ be the feasible range of solutions in a given search space, where $\mathbf{R}$ denotes the real valued numbers. The objective function

$$f_c : \Pi \longrightarrow \mathbf{R}_+^n$$

assigns a cost to each objective of a solution, where

$$f_c(x) = (f_{c_1}(x), f_{c_2}(x), \cdots, f_{c_n}(x)), x \in \Pi.$$

[1] Informations on *lexicographic* sorting of vectors can be found in standard mathematical literature.

To compare several multi-objective solutions *superior points* in the search space can be determined. In the following we restrict to minimization problems. The relation *dominate* is defined as proposed in [8]. Let $x, y \in \mathbf{R}_+^n$ be the costs of two different solutions. x *dominates* y ($x <_d y$), if no component of y is smaller than any component of x. More formally:

Definition 1.

$$\begin{aligned} x <_d y \Longleftrightarrow & \ (\quad \exists i : f_{c_i}(x) < f_{c_i}(y)) \\ & \wedge(\ \forall j \neq i : f_{c_j}(x) \leq f_{c_j}(y)) \end{aligned}$$

$$x \leq_d y \Longleftrightarrow \quad (\forall i : f_{c_i}(x) \leq f_{c_i}(y))$$

A non-dominated solution is called a *Pareto-optimal* solution. $\leq_d$ defines the set of all Pareto-optimal solutions, called the *Pareto-set*, and additionally $\leq_d$ is a partial order. All elements $x \in \Pi$ in the Pareto-set are equal or *not comparable.* Usually, all points in this set are of interest for the decision maker or designer.

3 The Model

Let $\Pi \subset \mathbf{R}^n$ be the set of all feasible solutions. The objective function $f_c : \Pi \longrightarrow \mathbf{R}_+^n$ assigns a cost to each $x \in \Pi$ as defined above. To classify solutions in *satisfiability classes* we define the relation *favor* ($\prec_f$):

Definition 2. Let $x, y \in \mathbf{R}_+^n$ and let $\prec_f$ be a relation on $\mathbf{R}^n$ which is defined as follows:

$$\begin{aligned} x \prec_f y \Longleftrightarrow \exists i, j : j < i : & \mid \{x_k < y_k | 1 \leq k \leq n\} \mid = i \\ & \wedge \mid \{x_l > y_l | 1 \leq l \leq n\} \mid = j \end{aligned}$$

Using Definition 2 we are able to compare elements $x, y \in \Pi$ pairwise more precisely. x is *favored* to y ($x \prec_f y$) iff i ($i \leq n$) components of x are smaller than the corresponding components of y and only j ($j < i$) components of y are smaller than the corresponding components of x.

We use a graph representation for the relation, where each element is a node and "preferences" are given by edges. Relation $\prec_f$ is not a partial order, because it is not transitive, as can be seen as follows:

Example 1. Consider some solution vectors from $\mathbf{R}^3$:

(8,7,1) (1,9,6) (7,0,9) (1,1,2) (2,1,1)

The relation graph of $\prec_f$ is given in Figure 1a). Vectors (1,1,2) and (2,1,1) are preferred to all other vectors, but they are not comparable. The remaining three vectors (8,7,1), (1,9,6), and (7,0,9) are pairwise comparable. But as can be seen in the relation graph they describe a "cycle". Thus relation $\prec_f$ is not transitive.

