Open access • Proceedings Article • DOI:10.1109/CEC.2007.4424570

SAT-decoding in evolutionary algorithms for discrete constrained optimization problems — Source link

Martin Lukasiewycz, Michael Glass, Christian Haubelt, Jürgen Teich

Institutions: University of Erlangen-Nuremberg

Published on: 01 Sep 2007 - Congress on Evolutionary Computation

Topics: Combinatorial optimization, Optimization problem, Metaheuristic, Evolutionary computation and Evolutionary algorithm

This is the author's version of the work. The definitive work was published in *Proceedings of IEEE Congress on Evolutionary Computation (CEC 2007), pp. 935-942, 2007.* The work is supported in part by the German Science Foundation (DFG), SFB 694.

SAT-Decoding in Evolutionary Algorithms for Discrete Constrained Optimization Problems

Martin Lukasiewycz, Michael Glaß, Christian Haubelt, and Jürgen Teich
Department of Computer Science 12, Hardware-Software-Co-Design
University of Erlangen-Nuremberg, Germany
{martin.lukasiewycz, glass, haubelt, teich}@cs.fau.de

***Abstract*— For complex optimization problems, several population-based heuristics like Multi-Objective Evolutionary Algorithms have been developed. These algorithms are aiming to deliver sufficiently good solutions in an acceptable time. However, for discrete problems that are restricted by several constraints it is mostly a hard problem to even find a single feasible solution. In these cases, the optimization heuristics typically perform poorly as they mainly focus on searching feasible solutions rather than optimizing the objectives.**

In this paper, we propose a novel methodology to obtain feasible solutions from constrained discrete problems in population-based optimization heuristics. At this juncture, the constraints have to be converted into the Propositional Satisfiability Problem (SAT). Obtaining a feasible solution is done by the DPLL algorithm which is the core of most modern SAT solvers. It is shown in detail how this methodology is implemented in Multi-objective Evolutionary Algorithms. The SAT solver is used to obtain feasible solutions from the genetic encoded information on arbitrarily hard solvable problems where common methods like penalty functions or repair strategies are failing. Handmade test cases are used to compare various configurations of the SAT solver. On an industrial example, the proposed methodology is compared to common strategies which are used to obtain feasible solutions.

I. Introduction

Evolutionary Algorithms (EAs) are heuristics that are based on the principles of biological evolution and are used for both decision and optimization problems. This paper focuses on discrete constrained optimization problems regardless of the number and kind of objective functions. EAs which can handle multiple objective functions are called *Multi-Objective Evolutionary Algorithms* (MOEAs). Instead of decision problems, where one feasible solution has to be found, an optimization with EAs is twofold: (1) The algorithm has to make an effort to stay in the valid search space to obtain feasible solutions whereas (2) the objectives have to be optimized. The first task is only of interest if the given problem is constrained and, as a matter of fact, most real-world applications are hard-constrained [1]. There are various strategies for obtaining feasible solutions [2], [3] such as by using penalty functions, the preservation of feasible solutions, prioritizing of feasible over infeasible solutions, and repairing strategies. These methods are widely and successfully used in many optimization problems, but tend to fail, if the search space is discrete and hard-constrained. If the constraints are linear or linearizable, searching for a feasible solution in discrete constrained optimization problems can be formulated as an *Integer Linear Program* (ILP) with an empty objective function. Hence, the ILP will return one feasible solution of the optimization problem. A special class of ILPs are *0-1 Integer Linear Programs* (0-1 ILP) where the decision variables are reduced to the binary values 0 and 1. These problems are also termed *Pseudo-Boolean* (PB) [4], whereas general ILPs can be described as 0-1 ILPs by a binary encoding. In fact, 0-1 ILPs are mentioned in Karp's 21 *NP-complete* problems [5]. That means obtaining feasible solutions in a discrete constrained optimization can even be NP-complete. However, specialized PB solvers are widely used to solve these problems efficiently [6].

One well known discrete constrained problem is the *Set Cover Problem*, cf. Figure 1. Given is a universe of elements $\mathcal{U}$ and a set $\mathcal{S}$ of subsets of $\mathcal{U}$. The task in the optimization version of the *Set Cover Problem* is to find a minimal set $\mathcal{C} \subseteq \mathcal{S}$ such that the union of $\mathcal{C}$ equals $\mathcal{U}$. This problem is known to be NP-hard [5]. Nevertheless, obtaining a single feasible solution is a trivial problem, as iteratively adding subsets of $\mathcal{S}$ to $\mathcal{C}$ until these sets are covering all elements can be used as a simple repair algorithm. If the problem is extended by the condition that the sets in $\mathcal{C}$ have to be pairwise disjoint, this repair algorithm can no longer be applied. In fact, obtaining a single feasible solution is an NP-complete problem known as the *Exact Cover Problem* [5]. Hence, many optimization procedures, in particular population-based heuristics, that are incorporating the common strategies like local repair or penalty functions, will rather be busy to find feasible solutions than optimizing the objective function. Note that this drawback will be apparent in any optimization problem where obtaining a feasible solution is a hard solvable problem.

