New Heuristics and Adaptive Memory Procedures for Boolean Optimization Problems

Lars M. Hvattum
Molde College, 6411 Molde, Norway.
Lars.M.Hvattum@himolde.no

Arne Løkketangen
Molde College, 6411 Molde, Norway.
Arne.Lokketangen@himolde.no

Fred Glover
Leeds School of Business, UCB 419
University of Colorado, Boulder, CO 80309, USA
Fred.Glover@Colorado.edu

Abstract

We describe new constructive and iterative search methods for Boolean Optimization Problems. Extending previous work by the authors, we describe the use of adaptive clause weights and probabilistic move acceptance for the adaptive memory search. We also describe how the use of the concepts *persistent attractiveness measures* and *marginal conditional validity* as search guidance mechanisms lead to very good results. Computational tests on portfolios of benchmark problems taken from the literature disclose that our method obtains results that improve on those previously published.

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 exact (provably convergent) optimization methods encounter severe performance difficulties in these particular applications, being dominated by heuristic search methods even for moderately sized instances.

Previous heuristic work on this problem is mainly by Davoine, Hammer and Vizvári (2003), employing a greedy heuristic based on pseudo-boolean functions. Hvattum, Løkketangen and Glover (2003) describe simple iterative heuristic methods for solving BOOP, starting from random initial solutions. Although equipped with no long-term mechanism apart from a random restart procedure, they obtain very good results compared to the work by Davoine, Hammer and

Vizvári, and also by an even greater margin when compared to CPLEX and XPRESS/MP on the larger instances.

The remainder of this paper is organized as follows. Section 2 provides BOOP problem formulations and details of previous work. Section 3 describes new local search mechanisms, designed to diversify the search, while Section 4 describes our new constructive methods. In Section 5 we address the Weighted Maximum Satisfiability problem (W-MAX_SAT), and show how to transform it into a BOOP formulation framework. Computational results are given in Section 6, followed by the conclusions in Section 7.

2 Problem formulation and search basics

2.1 Problem formulation

The Boolean Optimization Problem (BOOP), first formulated in Davoine, Hammer and Vizvári (2003), is based on logical expressions in prepositional, first-order logic, with an extra cost (or profit) associated with the variables having a *true* (or *false*) value. One formulation can be (assuming maximization)

$$Maxz = \sum_{i=1}^{N} (c_i \,|\, x_i = true \,/\, false)$$

such that

$$\Phi(x) = \Phi(x_1, ..., x_N) = \begin{cases} true \\ false \end{cases}$$

where $\Phi(x)$ is the logical expression, and N the number of variables. The solution to this problem is the set of truth value assignments to the x_i variables that yields the highest objective function value z, while satisfying the logical expression. The logical expression can in general be arbitrary, but we restrict ourselves to formulations in *conjunctive normal form*, *CNF*. (The *disjunctive normal form* can be obtained by a simple transformation.) Informally, a BOOP can be regarded as a *satisfiability problem* (SAT) with an objective function added on. For more info on SAT, see e.g. Cook (1971), and Du et al. (1997).

Applying simple transformations described in Hvattum, Løkketangen and Glover (2003), we get the following model by splitting each x_i into its *true* and *false* component y_i and $y_{i\#}$:

$$Max \quad z = \sum_{i=1}^{N} c_i y_i$$

s.t.

$$Dy \geq 1$$

$$y_i + y_{i\#} = 1$$

where D is the 0-1 matrix obtained by substituting the y's for the x_i's. The last constraint is handled implicitly in the search heuristics we introduce.

2.2 Local search basics

To better understand the mechanisms described in this paper, some background from previous work is helpful. For fuller details, see Hvattum, Løkketangen and Glover (2003). The basic strategy of this earlier work includes the following features.

- The *starting solution* (or starting point) is based on a random assignment to the variables. This solution may be primally infeasible, and hence the search must be able to move in infeasible space.
- A *move* is the flip of a variable by assigning the opposite value (i.e. change $1 \rightarrow 0$ or $0 \rightarrow 1$).
- The *search neighborhood* is the full set of possible flips, with a *neighborhood size* of N, the number of variables.
- *Move evaluation* is based on both the change in objective function value, and the change in amount of infeasibility.
- The *move selection* is greedy (i.e. take the best move according to the move evaluation).
- Simple randomized *tabu tenure* and a *new best* aspiration criterion are used.
- A random restart is applied after a certain number of moves, to *diversify* the search
- The *stopping criterion* is a simple time limit.