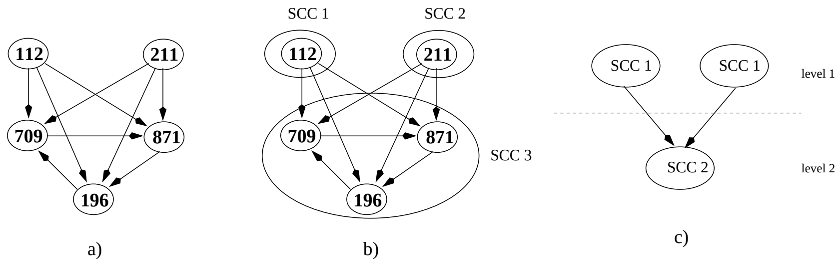

Fig. 1. Relation graph

To get some more insight in the structure of our model we briefly focus on the meaning of the cycles in the relation graph: We rank elements equally which are included in a cycle, because no element is superior to all the others. Elements that describe a cycle are denoted as not comparable. The determination of the *Strongly Connected Components* (SCC) of the relationgraph groups all elements which are not comparable in one SCC. The SCCs are computed by a DFS-based linear time graph algorithm [2]. The transitive closure is computed. A directed graph G_{SC} is constructed by replacing each SCC in G by one node representing this SCC. Thus, all cycles in G are eliminated.

In the following, we denote the relation that is represented by G_{SC} by $\prec_{sat}$. Relation $\prec_{sat}$ is asymmetric and transitive. It is not a partial order, since $\prec_{sat}$ is not reflexive. Nevertheless, the relation properties antisymmetry and transitivity are sufficient for our purposes. Elements are successfully ranked by $\prec_{sat}$. For each SCC in G_{SC} we define a *Satisfiability Class* (*SC*). Level sorting of the nodes in G_{SC} determines the ranking of the SCCs; each level contains at least one SCC. Then each level corresponds to a *SC*.

Example 2. In Figure 1b) SCC 1, SCC 2, and SCC 3 are illustrated. As can be seen the elements of SCC 1 and SCC 2 are superior to the elements of SCC 3. Figure 1c) shows the relation graph G_{SC} after level sorting. Level 1 corresponds to *SC* 1 and level 2 corresponds to *SC* 2.

Our model has several advantages over "classical" approaches.

- Relation $\prec_{sat}$ can also handle infeasible solutions. An infeasible component of the solution is considered as the worst possible value.
- Dependent on the population the model dynamically adapts the relation graph that performs the comparison of the fitness function. In each generation the relation graph is recomputed and the ranking is updated. (This is done totally automatic and no user interaction is required.) Thus the granularity of the *SCs* is dynamically adapted to present conditions.
- If the structure of the search space changes during the EA run these changes are directly included in the relation that is updated online.
- No problem specific knowledge is needed for the choice of the weights of the fitness function, like in the case of weighted sum.

– Due to the efficient representation based on graphs the runtimes are very low.
– For our method neither determination of weights (for weighted sum) is needed nor the values of the objectives have to be scaled. Only a ranking of the solutions is given by the relation $\prec_{sat}$.
– Handling of priorities of all or some objectives and of infeasible solutions is fully supported.

3.1 Application in Evolutionary Algorithms

The relation $\prec_{sat}$ describes a ranking of solution elements in the search space Π. Using $\prec_{sat}$ the individuals of a population are ordered. Then, for selection processes, a single fitness value *sf* is assigned to each individual. Elements of one *SC* have the same fitness value, i.e. the elements of SC 1 have the single fitness *sf*= 1. Doing so, the *tournament selection* as selection method is suggested, since *sf* does not take the "distances" of the solutions' fitness into account.

4 A Case Study: Heuristic Learning

In [5] a new approach to minimization of *Binary Decision Diagrams* (BDDs), the state-of-the-art data structure in VLSI CAD, has been proposed. They are very sensitive to the chosen variable ordering, i.e. the size may vary from linear to exponential. An EA has been used to learn heuristics for BDD minimization. The heuristics are given by sequences of basic operations, i.e. local search operations. Multiple optimization criteria have been considered in parallel in [5]. The evaluation was done by a weighted sum, where the parameters have been finetuned "by hand", i.e. many experiments had to be performed.

The model proposed in this paper has been applied to the optimization problem introduced above. Experiments are performed and a comparison to [5] is given. Many experiments were needed to obtain the high quality results in [5], since the weights of the weighted sum had to be determined. In the new approach no finetuning is needed and in only one program run the same optimization results are obtained.

Furthermore, the efficiency of this approach is demonstrated by a comparison to the method of *nondominated sorting* (NDOM) [11]. By our experiments it turns out that the new model for MOO is superior to NDOM.

