COMBINING SOLUTIONS OF THE OPTIMUM SATISFIABILITY PROBLEM USING EVOLUTIONARY TUNNELING

Rodrigo Ferreira da Silva[1], **Lars Magnus Hvattum**[2,✉], **Fred Glover**[3]

[1]Universidade Federal de Minas Gerais, Belo Horizonte, Brazil
[2]Faculty of Logistics, Molde University College, Molde, Norway
[3]Meta-Analytics, Inc., Boulder, CO, USA

rfsilva@dcc.ufmg.br[1], hvattum@himolde.no[2,✉], fredwglover@yahoo.com[3]

Abstract

The optimum satisfiability problem involves determining values for Boolean variables to satisfy a Boolean expression, while maximizing the sum of coefficients associated with the variables chosen to be true. Existing literature has identified a tabu search heuristic as the best method to deal with hard instances of the problem. This paper combines the tabu search with a simple evolutionary heuristic based on the idea of tunneling between local optima. When combining a set of solutions, variables with common values in all solutions are identified and fixed. The remaining free variables in the problem may be decomposed into several independent subproblems, so that parts of the solutions combined can be extracted and combined in an improved solution. This solution can be further improved by applying the tabu search in an improvement stage. The value of the new heuristic is demonstrated in extensive computational experiments on both existing and new test instances.

Keywords: *Zero-one integer programming, boolean optimization, metaheuristic, tabu search, adaptive memory programming, recombination operator.*

Received: 28 January 2020
Accepted: 17 June 2020
Published: 24 April 2020

1 Introduction

The *optimum satisfiability problem* (OptSAT) was introduced in [4], where it was referred to as the *Boolean optimization problem*. It subsumes a large class of binary optimization models, including weighted versions of *set covering*, *graph stability*, *set partitioning* and *maximum satisfiability problems*. These problems are all NP-hard, and exact optimization methods may require excessive computational resources to solve large instances. Previous research on OptSAT has therefore focused on developing efficient heuristic solution methods.

Formally, the OptSAT involves maximizing a linear objective function $z = \sum_{j=1}^{n} c_j x_j$ in binary variables, subject to a Boolean equation $\phi(x_1, \ldots, x_n) = \text{false}$. Every Boolean function can be written in disjunctive normal form, allowing ϕ to be expressed in the form $\phi = T_1 \vee \ldots \vee T_m$ where each term T_i is a conjunction of some non-negated variables x_j, $j \in A_i \subset \{1, \ldots, n\}$, and some negated variables $\overline{x_j}$, $j \in B_i \subset \{1, \ldots, n\}$: $T_i = (\bigwedge_{j \in A_i} x_j) \wedge (\bigwedge_{j \in B_i} \overline{x_j})$. This was the original formulation provided in [4].

A reformulation can be considered as observed by Hvattum et al. [11], based on transforming ϕ into conjunctive normal form by applying DeMorgan's law. Then, $\overline{\phi} = \overline{T_1 \vee \ldots \vee T_m} = \overline{T_1} \wedge \ldots \wedge \overline{T_m} = \text{true}$, where $\overline{T_i} = (\bigvee_{j \in A_i} \overline{x_j}) \vee (\bigvee_{j \in B_i} x_j)$. By converting the boolean variables into binary variables, this formulation can be expressed as a mixed integer programming (MIP) problem, with one constraint for each clause $\overline{T_i}$:

$$\max \; z = \sum_{j=1}^{n} c_j x_j$$

$$\sum_{j \in B_i} x_j + \sum_{j \in A_i} (1 - x_j) \geq 1, \qquad i \in \{1, \ldots, m\}$$

$$x_j \in \{0, 1\}, \qquad j \in \{1, \ldots, n\}$$

After moving the constant terms to the right hand side of each constraint and changing to less-than-or-equal constraints, an equivalent formulation is:

$$\max \; z = \sum_{j=1}^{n} c_j x_j$$

$$\sum_{j \in A_i} x_j - \sum_{j \in B_i} x_j \leq |A_i| - 1, \qquad i \in \{1, \ldots, m\} \tag{1}$$

$$x_j \in \{0, 1\}, \qquad j \in \{1, \ldots, n\}$$

Informally, the problem can be regarded as a *satisfiability problem* [2, 5] that is extended to include an objective function. To solve this problem, a greedy heuristic based on pseudo-boolean functions was devised in [4]. The heuristic was shown to perform better than commercially available MIP-solvers at the time. A simple tabu search (adaptive memory search) for solving OptSAT was in turn presented in [11], where new best results were obtained. A more advanced tabu search method was proposed in [12], and was shown to yield

