Algorithmic Operations Research

Comparisons of Commercial MIP Solvers and an Adaptive Memory (Tabu Search) Procedure for a Class of 0–1 Integer Programming Problems

Lars Magnus Hvattum, Arne Løkketangen and Fred Glover

Volume 7, Number 1, Spring 2012

URI: https://id.erudit.org/iderudit/aor7_1art02

See table of contents

Publisher(s)

Preeminent Academic Facets Inc.

ISSN

1718-3235 (digital)

Explore this journal

Cite this article

Hvattum, L. M., Løkketangen, A. & Glover, F. (2012). 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(1), 13–20.

Article abstract

The Boolean optimization problem *(BOOP) is a highly useful formulation that embraces a variety of* 0-1 integer programming problems, *including weighted versions of covering, partitioning and maximum satisfiability problems. In 2006 an adaptive memory (tabu search) method for BOOP was introduced, and was proved to be effective compared to competing approaches. However, in the intervening years, major advances have taken place in exact solvers for integer programming problems, leading to widely publicized successes by the leading commercial solvers XPRESS, CPLEX and GUROBI. The implicit message is that an alternative methodology for any broad class of IP problems such as BOOPs would now be dominated by the newer versions of these leading solvers. We test this hypothesis by performing new computational experiments comparing the tabu search method for the BOOP class against XPRESS, CPLEX and GUROBI, and documenting improvements provided by the exact codes. The outcomes are somewhat surprising.*

Algorithmic Operations Research Vol.7 (2012) 13–20

Comparisons of Commercial MIP Solvers and an Adaptive Memory (Tabu Search) Procedure for a Class of 0–1 Integer Programming Problems

Lars Magnus Hvattum [a] Arne Løkketangen [b] Fred Glover [c]

[a]Department of Industrial Economics and Technology Management, Norwegian University of Science and Technology, Trondheim, Norway
[b]Molde University College, Molde, Norway
[c]OptTek System Inc., Boulder, CO, USA

Abstract

The Boolean optimization problem *(BOOP) is a highly useful formulation that embraces a variety of* 0-1 integer programming problems*, including weighted versions of covering, partitioning and maximum satisfiability problems. In 2006 an adaptive memory (tabu search) method for BOOP was introduced, and was proved to be effective compared to competing approaches. However, in the intervening years, major advances have taken place in exact solvers for integer programming problems, leading to widely publicized successes by the leading commercial solvers XPRESS, CPLEX and GUROBI. The implicit message is that an alternative methodology for any broad class of IP problems such as BOOPs would now be dominated by the newer versions of these leading solvers. We test this hypothesis by performing new computational experiments comparing the tabu search method for the BOOP class against XPRESS, CPLEX and GUROBI, and documenting improvements provided by the exact codes. The outcomes are somewhat surprising.*

Key words: Zero-one integer programming, Boolean optimization, Commercial solver, Metaheuristic, Tabu search.

1. Introduction

The *Boolean optimization problem* (BOOP) represents 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 in the past exact (provably convergent) optimization methods have encountered severe performance difficulties in these particular applications.

The first computational study of this problem is by Davoine, Hammer and Vizvári [4], employing a greedy heuristic based on pseudo-boolean functions. Hvattum, Løkketangen and Glover [9] in turn introduced simple iterative heuristic methods for solving BOOP. A more advanced method, based on adaptive memory strategies from tabu search (TS) was proposed in [10], and was shown to yield significant computational advantages both in comparison with the preceding heuristic methods and in comparison with the leading commercial solvers XPRESS [15] and CPLEX [3] available at the time.

During the past several years, however, dramatic advances have been reported for the commercial solvers, enabling the three leading solvers, XPRESS, CPLEX and GUROBI [8] to solve integer programming problems far more effectively than in the past. According to [1], the speed-up of solving *mixed integer programming* (MIP) problems is a factor of about 30 from CPLEX 6.5 to CPLEX 11, and a factor of about 80 from CPLEX 6.5 to GUROBI 3.0. More moderate numbers are reported by Lodi [13], indicating a speed-up of 7.47 from CPLEX 6.5 to CPLEX 11, while on another set of instances the percentage of instances solved to optimality increased from 46.5 % for CPLEX 6.5 to 67.1 % for CPLEX 11. Koch et al. [11] report results showing a speed up factor of about 7.5 for XPRESS in versions released between 2003 and 2010.