The manner in which we incorporate these features, and add new ones to our current method, is sketched in the following sections.

2.3 Move evaluation function

The move evaluation function, F_{Mi}, has two components. The first is the change in objective function value. The cost coefficients, c_i, are initially normalized to lie in the range (0,1). This means that the change in objective function value per move, Δz_i, is in the range (-1, +1).

The second component is the change in the number of violated clauses (or constraint rows), for the flipping of each variable. This number, ΔV_i will usually be a small positive or negative integer. For a different way to handle infeasible solutions, see Løkketangen and Glover (1996).

These two components are combined to balance the considerations of obtaining solutions that are feasible and that have a good objective function value. The relative emphasis between the two

components is changed dynamically to focus the search in the vicinity of the feasibility boundary, using the following move evaluation function:

$$F_{Mi} = \Delta V_i + w^* \Delta z_i$$

The value of w, the adaptive component, is initially set to 1. It is adjusted after each move so that:

- If the current solution is feasible: $w = w + \Delta w_{inc}$
- If the current solution is not feasible, and $w > 1$: $w = w - \Delta w_{dec}$

The effect of this adaptation is to induce a strategic oscillation around the feasibility boundary. A different approach appears in Glover and Kochenberger (1996), where the oscillation is coupled with the use of a critical event memory, forcing the search into new areas.

3 Local Search Improvements

The simple local search described in Hvattum, Løkketangen and Glover (2003) relies on a sophisticated adaptive move evaluation scheme for achieving the type of balance between feasibility and objective function quality previously described. From their computational results, however, it is evident that for the larger test cases a better form of diversification than random restart is needed to be able to explore larger parts of the search space.

The extra mechanisms come at a cost. There is a tradeoff between the gains provided by improved search guidance or diversification, and the associated computational effort to perform the extra calculations and to maintain the auxiliary data structures. In the current setting, the additional mechanisms reduce the number of search iterations done in a given amount of computational time.

We have implemented two processes for diversification: *Adaptive Clause Weighting,* and *Probabilistic Move Acceptance*.

3.1 Adaptive Clause Weights

In the basic local search scheme, all violated clauses (i.e. constraint rows) contribute the same amount to the move evaluation function, F_{Mi}, as described in section 2.3. However, some of the clauses will be more difficult to satisfy than others, and should be given more emphasis. We achieve this by attaching a separate weight, CW, to each clause. Previous work on adaptive clause weights can be found in Løkketangen and Glover (1997).

All clauses start with $CW = 1$. The weight is updated only after iterations where a clause becomes violated, at which point the weight of the newly violated clause is incremented by a small amount, ΔCW. To prevent clause weights from growing prohibitively large, they are renormalized by dividing all the clause weights by a constant CW_{DIV}, whenever one weight becomes grater than some CW_{LIM}.

Such a procedure constitutes a long-time learning approach. The move evaluation function drives the search out of the feasible region to seek solutions with high objective function quality in nearby infeasible space. Having adaptive clause weights helps the search to better adapt to the infeasibility border of the search space, thus enabling the search to cross back over the border to find different, and better, feasible solutions. As shown in Section 6.1, the tradeoff between the extra time taken to update the weights, and the resulting improved search guidance pays the greatest dividends for the larger problems.

3.2 Probabilistic Move Acceptance

Every iteration the search method generates a list that identifies a subset of possible moves to execute, and the *best* move from this list is selected. Usually this *best* equates with best *move evaluation value.* But the move evaluation function is rather myopic, only looking at the local neighborhood, and we modify it by using recency and frequency measures as proposed in tabu search. (See, e.g., Glover and Laguna, 1997, and Gendreau, 2003)

In a sorted list of possible moves, the presumably *best* moves will be at the front of the list, but not necessarily in strict order. A simplified variant of this principle from Glover (1989) is also employed in GRASP, where the chosen move is randomly selected among the top half of the moves (see Feo and Resende, 1989).

We use this approach by selecting randomly from the top of the list, but in a way biased towards the moves having the highest evaluations. This is called *Probabilistic Move Acceptance*, *PMA*, as described in Løkketangen and Glover (1996). The selection method is as follows:

PMA:

1. Select a move acceptance probability, p.
2. Each iteration sort the admissible moves according to the move evaluation function
3. Reject moves with probability $(1 - p)$ until a move is accepted
4. Execute the selected move in the normal way
5. If not finished, go to 2 and start the next iteration

This can also be viewed as using randomness to diversify the search (as a substitute for deterministic use of memory structures), but in a guided way.

