This is the author's version of the work. The definitive work was published in *Proceedings of the 2008 IEEE Congress on Evolutionary Computation (CEC 2008), pp. 1968-1975, 2008.* The work is supported in part by the German Science Foundation (DFG), SFB 694.

A Feasibility-Preserving Local Search Operator for Constrained Discrete Optimization Problems

Martin Lukasiewycz, Michael Glaß, Christian Haubelt, and Jürgen Teich

***Abstract*— Meta-heuristic optimization approaches are commonly applied to many discrete optimization problems. Many of these optimization approaches are based on a local search operator like, e.g., the mutate or neighbor operator that are used in Evolution Strategies or Simulated Annealing, respectively. However, the straightforward implementations of these operators tend to deliver infeasible solutions in constrained optimization problems leading to a poor convergence. In this paper, a novel scheme for a local search operator for discrete constrained optimization problems is presented. By using a sophisticated methodology incorporating a backtracking-based ILP solver, the local search operator preserves the feasibility also on hard constrained problems. In detail, an implementation of the local serach operator as a feasibility-preserving mutate and neighbor operator is presented. To validate the usability of this approach, scalable discrete constrained testcases are introduced that allow to calculate the expected number of feasible solutions. Thus, the hardness of the testcases can be quantified. Hence, a sound comparison of different optimization methodologies is presented.**

I. Introduction

MANY meta-heuristic optimization approaches rely on a unary *local search operator*. In the domain of *Simulated Annealing* [1] and *Tabu Search* [2] this operator is known as the *neighbor* operator. A straightforward implementation of a neighbor operator on discrete problems is, e.g., a single bit-flip. Other optimization approaches like, e.g., *Evolution Strategies* [3] are based on a *mutate* operator. In this case, a common mutate implementation flips a single bit with a given probability. Hence, the common ground of these operators is the variation of a given solution to a specific amount.

For discrete problems with linear constraints it can be NP-complete to find a single feasible solution [4]. In general, for these constrained optimization problems, a local search for a varied feasible solution is NP-complete, too. Thus, it might appear that a straightforward implementation of the local search operator very often results in infeasible solutions. Hence, many approaches in optimization heuristics rely on local repair strategies whereas infeasible solutions are accepted. By using penalty functions, the objectives are deteriorated to guide the search toward the feasible solutions. However, in hard constrained problems, these approaches tend to fail since they are more focused on the search for feasible solutions than on the optimization of the objectives.

In this paper, a novel scheme for a local search operator for linear discrete constrained optimization problems that obtains feasible solutions only is presented. This scheme for local search operators incorporates a *Pseudo-Boolean* (PB) solver [5], [6], a specialized backtracking-based *Integer Linear Program* (ILP) with binary variables. In detail, an implementation of a neighbor operator in the context of a Simulated Annealing and a mutate operator in an Evolution Strategy is presented. To validate this methodology, scalable testcases are introduced. These testcases are scalable in the number of constraints and variables following a defined constructions procedure. This procedure allows a calculation of the average ratio of feasible solutions to the overall number of solutions as well as the expected absolute number of feasible solutions per testcase. Thus, the hardness of a testcase can be quantified and a sound comparison of the proposed local search operators to known methodologies is achieved, respectively.

The remainder of the paper is outlined as follows: Section II gives a short introduction of related work and Section III defines the problem of constrained discrete optimization. In Section IV, the feasibility-preserving local search operator is presented. Section V introduces scalable testcases for constrained discrete optimization. Experimental results are given and discussed in Section VI before the paper is concluded in Section VII.

II. Related Work

For discrete problems, it is common to use heuristic optimization approaches, mainly if the problem is too large to be solved completely or if it has multiple or non-linear objectives. However, the heuristic optimization of constrained discrete problems is still mainly unexplored. General methods for constraint handling are outlined in [7], [8], [9] and are summarized in the following:

A common method is the usage of penalty functions [7]. 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 well known strategy [10], too. For some problems, the feasibility of solutions can be preserved by eliminating decision variables or using specialized operators [11]. Furthermore, there exist local repair strategies which based on additional domain-specific information to fix an infeasible solution [7], [11]. However, repair strategies that are restricted to a local view are not able to guarantee to deliver feasible solutions only.

Hardware/Software Co-Design, Department of Computer Science, University of Erlangen-Nuremberg, Germany, {martin.lukasiewycz, glass, haubelt, teich}@cs.fau.de

Known methods for constrained handling are either restricted to real valued search spaces or tend to fail if the discrete problem is hard constrained since the optimization is more focused on the search for feasible solutions than optimizing the objectives. In [12], a decoding strategy is presented that overcomes these drawbacks, by using a PB solver as a decoder. Thus, only feasible solutions are obtained within the search process of an *Evolutionary Algorithm*. This decoder-based approach can focus on the optimization of the objectives and a good convergence towards the optimal solutions is reached. In this paper, an alternative methodology that uses a PB solver as a local search operator is presented instead. By using a PB solver on discrete constrained problems as a mutate or neighbor operator, only feasible solutions are obtained. Thus, the proposed search processes overcome the drawbacks of traditional straightforward local search operators which often stick to infeasible regions in the search space prohibiting the convergence towards optimal solutions.