As a remedy, we propose a new decoding strategy for EAs based on modern *SAT solvers* [7], which are programs and algorithms that are actually used to solve the *Propositional Satisfiability Problem* (SAT) [8] in conjunctive normal form. For this purpose, the used SAT solver is adapted to obtain feasible solutions whereas the search process of the solver and, thus, the resulting solutions are varied by the EA. Contrary to solving the SAT problem with EAs is a well researched topic [9], using SAT solvers in EAs is a novel approach. The proposed methodology is generally applicable to any discrete constrained optimization problem where the feasibility of a

Supported in part by the German Science Foundation (DFG), SFB 694

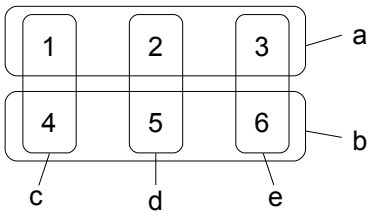

Fig. 1. Minimal Set Cover Problem or Minimal Exact Cover Problem, respectively. Universe containing the elements $\mathcal{U} = \{1, 2, 3, 4, 5, 6\}$ with $\mathcal{S} = \{a = \{1, 2, 3\}, b = \{4, 5, 6\}, c = \{1, 4\}, d = \{2, 5\}, e = \{3, 6\}\}$.

solution can be described by a set of linear PB constraints or any other form which is translatable into SAT. These are, for instance, Pseudo-Boolean optimization problems, many graph-based optimization problems for example the problem of *System-level Synthesis* [10], covering problems as they are used for logic minimization and technology mapping [11], and many more. As this methodology only deals with the search space, the objective space is not affected. Hence, the number of objectives is not limited, with nonlinear objective functions being allowed explicitly.

The remainder of the paper is outlined as follows: Section II gives a short introduction of related work and Section III of the preliminary work as well as the problem description. In Section IV the general SAT-decoding is described and additional enhancements are proposed in Section V. Experimental results will be discussed in Section VI before we conclude the paper in Section VII.

II. Related Work

While the optimization of the objective function f is one target of an EA, it has to be ensured that the found solutions are fulfilling the given constraints at the same time. Common strategies for obtaining feasible solutions in EAs on constrained problems are outlined in [2], [3] and are summarized in the following.

A common method is the usage of penalty functions. Depending on the number of unsatisfied constraints, a penalty value is added to the objective functions and, thus, the fitness of the individual is deteriorated. Thereby, feasible solutions and solutions with low penalty values are prioritized automatically in the optimization process. Prioritizing feasible solutions over infeasible by a clear distinction is a current strategy, too. In some cases, the feasibility of solutions can be preserved by eliminating decision variables or a neat structure of the chromosomes. Furthermore, there exist repair strategies which are using information from a chromosome to fix an infeasible solution. These repair strategies are also termed decoders and can be complete, thus, always obtain feasible solutions. On the other hand, due to the problem trait, they can be local strategies that are not able to guarantee a repair to feasible solutions. Moreover, these methods can be combined in various ways to hybrid strategies.

Another approach for a decoder-based EA is a *mapping strategy*. The information from the chromosome is mapped to the feasible search space. In [12], a mapping for continuous constrained optimization problems is done between an n-dimensional cube and the feasible search space X_f. However, this decoder is using a problem-specific mapping and, moreover, can not be applied to discrete constrained problems. In [13], we have proposed a specific SAT-based decoding approach for the problem of System-level Synthesis which was strongly connected to the given problem domain and of limited generality. This paper overcomes these limitations and proposes a general SAT-based decoding where a generic chromosome structure and mapping scheme will obtain feasible solutions. Therefore, this decoding is easily applicable to any optimization problem where obtaining a single feasible solution is of arbitrary hardness, and can be formulated as a 0-1 ILP or Satisfiability Problem, respectively.

III. Preliminary

A. Problem Formulation

This paper focuses on optimization problems with a various number of objectives and a set of linear discrete constraints. The constraints of the problems are restricting the search space such that only a fraction of the search space X results in *feasible* solutions. This feasible search space $X_f \subseteq X$ is containing all solutions that are fulfilling the given constraints. We will restrict the search space to vectors of binary variables, which, on the other hand, can represent binary encoded integers. Therefore, the objective function is a mapping from the search space $X = \{0, 1\}^n$ to the objective space $Y = \mathbb{R}^m$ given by $f : \{0, 1\}^n \rightarrow \mathbb{R}^m$, in which for an n-dimensional binary decision vector an m-dimensional objective vector is determined. Without loss of generality, we assume that all objectives have to be minimized.

Now, the goal in a multi-objective optimization problem is to find the set of Pareto-optimal solutions $X_p \subseteq X_f$ or the Pareto-optimal front $Y_p = \{f(x) | x \in X_p\}$, respectively. A solution $x_p \in X_p$ is said to be Pareto-optimal if its objective vector $f(x_p)$ is not dominated by any other objective vector $f(x)$ with $x \in X_f$, cf. Definition 1.

Definition 1 (Pareto dominance (cf. [14])) *For any two objective vectors a and b,*

$a \succ\succ b$ *(a strictly dominates b)*	*if* $\forall i : a_i < b_i$
$a \succ b$ *(a dominates b)*	*if* $\forall i : a_i \leq b_i \wedge \exists j : a_j < b_j$
$a \succeq b$ *(a weakly dominates b)*	*if* $\forall i : a_i \leq b_i$
$a \parallel b$ *(a is incomparable to b)*	*if* $\exists i, j : a_i > b_i \wedge a_j < b_j$.

B. The DPLL Algorithm

The Propositional Satisfiability Problem (SAT) is the first known NP-complete problem [8]. Given is a Boolean formula $f : \{0, 1\}^n \rightarrow \{0, 1\}$ in *conjunctive normal form* (CNF) whereas the question is if it is satisfiable respectively 1 or *true*, respectively, under any input. The main property of a CNF is that it is satisfied if every single clause is satisfied, where a clause is satisfied if at least one of its literals is *true*, cf. Figure 2.

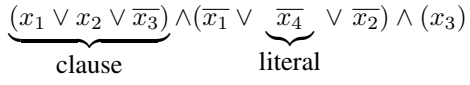

Fig. 2. A Boolean formula in conjunctive normal form

Though SAT is known to be NP-complete and, therefore, hard solvable, there exist algorithms and programs which are aiming to solve the SAT problem efficiently. These SAT solvers are designed and developed with high effort since they became essential in the field of *Electronic Design Automation* [15]. Modern complete SAT solvers [16], [17] are mostly based on the DPLL algorithm [7] and are often performing successfully even on huge instances with thousands of variables and clauses.