significant computational advantages both in comparison with the preceding heuristic methods and in comparison with the then leading commercial MIP-solvers, Xpress and CPLEX. The same paper also provided results by using two different constructive heuristics, although these were unable to improve on the results from the local search based methods. A system for automatic programming was used in [17] to improve the simple tabu search heuristic from [11]. Using a new set of hard test instances, the system was able to create code that improved on the simple tabu search heuristic. Up to date versions of commercial MIP solvers, including Xpress, CPLEX, and Gurobi, were considered in [13]. The testing performed showed that the best tabu search heuristic still outperformed the commercial solvers, despite the improvements of commercial solvers suggested by several authors since the original experiments [1, 14, 15].

Recently, a deterministic recombination operator that is said to tunnel between local optima was described by Whitley [19]: given two locally optimal solutions, one can remove variables with common assignments and identify separated components. A new solution can then be built by combining the best parts of the two solutions for each separated component. The main contribution of this paper is two-fold. First, we show that the tunneling operator can be applied to the OptSAT problem. Second, we design a simple evolutionary heuristic that combines the use of a tabu search with the tunneling operator, and show through computational experiments how it performs on instances of OptSAT.

The remainder of the paper is structured as follows. In Section 2 the concept of evolutionary tunneling for OptSAT is explored. Section 3 describes both an existing tabu search for OptSAT, as well as a new heuristic combining the tabu search with a recombination operator working on a set of solutions. Results of computational experiments are reported in Section 4, before concluding remarks are given in Section 5.

2 Evolutionary tunneling for OptSAT

The recombination operator suggested in [19], tunneling between local optima, is based on the idea that once common parts of two solutions are disregarded, the remaining parts become separate subproblems that do not interact with each other. We will first examine whether this may happen to instances of OptSAT, or whether these instances will remain connected after fixing the common variable assignments of different solutions.

Consider a set of solutions, and fix the common values. The remaining problem is thus simplified: some variables disappear (those that are fixed) and some constraints can be ignored (those corresponding to clauses where now at least one literal is guaranteed to be true). The question is whether this will split the problem into several separate components, or just leave a reduced problem that is still connected. To see whether a problem can be split into several components, we draw a graph where the nodes represent variables and where edges represent clauses (or constraints in the 0-1 IP formulation) such that there is an edge between any two variables that have literals appearing in the same clause. Let us illustrate using the following artificial example:

$$\begin{aligned} \max\ & z = 7x_1 + 6x_2 + 5x_3 + 4x_4 + 3x_5 + 2x_6 + x_7 \\ & \phi = x_1x_3x_5 \vee x_2x_5x_6 \vee x_4x_6x_7 = 0 \end{aligned}$$

Using formulation (1), this instance can also be written as:

$$\begin{aligned} \max\ & z = 7x_1 + 6x_2 + 5x_3 + 4x_4 + 3x_5 + 2x_6 + x_7 \\ & x_1 + x_3 + x_5 \leq 2 \\ & x_2 + x_5 + x_6 \leq 2 \\ & x_4 + x_6 + x_7 \leq 2 \\ & x_j \in \{0,1\}, j \in \{1,\ldots,7\} \end{aligned}$$

The corresponding graph (without fixing any variables), referred to as a variable interaction graph [19], is shown in Figure 1. As an example, there is no edge between node 1 and node 2, because there is no clause in ϕ containing both variables x_1 and x_2. Let us now assume that we fix $x_2 = 0$. The clause $x_2x_5x_6$ can then be removed, as it will contain at least one false literal. This results in the variable interaction graph shown in Figure 2.

The graph is now disconnected, with two components, indicating that the remaining problem can be solved separately for each component, and then combined into a single solution afterwards. This separation into smaller problems improves the computational efficiency of the method used to solve the remaining problem. If instead of fixing x_2, we had been fixing $x_5 = 1$ or $x_6 = 1$ the graph would also become disconnected.