To evaluate the performance of the proposed methodology, appropriate testcases are needed. Common benchmarks for discrete constrained optimization with meta-heuristic approaches are, e.g., the *0/1 Knapsack Problem* [13] or the *Traveling Salesman Problem* [14]. In fact, obtaining a feasible solution for these problems is trivial. With domain-specific knowledge, it is a common method to apply a straightforward repair algorithm [13] or operators that preserve the feasibility of the solution [14], respectively. In [12] and [15], also problems where obtaining one feasible solution is NP-complete are created. However, these testcases are not scalable in the degree of hardness. On the other hand, there exists a set of handmade and industrial benchmarks for the optimization of Integer Linear Programs with binary variables [16]. Though, these could be easily extended to multiple and also non-linear objectives, none of these testcases gives information about the difficulty to obtain one feasible solution since these benchmarks are not used for meta-heuristic approaches but for complete ILP and PB solvers.

In this paper, a construction procedure for scalable testcases is introduced. By using this construction procedure, the calculation of the average ratio of feasible solutions to the size of the complete search space is enabled. These testcases are scalable in the number of variables and constraints. Thus, a sound comparison of the proposed methodology is obtained.

III. Problem Formulation

For constrained discrete problems the task of meta-heuristic optimization approach is twofold: (1) The objectives have to be optimized whereas (2) all constraints have to be fulfilled. In this paper, all variables are assumed to be binary variables. Thus, the search space is $X = \{0,1\}^n$. All constraints are linear, with a single constraint formulated as

$$a^T x \circ b \tag{1}$$

with $a \in \mathbb{Z}^n$, $b \in \mathbb{Z}$ and $\circ \in \{<, \leq, =, \geq, >\}$. Thus, by introducing a set of constraints, the feasible search space $X_f \subseteq X$ is defined as the set of all solutions x that fulfill all constraints. Though the demand that all constraints have to be linear with binary variables sounds rather restrictive, a linearization of products of binary variables is allowed by introducing additional variables and constraints [1]. Moreover, integer variables can be encoded binary.

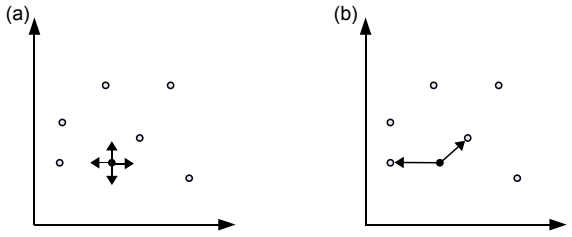

Fig. 1. Illustration of different local search operators in the search space of a constrained problem. The circles are feasible solutions and the arrows show one possible local search operation. (a) illustrates a straightforward local search operator that results in infeasible solutions only. (b) is a feasibility-preserving local search operator.

Given the definition of the feasible search space X_f and the objective function f, the *Constrained Combinatorial Optimization* is defined as the following multi-objective optimization problem:

Definition 1 (Constrained Combinatorial Optimization)

optimize $f(x)$
subject to:
$x \in X_f$

The objective function f consists of multiple functions including also non-linear calculations. In a single-objective optimization, the set of feasible solutions is totally ordered, whereas in multi-objective optimization problems, the feasible set is only partially ordered and, thus, there is generally not only one global optimum, but a set of *Pareto solutions*. A Pareto-optimal solution is not worse or equal in all objectives than any other feasible solution in the design space [17].

In hard constrained problems the set of feasible solutions X_f is significantly smaller than X and the search space contains a high number of infeasible solutions, respectively. If there exists no domain specific repair algorithm or feasibility-preserving operator, a straightforward local search tends to obtain infeasible solutions mainly. This problem is illustrated in Fig. 1(a). For optimization problems like the *Minimal Hamiltonian Circuit* and the optimization version of the *Exact Cover Problem* [4] as well as the industrial application of *Design Space Exploration* [18] it is a NP-complete problem to obtain a single feasible solution and can be formulated as a set of linear constraints with binary variables. In this paper, a procedure for a construction of testcases where obtaining a single feasible solution is NP-complete is presented. Moreover, this construction procedure allowes to formalize the hardness of this problem as the

[1] linearization substitution rule: $x_1 \cdot x_2 \leftrightarrow x_3$ with $x_1 - x_3 \geq 0 \wedge x_2 - x_3 \wedge x_1 + x_2 - x_3 < 2$

Algorithm 1 DPLL backtracking algorithm

```
1: while true do
2:     branch(ρ, σ)
3:     if CONFLICT then
4:         backtrack()
5:     else if SATISFIED then
6:         return x
7:     end if
8: end while
```

expected number of feasible solutions or the average ratio of feasible solution.

With a decreasing ratio of feasible to infeasible solutions, a straighforward local search operator obtains a growing number of infeasible solutions. A meta-heuristic approach that uses a naive operator on this class of prolems tends to obtain mainly infeasible solutions. Therefore, these approaches are more focused on the search for feasible solutions than on the optimization of the objectives.