In our local search setting, using PMA generally yields worse results than the deterministic approach of always taking the best non-tabu move. This implies that the move evaluation function is good, and that rejecting the top moves deteriorates the search.

The inclusion of PMA yields better results for the largest problems (with up to 1000 variables and 10000 clauses). This indicates that the PMA introduces some necessary diversification that the basic mechanisms lack.

4 Constructive Methods

Constructive methods in the literature are mainly used for creating good, or feasible, starting solutions for subsequent local search heuristics. We show how proper use of adaptive memory structures can be used to create iterated constructive learning heuristics. These generate a series of solutions, where the constructive guidance is modified by the outcome of the previous searches. Our ideas are based on principles for exploiting adaptive memory to enhance multi-start methods given in Glover (2000).

We focus in particular on implementing the principles embodied in the *PAM* (Persistent Attractiveness Measure) and *MCV* (Marginal Conditional Validity) concepts. As is customary, our methods also incorporate a short local search after each constructive phase.

4.1 PAM – Persistent Attractiveness Measure

The *Persistent Attractiveness Measure*, PAM, is a measure of how often a specific value assignment to a variable is considered attractive, without actually being selected for inclusion in the solution. It is reasonable to assume that early assignments in the construction phase should have more importance than later assignments, and that good evaluations should be better than bad ones. The attractiveness should also increase for a variable assignment that repeatedly is ranked high without being chosen.

If we index the assignment steps with s, and the individual rankings in a step with r, we would like the PAM to have the following properties:

PAM(r,s) is decreasing for increasing s (earlier steps are more important)

PAM(r,s) is decreasing for increasing r (higher rank is better)

We only calculate PAM for the top ranked moves.

The PAM evaluator can be formulated as E(s,r) = E'(s) + E''(r), where, for BOOP, we set

$E'(s) = as^* - as$ and
$E''(r) = br^* - br$

where $s^* = N$ (number of variables) and r is a parameter. The constants a and b are determined experimentally as subsequently described.

The PAM-values corresponding to a given assignment are summed over all the constructive steps, and added (with exponential smoothing over the constructive runs) to yield an overall measure of attractiveness for each possible assignment.

PAM values for several consecutive constructive runs can be accumulated in a measure of attractiveness e.g. by exponential smoothing:

New PAM = (Last PAM + Last Accumulated Total PAM)/2

We thus expand the move evaluation indicated earlier to become:

$$F(y_{i(\#)}) = ?V_{i(\#)} + w*(z_{i(\#)} + PAM_{i(\#)})$$

The values for the PAM-measure are limited to an interval $[0,k]$, with k chosen to match w in some way, again as specified later.

4.2 MCV – Marginal Conditional Validity

The choices made at the beginning of a constructive search are based on less information than later choices, and are thus more likely to be *bad*. When later choices are made, the problem has been reduced by the earlier choices, and better choices can be made (but in the context of the earlier ones). Later decisions are thus likely to make earlier decisions look better. We call this the *Marginal Conditional Validity* principle.

After the constructive phase we analyze the completed solution to find assignments that should have been different. There are two cases that can be used as a foundation:

1. A variable is *true*, but there are unsatisfied rows where the negated variable is present.
2. A variable is *false*, but the negated variable is present only in rows that would be satisfied even if the variable had been flipped.

In the first case the opposite value assignment to the variable would satisfy more rows, while in the second case we would get an increase in the objective function value, without violating any new constraints. These are then used as a start of the next iteration, with a probability *p*.

4.3 A Comparison with GRASP

The *Greedy Randomized Adaptive Search Procedure*, or GRASP, is a well known, memoryless, constructive heuristic relying heavily on randomization (see Resende, Pitsoulis, and Pardalos, 1997). A constructive run can be followed by a short greedy local search.

We have adapted and implemented GRASP to work for BOOP, for comparison purposes. We use the same basic objective function value as before, but without any adaptive memory or learning. The only parameter required for GRASP is the proportion of moves to be considered for execution in each constructive assignment, called α.

GRASP:

1. Start with all variables unassigned, rate all possible assignments.
2. Select an assignment randomly among those who are within α% of best evaluation
3. When all variables are assigned, possibly do a local search
4. Go to 1, if not finished

We use time as a stopping criterion, and try three versions of local search: No LS, complete LS or "steepest ascent" LS.