These impressive performance gains raise the question of whether the modern commercial solvers will in fact dominate alternative methods previously proposed for various integer programming models, particularly in the case of models like BOOP that encompass a significant range of problems. Indeed, Lodi [13] states that, as techniques from metaheuristics have been incorporated in the MIP solvers, they can now be seen as com-

Email: Lars Magnus Hvattum [Lars.M.Hvattum@iot.ntnu.no], Arne Løkketangen [Arne.Lokketangen@himolde.no], Fred Glover [Glover@opttek.com].

petitive heuristic techniques if used in a truncated way. For more details on which improvements have been incorporated into commercial solvers, see the article by Linderoth and Lodi [12].

We examine this question by performing new computational experiments comparing XPRESS, CPLEX, and GUROBI against the tabu search for BOOP. In making these comparisons, we document improvements provided by the exact codes and disclose how different methods perform differently based on the subclass of test cases considered.

The remainder of this paper is organized as follows. Section 2. provides a 0–1 integer programming model for the BOOP and provides a survey of previous work. Detailed computational outcomes are given in Section 3., followed by a summary of results and associated conclusions in Section 4..

2. Problem formulation and search basics

2.1. *Problem formulation*

The BOOP model involves the objective of maximizing a linear objective function $z = \sum_{j=1}^{n} c_j x_j$ in binary variables, subject to a Boolean equation $\phi(x_1, \ldots, x_n) = 0$. 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 T_i is a product of some non-negated variables x_j, $j \in A_i \subset \{1, \ldots, n\}$, and some negated variables $\overline{x_j} = 1 - x_j$, $j \in B_i \subset \{1, \ldots, n\}$: $T_i = (\prod_{j \in A_i} x_j)(\prod_{j \in B_i} \overline{x_j})$. This was the original formulation by Davoine, Hammer and Vizvari [4]. Following [9], we consider a reformulation based on transforming ϕ into conjunctive normal form, giving rise to the following mixed integer programming formulation:

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

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

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

Informally, a BOOP can be regarded as a *satisfiability problem* (SAT) that is extended to include an objective function. For more information on SAT, see for example [2,6].

2.2. *The TS-ACW method: basic search components*

We briefly sketch the basics of the tabu search procedure we employ, called TS-ACW (for Tabu Search with Adaptive Clause Weights) to provide a general overview of its operation. A certain familiarity with basic TS and local search is assumed, and otherwise the reader is referred to [7]. 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, may be summarized as follows.

The starting solution is randomly generated. A move is a flip of a variable, as expressed by $0 \to 1$ or $1 \to 0$. 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, i.e., always chooses a non-tabu move having the highest move evaluation. Tabu tenure is drawn at random from the range $\{10, \ldots, 15\}$. The aspiration criterion allows a move to be selected in spite of being tabu if it leads to a new best solution.

The move evaluation function, F_i has two components. One is the change in the objective function value, and the other is the change in the number of infeasible rows (or clauses). The cost coefficients are normalized to the range $[0, 1]$, and hence the change Δz_j in the objective function lies in the range $[-1, +1]$. The change in the number of infeasible rows, ΔV_j, is usually a small positive or negative integer. These two components are dynamically balanced by a weight, w, to keep the search focused around the infeasibility barrier, thus yielding a move evaluation function given by $F_j = \Delta V_j + w * \Delta z_j$. The adaptive weight w is initially set to 1, and is updated every iteration as follows: If the current solution is feasible, then $w = w + w^{inc}$, otherwise $w = w - w^{dec}$. Computational tests lead to the choices of $w^{inc} = 0.90$ and $w^{dec} = 0.35$. The effect of this adaptation is to induce a strategic oscillation around the feasibility boundary. The move selection and update function thus becomes as shown in Figure 1.

Adaptive clause weighting is a long-term learning approach that operates in conjunction with the move evaluation function to diversify the search, and is a variant of an approach from [14]. 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.

```
1: for each variable, evaluate move
2: if aspiration then
3:     select best move
4: else
5:     select best non-tabu move
6: end if
7: execute selected move
8: set tabu tenure drawn from $\{10, \ldots, 15\}$
9: update $w$ and clause weights
```

Fig. 1. BOOP move selection and update function.

Full details of the TS-ACW method are given in [10].