Thus, a local search operator that only delivers feasible solutions is desired. Such an operator is illustrated in Fig. 1(b). The illustrated approach preserves the feasibility of solutions regardless of the degree of hardness of the problem. Moreover, this methodology is not domain-specific and can be applied to any problem where obtaining one feasible solution can be encoded into binary linear constraints.

IV. Feasibility-preserving Local Search Operator

This section presents a scheme for a feasibility-preserving local search operator for linear constrained discrete problems. This scheme incorporates a *Pseudo-Boolean* (PB) solver [5], [6]. A PB solver uses a backtracking algorithm to solve Integer Linear Programs with an empty objective function and binary variables. The introduced scheme is used to implement a feasibility-preserving *neighbor* and *mutate* operator. Moreover, these operators can be used to effectively reduce the search space within the search process since these operators recognize if some variables have to be assigned a specific value to obtain a feasible solution.

A. PB Solver

The task of a PB solver is to find an $\mathbf{x} \in \{0,1\}^n$ that satisfies a set of linear constraints. In fact, this NP-complete problem [4] is an ILP with binary variables and an empty objective function that can be solved by a common ILP solver. However, due to the boolean nature, the specialized PB solvers tend to outrun common ILP solvers on these problems [19]. These PB solvers are extended *SAT* solvers that are actually used to solve the *Satisfiability problem* and are based on a backtracking strategy. This strategy is known as the DPLL algorithm [20] and is outlined in Algorithm 1.

The algorithm efficiently searches for a solution $\mathbf{x}$ that fulfills all constraints, cf. [5], [6]: Starting with completely unassigned variables, the operation $branch(\rho, \sigma)$ chooses an unassigned variable and assigns it a value (line 2). The branching rule which variable is chosen and value is assigned is called *decision strategy*. Decision strategies are often guided by two vectors $\rho \in \mathbb{R}^n$ and $\sigma \in \{0,1\}^n$. Unassigned variables $\mathbf{x}_i$ with the highest value ρ_i are prioritized and are set to the value σ_i. The configuration of the vectors ρ and σ is called *branching strategy* since it determines the backtracking branching process. After each variable assignment, possible conflicts are recognized (line 3). That means, if any constraint is not satisfiable anymore, the backtracking is triggered (line 4), i.e., variable assignments are reverted. In the case that all variables have an assignment and there exists no conflict (line 5), a feasible solution $\mathbf{x}$ is found and returned (line 6).

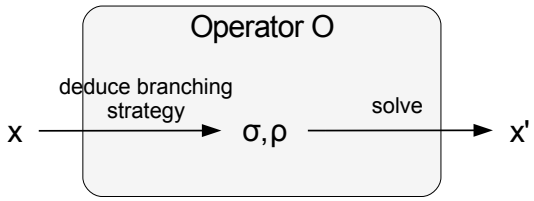

Fig. 2. PB solver based local search operator. Given is the solution $x \in X_f$. Following x, a branching strategy ρ, σ for a PB solver is deduced. With this branching strategy, the PB solver finds an feasible $x' \in X_f$ automatically.

B. Feasibility-preserving Operator Scheme

The general task of a feasibility-preserving local search operator is: For a given solution $x \in X_f$ to find a feasible solution $x' \in X_f$. This local search operator $O : X_f \rightarrow X_f$ is illustrated in Fig. 2.

The function O is twofold: First, a branching strategy σ, ρ is deduced from the solution x. Second, the PB solver generates a feasible solution x' using the deduced branching strategy.

In fact, the deduced branching strategy has a huge impact on the found solution. For a given σ and two feasible solutions $x', x'' \in X_f$ it holds: The solution x' is more similar to σ than x'' if

$$|\{i|x'_i = \sigma_i\}| > |\{i|x''_i = \sigma_i\}|. \tag{2}$$

Now, given a σ and random priorities ρ, the probability that x' is found by a backtracking strategy in a PB solver is higher than the probability that x'' is found. This is due to the fact that with Equation (2) it holds

$$|\{i|x'_i \neq \sigma_i\}| < |\{i|x''_i \neq \sigma_i\}|. \tag{3}$$

Thus, with randomly assigned priorities, x'' is excluded with a higher probability early in the backtracking search process compared to x'. Hence, x' is reached with a higher probability than x''. Moreover, in a special case where for all $x''_i = \sigma_i$ holds that $x'_i = \sigma_i$, the solution x'' is not reachable at all due to the presence of x'.

Thus, the backtracking strategy tends to find an x' that is more similar to σ than other feasible solutions. This holds under the assumption that the priorities are random values. Thus, the vector σ allows to influence the found solution of a PB solver based local search operator, i.e., the similarity

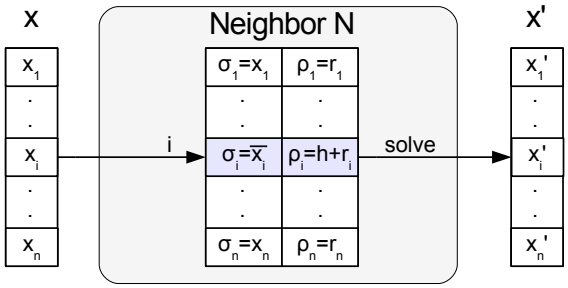

Fig. 3. Feasibility-preserving Neighbor Operator. One bit i is forced to flip with a high priority.