We also tried to augment GRASP with learning capabilities similar to the adaptive clause weights outlined in section 3.1. After each GRASP run, the clause weights are updated analogous to the iterative case. Computational results are in 6.4.

5 Weighted Maximum Satisfiability

To support the claim that BOOP can represent many different problem classes, this section outlines how Weighted Maximum Satisfiability problems (W-MAX_SAT) can be easily transformed to BOOP. Section 6.5 gives computational results for this case, without any effort to specialize our procedure to handle the special structure of this problem.

A W-MAX_SAT instance can informally be regarded as an unsatisfiable instance of a SAT problem that in addition has *weights* on the clauses (rows). The objective is then to find a truth assignment that maximizes the sum of the weights on the satisfied clauses. This is similar to BOOP, except that weights are attached to the clauses rather than the variables. A W-MAX-SAT instance can be transformed to BOOP by adding a new variable to each clause to carry information about weights. The clause weights are transformed to objective function value coefficients, while the original *n* variables will have objective function value coefficients of 0.

Thus, if the W-MAX-SAT has *n* variables and *m* clauses, the BOOP will have (*n*+*m*) variables and *m* clauses. The number of clauses (rows), *m*, is often large compared to the number of variables, *n*, giving a BOOP encoding for W-MAX-SAT having many more variables. (In the test instances used in section 6.5 n is 100 and m is 800-900, giving 900-1000 variables for the BOOP encoding, compared to 100 for W-MAX-SAT).

As we can see in the computational results section 6.5, our BOOP code compares favourably to the GRASP heuristic on the same problem instances (Resende, Pitsoulis and Pardalos, 1997), and is only slightly worse than the special purpose method of Shang (1997) in spite of the fact that no specialization is used in our procedure.

6 Computational Results

This section reports the final parameter settings applied to each of the different methods or mechanisms during testing, as well as overall computational results. Section 6.6 attempts to compare all the different methods and mechanisms in a meaningful way.

The same BOOP test cases as used in the previous work (Hvattum, Løkketangen and Glover, 2003, and Davoine, Hammer and Vizvári, 2003) are used for testing. The test-set consists of 5485 instances, ranging in size from 50 to 1000 variables, and 200 to 10000 clauses (rows). Results are reported as the average of solution values relative to results obtained by Davoine, Hammer and Vizvári using CPLEX 6.0.

The testing of W-MAX_SAT is based on modifying the unsatisfiable *jnh**, as used in Resende, Pitsoulis and Pardalos (1997). These all have 100 variables and 800 to 900 clauses. For preliminary testing to fix parameter values, we selected the same 3 test cases as in Hvattum, Løkketangen and Glover (2003).

6.1 Effect of Adaptive Clause Weights

The first addition to the mechanisms for BOOP described in Hvattum, Løkketangen and Glover (2003), is the inclusion of adaptive clause weights (see 3.1). Preliminary testing showed that the results were not very sensitive to the values of CW_{LIM} (the maximum weight value) or CW_{DIV} (the renormalization factor). For our final testing we used CW_{LIM} = 4.0 and CW_{DIV} = 2.0. The best value for ΔCW was chosen to be 0.003, based on preliminary testing. The actual value is not sensitive, but it should be much smaller than 1.

Computational results are shown in Table 1. The results using Adaptive Clause Weights (ACW) are compared to the results from Hvattum, Løkketangen and Glover (2003) (TS), with computational time of 5 and 60 seconds.

Table 1. Adaptive Clause Weights

	TS 5	TS 60	ACW 5	ACW 60
Classes 1-22	100,001	100,001	100,002	100,002
Classes 23-49	101,214	101,215	101,212	101,214
Classes 50-54	106,305	106,982	107,628	107,866
Classes 55-63	102,463	102,465	102,462	102,464
Classes 1-63	101,373	101,427	101,477	101,497

As can be seen, the overall results show an improvement for both the 5 second and 60 second cutoff. For classes 55-63 the results are slightly inferior to those of our earlier approach.

6.2 Effect of Probabilistic Move Acceptance

The important parameter for PMA is the probability of move acceptance, p. (See 3.2). In Table 2 and 3 are shown the results for a selected test case for various values of p without and with adaptive clause weights (ACW). As is indicated in the tables, a fairly large value should be chosen for p. In our subsequent test we use the value 0.9. Overall computational results are shown in Table 4. The use of PMA gives in general slightly inferior results, except for the largest problems. This is as expected, as the search guidance (through the move evaluation value) should be better for smaller problems. The PMA also introduces a certain amount of diversification that is helpful for the larger problems.