3. Computational results

As stated earlier, our goal is to get an indication of the increased power of the leading general purpose exact solvers, CPLEX, XPRESS and GUROBI, and to address the question of how the improvements in these methods affect their performance in relation to the tabu search method TS-ACW for solving problems from the BOOP class.

The various versions of the exact solvers are shown in Table 1. To carry out a comparative analysis we must also consider the difference in hardware performance. Previous tests comparing CPLEX and XPRESS to TS-ACW were conducted by running CPLEX and TS-ACW on a 1 Ghz, Intel P3, under Windows 2000, and running XPRESS on a 400 MHzSun UltraSparc under SunOS 5.7. All the new tests were run on the same machine, an Intel Core2 Duo, 2.66 GHz, under Windows XP. These machines have ratings of 769, 154 and 1429 MIPS (whetstones) respectively. Based on this rating, on data from [5], and on the number of iterations performed of TS-ACW per second on the different computers, we have normalized the computational times presented later so as to correspond to running times on the newer Intel Core2 Duo machine. Running times on the UltraSparc computer have thus been divided by a factor of 7.7, and the running times on the Intel P3 computer have been divided by a factor of 3.

The original set of test-cases consisted of 63 major classes, all generated by Davoine, Hammer and Vizvari [4]. All together there were 5400 instances. We have selected instances from classes 23 - 63, as the smaller test problems were considerably easier, and not as suitable for discerning the differences between the various methods. These selected test-sets thus include instances containing from 100 variables and 400 clauses (or constraints) to those containing 1000 variables and 10,000 clauses. Each test set comprises five different instances sharing the same size and structure, bringing the total to 205. As these are optimization problems, obtaining feasibility was not a major issue. Detailed results per problem class are presented after the Acknowledgements. In the following we discuss results at an aggregated level.

Table 1

Solver	Original version	New version
XPRESS	12	20
CPLEX	6.5	12.1
GUROBI	NA	3.0.1

Solver versions.

The TS-ACW metaheuristic has been run on the new computer using a time limit of 60 seconds and ten separate runs per instance. We report average results for the latter experiment. The three exact solvers were tested with a time-limit of 4 hours per instance, with the exception of XPRESS 12 which was allowed to be run for up to 24 hours on some selected instance classes.

Table 2 reports the results relative to the performance of CPLEX 6.0 as run by Davoine, Hammer and Vizvari [4]. Each value in Table 2 is the average over all instances of the ratio of the objective function value obtained by the indicated method to the value obtained by CPLEX 6.0. Since the objective is that of maximization, larger values are better. All numbers referring to results relative to CPLEX 6.0 are given in percentage points.

Table 2

Solver	Original version	New version
XPRESS	99.465	100.865
CPLEX	100.902	102.112
GUROBI	NA	102.190
TS-ACW	NA	102.302

Average results (in ratios) over all 205 instances.

We point out that relatively small differences in ratios amount to somewhat more significant differences in the actual objective function values obtained. In Table 3 we compare the average results of TS-ACW with the results of the new versions of three exact methods, reporting the minimum, average, and maximum difference on the complete set of instances. If instead we were to use the best values from the 10 runs of TS-ACW, the minimum difference would become 0 as none of the exact methods are able to find better solutions on any instance.

To further highlight the differences in performance between the four methods, a pair-wise t-test shows that TS-ACW is significantly better than any other method.

P-values from two-tailed tests with TS-ACW (using average results over 10 runs) against XPRESS, CPLEX, and GUROBI were 0.0002, 0.0064, and 0.0004, respectively. Comparing XPRESS 12 against XPRESS 20 yields a P-value of 0.0073 and comparing CPLEX 6.5 against CPLEX 12.1 yields a P-value of 0.0018. Hence, there is a significant improvement in the results obtained by the new versions of XPRESS and CPLEX. The difference between CPLEX and GUROBI is not significant on a 0.05 level, whereas both are significantly better than XPRESS on any reasonable level.

Table 3

	XPRESS	CPLEX	GUROBI
Minimum	-15.0	-15.0	-15.0
Average	1147.3	184.1	100.6
Maximum	31354.7	10552.7	3339.7

Comparing nominal solution values of TS-ACW with the three exact methods over all the 205 instances. Fractional differences arise due to using the average result over 10 runs for TS-ACW.