Now, consider what happens when we fix $x_2 = 1$ instead of $x_2 = 0$. We then only drop the node for x_2, and none of the clauses are guaranteed to be satisfied. The resulting graph, shown in Figure 3 has a bridge between the node for x_5 and the node for x_6, indicating that there is a clause that contains only these two variables, and that without this clause, the problem would decompose as in the previous example. Instead of solving the remaining problem (after fixing $x_2 = 1$) as a single instance, we may try, in turn, to fix either x_5 or x_6 to make sure the bridge-clause is satisfied, and solve the resulting decomposed problem as separate parts. After fixing variables, it may therefore be beneficial to look both for separated components and bridges.

Finally, note that in the original example (without fixing any variables), there are two vertex cuts of size 1, consisting of node 5 and node 6, respectively. These already behave similarly to a bridge: it is possible to fix the value of x_5 (or x_6) and the problem will decompose into separate parts. These can be found as bridges in the corresponding line graph, where nodes correspond

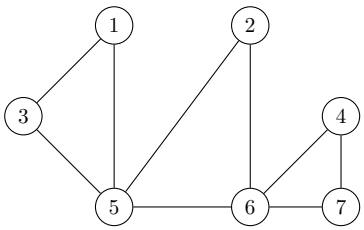

Figure 1: Variable interaction graph for an example with no fixed variables.

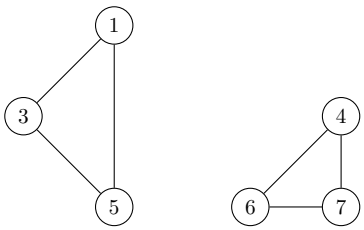

Figure 2: Variable interaction graph for an example with x_2 fixed to 0.

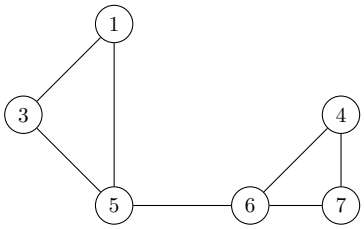

Figure 3: Variable interaction graph for an example with x_2 fixed to 1.

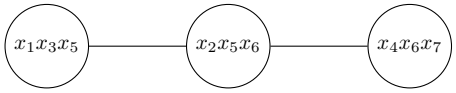

Figure 4: Line graph of an example with no fixed variables.

to clauses and edges to variables that appear in the clauses corresponding to the edge's end points. For example, the line graph of the original example is shown in Figure 4.

Preliminary tests indicated that fixing the common parts of two solutions often results in either a disconnected graph or a graph with bridges. Figure 5 shows the full graph for instance with 100 variables and 400 clauses from [4]. Figure 6 shows the two resulting components after fixing the common parts of a pair of example solutions (one component in blue and the other in red).

3 Heuristics

This section first describes a state-of-the-art tabu search for solving OptSAT. This tabu search is used as a stand-alone method, but also as a part of an evolutionary heuristic that is described next, and in which the recombination operator is based on tunneling.

3.1 Tabu search for OptSAT

For large and difficult instances of the OptSAT, the current best heuristic is a tabu search heuristic with adaptive clause weights (TS-ACW) from [12]. The search is designed to be able to move in infeasible space by penalizing an infeasibility measure. The basic components of the method, which derive from a simple version of tabu search [9], may be summarized as follows.

The starting solution is randomly generated. A move is the flip of a variable, that is, changing a variable from 0 to 1 or vice versa. The search neighborhood is the set of possible flips, which implies that it has a cardinality equal to the number of variables. The move selection is greedy, always choosing the non-tabu move having the highest move evaluation. Tabu tenure is drawn at random from the range $\{10, \ldots, 15\}$, and an aspiration criterion allows a move to be selected in spite of being tabu if it leads to a new best solution.

The evaluation of a move is based on the change in the objective function value and in the change of the number of violated constraints (or clauses). First, the objective function coefficients are normalized to the range $[0, 1\rangle$, and ΔZ_j is defined as the change in the normalized objective function when flipping variable j. The range of ΔZ_j is therefore $\langle -1, +1\rangle$. Second, ΔV_j is defined as the change in the number of violated rows when flipping variable i. The value of ΔV_j is in $\{-m, -m+1, \ldots, m-1, m\}$, but is usually an integer close to 0. The move evaluation function is then defined as $F_j = \Delta V_j + w * \Delta Z_j$, where w is a weight that dynamically balances the two components to keep the search focused around the infeasibility barrier. That is, w is updated every iteration to induce a strategic oscillation around the feasibility boundary.