Table 2. PMA without ACW

p	obj. value	time
0,1	142796	3,86
0,2	143138	3,47
0,3	143255	7,05
0,4	143315	5,27
0,5	143356	4,17
0,6	143367	4,02
0,7	143372	1,98
0,8	143372	1,20
0,9	143372	1,59
1,0	143372	1,33

Table 3. PMA with ACW

p	obj.value	time
0,1	142845	5,29
0,2	143148	6,35
0,3	143246	6,18
0,4	143323	5,51
0,5	143357	6,11
0,6	143365	4,62
0,7	143369	2,27
0,8	143372	1,96
0,9	143372	1,27
1,0	143372	0,82

Table 4. Results for PMA

	Tabu Search		PMA w.o. ACW		PMA w. ACW	
	TS 5	TS 60	PMA 5	PMA 60	PMA 5	PMA 60
Classes 1-22	100,001	100,001	99,998	100,000	99,996	99,998
Classes 23-49	101,214	101,215	101,205	101,213	101,205	101,211
Classes 50-54	106,305	106,982	105,787	106,168	107,438	107,778
Classes 55-63	102,463	102,465	102,446	102,463	102,450	102,461
Classes 1-63	101,373	101,427	101,324	101,361	101,455	101,487

6.3 PAM and MCV

Preliminary testing gave the following values for the *PAM* (*Persistent Attractiveness Measure*) and *MCV* (*Marginal Conditional Validity*) parameters, whose role is sketched in Sections 4.1 and 4.2:

$a = 2$
$b = 3$
$r^* = 4$

The PAM-value of each variable assignment is scaled to lie between 0 and 0.3 before it is used in the move evaluation function as specified in section 4.1.

Figure 1 shows results for the given test case for various values of p, the probability that determines when to apply the MCV principle. For this test case the best results are when the MCV principle ($p = 0$) is not applied. The results with $p = 0.4$ gives best results when applying MCV, and this value is used in the computational testing.

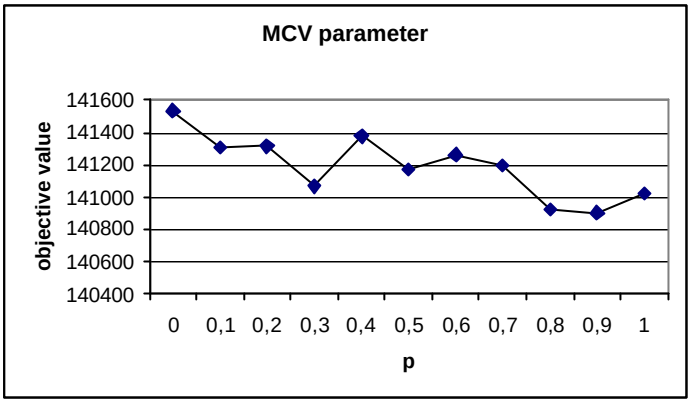

Figure 1. Values for *p* for MCV

Table 5 shows the computational results for our constructive learning heuristic applying both PAM and MCV. The column **PAM/MCV - NO LS** gives the results when no local search was applied after each constructive run. **PAM/MCV - STEEP** indicates that a steepest descent local search was applied after each construction, and **PAM/MCV - TS 500** indicates that a tabu search limited to 500 iterations, as described in 3.1, was used for improvement. All the runs were for 60 seconds. This new constructive method, even without the local search, performs much better than the basic GRASP approach (see 6.4). The constructive approach without local search (LS) also beats the results in Davoine, Hammer and Vizvári (2003) on small instances, and beats, with the addition of a short LS to the constructive approach, these results on all the instances.

Table 5. Results for PAM and MCV

	PAM/MCV - NO LS	PAM/MCV - STEEP	PAM/MCV - TS 500
Classes 1-22	99,359	99,780	100,002
Classes 23-49	99,571	100,074	101,205
Classes 50-54	97,202	98,545	106,778
Classes 55-63	99,581	99,942	102,448
Classes 1-63	99,310	99,831	101,405

It seems that when combining the constructive learning heuristic with the TS from 3.1, most of the benefit comes from the TS. However, the method **PAM/MCV - TS 500** was the only method that finds the optimum of all the small instances (class 1-22, 5280 instances). In fact, all the optima were found within 2 seconds. This seems to reflect the trend we have observed for our constructive heuristics, that they are more effective for the smaller problem instances and do not often contribute improved results for the largest problem instances.