Although these results show the performance outcomes when the methods are run to their time limits, they do not indicate how much time each method required to obtain the best solution that it found. In this connection, it should be pointed out that the new version of CPLEX used more time on average than its original version to find the best solution, even when taking into account that the hardware is about three times faster. This reflects the fact that CPLEX finds much better solutions than before, and of course has to work for it. Table 4 shows the time required for each method to obtain the best solutions found.

For 18 of the 41 instance classes, all versions of all commercial solvers were able to prove optimality on all instances. The newest version of CPLEX proved optimality for all instances in 23 classes, but for the remaining 18 classes, only a few instances could be solved to optimality. Table 5 illustrates the running times required by the different methods to find optimal solutions and to prove optimality for the 18 out of 41 classes where all commercial methods prove optimality. Recall that running times have been normalized to correspond to seconds on the newest computer used.

The above results were all obtained using the standard settings of the commercial solvers. However, both CPLEX and XPRESS contain some support for automatically tuning the parameters of the solvers. Three instances, one from each of classes 27, 52, and 61, were selected and the parameter tuning procedures run with ample running time allowed, although limiting the running time for each instance on each parameter set to 300 seconds.

Table 4

Solver	Original version	New version
XPRESS	680.0	2929.8
CPLEX	952.7	2981.5
GUROBI	NA	229.4
TS-ACW	NA	3.8

Average time in seconds to find best solution over all 205 instances.

Table 5

	Seconds to find optimal solution		Seconds to prove optimality	
Solver	Original version	New version	Original version	New version
XPRESS	53.3	155.8	162.8	182.9
CPLEX	178.1	39.3	204.3	58.5
GUROBI	NA	8.3	NA	68.0
TS-ACW	NA	3.8	NA	NA

Results on a subset of 18 classes where all methods find optimal solutions.

For XPRESS the following settings were adjusted by the tuning process: 1) primal simplex was chosen as default instead of automatically deciding the type of LP-solver, 2) the feasibility pump was set to be active always, instead of being turned off, 3) Gomory cuts were turned off except in the root node, and 4) selection of heuristics in the search tree was set to an undocumented value. For CPLEX the following settings were adjusted: 1) a limit on 10 passes to generate Gomory cuts instead of the default which is to let CPLEX decide and 2) the maximum number of candidate variables for generating Gomory fractional cuts is increased from 200 to 10,000. The changed parameters are summarized in Table 6 for XPRESS and Table 7 for CPLEX.

Table 6

Parameter	Initial value	Value after tuning
DEFAULTALG	1	3
FEASIBILITYPUMP	0	1
HEURSEARCHTREESELECT	1	3
TREEGOMCUTS	0	1

Parameter values for XPRESS 20 that were changed after tuning.

The tuned versions of XPRESS and CPLEX were run again on the 70 most challenging instances (those

Table 7

Parameter	Initial value	Value after tuning
gomorycand	200	10,000
gomorypass	0	10

Parameter values for CPLEX 12.1 that were changed after tuning (names of parameters given as when using the interactive optimizer).

instance classes where no exact solver had been able to prove optimality for any instance). The results are summarized in Table 8, and show that both XPRESS and CPLEX obtain better results after the tuning process. However, only the improvement by XPRESS is statistically significant, with a P-value of 0.0002 in a two-tailed pair-wise t-test. Since CPLEX and XPRESS makes different choices with respect to Gomory cuts in the parameter tuning process, it may appear that these choices are not important when solving instances of the BOOP. For XPRESS, the choice of always using dual simplex would not degrade the performance by much. We are thus inclined to suggest that the choices with respect to running the feasibility pump and the selection of heuristics in the search tree had an effect on the performance of XPRESS.

4. Summary of results and conclusions

GUROBI is the fastest MIP solver, and gives slightly better results than CPLEX. CPLEX is a bit faster than GUROBI in terms of proving the optimal solution. XPRESS finds worse solutions than the two other methods, and uses more time both to find these solutions and to prove optimality where applicable.

We estimate that CPLEX 12.1 is about 3.5 times faster than CPLEX 6.5 in terms of solving medium difficult BOOP instances to optimality. The percentage of instances solved to optimality has increased from 52.2 % to 62.0 %, but this increase is an overestimation since the effective time limit was higher for CPLEX 12.1. No speed increase is observed from XPRESS 12 to XPRESS 20, but the results obtained on instances where optimality is not proven are significantly improved. Also, the percentage of instances solved to optimality increased from 50.7 % to 58.0 %.