Adaptive clause weighting is a long-term learning approach that operates in conjunction with the move evaluation function to diversify the search [16]. Considering that some clauses may be harder to satisfy than others, each clause is assigned a weight, W_i. When a clause becomes violated during the search, the associated weight is incremented. This information is then used to modify the move evaluation function by using a weighted version of ΔV_j. Full details of the TS-ACW method are given in [12].

3.2 Evolutionary heuristic with tunneling

To test the idea of tunneling for OptSAT, a simple evolutionary heuristic has been developed. The heuristic starts by creating an initial reference set of $\mu = 400$ random solutions. In each iteration, a subset of T solutions are selected at random from the current reference set, with T being randomly chosen between T_L and T_U. These T solutions are then used to generate

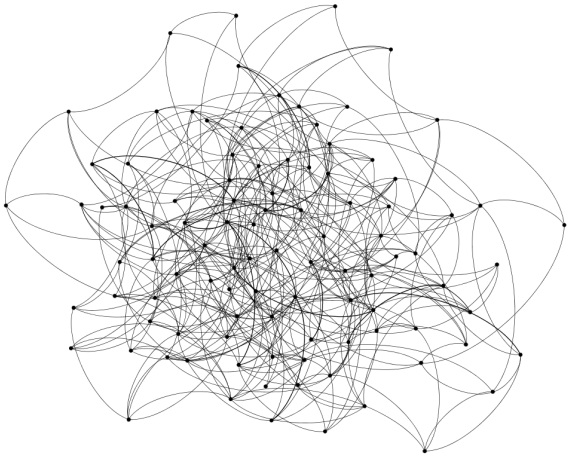

Figure 5: Variable interaction graph of instance qn100m400t2s0c0num0 from [4].

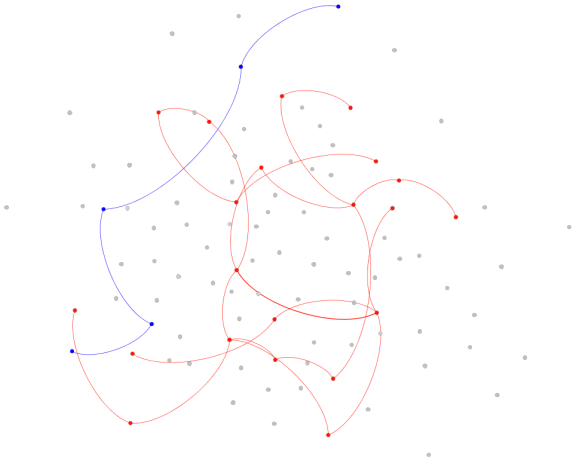

Figure 6: Variable interaction graph after fixing common parts of two selected solutions.

one new solution to be added to the reference set.

Using the tunneling operator, the variables with common assignments in the T solutions are fixed to their respective values. Then, a variable interaction graph is created and disconnected components are identified. For each disconnected component, the best variable assignment from the T solutions is selected. Once each variable has been assigned a value, the resulting solution is improved by running the TS-ACW for a number of $I = 1000$ iterations, while keeping fixed the variables that are common in all the T combined solutions. Since identifying disconnected components leads to an increased computational time, in particular for large problem instances, a version of the search is also considered where we simply fix the common variables and start the TS-ACW from the best of the T solutions. The exploitation of variables with common assignments is related to the concepts of strongly determined and consistent variables [7] and path relinking [6]. Temporarily fixing the values of variables with common values and subjecting the reduced space to more intensive algorithmic exploration is an instance of SIN-space optimization in mixed integer optimization [8].

If the new solution obtained is distinct from those in the reference set, it is added to the set. If the size of the reference set reaches $\mu + \alpha$, with $\alpha = 100$, the set is reduced by eliminating the α worst solutions. If a number of iterations is performed without finding a new best solution, a diversification process is started, where the reference set is reduced to its $\mu/4$ best solutions, and $3\mu/4$ new random solutions are generated and added. Then, I is increased by a factor of $I_F = 20$, to allow a more thorough improvement phase. A time limit is used as the overall stopping criterion for the method.