The DPLL algorithm is a backtracking algorithm that tries to find a binary input vector that fulfills f. The algorithm starts with completely unassigned variables whereas in an iterative process assignments are done and conflicts are resolved by a backtracking procedure. The algorithm is searching a satisfiable variable assignment until the function is proven to be satisfiable or unsatisfiable. The decision which unassigned variable is chosen and what assigned value, the *phase* 0 or 1, it gets, is called *decision strategy*. If the algorithm recognizes that all variables have an assignment the algorithm stops and f is recognized as satisfiable whereas the current variable assignment is called *witness*. One of the main principles of a SAT solver are the *implications*. An implication is done if a still unsatisfied clause has only one unassigned literal. The variable in this literal gets the corresponding value to satisfy the clause and keeps the CNF satisfiable. In modern SAT solvers, most of the time is spent on the propagation of implications. Concepts like watched literals were proposed to improve this process [16]. Moreover, modern SAT solvers are improved by an enhanced conflict resolution [18], a clause learning scheme [19] and random restarts [16].

IV. SAT-Decoding

In the proposed approach, a SAT solver is used to obtain feasible solutions and, by that, utilize the advantageous concepts of modern SAT solvers.

Figure 3 is illustrating the concept of an optimization heuristic using *SAT-decoding*. Encoding the solutions directly into the chromosome makes it hard to find feasible solutions in many hard constrained problems. Therefore, a clear distinction between the *genotype* and the *phenotype* is made. The genotype, a specific vector of variables $v \in V$, is varied in the chromosome space which is simply bounded. By using the mapping function $g : V \rightarrow X_f$, any genotype v is mapped to a feasible solution $x \in X_f$, the phenotype, in the decision space. Actually, the mapping scheme g is guided by the information of the the chromosome v. Therefore, it must be ensured that the chromosome space and mapping scheme is chosen in a way, such that at least each Pareto-optimal solution can be reached. More formally,

$$\forall x \in X_p \; \exists v \in V : x = g(v).$$

That means an EA is no longer varying the solutions in the decision space and running into many infeasible solutions if the decision space contains only few feasible solutions. Instead of that, the EA is varying the vectors in the chromosome space V whereas these bounds are clearly defined and can simply be preserved. By the mapping scheme g, which is realized by a SAT solver, a chromosome v is mapped to a feasible solutions $x \in X_f$. The mapping process with a SAT solver is referred to as *SAT-decoding*. Evaluating the objectives of a feasible solution is done by the function f as usual.

A. Converting Model into SAT

Preliminary, the given problem, excluding the objective functions, has to be converted into SAT. In particular, each feasible solution of the problem must have a solution in the SAT problem and vice versa. To overcome the limited expressiveness of a CNF, the problem can also be encoded as a Pseudo-Boolean problem with an empty objective function. Hence, each linear PB constraint has to be converted into a set of clauses. In [20] a methodology for that purpose is described in detail. The two-staged process which prevents a resulting exponential number of clauses works as follows: First, each PB constraint is converted into a hardware circuit by using BDDs, Adders, or Sorters. Second, the resulting circuits are converted linearly into a set of clauses by a transformation that introduces additional variables. Alternatively, there exist specialized SAT solvers that beside clauses in the conjunctive normal form also support natively PB constraints [6], [21], [22]. Moreover, there are approaches for converting problem specifications in specialized description languages automatically into SAT [23].

Exemplary for the problems defined in Figure 1, the constraints can be defined as follows: For each set $s \in \mathcal{S}$, a binary variable is introduced by the function $\sigma : \mathcal{S} \rightarrow \{0, 1\}$ indicating whether s is (1) or is not (0) contained in $\mathcal{C}$. The constraints of the *Minimal Set Cover Problem* are then

$$\forall u \in \mathcal{U} : \sum_{s \in \mathcal{S} \wedge u \in s} \sigma(s) > 0,$$

which means that each element of the universe $\mathcal{U}$ has to be in at least one set that is part of $\mathcal{C}$. For the example in Figure 1, these are six constraints:

$$\sigma(a) + \sigma(c) > 0 \quad (1)$$
$$\sigma(a) + \sigma(d) > 0 \quad (2)$$
$$\sigma(a) + \sigma(e) > 0 \quad (3)$$
$$\sigma(b) + \sigma(c) > 0 \quad (4)$$
$$\sigma(b) + \sigma(d) > 0 \quad (5)$$
$$\sigma(b) + \sigma(e) > 0 \quad (6)$$

Accordingly, the constraints of the *Minimal Exact Cover Problem* are

$$\forall u \in \mathcal{U} : \sum_{s \in \mathcal{S} \wedge u \in s} \sigma(s) = 1,$$

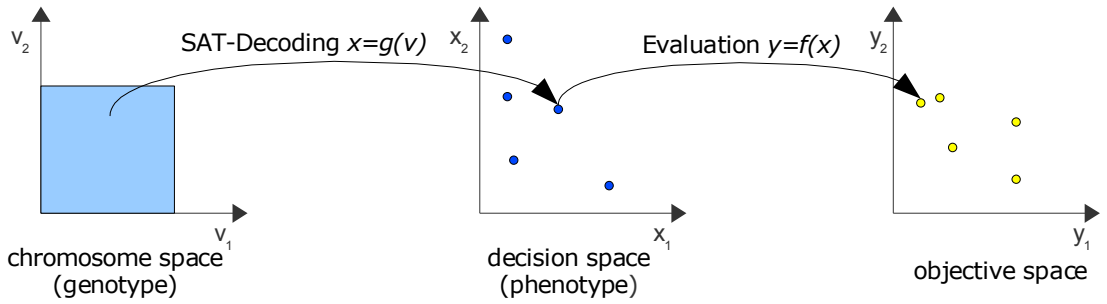

Fig. 3. The SAT-decoding $x = g(v)$ is mapping a vector $v \in V$ from the bounded chromosome space to a feasible solution $x \in X_f$ in the decision space. The evaluation of the objectives is done by the function $y = f(x)$.