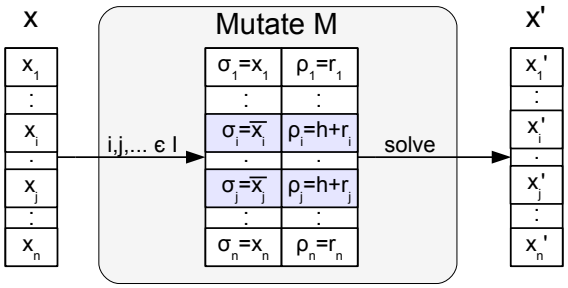

Fig. 4. Feasibility-preserving Mutate Operator. A certain amount of bits I are prioritized to flip. This set I deduced by the mutation rate.

of x and x' can be influenced by the way σ is deduced from x. This is utilized by the operators that are introduced in the following.

C. Feasibility-preserving Neighbor Operator

The most common optimization heuristics that are using a neighbor operator are Simulated Annealing and Tabu Search. Given a solution $x \in X$, the task of the neighbor operator is to find a neighbor $x' \in X$. In a constrained optimization, it is aspired that both solutions are feasible. With the scheme defined in the previous section, the proposed neighbor operator $N : X_f \rightarrow X_f$ is realized by a PB solver that preserves the feasibility of the solutions. A good neighbor operator fulfills two conditions:

1) The neighbor is never equal to the original solution, i.e., $x' \neq x$.
2) The difference between the solutions x and x' is aspired to be as small as possible.

The novel feasibility-preserving neighbor operator N for constrained discrete optimization problems is illustrated in Fig. 3. In the deduced branching strategy, at least one bit flip is forced by randomly selecting one $i \in \{1, ..., n\}$ and setting the corresponding σ_i to $\overline{x_i}$ (the negated value of x_i). The priority of the i-th bit is increased by a large number h $(max_i r_i + 1)$ to ensure that the PB solver flips the i-th bit. The remaining σ values are inherited from x such that the branching strategy tends to find a solution x' that is to a large extend similar to x. Thus, the priorities within the branching strategy have to be randomly selected from a uniform distribution, i.e., each ρ value is a random number $r \in \mathbb{R}$. This random priority assigment is done to fulfill the assumption from the previous section such that a near neighbor is found.

In the backtracking search process, first the i-th variable is set to $\overline{x_i}$. In the following, the PB solver tries to find an assignments for the remaining variables that fulfills all constraints. Since the remaining σ is inherited from x, the search tends to find a similar solution. In case the problem is unconstrained, no conflict will occur. Thus, this operator converges to the standard neighbor operator that flips a single bit since x' equals σ.

D. Feasibility-preserving Mutate Operator

Mutate operators are applied to many meta-heuristic optimization methods. The Evolution Strategy is a popular methodology that relies on a mutate operator merely. In fact, a mutate operator has several similarities to the neighbor operator: Given a solution $x \in X$, the task of the mutate operator is to find an $x' \in X$. In a constrained optimization both solutions are aspired to be feasible. Thus, the feasibility-preserving mutate operator $M : X_f \rightarrow X_f$ is implemented following the proposed operator scheme. A good mutate operator fulfills the following condition:

1) The difference between x and x' are of a distinct degree.

The novel feasibility-preserving mutate operator M for discrete constrained optimization problems is illustrated in Fig. 4. In a first step, a set $I \subseteq \{1, ..., n\}$ is selected by adding each element with a probability of the mutation rate. With a growing mutation rate the size of the set I tends to increase. The mutate operator sets for each variable $i \in I$ the corresponding σ_i to $\overline{x_i}$, thus, prioritizing a mutation of these bits. Therefore, the priority ρ_i for the bit $i \in I$ is increased by the large number h $(max_i r_i + 1)$ to ensure the mutation of these variables. Correspondingly, for each bit $i \notin I$ the σ_i is set to x_i. Like in the neighbor operator, the priorities are randomly selected from $\mathbb{R}$.

With a growing size of I, the number of forced bit flips increases and, thus, the resulting solutions x' tends to have a greater difference to x. In fact, the neighbor operator is a special case of the mutation operator with $|I| = 1$. In case the problem is unconstrained, this operator converges to the standard mutate operator that flips a single bit with a given probability.

E. Search Space Reduction

The presented operators always obtain feasible solutions. Under some circumstances, this allows the search space reduction of an optimization heuristic that relies on these operators at runtime. For instance, if the neighbor operator selects the i-th bits to be set to $\overline{x_i}$ with a very high priority, the PB solver is forced to find such a solution. But, if there exists no solution where is the i-th bit is $\overline{x_i}$, the PB solver will return a x' that equals x. On one hand, this violates the condition that the neighbor x' is not allowed to be equal to

x. On the other hand, in the further optimization process the i-th bit can be ignored since there only exists solutions with x_i. Thus, the search space can be reduced efficiently within the optimization process.