6.4 Comparison with GRASP

Results for the GRASP heuristic outlined in section 4.3 are shown in Table 6, allowing for 5 or 60 seconds search time. A value of $\alpha = 0.5$ was used. The column **GRASP – NO LS** shows the results when no local search is applied after the constructive phase, and **GRASP – CLS** shows

the results when a complete, recursive, local search is applied. **GRASP – STEEP** shows the results when steepest descent is used.

Table 6. Results for GRASP

	GRASP – NO LS		GRASP - CLS		GRASP - STEEP	
	5 sec.	60 sec.	5 sec.	60 sec.	5 sec.	60 sec.
Classes 1-22	97,483	98,413	99,387	99,680	99,389	99,662
Classes 23-49	95,400	96,149	97,216	97,983	98,326	98,826
Classes 50-54	86,554	89,748	90,164	91,857	93,844	95,969
Classes 55-63	95,647	96,550	97,114	97,950	97,302	97,988
Classes 1-63	95,461	96,489	97,400	98,085	98,195	98,772

These results indicate that GRASP is better than Davoine, Hammer and Vizvári (2003) on small instances, but does not scale well for the larger problems.

When we apply our adaptive learning weights, GRASP functions much better. Table 7 shows the results with adaptive clause weights and complete local search. The same values are used for Δw_{inc} (= 0.20) and Δw_{dec} (= 0.15) as for the TS. The results are now better than Davoine, Hammer and Vizvári (2003), except on classes 50-54. This shows that a modified GRASP can compete with other heuristics on small and medium sized instances, while other mechanisms may be needed for the larger ones. The recent work on marrying GRASP with path relinking offers promise in this regard. (See Resende and Ribeiro, 2003)

Table 7. Results for GRASP with learning

	GRASP w. Learning
Classes 1-22	99,972
Classes 23-49	100,717
Classes 50-54	96,587
Classes 55-63	101,901
Classes 1-63	100,298

6.5 Results for Weighted Maximum Satisfiability

We use the encoding of W-MAX_SAT in the BOOP framework outlined in Section 5. Our problem instances are from Resende, Pitsoulis and Pardalos (1997), based on the unsatisfiable "jnh" instances from 2nd DIMACS Implementation Challenge. These test cases have 100 variables, and 800 to 900 clauses (rows). Our BOOP encoding of these problems thus has 900 to 1000 variables and 800 to 900 rows, being somewhat inflated compared to the original encoding.

Computational results are shown in Table 8. The settings for **ACW 60** (see 6.1), without any changes, are used. **GRASP*** shows the results reported in Resende, Pitsoulis and Pardalos (1997). The column **DML** shows the results for DML, a Lagrange-based method specially tailored to the problem (Shang, 1997).

Table 8. Results for W-MAX_SAT

Problem	Optimal	GRASP*	DML	ACW 60
jnh1	420925	-188	0	0
jnh4	420830	-215	-41	-85
jnh5	420742	-254	0	-116
jnh6	420826	-11	0	-15
jnh7	420925	0	0	0
jnh8	420463	-578	0	0
jnh9	420592	-514	-7	-327
jnh10	420840	-275	0	0
jnh11	420753	-111	0	-250
jnh12	420925	-188	0	0
jnh13	420816	-283	0	0
jnh14	420824	-314	0	-172
jnh15	420719	-359	0	-52
jnh16	420919	-68	0	-5
jnh17	420925	-118	0	0
jnh18	420795	-423	0	-207
jnh19	420759	-436	0	0
jnh201	394238	0	0	0
jnh202	394170	-187	0	-126
jnh203	394199	-310	0	-137
jnh205	394238	-14	0	0
jnh207	394238	-137	0	-9
jnh208	394159	-172	0	-162
jnh209	394238	-207	0	0
jnh210	394238	0	0	0
jnh211	393979	-240	0	0
jnh212	394238	-195	0	0
jnh214	394163	-462	0	0
jnh215	394150	-292	0	-199
jnh216	394226	-197	0	0
jnh217	394238	-6	0	0
jnh218	394238	-139	0	0
jnh219	394156	-436	0	-103
jnh220	394238	-185	0	-33
jnh301	444854	-184	0	0
jnh302	444459	-211	-338	0
jnh303	444503	-259	-143	-414
jnh304	444533	-319	0	-570
jnh305	444112	-609	-194	-299
jnh306	444838	-180	0	0
jnh307	444314	-155	0	-685
jnh308	444724	-502	0	-699
jnh309	444578	-229	0	0
jnh310	444391	-109	0	0
Average	415914	-233	-16	-106

Our computational results compare favourably to those of the GRASP heuristic on the same problem instances. Our outcomes are only slightly worse than those of the special purpose DML method of Shang (1997), although we are undertaking to solve the much larger transformed problem and make no use of any specialization.