where each element of the universe $\mathcal{U}$ has to be in exactly one set which is part of $\mathcal{C}$. For the example in Figure 1, the six constraints are:

$$\sigma(a) + \sigma(c) = 1 \quad (1)$$
$$\sigma(a) + \sigma(d) = 1 \quad (2)$$
$$\sigma(a) + \sigma(e) = 1 \quad (3)$$
$$\sigma(b) + \sigma(c) = 1 \quad (4)$$
$$\sigma(b) + \sigma(d) = 1 \quad (5)$$
$$\sigma(b) + \sigma(e) = 1 \quad (6)$$

B. Search Process

Now, once the given problem is translated into a SAT problem, the mapping function $g : V \rightarrow X_f$ has to be realized such that the search process of the SAT solver is guided by the information in the chromosome $v \in V$. Here, the main task is to cover all feasible solutions of the SAT problem and the underlying problem, respectively. This means, that for each feasible solution $x \in X_f$ there must exist at least one chromosome $v \in V$ such that $x = g(v)$. By covering the whole feasible search space X_f, it is implicitly ensured that the Pareto-optimal solutions $X_p \subseteq X_f$ can be reached.

Many modern SAT solvers are using an *activity*-based decision strategy. Hereby, each variable is tagged by an activity value whereas in the decision strategy, the next assignment takes place for the unassigned variable with the highest activity value. The decision phase is set statically to 0 or 1, respectively, cf. [17]. Each time a variable is involved into a conflict, the activity value of the variable is increased by a so-called *bumping value* which is constantly scaled by a value that is greater or equal to 1. Usually, an initial activity for each variable is calculated at the start of the algorithm depending on the occurrence of the variable in the CNF. It is obvious that an initial activity assignment has a huge impact on the search process and the solution which is found by the SAT solver.

Adapting this process, the chromosome v is now holding (a) the initial *activity* as a real number, and (b) a prioritized *phase* as a binary value for each variable of the underlying problem. Variables that are introduced during the transformation of linear constraints into clauses are not considered, because they are used to discover conflicts within the linear constraints and are automatically set by implications during the search process. The initial activity of the variables that are not part of the chromosome are set to 0. It is clear that all feasible solutions can be reached without any conflict if each variable has the same prioritized phase in the chromosome as the corresponding binary value of the solution. Therefore, the whole feasible decision space is covered. Hence, the requirement that there must exist a mapping to all Pareto-optimal solutions is fulfilled.

	$\sigma(a)$	$\sigma(b)$	$\sigma(c)$	$\sigma(d)$	$\sigma(e)$
activity	0.5	1	0.2	0.7	0.9
phase	0	1	1	0	1

Fig. 4. Chromosome for the example from Figure 1

Exemplary, the search process for the chromosome in Figure 4 and the *Minimal Set Cover Problem* from Figure 1 is as follows:

$$\sigma(b) = 1$$
$$\sigma(e) = 1$$
$$\sigma(d) = 0$$
$$\hookrightarrow \sigma(a) = 1 \text{ (implication constraint 2)}$$
$$\sigma(c) = 1$$

This means for the used chromosome, the SAT-decoding leads to the feasible solution $\mathcal{C} = \{a, b, c, e\}$. Correspondingly for the *Minimal Exact Cover Problem* from Figure 1, the search process is:

$$\sigma(b) = 1$$
$$\hookrightarrow \sigma(c) = 0 \text{ (implication constraint 4)}$$
$$\sigma(d) = 0 \text{ (implication constraint 5)}$$
$$\sigma(e) = 0 \text{ (implication constraint 6)}$$
$$\hookrightarrow \sigma(a) = 1 \text{ (implication constraints 1-3)}$$

In this case, the SAT-decoding finds the feasible solution $\mathcal{C} = \{a, b\}$.

C. Static/Dynamic Decision Strategy

Depending on the bumping value, the decision strategy can be either *static* or *dynamic*. A static decision strategy

is achieved by setting the bumping value to 0 that the scaling value does not affect. That means that the priority of the variable keeps the same throughout the whole search process. At a first glance, a dynamic decision strategy with a bumping value greater than 0 is less strict than a static decision strategy regarding a mapping from the chromosome to the search space. But this holds only for the decision order, not for the prioritized decision phase. Therefore, with a dynamic decision strategy, the decision order is not strictly followed. On the other hand, the found solutions are closer to the information in the chromosome regarding the prioritized decision phase.

This means deciding between a static and a dynamic decision strategy turns out to be a trade-off between the *strictness* of the variable order and the *closeness* to the prioritized decision phase. The requirement that all feasible solutions can be reached still holds for a dynamic decision strategy as any solution can be reached without a conflict. Moreover, one can expect that a dynamic decision strategy is improving the decoding speed on problems where obtaining a single feasible solution is a hard problem. In these cases, the decision order should improve throughout the search process such that variables that are frequently involved in conflicts are ordered to the front and a satisfiable assignment is achieved faster.

V. Enhancements

The requirement that the whole feasible search space has to be covered by the mapping from the chromosome was claimed such that all Pareto-optimal points are covered. But, at this juncture, the mapping scheme can be modified such that a smaller set of feasible solutions is covered whereas this set still contains all Pareto-optimal solutions. The proposed improvements are minimizing the chromosome space V by removing variables and biasing variables with a fixed decision phase in the search process.

These enhancements have to be done with respect to the objective functions and are affecting the mapping scheme in such way that obviously suboptimal solutions are not reached by the decoding. Thereby, a faster convergence to the Pareto-optimal points is reached. On the other hand, there must be a substantial knowledge of the objective functions, with reasoning being possible from a local point of view. In the following, some rules for enhancements are proposed whereas it is described exemplary how they are applied if all objective functions are linear.