There are four versions of the evolutionary heuristic considered: In E_2^D and E_{5-15}^D, disconnected components are identified and the improvement phase starts from the best possible solution obtained by combining parts from different solutions, whereas in E_2 and E_{5-15}, the improvement phase is started from the best of the combined solutions after fixing the variables with common assignments in all the combined solutions. In E_2^D and E_2, $T_L = T_U = 2$, whereas in E_{5-15}^D and E_{5-15}, $T_L = 5$ and $T_U = 15$. The variant E_{5-15}^D, with all the parameter settings as described above, was obtained by using the software SMAC [10] to automatically tune the parameters.

4 Computational experiments

This section presents results by the evolutionary heuristic with tunneling for OptSAT instances from the literature, as well as some new, randomly generated, hard instances.

4.1 Results on instances from the literature

Instances for the OptSAT were provided in [4]. When measuring the performance of heuristics on these instances, we follow the convention used in previous research [4, 11, 12, 13]: the obtained objective function value is divided by the value of the best solution found by a commercial MIP solver, CPLEX 6.0, as run with a fixed time limit of up to 600 seconds per instance on a 50 MHz SUN Sparcstation 5 by [4]. Presenting results in percentages, this means that values > 100 correspond to solutions that are better than the ones found by CPLEX, and values < 100 correspond to solutions worse than those found by CPLEX. In the following, we focus on the hardest instances from the literature, as identified by [13].

Table 1 shows the results of five methods: TS-ACW and the four variants of the evolutionary heuristic, each

Table 1: Results on hard instances from the literature.

	TS-ACW		E_2^D		E_{5-15}^D		E_2		E_{5-15}	
	Avg	Sec B	Avg	Sec B	Avg	Sec B	Avg	Sec B	Avg	Sec B
Class 27	111.195	4.8	111.187	5.0	111.185	8.1	111.148	8.2	111.195	4.5
Class 29	101.290	23.9	101.270	36.6	101.270	29.2	101.262	11.5	101.290	25.0
Class 30	100.391	1.5	100.386	7.9	100.381	34.4	100.384	32.8	100.391	1.5
Class 31	101.452	27.4	101.364	4.2	101.398	39.2	101.411	18.1	101.452	27.9
Class 38	100.987	8.1	100.978	29.0	100.980	28.4	100.973	13.8	100.987	8.7
Class 40	101.618	22.8	101.556	10.7	101.581	36.3	101.593	18.8	101.618	24.4
Class 49	101.517	16.4	101.463	12.4	101.479	40.1	101.492	20.8	101.517	16.8
Class 52	109.931	1.9	109.916	6.3	109.873	6.2	109.738	3.5	109.931	1.6
Class 53	113.226	2.3	113.226	2.4	113.163	13.7	112.144	11.5	113.226	2.5
Class 54	114.948	32.6	114.826	21.3	113.324	47.6	112.965	27.4	114.967	33.3
Class 59	107.363	0.6	107.362	4.5	107.353	10.2	107.300	8.5	107.363	0.6
Class 61	101.557	7.0	101.536	35.6	101.533	27.0	101.523	22.0	101.557	7.7
Class 62	100.984	8.6	100.970	11.0	100.958	31.3	100.966	34.1	100.984	8.1
Class 63	103.583	15.4	103.498	3.8	103.536	39.8	103.539	16.0	103.583	17.2
Average	105.003	12.4	104.967	13.6	104.858	28.0	104.746	17.6	105.004	12.8

using a time limit of 60 seconds. The heuristics were all executed on an Intel Core i7 CPU at 3.33GHz, with 24 GB RAM, and using the operating system Ubuntu 12.04. For each method, the average solution quality ("Avg") and the time in seconds to find the best solution ("Sec B") are reported. Results are reported individually for 14 problem classes, each containing five instances. Each instance is run ten times with different random seeds, and the numbers presented are the average values over the fifty runs made within each class.

Although the recombination operator with tunneling succeeds in identifying disconnected components and combining the best components before applying the TS-ACW as an additional improvement method, E_2^D and E_{5-15}^D in general perform worse than E_{5-15}, where the identification of disconnected components is skipped. When fixing relatively fewer variables, in E_{5-15}^D compared to E_2^D, the time to identify and exploit disconnected components increases further, and results become worse. However, the method E_2 performs worse than E_{5-15} because the number of fixed variables becomes very large when only $T = 2$ solutions are combined, in particular as the reference set has started to converge towards good solutions. When that happens, the TS-ACW used as a subsolver can no longer effectively improve the solution with fixed variables.