6.6 Performance Profiles

It is always very difficult to compare different methods based on tables of computational results, unless one method is best on all the tests. We therefore also compare our methods using the ideas given in Dolan and Moré (2001). Based on the time used to find the best solution, we can construct a performance profile as follows.

Let P be the set of problem instances, S be the set of solvers, and n_p be the number of problems. Define $t_{p,s}$ to be the time used by solver s to solve problem p. Let

$$r_{p,s} = \frac{t_{p,s}}{\min\{t_{p,s^*} \mid s^* \in S\}}$$

be the ratio between the performances of solver s to the *best* solver on the problem p. If a solver fails to solve a problem, then set $r_{p,s} = r_M$, where $r_M \geq r_{p,s}$ for all p and s.

A measure of the performance of a solver can be given by

$$r_s(t) = \frac{1}{n_p} size\{p \in P \mid r_{p,s} \leq t\}$$

where $r_s(t)$ is the probability that for solver s, the ratio of performance $r_{p,s}$ is within a factor t of the best ratio. A plot of $r_s(t)$ for the different solvers will give interesting characteristics of the solvers. Please note that $r_s(1)$ gives the proportion of problems where s is winning over the other solvers.

For many of our problem instances the reported solution time is *very* small, and the solvers report 0. All these instances are removed from this comparison. This is not necessarily a drawback, as the remaining problems instances presumably are the most interesting ones.

Figure 2 shows performance profiles for the following 6 methods:

- **ACW 60** – Tabu Search with adaptive clause weights
- **TS 60** – Tabu Search without adaptive clause weights
- **PMA 60 TS** – Tabu Search without adaptive clause weights, but with PMA
- **CON ACW** – Constructive Search, followed by TS
- **CON LS** – Constructive Search, followed by Steepest Ascent
- **CON** – Constructive Search – No LS

The allotted solution times are 60 seconds per problem instance.

Of the original 5485 problem instances, 352 were left after removing those where at least one of the solvers reported a solution time of 0 seconds. Of these remaining problems there are 299 where not all the solvers find the same solution value.