1) Removing Indifferent Variables: The first simplification is done by removing variables from the chromosome that do not affect the objective functions in any way. Instead, they get an assignment for the initial activity of 0 and a random prioritized decision phase, which in the used test cases was set statically to 0. More formally, for a chromosome of length n, the variable x_i can be removed if

$$\begin{aligned}&\forall x_1, ..., x_{i-1}, x_{i+1}, ..., x_n \in \{0,1\}\\ &\quad \wedge\, x^0 = (x_1, ..., x_i = 0, ..., x_n) \in X_f\\ &\quad \wedge\, x^1 = (x_1, ..., x_i = 1, ..., x_n) \in X_f :\\ &\quad f(x^0) = f(x^1).\end{aligned}$$

We also say that the variable x_i is *indifferent* to f. In a problem with linear objective functions, any variable that has the coefficient 0 in all objective functions can be removed from the chromosome. Removing these indifferent variables changes the mapping scheme such that it can no longer be guaranteed that all feasible solutions in the search space are reached. On the other hand, at least one of these solutions can be reached, and as the others are equal regarding the objective functions there is no need to reach them either.

2) Phase Biasing on Dominant Variables: The second rule is biasing variables to a specific phase. That means that the chromosome is still holding an initial activity value for these variables, but the decision phase is statically set to 0 or 1, respectively. This modification can be made if a specific phase only has positive influence on the objective functions compared to the decision to the contrary phase. Stated formally, for a chromosome of the length n, a variable x_i is biased to the decision phase p if

$$\begin{aligned}&\exists p \in \{0,1\} :\\ &\forall x_1, ..., x_{i-1}, x_{i+1}, ..., x_n \in \{0,1\}\\ &\quad \wedge\, x^0 = (x_1, ..., x_i = 0, ..., x_n) \in X_f\\ &\quad \wedge\, x^1 = (x_1, ..., x_i = 1, ..., x_n) \in X_f :\\ &\quad f(x^p) \succeq f(x^{\overline{p}}).\end{aligned}$$

We also say that the variable x_i is dominant with the phase p. Any solution $x^{\overline{p}} \in X_f$ is still reachable if the corresponding $x^p \notin X_f$ is not feasible. In that case, x_i will be set to the value $\overline{p}$ during the search process by implications or conflicts which are recognized by the SAT solver. Focusing on linear objective functions, a variable can be biased to 0 if the corresponding coefficients are positive in all objective functions. On the other hand, a biasing to the phase 1 can be done if the coefficients are negative.

3) Special Handling of One-Hot Constraints: Constraints of the form $x_i + ... + x_j = 1$ are appearing frequently in many linear constrained problems. These constraints have the property that exactly one variable from $x_i, ..., x_j$ has to be 1, whereas the remaining variables have to be 0. In fact, a one-hot encoding of the variables $x_i, ..., x_j$ is preserved. Biasing all variables from such a constraint to the same static phase 0 or 1, respectively, will, on the one hand, simplify the chromosome structure and, on the other hand, not restrict the set of covered feasible solutions in the decision space. The coverage is still reached through implications that are done to fulfill the one-hot constraint and the fact that with an initial activity assignment of the variables $x_i, ..., x_j$, any decision order can be achieved.

Biasing the variables of a one-hot constraint to the phase 1 will speed up the decoding process, as many implications are done at the same time. But the number of implications

has a huge influence on the quality of the resulting solution. That means, less implications will lead to a decoding that is paying more attention to the information in the chromosome. Therefore, biasing the variables to a 0 will be slower, but, on the other hand, they are more exact relating to the information of the chromosome.

Additionally, variables can be removed from the search process and their initial activity can be set to 0 under certain circumstances. In the case that $x_i, ..., x_j$ are biased to 0, a variable $x_k \in \{x_i, ..., x_j\}$ can be removed corresponding to the phase biasing rule if x_k is dominant with the phase 0. This means formally that

$$\begin{aligned}&\exists x_k \in \{x_i, ..., x_j\} : \\ &\forall x \in \{x_1, ..., x_n\} \backslash \{x_i, ..., x_j\} \in \{0, 1\} \\ &\quad \wedge\ x^0 = (x_1, ..., x_k = 0, ..., x_n) \in X_f \\ &\quad \wedge\ x^1 = (x_1, ..., x_k = 1, ..., x_n) \in X_f : \\ &\quad f(x^1) \succeq f(x^0).\end{aligned}$$

Accordingly, if $x_i, ..., x_j$ are biased to 1, x_k can be removed if

$$\begin{aligned}&\exists x_k \in \{x_i, ..., x_j\} : \\ &\forall x \in \{x_1, ..., x_n\} \backslash \{x_i, ..., x_j\} \in \{0, 1\} \\ &\quad \wedge\ x^0 = (x_1, ..., x_k = 0, ..., x_n) \in X_f \\ &\quad \wedge\ x^1 = (x_1, ..., x_k = 1, ..., x_n) \in X_f : \\ &\quad f(x^0) \succeq f(x^1).\end{aligned}$$

By removing these variables, the search space can effectively be reduced such that suboptimal solutions are not reached by the mapping scheme.

VI. Experimental Results

The experimental results are based on an implementation of the SAT-decoding using the state-of-the-art SAT solver MiniSat v1.14 [17]. For the linear constraints, the translation scheme into SAT of the PB solver MiniSat+ [20] is used. All test cases were carried out on an Intel Pentium 4 3.20 GHz machine with 1 GB RAM whereas for each handmade test case, 10 instances were created and a representative average was calculated.