The tabu search method yields solutions of the same quality as those obtained by the exact methods when optimality is proven, and yields better solutions otherwise. Solutions are found much faster using the tabu search approach than any of the exact methods, representing a difference of roughly two orders of magnitude. In general, TS-ACW performed somewhat better than in the previous tests, due to better hardware which allowed the search to perform approximately three times as many iterations within a fixed time limit.

In summary, the computational tests comparing the TS-ACW method to the best performing commercial solver GUROBI produced the following outcomes.

Computed over all 205 test problems (Class 23 to Class 63), the average time for each method to obtain the best solution it found is 4.1 seconds for TS-ACW (with a limit of 60 seconds), and 229.4 seconds for GUROBI.

The average of the objective function values (for a maximization objective) expressed as a ratio to the average obtained by CPLEX 6.0 is 102.302 for TS-ACW (when taking the average objective function values, or 102.310 if taking the best objective function value over 10 independent runs), and 102.190 for GUROBI. Expressed in terms of actual differences in objective function values, these amounts correspond to an average difference of 100.6 between TS-ACW and GUROBI, with a maximum difference of 3339.7. More significantly, even if terminated after 5 seconds the tabu search approach yields solutions having the same quality as obtained by the GUROBI exact method when optimality is proven (126 out of 205 problems, one less than CPLEX 12.1), and TS-ACW yields solutions whose quality is superior to that of the exact method otherwise (79 out of 205 problems), even if restricted to 5 seconds per run. The other commercial solvers did not perform as well compared to our TS-ACW methods, as shown in Section 3.

We observe that these results demonstrate the considerable value of the exact solvers in cases where the overriding concern is to know whether or not optimal solutions have been found. Specifically, as noted, in 126 of the 205 problems tested (having disregarded the majority of the instances in the original test bed due to being too easy to solve) the best of the exact solvers did in fact verify optimality. Although it is noteworthy that our method likewise obtained these same solutions in a fraction of the time required by the exact solver, we emphasize that there are contexts in which an assurance of optimality (when an exact method succeeds in achieving and verifying optimality) can be important.

Acknowledgements

The authors thank an anonymous referee for helpful comments.