A general rule for the reduction of the search space is as following:

Definition 2 (Search Space Reduction Rule) *For a given set of I bits that are prioritized to be flipped and a PB solver based feasibility-preserving operator $O : X_f \rightarrow X_f$, it holds: If the solution $x' = O(x)$ equals the original solution x, the bits in I can be removed from the search space.*

The variables that are removed from the search space are set to the values from the solution x or x', respectively, for the following solutions. For the neighbor operator this rule applies to $I = \{i\}$.

V. Scalable Testcases

This section presents scalable testcases for discrete constrained optimization problems where obtaining a single feasible solution is an NP-complete problem. First, the construction procedure for these testcases that are scalable in the number of variables and constraints are introduced. Moreover, a determination of the hardness of the testcases is presented.

A. Constraint Construction Scheme

The construction of the constraints for the scalable testcases relies on three variables:

- $n \in \mathbb{N}$ - number of binary variables
- $c \in \mathbb{N}$ - number of constraints
- $m \in \mathbb{N}$ with $\frac{c}{m} \in \mathbb{N}$ - occurrence of each variable among the constraints

The constraints of this problem are generated as follows: Exactly c empty sets $S_1, ..., S_c$ of variables are created. Each variable is randomly distributed over these sets such that each variable occurs in exactly m sets. For each set $S_i \in \{S_1, ..., S_c\}$ a constraints of the form

$$\sum_{x \in S_i} x = 1$$

is created. It has to be ensured that each set S is non-empty to allow a feasible solution. Under the assumption $\frac{c}{m} \in \mathbb{N}$ and each variable occurs exactly m times among the constraints, it is obvious that if a solution exists, it has exactly $\frac{c}{m}$ variables set to 1 and $n - \frac{c}{m}$ variables set to 0. Moreover, this problem equals *Exact Cover Problem* that is known to be NP-complete.

The strict construction procedure allows to calculate the expected number of feasible solutions $|X_f|$, i.e., the cardinality of the set X_f, for one testcases (for Proof see Appendix):

Theorem 1

$$E(|X_f|) = \binom{n}{\frac{c}{m}} \cdot \frac{(c-m)!^{\frac{c}{m}}}{c!^{\frac{c}{m}-1}}$$

testcase	n	c	m	$E(\lvert X_f \rvert)$	$p(x \in X_f)$
TC1a	200	80	2	$1.37 \cdot 10^{9}$	$8.54 \cdot 10^{-52}$
TC1b	200	30	3	$8.09 \cdot 10^{4}$	$5.03 \cdot 10^{-56}$
TC2a	300	120	2	$3.15 \cdot 10^{13}$	$1.54 \cdot 10^{-77}$
TC2b	300	60	3	$3.52 \cdot 10^{6}$	$1.72 \cdot 10^{-84}$

TABLE I

Parameters for the testcases: The number of variables c, the number of constraints n, and the number of occurences of each variable within the constraints m. Also given is the number of expected feasible solutions $E(|X_f|)$ and the average ratio of feasible solutions to the size of the search space $p(x \in X_f)$.

Thus, the average ratio of feasible solutions is calculated as following:

$$p(x \in X_f) = \frac{E(|X_f|)}{|X|} = \frac{E(|X_f|)}{2^n}$$

Using this ratio of feasible solutions to the size of the search space allows the quantification of the hardness of the testcase.

B. Objective Function

The objective functions can be selected indepently from the constraints. Here, the objective functions are linear since the focus of this paper is on constraint handling techniques. Each single objective function is

$$f(x) = \sum_{i=1}^{n} r_i \cdot x_i$$

with $r_i \in \mathbb{N}$ the randomly generated in $[0; 100]$. Non-linear objective functions like NK landscapes [21] are applicable, too.

VI. Experimental Results

In this section, experimental results are presented. After an introduction of the used testcases, the proposed neighbor and the proposed mutate operator are analyzed. Afterwards, the performance of these operators using a Simulated Annealing and an Evolution Strategy approach are compared to a state-of-the-art SAT-decoding based Evolutionary Algorithm presented in [12]. The section is concluded with an application of the proposed methodology to a real-world example from the automotive network area.

The experimental results are based on an implementation of the local search operators using the PB solver SAT4J [22]. All test cases were carried out on an Intel Pentium 4 3.20 GHz machine with 1GB RAM.

A. Testcases

Four testcases with 10 instances per testcase were created following Table I. The size of the search space for TC1a and TC1b is 2^{200} and for TC2a and TC2b 2^{300}. For each testcase, a single-objective and a multi-objective (two objective functions) problem is generated.

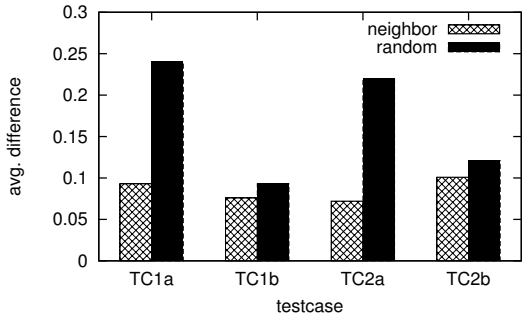

Fig. 5. Comparison of the difference for the neighbor operator as well as two random solutions.