4.1 The Learning Model

To make the paper self-contained the basic learning model is briefly reviewed [4]. It is assumed that the problem to be solved has the following property: A non empty set of optimization procedures can be applied to a given (non-optimal) solution in order to further improve its quality. (These procedures are called *Basic Optimization Modules* (BOMs).) Each heuristic is a sequence of BOMs. The length of the sequence depends on the problem instances that are considered.

The goal of the approach is to determine a good (or even optimal) sequence of BOMs such that the overall results obtained by the heuristic are improved.

The set of BOMs defines the set H of all possible heuristics that are applicable to the problem to be solved in the given environment. H may include problem specific heuristics, local search operators but can also include some random operators.

The individuals of our EA make use of multi-valued strings. The sequences of different length are modeled by a variable size representation of the individuals.

To each BOM $h \in H$ we associate a *cost function cost*: $H \rightarrow \mathbf{R}$. *cost* estimates the resources that are needed for a heuristic, e.g. execution time of the heuristics. The quality fitness measures the quality of the heuristic that is applied to a given example.

For more details about the learning model see also [4, 5], where the complete model, using a weighted sum for measuring the fitness, has been presented.

4.2 Binary Decision Diagrams

As well-known each Boolean function $f : \mathbf{B}^n \rightarrow \mathbf{B}$ can be represented by a *Binary Decision Diagram* (BDD), i.e. a directed acyclic graph where a Shannon decomposition is carried out in each node.

We make use of reduced and ordered BDDs [1]. For briefness they are called OBDDs in the following.

Example 3 shows the importance of good variable orderings for OBDDs:

Example 3. In Figure 2 the OBDDs of the function $f = x_1x_2 + x_3x_4 + x_5x_6$ with variable orderings $x_1x_2x_3x_4x_5x_6$ and $x_1x_3x_5x_2x_4x_6$ are illustrated. As can be seen the choice of the variable ordering largely influences the size of the OBDDs. The left (right) outgoing edge of each node x_i denotes $f_{x_i=1}$ ($f_{x_i=0}$).

We now consider the following problem that will be solved using EAs:

> *How can we develop a good heuristic to determine variable orderings for an OBDD representing a given Boolean function f such that the number of nodes in the OBDD is minimized?*

Notice we do *not* optimize OBDDs by EAs. Instead we optimize the heuristic that is applied to OBDD minimization.

Dynamic Variable Ordering The algorithms that are used as BOMs in the EA are based on dynamic variable ordering. *Sifting* (S) is a local search operation for variable ordering of OBDDs which allows hill climbing. *Siftlight* (L) is a restricted form of sifting where hill climbing is not allowed. The third BOM is called *inversion* (I) which inverts the variable ordering of an OBDD. For more details see [5].

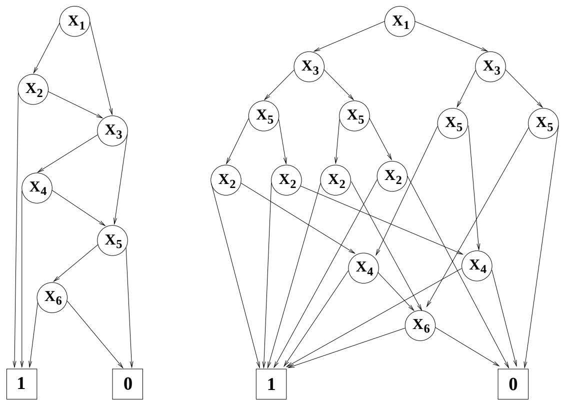

Fig. 2. OBDDs of $f = x_1x_2 + x_3x_4 + x_5x_6$

4.3 Evolutionary Algorithm

Representation In our application we use a multi-valued encoding, such that the problem can easily be formulated. Each position in a string represents an application of a BOM. Thus, a string represents a sequence of BOMs which corresponds to a heuristic. The size of the string has an upper limit of size l_{max} which is given by the designer and limits the maximum running time of the heuristic.