The results therefore show that, although the identification of disconnected components in OptSAT works as intended, the time overhead is too large compared to simply fixing the common variables and using tabu search to improve the remaining components, without necessarily identifying them as being disconnected. However, despite selecting the most difficult of the instances from existing literature, current hardware enables both TS-ACW and E_{5-15} to find what is likely optimal solutions to almost all the instances. To better judge the difference between E_{5-15} and TS-ACW, harder instances must be used.

4.2 Results on new instances

To gauge the effectiveness of E_{5-15} compared to TS-ACW, new sets of hard test instances are generated, as in [17]. The instance generator used takes as input: the number of variables n, the number of terms m, the number of literals in each term, and the probability that each literal is negated. In the instance generated for the tests reported below, each term is generated with 3 literals, and none of the literals are negated. The objective function coefficients, c_j are randomly generated between n and $2n$. The instance generator ensures that each instance has at least one feasible solution.

Hard 3-SAT instances are obtained when the ratio of clauses to variables is between four and five [18]. We generated six new classes of instances, labelled as class 64 to class 69, with five new instances in each class. Each class has a different combination of values for n and m, with a ratio of $n/m = 5$. The largest existing instances, for class 54 in Table 1, have $n = 1000$ and $m = 10000$, but only 2 literals per clause.

Results are presented in Table 2 for TS-ACW and E_{5-15}. Given that these instances are meant to be hard to solve, the heuristics are run both for 60 seconds and 600 seconds, with 10 runs for each instance. Results are again presented relative to the best solutions obtained by a commercial MIP solver. These were obtained using CPLEX 12.9 [3] with a time limit of four hours (14,400 seconds) on a single thread, on a computer with Windows 7, a 64-bit Intel i5-4690 CPU at 3.5GHz, and 16 GB RAM. For instances of class 64 and class 65, CPLEX would occasionally run out of memory before the time limit was reached. CPLEX is not able to solve any of the new instances to optimality, and the dual bounds at the end of the runs are between 25 % and 40 % above the primal bounds.

The results show that TS-ACW with 60 seconds of running time is able to consistently outperform CPLEX, and also seems better than E_{5-15} with 60 seconds running time on the more difficult instances.

Table 2: Results on new and larger instances.

		TS-ACW (60)		TS-ACW (600)		E_{5-15} (60)		E_{5-15} (600)	
	(n, m)	Avg	Sec B	Avg	Sec B	Avg	Sec B	Avg	Sec B
Class 64	$(500, 2500)$	103.334	26.9	103.462	262.1	103.367	29.9	103.420	113.8
Class 65	$(1000, 5000)$	102.283	17.8	102.505	153.1	102.638	44.7	102.992	288.9
Class 66	$(1500, 7500)$	100.939	13.0	101.190	144.1	100.968	46.4	101.868	350.0
Class 67	$(2000, 10000)$	101.188	7.9	101.399	115.4	100.868	48.2	102.182	444.0
Class 68	$(2500, 12500)$	101.387	6.6	101.595	86.9	100.507	50.9	102.203	475.6
Class 69	$(3000, 15000)$	101.324	5.1	101.413	45.5	100.004	49.7	101.948	462.1
Average		101.742	12.9	101.928	134.5	101.392	45.0	102.436	355.7

However, with 600 seconds of running time, E_{5-15} provides the best results for the larger test instances. This shows that the recombination operator based on variable fixing is useful when solving large instances of OptSAT, but that sufficient running time is required before the positive effect is gained.

In additional experiments, not reported in full, we have observed that the relative performance of CPLEX improves when the randomly generated clauses have some negated literals, in particular on larger instances. On the other hand, the relative performance of the heuristics improves when the generated clauses have a larger number of literals.

5 Concluding remarks

A recombination operator recently proposed for combinatorial optimization problems has the ability to tunnel between local optima [19]. When combining a set of solutions, variables with common assignments are fixed, whereupon the optimization problem may decompose into independent subproblems. A new solution can then be reached by combining the best solutions for each independent subproblem. Here, this idea is tested for the *optimum satisfiability problem* (OptSAT). For that purpose, a simple evolutionary heuristic is developed, starting from a set of random solutions and in each iteration applying the recombination operator to a randomly selected subset of solutions. After the recombination, the solution is improved further by applying a tabu search described in existing literature.