Table 8

	TS-ACW		CPLEX 6.5			CPLEX 12.1			XPRESS 12			XPRESS 20			GUROBI 3.0.1		
	Avg	**Sec B**	**Avg**	**Sec B**	**Sec T**	**Avg**	**Sec B**	**Sec T**	**Avg**	**Sec B**	**Sec T**	**Avg**	**Sec B**	**Sec T**	**Avg**	**Sec B**	**Sec T**
Class 23	101.933	0.1	**101.933**	43.6	50.6	**101.933**	7.1	31.6	**101.933**	29.2	52.5	**101.933**	129.8	135.6	**101.933**	3.4	36.7
Class 24	100.004	0.0	**100.004**	3.2	6.1	**100.004**	2.3	3.3	**100.004**	3.4	4.8	**100.004**	3.4	4.6	**100.004**	2.4	5.9
Class 26	100.034	0.1	**100.034**	40.1	53.1	**100.034**	14.0	23.0	**100.034**	27.7	43.5	**100.034**	35.8	47.6	**100.034**	8.2	33.5
Class 32	100.359	0.0	**100.359**	9.4	10.5	**100.359**	8.6	9.5	**100.359**	7.0	10.5	**100.359**	15.8	18.4	**100.359**	6.6	12.4
Class 33	100.000	0.0	**100.000**	0.0	1.5	**100.000**	0.4	0.6	**100.000**	0.0	0.3	**100.000**	1.0	1.0	**100.000**	0.4	1.1
Class 34	100.031	0.4	**100.031**	15.9	19.6	**100.031**	3.7	9.9	**100.031**	7.8	15.4	**100.031**	31.8	35.0	**100.031**	3.8	15.7
Class 35	100.000	0.0	**100.000**	6.5	10.3	**100.000**	3.9	5.0	**100.000**	3.2	7.3	**100.000**	6.8	9.6	**100.000**	5.6	9.3
Class 41	100.012	0.0	**100.012**	0.0	0.7	**100.012**	0.0	0.3	**100.012**	0.3	0.4	**100.012**	0.4	0.4	**100.012**	0.0	0.4
Class 42	100.010	0.0	**100.010**	0.0	0.7	**100.010**	0.0	0.2	**100.010**	0.1	0.2	**100.010**	0.6	0.6	**100.010**	0.0	0.4
Class 43	100.000	0.0	**100.000**	0.0	0.7	**100.000**	0.0	0.2	**100.000**	0.1	0.2	**100.000**	0.7	0.8	**100.000**	0.2	0.7
Class 44	100.000	0.0	**100.000**	0.0	1.2	**100.000**	0.0	0.4	**100.000**	0.3	0.4	**100.000**	0.5	1.0	**100.000**	0.0	0.8
Class 45	101.752	0.2	**101.752**	876.8	915.0	**101.752**	90.6	210.9	**101.752**	248.4	409.9	**101.752**	1016.4	1110.0	**101.752**	29.0	248.0
Class 46	100.000	0.0	**100.000**	53.1	62.4	**100.000**	8.0	15.8	**100.000**	18.3	25.2	**100.000**	24.0	27.4	**100.000**	3.8	24.9
Class 48	100.115	0.5	**100.115**	1944.6	2309.6	**100.115**	495.2	628.7	**100.115**	449.8	1618.6	**100.115**	1282.2	1558.0	**100.115**	61.0	735.3
Class 50	100.000	0.0	**100.000**	0.0	0.3	**100.000**	0.0	0.2	**100.000**	0.0	0.1	**100.000**	0.0	0.0	**100.000**	0.0	0.1
Class 51	101.502	0.0	**101.502**	155.8	177.7	**101.502**	36.0	51.3	**101.502**	91.6	709.4	**101.502**	153.0	169.4	**101.502**	9.2	43.3
Class 55	103.280	0.0	**103.280**	125.0	141.2	**103.280**	52.7	75.7	**103.280**	67.5	119.1	**103.280**	182.8	257.6	**103.280**	13.8	71.8
Class 56	100.000	0.0	**100.000**	5.8	9.1	**100.000**	2.7	4.2	**100.000**	3.8	5.9	**100.000**	7.0	7.6	**100.000**	2.6	7.8
Class 58	100.049	0.0	**100.049**	104.6	113.3	**100.049**	22.6	40.9	**100.049**	53.2	69.4	**100.049**	69.0	89.6	**100.049**	6.8	44.5
Avg.	100.478	0.1	100.478	178.1	204.4	100.478	39.3	58.5	100.478	53.3	162.8	100.478	155.8	182.9	100.478	8.3	68.0

Results on classes where all exact methods are able to prove optimality within the time limit. Running times are normalized to match those of the fastest computer used, reporting time in seconds to best solution (B) and time in seconds to prove optimality total (T). Results for TS-ACW are averaged over ten runs, each run being terminated after 60 seconds.

Table 9

	TS-ACW		CPLEX 6.5			CPLEX 12.1			XPRESS 12			XPRESS 20			GUROBI 3.0.1		
	Avg	**B**	**Avg**	**B**	**T**	**Avg**	**B**	**T**	**Avg**	**B**	**T**	**Avg**	**B**	**T**	**Avg**	**B**	**T**
Class 25	102.610	0.9	102.610	1962.5	2167.9	**102.610**	302.0	544.8	**102.610**	389.8	1198.8	**102.610**	2046.8	2589.8	**102.610**	12.8	954.3
Class 28	101.077	0.0	101.045	2048.7	2663.6	**101.077**	57.0	563.6	**101.077**	695.8	1431.8	**101.077**	742.0	1383.8	**101.077**	25.2	607.1
Class 36	105.074	2.7	104.463	844.9	4800.0	105.049	2712.5	13509.0	104.904	662.8	1862.4	104.789	3946.0	14400.0	105.052	35.8	13545.2
Class 37	100.543	0.1	100.543	1369.1	1573.6	**100.543**	118.3	314.0	**100.543**	317.7	807.6	**100.543**	614.2	736.8	**100.543**	32.2	392.2
Class 39	100.299	1.1	100.178	1601.5	4800.1	100.296	5062.4	10488.3	100.267	1300.1	1862.4	100.272	8829.8	14400.0	100.292	108.6	12452.0
Class 47	100.481	3.0	100.386	1781.5	4800.0	100.484	181.5	4630.5	100.477	1180.6	1862.4	100.484	1790.2	10083.2	100.484	53.6	5523.6
Class 57	104.162	0.8	103.817	1997.8	4251.1	**104.162**	1249.2	6659.5	104.127	468.3	6803.6	103.957	4692.6	9135.2	104.147	14.0	8513.1
Avg.	102.035	1.2	101.863	1658.0	3579.5	102.032	1383.3	5244.2	102.001	716.4	2261.3	101.962	3237.4	7532.7	102.029	40.3	5998.2