The used MOEA was the elitist SPEA2 [24] algorithm. In all test cases, the population size was 100, and each generation 25 offspring were created from 25 parents by using crossover and mutation operators. The initial activity for each variable that is part of the chromosome is a real number value in the bounds $\mathbb{R} \in [0, 1)$. For the binary values, a naive crossover strategy was used, that means, the value of one parent was randomly selected for the offspring. The mutation rate for the binary values was set accordingly to the number of the real $\#r$ and binary $\#b$ values of the chromosome to $p = \frac{1}{\#r+\#b}$. This means that a binary value was flipped with the possibility p. The crossover of the real number values is based on the *Simulated Binary Crossover* operator [25] followed by a mutation by adding a number from the the natural distribution $\mathcal{N}(0, p)$. In the case that a dynamic decision strategy was used, the initial bumping value of the SAT solver was set to p and the scaling factor to $\frac{1}{0.95}$.

$\lvert\mathcal{U}\rvert$	$\lvert\mathcal{S}\rvert$	$\lvert s \in \mathcal{S}\rvert$	Set Cover	Exact Cover
50	250	$[1, 8]$	TC1.1	TC2.1
100	500	$[1, 8]$	TC1.2	TC2.2

TABLE I

Test cases for the minimal weighted problems based on the *Set Cover* and *Exact Cover Problem*. Given is the size of the universe $\mathcal{U}$, the size of the set $\mathcal{S}$, the size of each element in $|\mathcal{S}|$, and the labels of the test cases.

In order to evaluate the quality of the methods, we use the ϵ-dominance [14] criterion. This measurement is used to specify the convergence of multi-objective optimization methods to the front of Pareto-optimal solutions. The ϵ-dominance calculates the relation of a set of solutions A to the set of the Pareto-optimal solutions B, which is approximated by the best solutions found by all methods in all runs.

$$\mathcal{D}_\epsilon(A, B) = \inf_\epsilon \{b \in B \mid \exists a \in A : a \succeq_\epsilon b\}$$

Thus, the ϵ-dominance is the smallest value ϵ with that a set of Pareto-optimal solutions has to be scaled in order to be weakly dominated by the set A. The scaling is normalized in such a way that the value of $\mathcal{D}_\epsilon(A, B)$ is between 1 and 2. Hence, a small value for ϵ is aspired.

A. Handmade Test Cases

The handmade test cases are based on the *Minimal Set Cover Problem* and the *Minimal Exact Cover Problem* which were presented in the Introduction, and are illustrated in Figure 1. Both problems were extended to multi-objective optimization problems by introducing a costs function $w_i : \mathcal{S} \to \mathbb{N}$ for each dimension i. The goal of the n-dimensional problem is to minimize the linear objective functions

$$\forall i \in \{1, ..., n\} : f_i(\mathcal{C}) = \sum_{A \in \mathcal{C}} w_i(A).$$

All test cases were created randomly following Table I, ensuring that there exists at least one feasible solution. In all test cases, there exist three objective functions whereas each random cost function is bounded by $[1, 100]$.

The test case groups TC1.1 and TC1.2 are both mapped by the SAT solver without any conflict. Through implications, conflicts are avoided. Therefore, there is no difference between the dynamic and static decision strategy. Thus, only a basic SAT-decoder and an enhanced SAT-decoder that is implementing the introduced rules for variable reduction and phase biasing are compared. The enhanced SAT-decoding is superior in these test cases as Figure 5 shows. The decoding time for one chromosome is nearly the same in all SAT-decoding schemes with approximately 0.5 milliseconds for TC1.1 and 1 millisecond for TC2.1. In fact, the usage of a SAT solver does not produce an overhead compared to a simple repair strategy.

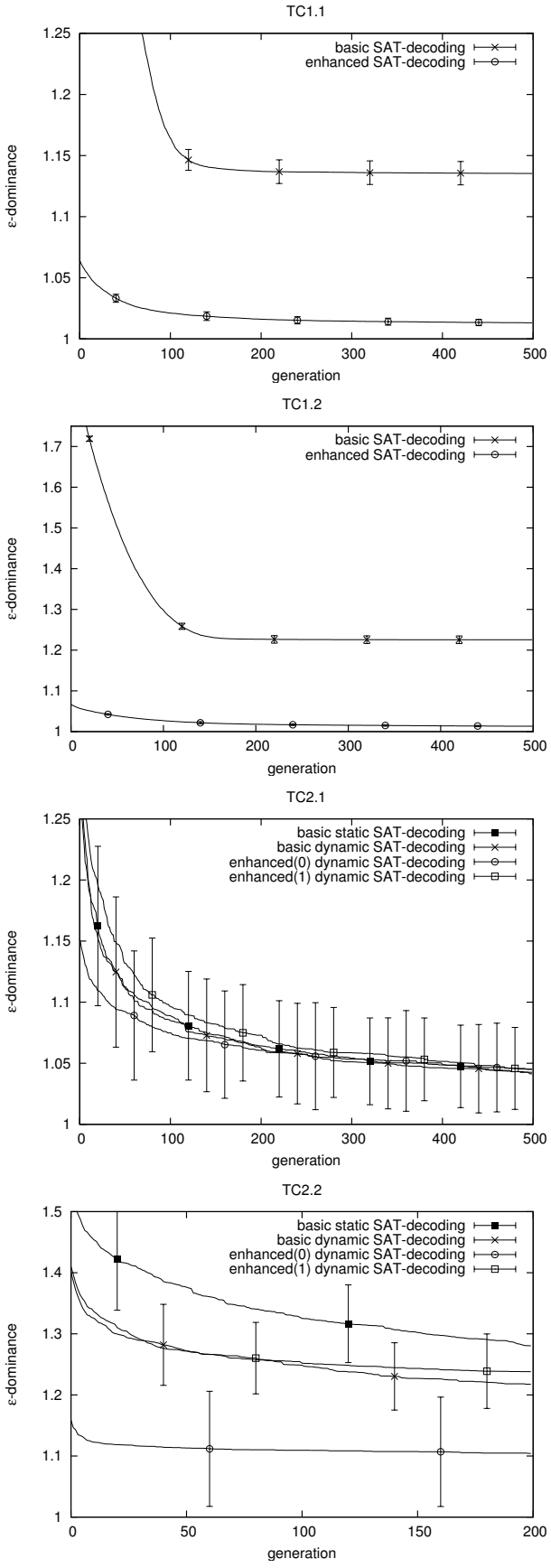

Fig. 5. Results for TC1.1, TC1.2, TC2.1, and TC2.2. The vertical bars indicate the standard deviation.