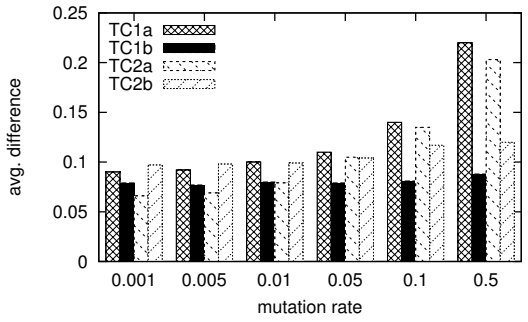

Fig. 6. Influence of the mutation rate on the difference that is induced by the mutate operator.

B. Operator Analysis

For the analysis of the proposed novel operators, the normalized *difference* metric is used. Given two solutions $x, x' \in X$ the difference is:

$$D(x, x') = \frac{\sum_{i=1 \wedge x_i = x'_i}^{n} 1}{n}$$

A small value indicates that x and x' are similar, a big value that these solution are correspondingly different. All results in the following are an average of 1000 operator calls.

In Fig. 5, the difference of two solutions induced by the neighbor operator is stated. To enable a sound comparison, the difference of pairwise random solutions is given, too. In TC1a and TC2a, the difference induced by the neighbor operator is clearly smaller than the random value and, thus, the neighbor operator works as expected. On the other hand, TC1b and TC2b have a sparse neighborhood leading to a comparatively small difference distinction to the pairwise random solutions.

Figure 6 shows the influence of the mutation rate on the difference that is induced by the proposed mutate operator. As expected, all testcases show a growing difference with an increasing mutation rate. Correspondingly to the results of the neighbor operator, this influence is more apparent on the testcases TC1a and TC2a.

C. Optimization Analysis

The novel constraint handling techniques are compared to the Evolutionary Algorithm based SAT-decoding, the state-of-the-art methodology for constrained discrete problems with linear constraints [12]. For the SAT-decoding, the used Evolutionary Algorithm (EA) was the elitist SPEA2 [23] algorithm. The population size was set to 100, and in each generation, 25 offspring were created from 25 parents by using crossover and mutation operators with the operators proposed in [12].

To validate the neighbor operator, we used a standard Simulated Annealing (SA) approach [1]. The cooling schedule is linear, starting from a temperature of 100 and ending at 0. The SA is applied to the single-objective optimization only.

The presented mutate operator is tested with an $(\lambda + \mu)$ Multi-Objective Evolution Strategy (ES) with $\lambda = 25$ and $\mu = 25$ [3]. The mutation mechanism is done by the $1/5$ rule. On all handmade testcases the number of evaluations for all methods is set to 25000.

In order to evaluate the quality of the methods, the ϵ-dominance [17] criterion is used. 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 and, in the following, the reciprocal value is used such that a high value is aspired.

Table II shows the runtimes of the compared methods. In TC1a and TC2a the runtimes of the SA approach with the proposed neighbor operator is significantly lower. This is due the comparatively close neighborhood on these testcases. In contrast, in particularly on TC2b the ES and SA are significantly slower than the EA SAT-decoding approach due to the large search space and sparse neighborhood.

Figure 7 shows the results of the single-objective optimization. The SA approach shows the best convergence on all testcases except TC2b. In TC2b, the EA SAT-decoding is significantly better than the proposed local search operator based approaches. In fact, this can be explained by the

Runtimes[s] of the single-objective optimization:

	TC1a	TC1b	TC2a	TC2b
EA (SAT-decoding)	58.4 (2.9)	89.3 (6.3)	70.2 (4.1)	673 (101)
ES (Mutate Operator)	61.1 (7.0)	108 (9.9)	65.6 (8.5)	2466 (564)
SA (Neighbor Operator)	24.2 (0.4)	96.0 (13.4)	36.6 (1.9)	2723 (511)

Runtimes[s] of the multi-objective optimization:

	TC1a	TC1b	TC2a	TC2b
EA (SAT-decoding)	57.2 (2.2)	86.5(3.5)	72.9 (1.4)	830 (124)
ES (Mutate Operator)	50.9 (0.3)	107 (4.3)	58.3 (4.1)	2424 (569)

TABLE II

OPTIMIZATION RUNTIMES OF THE COMPARED METHODS. GIVEN IS THE AVERAGE RUNTIME SECONDS AND THE STANDARD DEVIATION IN THE BRACKETS.

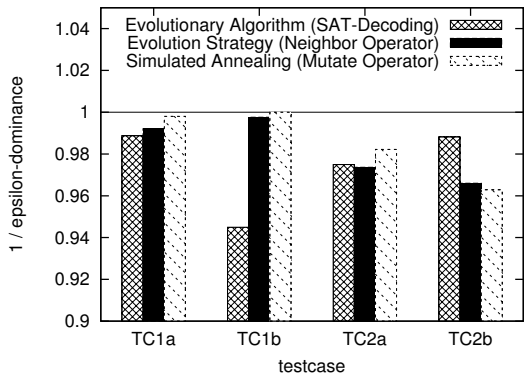

Fig. 7. Results of the single-objective optimization.