Results on classes containing some instances where only a subset of the exact methods is able to prove optimality within the time limit. Running times are normalized to match those of the fastest computer used, reporting time in seconds to best solution (B) and time in seconds to prove optimality total (T). Results for TS-ACW are averaged over ten runs, each run being terminated after 60 seconds.

Table 10

	TS-ACW		CPLEX 6.5			CPLEX 12.1			XPRESS 12			XPRESS 20			GUROBI 3.0.1		
	Avg	**B**	**Avg**	**B**	**T**	**Avg**	**B**	**T**	**Avg**	**B**	**T**	**Avg**	**B**	**T**	**Avg**	**B**	**T**
Class 27	111.192	1.9	110.430	2251.8	4800.0	111.028	3485.9	14400.0	110.938	853.0	11174.4	110.376	10708.6	14400.0	111.098	39.2	14400.0
Class 29	101.288	14.2	100.992	358.0	4800.0	101.221	5449.7	14400.0	101.065	494.2	1862.4	100.779	10384.8	14400.0	101.209	110.6	14400.0
Class 30	100.391	1.4	100.290	225.9	4800.1	100.390	6063.3	14400.0	100.368	1239.2	1862.4	100.370	8708.2	14400.0	100.384	108.2	14400.0
Class 31	101.448	25.0	100.967	2949.9	4800.1	101.216	8985.2	14400.0	100.974	728.7	1862.4	99.965	3697.2	14400.0	101.313	583.8	14400.0
Class 38	100.985	8.8	100.796	743.8	4800.0	100.964	5738.5	14400.0	100.868	982.6	1862.4	100.657	7599.0	14400.0	100.952	120.4	14400.0
Class 40	101.610	13.9	101.078	1581.1	4800.1	101.517	3951.3	14400.0	101.056	758.4	1862.4	100.480	3386.2	14400.0	101.512	642.2	14400.0
Class 49	101.515	15.4	101.060	1649.4	4800.1	101.359	11180.5	14400.0	101.027	822.2	1862.4	100.511	8056.0	14400.0	101.388	652.0	14400.0
Class 52	109.910	4.6	103.342	2040.3	4800.1	109.603	8008.7	14400.0	79.565	724.6	1862.4	103.194	6993.8	14400.0	109.825	75.0	14400.0
Class 53	113.226	0.9	95.555	808.9	4800.0	111.433	10673.6	14400.0	79.487	957.2	1862.4	92.639	124.6	14400.0	111.805	505.8	14400.0
Class 54	114.816	27.1	87.694	4800.1	4800.1	110.368	14118.9	14400.0	66.231	1077.7	1862.4	92.564	121.1	14400.0	112.832	4078.2	14400.8
Class 59	107.363	1.9	106.545	933.3	4800.0	107.194	6202.3	14400.0	106.910	1786.2	11174.4	105.983	10433.4	14400.0	107.184	64.6	14400.0
Class 61	101.556	6.7	101.155	807.4	4800.0	101.408	10844.2	14400.0	101.299	4368.4	11174.4	100.894	3242.2	14400.0	101.441	447.2	14400.0
Class 62	100.984	7.6	100.867	1026.7	4800.1	100.975	1319.7	14400.0	100.960	2110.5	11174.4	100.900	8596.8	14400.0	100.949	89.0	14400.0
Class 63	103.579	13.7	103.103	2715.8	4800.1	103.415	13884.3	14400.0	103.014	4008.4	11174.4	102.159	9147.6	14400.0	103.421	1396.8	14400.0
Avg.	104.990	10.2	100.991	1635.2	4800.1	104.435	7850.4	14400.0	96.697	1493.7	5188.1	100.819	6514.2	14400.0	104.665	636.6	14400.1

Results on classes where none of the exact methods has been able to prove optimality within the time limit on any instances. Running times are normalized to match those of the fastest computer used, reporting time in seconds to best solution (B) and time in seconds to prove optimality total (T). Results for TS-ACW are averaged over ten runs, each run being terminated after 60 seconds.