In the following we consider three-valued vectors: S (L, I) represents sifting (siftlight, inversion).

Objective Function As an *objective function* that measures the *fitness* of each element we apply the heuristics to k benchmark functions in a *training set* $T = \{b_1, \cdots, b_k\}$. The quality of an individual is calculated by constructing the OBDD and counting the number of nodes for each $b_i, 1 \leq i \leq k$. Additionally, the execution time (measured in CPU seconds) that is used for the application of the newly generated heuristic is minimized. Then, the objective function is a $(k+1)$-tuple and it is given by:

$$f_c(T) = (\#nodes(b_1), \cdots, \#nodes(b_k), \sum_{i=1}^{k} time(b_i)),$$

where $\#nodes(b_i), 1 \leq i \leq k$, denotes the number of OBDD nodes that are needed to represent function b_i. $time(b_i), 1 \leq i \leq k$ is the execution time of the newly created heuristic applied to function b_i.

The choice of the fitness function largely influences the optimization procedure. It also possible to chose a fitness function as a tuple of length $2 \cdot k$ by considering each $time(b_i)$ separately, instead of using the sum. By our choice the EA focuses more on quality of the result than on the runtime needed.

solutions		OBDD size				
	add6	addm4	cm85a	m181	risc	$\sum$time
dom 1.	28	181	27	60	65	0.17
2.	28	163	27	59	65	0.33
3.	68	163	35	60	66	0.17
4.	68	163	35	54	70	0.18
5.	28	236	37	60	65	0.16
6.	28	175	27	60	65	0.28
7.	68	163	35	60	66	0.17
8.	54	163	35	56	75	0.17
9.	68	177	38	63	76	0.12
10.	28	165	36	61	66	0.28
11.	68	163	35	54	65	0.21
favor 1.	28	163	27	54	65	0.29
2.	28	163	27	60	65	0.26
exact	28	163	27	54	65	*

Table 1. Training set 1

Algorithm

1. The initial population $\mathcal{P}$ is randomly generated.
2. Then $\frac{\mathcal{P}}{2}$ elements are generated by the genetic operators *reproduction* and *crossover*. The newly created elements are then mutated.
3. The offsprings are evaluated. Then the new population is sorted using relation $\prec_f$.
4. If no improvement is obtained for 50 generations the algorithm stops.

For more details of the algorithm see also [5].

4.4 Experimental Results

In this section we present results of experiments that were carried out on a *SUN Ultra 1* workstation. All running times are given in CPU seconds. We performed our experiments using examples of Boolean functions that are taken from LGSynth91 [12].

In a first series of experiments we compare the model proposed in this paper to the method of nondominated sorting [8]. The results are summarized in Table 1. The EA is applied to a training set of size five that is used to develop heuristics for OBDD minimization. The size of the population is set to 20. In the first row of Table 1 the names of the five benchmark functions and the execution time are given. The next block of rows show the best solutions resulting from nondominated sorting (*dom*). All these solutions are not comparable with respect to relation *dominate*, i.e., this set of solutions is the Pareto-set. As can be seen no solution hits the minimum OBDD size for each function (see

	training set 2	
	dom	*favor*
# solutions	18	2
# SCs	3	2
time EA	4h 17m s	3h 50m

Table 2. Comparison of *dom* and *favor*

row exact). Analogously, the best solutions of the EA using the new model of ranking MOO solutions can be seen in the next two rows (*favor*). In this case the EA comes up with only two concurring solutions. Both are superior to the solutions from *dom*. Notice that solution 1 even hits the minimum OBDD size in each case. Each component of solution *favor* 2 is even better or equal than the corresponding components from some solutions in *dom*, e.g. solution *dom* 10. The granularity of the search space partition of relation $\prec_f$ is clearly better in comparison to *dominate*. Only two instead of 11 solutions are ranked in the same level. Additionally, the overall quality of the EA is improved by relation $\prec_f$ as can be seen by the quality of solutions *favor* 1 and *favor* 2.