SAT-decoding	TC2.1	TC2.2
basic static	13.2 ms	290 ms
basic dynamic	3.9 ms	45 ms
enhanced(0) dynamic	22.4 ms	135 ms
enhanced(1) dynamic	2.9 ms	36 ms

TABLE II
TIME GIVEN IN MILLISECONDS PER SAT-DECODING ON THE TC2.1 AND T2.2.

Obtaining feasible solutions for the test cases groups TC2.1 and TC2.2 is an NP-complete problem. A common strategy that is counting the violated constraints as an error objective which has to be minimized, is failing since it does not even find a single feasible solution for any test case within the number of generations stated in Figure 5. Moreover, a repair strategy can not be easily obtained like in TC1.1 and TC1.2. As Figure 5 shows, the basic decoding scheme is performing better with the dynamic decision strategy. Moreover, the dynamic decision strategy is working faster as Table II shows. This is more noticeable on TC2.2 where obtaining feasible solutions is harder than in TC2.1 due to the problem size.

Therefore, the enhanced SAT-decoding schemes were used with a dynamic decision strategy. As all constraints in this problem class are one-hot constraints, the enhanced SAT-decoding was compared on the phase biasing on the one-hot constraints. One can see that biasing the variables with 0 delivers much better solutions than biasing with 1. On the other hand, biasing with 1 leads to a faster decoding scheme as Table II shows. In cases where the evaluation of the objective is a time-consuming procedure, one has to take into account that the decoding time can be just a fraction of the whole run time.

B. Industrial Test Case

The so-called *adaptive light control* (TC3) is an automotive design problem from the area of System-level Synthesis [10]. Obtaining feasible solutions can be formulated directly into SAT by following [26]. Moreover, it was shown that obtaining a feasible solution is in general an NP-complete problem. The problem consists of 234 process, 1103 resources and 1851 mappings edges. This leads to approximately 2^{375} possible solutions. The optimization problem has two objectives, the power as well as the area-consumption.

In TC3, the general SAT-decoding scheme using the introduced enhancements and a negative phase biasing on one-hot constraints has an even better convergence than the specialized SAT-based decoding scheme based on priority lists for System-level Synthesis problems [13]. At this juncture, the decoding time for all methods was nearly the same with 3 milliseconds. Existing methods applying local repair strategies and penalty functions are inferior to the SAT-decoding variants as only a fraction of the found solutions are feasible.

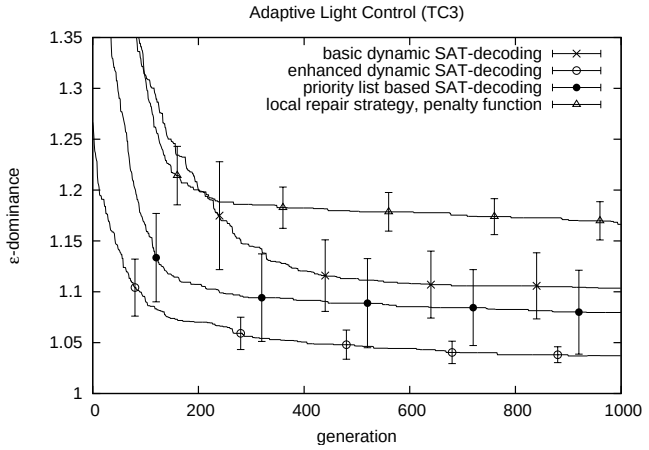

Fig. 6. Results for TC3. The vertical bars indicate the standard deviation.

VII. Conclusions

In this paper we proposed a general methodology for integrating modern SAT solvers into EAs for obtaining feasible solutions on hard-constrained discrete optimization problems. For this purpose, the problem of obtaining a feasible solution has to be converted into the Satisfiability Problem or a set of linear constraints with binary variables, respectively. The task of the EA is no longer to vary the solutions in the decision space and thereby risking to obtain many infeasible solutions on specific hard-constrained problems. In our approach, the EA varies the search process of the SAT solver which is guided by the information stored in the chromosome. The SAT solver, based on the DPLL algorithm, is used as a decoder to map the chromosome to a feasible solution in the decision space.

The experimental results show that the methodology is even applicable on problems where obtaining a feasible solution is an NP-complete problem. On these hard combinatorial problems, common strategies like local repair or penalty functions are failing since they do not find any feasible solution. On the other hand, the SAT-decoding was also tested successfully on simple problems where it creates no noticeable overhead. Therefore, the methodology is applicable and of great benefit on many discrete optimization problems were obtaining feasible solutions is a hard problem.

References

[1] K. Deb, *Optimization for Engineering Design*. Prentice-Hall of India Pvt.Ltd, 1995.

[2] Z. Michalewicz and M. Schoenauer, "Evolutionary algorithms for constrained parameter optimization problems," *Evolutionary Computation*, vol. 4, no. 1, pp. 1–32, 1996.

[3] Z. Michalewicz, K. Deb, M. Schmidt, and T. Stidsen, "Test-case generator for nonlinear continuous parameter optimization techniques," *IEEE Transactions on Evolutionary Computation*, vol. 4, no. 3, pp. 197–215, 2000.