Table 11

	CPLEX 12.1 TUNED			XPRESS 20 TUNED		
	Avg	**B**	**T**	**Avg**	**B**	**T**
Class 27	111.093	5145.0	14400.0	110.698	5455.0	14400.0
Class 29	101.241	10712.5	14400.0	101.010	5701.2	14400.0
Class 30	100.391	7211.2	14400.0	100.348	5693.7	14400.0
Class 31	101.290	6588.8	14400.0	101.228	5913.0	14400.0
Class 38	100.960	10707.7	14400.0	100.930	2277.8	14400.0
Class 40	101.486	6277.5	14400.0	101.412	6610.0	14400.0
Class 49	101.363	6225.5	14400.0	101.264	5729.2	14400.0
Class 52	109.769	10530.6	14400.0	109.073	2424.2	14400.0
Class 53	111.451	13378.7	14400.0	110.779	2167.0	14400.0
Class 54	111.155	14279.5	14400.0	110.036	5074.4	14400.0
Class 59	107.191	10738.6	14400.0	106.886	2300.4	14400.0
Class 61	101.436	8288.7	14400.0	101.340	4322.8	14400.0
Class 62	100.970	5286.1	14400.0	100.931	5613.2	14400.0
Class 63	103.359	11368.5	14400.0	103.106	2506.4	14400.0
Avg.	104.511	9052.8	14400.0	104.217	4413.4	14400.0

Results on classes where none of the exact methods has been able to prove optimality within the time limit on any instances. Running times are normalized to match those of the fastest computer used, reporting time in seconds to best solution (B) and time in seconds to prove optimality total (T).

Table 12

	#vars	#terms	%compl	#inst
Class 23	100	400	25	5
Class 24	100	400	25	5
Class 25	200	400	25	5
Class 26	200	400	25	5
Class 27	200	1000	25	5
Class 28	200	1000	25	5
Class 29	500	1000	25	5
Class 30	500	1000	25	5
Class 31	500	2500	25	5
Class 32	100	400	50	5
Class 33	100	400	50	5
Class 34	200	400	50	5
Class 35	200	400	50	5
Class 36	200	1000	50	5
Class 37	200	1000	50	5
Class 38	500	1000	50	5
Class 39	500	1000	50	5
Class 40	500	2500	50	5
Class 41	100	400	75	5
Class 42	100	400	75	5
Class 43	200	400	75	5
Class 44	200	400	75	5
Class 45	200	1000	75	5
Class 46	200	1000	75	5
Class 47	500	1000	75	5
Class 48	500	1000	75	5
Class 49	500	2500	75	5
Class 50	100	400	0	5
Class 51	200	1000	0	5
Class 52	500	2500	0	5
Class 53	500	5000	0	5
Class 54	1000	10000	0	5
Class 55	100	400	0	5
Class 56	100	400	0	5
Class 57	200	400	0	5
Class 58	200	400	0	5
Class 59	200	1000	0	5
Class 60	200	1000	0	5
Class 61	500	1000	0	5
Class 62	500	1000	0	5
Class 63	500	2500	0	5

Details of the test instances used, reporting the number of variables, the total number of terms (non-zero elements in the constraint matrix), and the percentage of complemented terms. Classes 23–49 are random instances, classes 50–54 are based on graph stability problems, and classes 55–63 are based on set covering problems.

Received 25-7-2011; revised 14-2-2012; accepted 22-2-2012

References

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

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

[3] CPLEX, 2011. `http://www-01.ibm.com/software/integration/optimization/cplex-optimizer/`.

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

[5] J.J. Dongarra. Performance of various computers using standard linear equations software. Technical Report CS-(2010) 89-85.

[6] D. Du, J. Gu, and P.M. Pardalos, editors. *Satisfiability Problem: Theory and Applications*, *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*. 35(1997).

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

[8] GUROBI, 2011. `http://www.gurobi.com/`.

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

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

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

[12] J.T. Linderoth and A. Lodi. MILP software. In J.J. Cochran, L.A. Cox Jr., P. Keskinocak, J.P. Kharoufeh, and J.C. Smith, editors, *Wiley Encyclopedia of Operations Research and Management Science*. Wiley, 2011.

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

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

[15] XPRESS, 2011. `http://www.fico.com/en/Products/DMTools/Pages/FICO-Xpress-Optimization-Suite.aspx`.