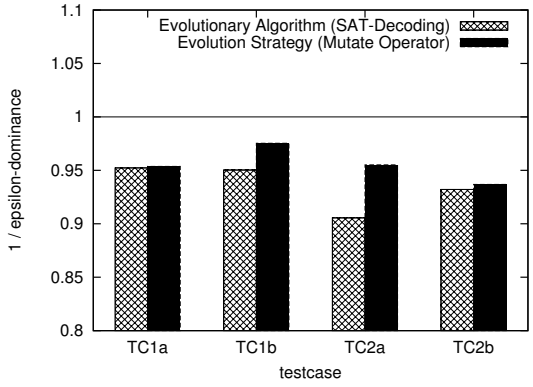

Fig. 8. Results of the multi-objective optimization.

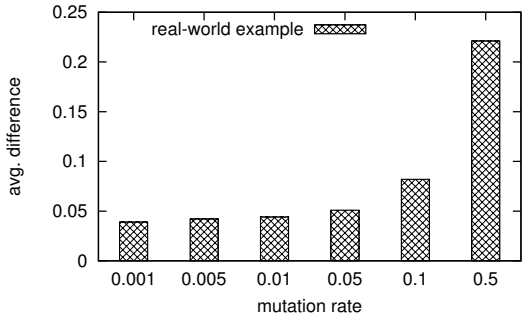

Fig. 9. Influence of the mutation rate on the difference that is induced by the mutate operator.

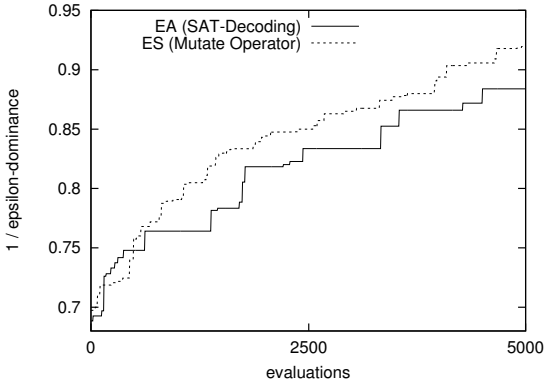

Fig. 10. Results of the real-world example optimization.

sparse neighborhood and, thus, these operators are tending to perform similar to a random search. On the other hand, though TC1b has a sparse neighborhood, too, this search is more successful since the average number of feasible solutions is only about 3-4 times higher than the number of evaluations.

Figure 8 shows the results of the multi-objective optimization. The ES optimization based on the proposed mutate operator is better on all testcases compared to the EA based approach. The clearly improved results on TC2b can be explained by a weak performance of the SAT-decoding that leads to a premature convergence. Compared to that, the ES steadily exploits the search space.

D. Real-World Example

Finally, a real-world example is used to show the practicability of the proposed local search operator. The real-world example is from the automotive network area and consists of 19877 constraints with 7731 binary variables. This example has two objectives, the first is linear (costs) and the second is non-linear (average response time). Thus, the proposed mutate operator within an Evolution Strategy is used and compared to the EA-based SAT-decoding.

First, the mutate operator is analyzed on this problem. Figure 9 shows the results. The mutate operator works as expected on this problem as the average difference increases with a growing mutation rate.

Each method was started 3 times such that a sound average value is calculated. Each optimization run was aborted after 5000 evaluations. The average runtime of the ES was 1260 seconds and 1440 seconds for the EA. Figure 10 shows the corresponding results. Thus, the novel mutate operator in combination with the ES shows a better convergence compared to the EA-based SAT-decoding. Moreover, the novel mutate operator within the ES is faster than the decoding strategy with the EA.

VII. Conclusion

In this paper, a novel scheme for a local search operator for constrained discrete optimization problems as well as scalable testcases that allow to calculate the expected number of feasible solutions are introduced. Based on this scheme, a feasibility-preserving neighbor and mutate operator are presented and applied to a Simulated Annealing and an Evolution Strategy, respectively. Hereby, a branching strategy for a PB solver is deduced from a given feasible solution and, within a backtracking process, a varied feasible solution is found. With a decreasing hardness of the problem, the presented operators converge to a straightforward implementation of the neighbor and mutate operator, respectively. Thus, these operators are applicable regardless of the hardness of the problem that is defined by the constraints.

The proposed scalable testcases allow a sound comparison of the presented methodology to former approaches since their hardness can be estimated by the average ratio of

feasible to infeasible solutions. The presented handmade test-cases validate the competitiveness of the novel local search operators compared to the state-of-the-art SAT-decoding based Evolutionary Algorithm. Moreover, an application of an Evolution Strategy using the proposed mutate operator performed better than the SAT-decoding approach on a real-world example from the automotive network area.