[4] P. Barth, "A Davis-Putnam based enumeration algorithm for linear pseudo-Boolean optimization," Max-Planck-Institut für Informatik, Im Stadtwald, D-66123 Saarbrücken, Germany, Research Report MPI-I-95-2-003, January 1995.

[5] R. M. Karp, "Reducibility among combinatorial problems," in *Complexity of Computer Computations*, R. E. Miller and J. W. Thatcher, Eds. Plenum Press, 1972, pp. 85–103.

[6] F. A. Aloul, A. Ramani, I. L. Markov, and K. A. Sakallah, "Generic ilp versus specialized 0-1 ilp: an update," in *ICCAD '02: Proceedings of the 2002 IEEE/ACM international conference on Computer-aided design*. New York, NY, USA: ACM Press, 2002, pp. 450–457.

[7] M. Davis, G. Logemann, and D. Loveland, "A machine program for theorem-proving," *Commun. ACM*, vol. 5, no. 7, pp. 394–397, 1962.

[8] S. A. Cook, "The complexity of theorem-proving procedures," in *STOC '71: Proceedings of the third annual ACM symposium on Theory of computing*. New York, NY, USA: ACM Press, 1971, pp. 151–158.

[9] J. Gottlieb, E. Marchiori, and C. Rossi, "Evolutionary algorithms for the satisfiability problem," *Evol. Comput.*, vol. 10, no. 1, pp. 35–50, 2002.

[10] J. Teich, T. Blickle, and L. Thiele, "An evolutionary approach to system-level synthesis," in *CODES '97: Proceedings of the 5th International Workshop on Hardware/Software Co-Design*. Washington, DC, USA: IEEE Computer Society, 1997, p. 167.

[11] X. Y. Li, M. F. Stallmann, and F. Brglez, "Effective bounding techniques for solving unate and binate covering problems," in *DAC '05: Proceedings of the 42nd annual conference on Design automation*. New York, NY, USA: ACM Press, 2005, pp. 385–390.

[12] S. Koziel and Z. Michalewicz, "A decoder-based evolutionary algorithm for constrained parameter optimization problems," in *Proceedings of the 5th International Conference on Parallel Problem Solving from Nature*. London, UK: Springer-Verlag, 1998, pp. 231–240.

[13] T. Schlichter, M. Lukasiewycz, C. Haubelt, and J. Teich, "Improving system level design space exploration by incorporating sat-solvers into multi-objective evolutionary algorithms," in *Proceedings of Annual Symposium on VLSI*. Karlsruhe, Germany: IEEE Computer Society, 2006, pp. 309–314.

[14] E. Zitzler, L. Thiele, M. Laumanns, C. M. Fonseca, and V. G. da Fonseca, "Performance assessment of multiobjective optimizers: an analysis and review." *IEEE Trans. Evolutionary Computation*, vol. 7, no. 2, pp. 117–132, 2003.

[15] J. P. Marques-Silva and K. A. Sakallah, "Boolean satisfiability in electronic design automation," in *DAC '00: Proceedings of the 37th conference on Design automation*. New York, NY, USA: ACM Press, 2000, pp. 675–680.

[16] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik, "Chaff: engineering an efficient sat solver," in *DAC '01: Proceedings of the 38th conference on Design automation*. New York, NY, USA: ACM Press, 2001, pp. 530–535.

[17] N. Eén and N. Sörensson, "An extensible sat-solver." in *Conference on Theory and Application of Satisfiability Testing*, 2003, pp. 502–518.

[18] J. P. M. Silva and K. A. Sakallah, "Grasp - a new search algorithm for satisfiability," in *ICCAD '96: Proceedings of the 1996 IEEE/ACM international conference on Computer-aided design*. Washington, DC, USA: IEEE Computer Society, 1996, pp. 220–227.

[19] L. Zhang, C. F. Madigan, M. H. Moskewicz, and S. Malik, "Efficient conflict driven learning in a boolean satisfiability solver," in *Proceedings of the 2001 IEEE/ACM international conference on Computer-aided design*. Piscataway, NJ, USA: IEEE Press, 2001, pp. 279–285.

[20] N. Eén and N. Sörensson, "Translating Pseudo-Boolean Constraints into SAT," *Journal on Satisfiability, Boolean Moelding and Computation*, vol. 2, pp. 1–25, 2006.

[21] H. M. Sheini and K. A. Sakallah, "Pueblo: A modern pseudo-boolean sat solver," in *DATE '05: Proc. of the conf. on Design, Automation and Test in Europe*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 684–685.

[22] D. Chai and A. Kuehlmann, "A fast pseudo-boolean constraint solver," in *DAC '03: Proceedings of the 40th conference on Design automation*. New York, NY, USA: ACM Press, 2003, pp. 830–835.

[23] M. Cadoli and A. Schaerf, "Compiling problem specification into SAT," *Artificial Intelligence*, vol. 162, no. 1-2, pp. 89–120, 2005.

[24] E. Zitzler, M. Laumanns, and L. Thiele, "SPEA2: Improving the Strength Pareto Evolutionary Algorithm," in *EUROGEN 2001. Evolutionary Methods for Design, Optimization and Control with Applications to Industrial Problems*, K. Giannakoglou, D. Tsahalis, J. Periaux, P. Papailou, and T. Fogarty, Eds., Athens, Greece, 2002, pp. 95–100.

[25] K. Deb and R. B. Agrawal, "Simulated binary crossover for continuous search space," *Complex Systems*, vol. 9, pp. 115–148, 1995.

[26] R. Feldmann, C. Haubelt, B. Monien, and J. Teich, "Fault tolerance analysis of distributed reconfigurable systems using sat-based techniques." in *Proc. of the 13th International Conference on Field Programmable Logic and Applications.*, ser. Springer Lecture Notes in Computer Science LNCS, 2003, pp. 478–487.