In a next series of experiments a larger training set consisting of 8 functions of varying sizes has been considered to give some more insight in the effect of using our approach. The results are summarized in Table 2.

For training set 2 the number of solutions in the final solution set is shown. Again it can be seen in row *# solutions* that the method of nondominated sorting gives 18 best solutions in the Pareto-set. Using the technique proposed in this paper again the differentiation of the elements is better, i.e. only 2 best elements, and thus the convergence of the EA is much stronger. To show the effect of our method we apply the concept of SCs to the 18 elements resulting from nondominated sorting. As can be seen in row *# SCs* the number of elements can be drastically reduced. (Obviously, for *favor* the number does not change.) But still both solution elements from experiment *favor* are better than the best solution from *dom*, if the comparison is based on relation $\prec_f$. Clearly, this results in significant reductions in runtime of the EA. In [3] it has been observed that most of the runtime of EAs applied in VLSI CAD is needed for evaluation of the fitness function. Using the model presented in this paper, the number of evaluations can be significantly reduced by applying operations on graphs. These operations have polynomial runtime and can be carried out efficiently. For our example the runtime is given in the last row for both approaches. As can be seen *favor* is even faster and also gives better results.

5 Conclusion

A new model for *Multi-Objective Optimization* has been proposed. We could overcome the limitations of classical EA approaches that often require several runs to determine good starting parameters. Furthermore, our model gives very robust results since the number of parameters is reduced without reducing the quality of the result. Only "non-promising" candidates (for which can be guaranteed that they are not optimal and better individuals already exist) are not considered. An additional advantage is the fact that components with different measures have not to be scaled. This is done automatically comparing elements with relation $\prec_f$. This pruning of the search space even may result in significant speed-ups in EAs and simplifies the handling of the algorithms.

A comparison to the approach of *nondominated sorting* has shown that relation $\prec_f$ performs a ranking of finer granularity. In the case study it turned out that the overall quality of the EA increased using the new model for MOO based on satisfiability classes and furthermore speeds up the computation since less evaluations of the fitness function are needed. This is especially useful in cases where the evaluation of the fitness function dominates the overall execution time, as it is often the case in applications in VLSI CAD.

References

1. R.E. Bryant. Graph - based algorithms for Boolean function manipulation. *IEEE Trans. on Comp.*, 35(8):677–691, 1986.
2. T.H. Cormen, C.E. Leierson, and R.C. Rivest. *Introduction to Algorithms.* MIT Press, McGraw-Hill Book Company, 1990.
3. R. Drechsler. *Evolutionary Algorithms for VLSI CAD.* Kluwer Academic Publisher, 1998.
4. R. Drechsler and B. Becker. Learning heuristics by genetic algorithms. In *ASP Design Automation Conf.*, pages 349–352, 1995.
5. R. Drechsler, N. Göckel, and B. Becker. Learning heuristics for OBDD minimization by evolutionary algorithms. In *Parallel Problem Solving from Nature, LNCS 1141*, pages 730–739, 1996.
6. H. Esbensen. Defining solution set quality. Technical report, UCB/ERL M96/1, University of Berkeley, 1996.
7. H. Esbensen and E.S. Kuh. EXPLORER: an interactive floorplaner for design space exploration. In *European Design Automation Conf.*, pages 356–361, 1996.
8. D.E. Goldberg. *Genetic Algorithms in Search, Optimization & Machine Learning.* Addision-Wesley Publisher Company, Inc., 1989.
9. J. Horn, N. Nafpliotis, and D. Goldberg. A niched pareto genetic algorithm for multiobjective optimization. In *Int'l Conference on Evolutionary Computation*, 1994.
10. Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs.* Springer-Verlag, 1994.
11. N. Srinivas and K. Deb. Multiobjective optimization using nondominated sorting in genetic algorithms. *Evolutionary Computation*, 2(3):221–248, 1995.
12. S. Yang. Logic synthesis and optimization benchmarks user guide. Technical Report 1/95, Microelectronic Center of North Carolina, Jan. 1991.