It turns out that for OptSAT, combining a set of solutions does indeed often lead to a decomposition into independent subproblems. However, given the time used to identify and combine solutions for the subproblems, it turns out to be more efficient simply let a tabu search improve the best of the combined solutions after fixing the variables with common values.

To demonstrate the effectiveness of the recombination operator where consistent variables are fixed and a tabu search is used to improve the remaining variables, computational experiments are performed on both existing test instances from the literature and on new instances. These highlight the usefulness of combining solutions, as with longer running times the evolutionary heuristic outperforms the stand-alone tabu search on a set of large instances.

Acknowledgement: The authors wish to thank two anonymous reviewers for their insightful and helpful comments.

References

[1] Bixby, R. Mixed integer programming: It works better than you may think. slide presentation, Gurobi Optimization, 2010.

[2] Cook, S. The complexity of theorem-proving procedures. In *Proceedings of the Third ACM Symposium on Theory of Computing* (1971), pp. 151–158.

[3] CPLEX, 2020. https://www.ibm.com/support/knowledgecenter/en/SSSA5P_12.9.0/.

[4] Davoine, T., Hammer, P., and Vizvári, B. A heuristic for boolean optimization problems. *Journal of Heuristics 9* (2003), 229–247.

[5] Du, D., Gu, J., and Pardalos, P., Eds. *Satisfiability Problem: Theory and Applications*, vol. 35 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*. 1997.

[6] Glover, F. A template for scatter search and path relinking. In *Artificial Evolution*, J.-K. Hao, E. Lutton, E. Ronald, M. Schoenauer, and D. Snyers, Eds., vol. 1363 of *Lecture Notes in Computer Science*. Springer, 1997, pp. 13–54.

[7] Glover, F. *Adaptive Memory Projection Methods for Integer Programming*. Springer US, Boston, MA, 2005, pp. 425–440.

[8] Glover, F. Parametric tabu search for mixed integer programs. *Computers and Operations Research 33* (2006), 2449–2494.

[9] Glover, F., and Laguna, M. *Tabu Search*. Kluwer Academic Publisher, Boston, Dordrecht, London, 1997.

[10] Hutter, F., Hoos, H., and Leyton-Brown, K. Sequential model-based optimization for general algorithm configuration. In *LION-5* (2011), LNCS, pp. 507–523.

[11] Hvattum, L., Løkketangen, A., and Glover, F. Adaptive memory search for boolean optimization problems. *Discrete Applied Mathematics 142* (2004), 99–109.

[12] Hvattum, L., Løkketangen, A., and Glover, F. New heuristics and adaptive memory procedures for boolean optimization problems. In *Integer Programming: Theory and Practice*, J. Karlof, Ed. CRC Press, Boca Raton, FL, 2006, pp. 1–18.

[13] Hvattum, L., Løkketangen, A., and Glover, F. Comparisons of commercial MIP solvers and an adaptive memory (tabu search) procedure for a class of 0–1 integer programming problems. *Algorithmic Operations Research 7* (2012), 13–21.

[14] Koch, T., Achterberg, T., Andersen, E., Bastert, O., Berthold, T., Bixby, R., Danna, E., Gamrath, G., Gleixner, A., Heinz, S., Lodi, A., Mittelmann, H., Ralphs, T., Salvagnin, D., Steffy, D., and Wolter, K. MIPLIB 2010 - mixed integer programming library version 5. *Mathematical Programming Computation 3* (2011), 103–163.

[15] Lodi, A. MIP computation and beyond. Technical Report ARRIVAL-TR-0229, 2008.

[16] Løkketangen, A., and Glover, F. Surrogate constraint analysis — new heuristics and learning schemes for satisfiability problems. In *Satisfiability Problem: Theory and Applications*, D. Du, J. Gu, and P. Pardalos, Eds., vol. 35 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*. 1997.

[17] Løkketangen, A., and Olsson, R. Generating meta-heuristic optimization code using ADATE. *Journal of Heuristics 16* (2010), 911–930.

[18] Selman, B., Mitchell, D., and Levesque, H. Generating hard satisfiability problems. *Artificial Intelligence 81* (1996), 17–29.

[19] Whitley, D. Next generation genetic algorithms: a user's guide and tutorial. In *Handbook of Metaheuristics*, M. Gendreau and J.-Y. Potvin, Eds., 3rd ed., vol. 272 of *International Series in Operations Research & Management Science*. Springer, Switzerland, 2019, pp. 245–274.