APPENDIX

Proof: In the following, two variables are named *disjunctive* if they do not appear in the same constraints. The probability that one variable is disjunctive to k other variables (that are pairwise disjunctive) calculates to:

$$p(k) = \prod_{i=0}^{m-1} \frac{c - k \cdot m - i}{c - i} \text{ with } 0 \leq k < \frac{c}{m}$$

It is obvious, that each feasible solution $x \in X_f$ contains exactly $\frac{c}{m}$ variables that are 1 and $n - \frac{c}{m}$ variables that are 0. A set that fulfills this requirement is named X_p in the following with $X_f \subseteq X_p \subseteq X$. If $\frac{c}{m}$ randomly chosen variables are disjunctive, these variables can be set to 1 and a feasible solution is found. Thus, the probability for a feasible solution of a random $x \in X_p$ is:

$$p(x \in X_f | x \in X_p) = \prod_{k=0}^{\frac{c}{m}-1} p(k) = \prod_{k=0}^{\frac{c}{m}-1} \prod_{i=0}^{m-1} \frac{c - k \cdot m - i}{c - i}$$

The denominator can be simplified as following:

$$\prod_{k=0}^{\frac{c}{m}-1} \prod_{i=0}^{m-1} \frac{1}{c-i} = \prod_{k=0}^{\frac{c}{m}-1} \frac{1}{\frac{c!}{(c-m)!}} = \frac{1}{\frac{c!}{(c-m)!}^{\frac{c}{m}}}$$

The numerator can be simplified as following:

$$\prod_{k=0}^{\frac{c}{m}-1} \prod_{i=0}^{m-1} c - k \cdot m - i = \prod_{j=0}^{c-1} c - j = c!$$

An by combination:

$$p(x \in X_f | x \in X_p) = \frac{c!}{\frac{c!}{(c-m)!}^{\frac{c}{m}}} = \frac{(c-m)!^{\frac{c}{m}}}{c!^{\frac{c}{m}-1}}$$

With *Bayes' theorem*, the number of expected feasible solutions is:

$$E(|X_f|) = |X_p| \cdot p(x \in X_f | x \in X_p) = \binom{n}{\frac{c}{m}} \cdot \frac{(c-m)!^{\frac{c}{m}}}{c!^{\frac{c}{m}-1}}$$

■

REFERENCES

[1] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science, Number 4598, 13 May 1983*, vol. 220, 4598, pp. 671–680, 1983.

[2] F. Glover, "Future paths for integer programming and links to artificial intelligence," *Computers and Operations Research*, vol. 13, no. 5, pp. 533–549, 1986.

[3] H.-G. Beyer, *The theory of evolution strategies*. New York, NY, USA: Springer-Verlag New York, Inc., 2001.

[4] 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.

[5] 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.

[6] 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.

[7] C. Coello, "Theoretical and numerical constraint handling techniques used with evolutionary algorithms: A survey of the state of the art," *Art. Computer Methods in Applied Mechanics and Engineering*, vol. 191(11-12), pp. 1245–1287, 2002.

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

[9] 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.

[10] K. Deb, "An efficient constraint handling method for genetic algorithms," *Computer Methods in Applied Mechanics and Engineering*, vol. 186, pp. 311–338, 2000.

[11] Z. Michalewicz and G. Nazhiyath, "Genocop iii: A co-evolutionary algorithm for numerical optimization problems with nonlinear constraints," in *Proceedings of the Second IEEE Conference on Evolutionary Computation*, 1995, pp. 647–651.

[12] M. Lukasiewycz, M. Glaß, C. Haubelt, and J. Teich, "SAT-Decoding in Evolutionary Algorithms for Discrete Constrained Optimization Problems," in *Proc. of CEC '07*, 2007, pp. 935–942.

[13] E. Zitzler and L. Thiele, "Multiobjective Evolutionary Algorithms: A Comparative Case Study and the Strength Pareto Approach," *IEEE Transactions on Evolutionary Computation*, vol. 3, no. 4, pp. 257–271, 1999.

[14] P. Larranaga, C. M. H. Kuijpers, R. H. Murga, I. Inza, and S. Dizdarevic, "Genetic algorithms for the travelling salesman problem: A review of representations and operators," *Artif. Intell. Rev.*, vol. 13, no. 2, pp. 129–170, 1999.

[15] K. Xu, F. Boussemart, F. Hemery, and C. Lecoutre, "Random constraint satisfaction: Easy generation of hard (satisfiable) instances," *Artif. Intell.*, vol. 171, no. 8-9, pp. 514–534, 2007.

[16] V. Manquinho and O. Roussel, "Pseudo boolean evaluation 2007," Website, 2006, available online at http://www.cril.univ-artois.fr/PB07/.

[17] 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.

[18] C. Haubelt, J. Teich, R. Feldmann, and B. Monien, "SAT-Based Techniques in System Design," in *Proceedings of Design, Automation and Test in Europe*, N. Wehn and D. Verkest, Eds. Munich, Germany: IEEE Computer Society, Mar. 2003, pp. 1168–1169.

[19] F. A. Aloul, A. Ramani, I. L. Markov, and K. A. Sakallah, "Generic ilp versus specialized 0-1 ilp: an update," in *Proc. of ICCAD '02*, 2002, pp. 450–457.

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

[21] S.-S. Choi, K. Jung, and J. H. Kim, "Phase transition in a random nk landscape model," in *Proc. of GECCO '05*, 2005, pp. 1241–1248.

[22] SAT4J, "A satisfiability library for Java," http://www.sat4j.org/.

[23] 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*, 2002, pp. 95–100.