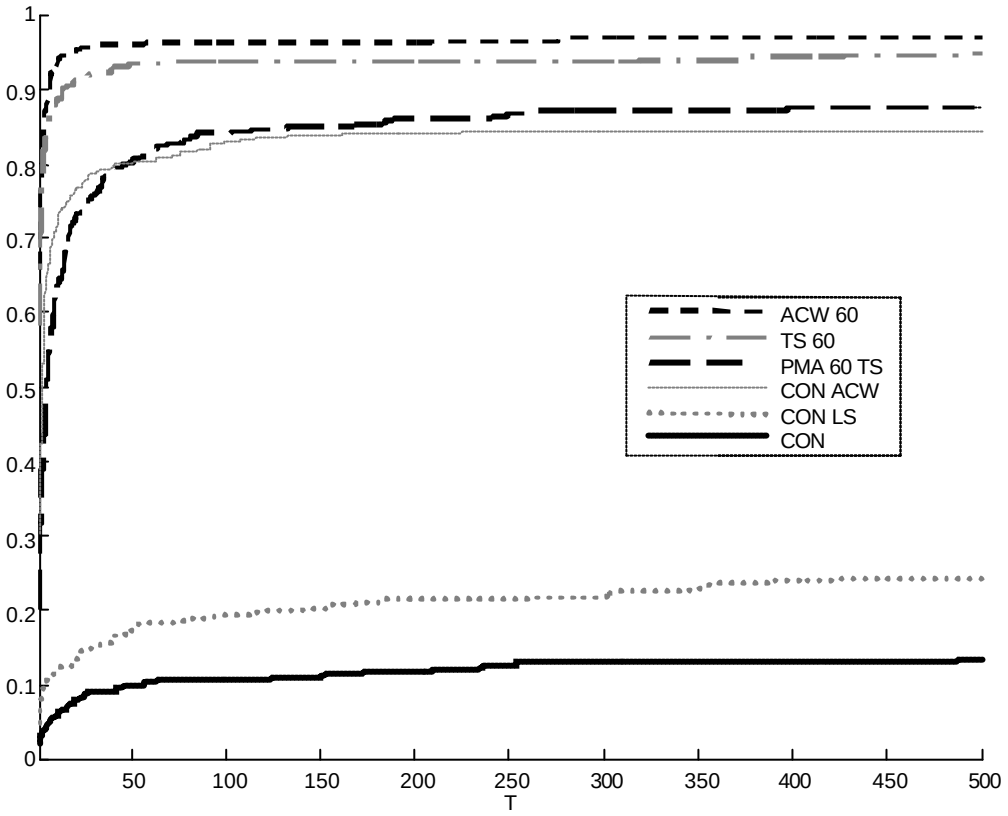

Figure 2. Performance Profiles

As can be seen, the method **ACW 60** is the best. It is of interest to note that when solution time is small (less than a factor 40 from the best solver on each particular instance), is that **CON ACW** is better than **PMA 60 TS**, while for longer solution times **PMA 60 TS** is better.

7 Conclusions

We have shown the value of certain types of adaptive memory to improve the performance of heuristics, both iterative and constructive. Our results compare very favorably to previous published results, and are significantly better than those obtained by exact solvers (XPRESS and CPLEX)

For BOOP, we have achieved the best results by using a tabu search based heuristic, augmented by an adaptive move evaluation function, and adaptive clause weights. Very good results are also

obtained for constructive heuristics augmented by the learning schemes of *PAM* (*Persistent Attractiveness Measure*) and *MCV* (*Marginal Conditional Validity*).

We also show that our approach can be applied to Weighted Maximum Satisfiability problems by transforming them into (larger) BOOP problems, and that without specialization to the W-MAX_SAT setting we obtain results comparable to those of the better specialized methods from the literature.

References

Cook, S. A. (1971). "The complexity of theorem-proving procedures". Proceedings of the Third ACM Symposium on Theory of Computing, pp 151-158.

Davoine, T., P. L. Hammer and B. Vizvári. (2003). "A Heuristic for Boolean optimization problems". *Journal of Heuristics* 9, pp 229-247.

Dolan, E. D. and J. J. Moré. (2001). "Benchmarking optimization software with performance profiles". Preprint ANL/MCS-P861-1200, Mathematics and Computer Science Division, Argonne National Laboratory.

Feo, T. A. and M. G. C. Resende. (1989). "A probabilistic heuristic for a computationally difficult set covering problem". *Operations Research Letters* 8, pp. 67-71.

Gendreau, M. (2003). "An Introduction to Tabu Search". In: **Handbook of Metaheuristics**, Kluwer Academic Publishers. Eds.: F. Glover and G. Kochenberger.

Glover, F. (1989). "Tabu Search – Part I". *ORSA Journal on Computing* 1, pp. 190-206.

Glover, F. (2000). "Multi-Start and Strategic Oscillation Methods – Principles to Exploit Adaptive Memory". In: **OR Computing Tools for Modeling, Optimization and Simulation: Interfaces in Computer Science and Operations Research**. Eds.: M. Laguna and J. L. González-Velarde, pp. 1-24.

Glover, F. and G. Kochenberger. (1996). "Critical Event Tabu Search for Multidimensional Knapsack Problems", In: **Meta Heuristics: Theory and Applications**, Kluwer Academic Publishers. Eds.: I. H. Osman and J. P. Kelly, pp. 407 – 427.

Glover, F. and M. Laguna. (1997). **Tabu Search**. Kluwer Academic Publishers.

Hvattum, L. M., A. Løkketangen and F. Glover. (2003). "Adaptive Memory Search for Boolean Optimization Problems". Forthcoming in the special issue of *Discrete Applied Mathematics on boolean and pseudo-boolean functions*.

Løkketangen, A. og F. Glover. (1996). "Probabilistic Move Selection in Tabu Search for 0/1 Mixed Integer Programming Problems". In: **Metaheuristics: Theory and Applications**, Eds.: I. H. Osman and J. P. Kelly, Kluwer, pp 467-488.

Løkketangen, A. og F. Glover. (1997). "Surrogate Constraint Analysis - New Heuristics and Learning Schemes for Satisfiability Problems". In: **Satisfiability Problem: Theory and Applications**. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Vol. 35. AMS. Eds.: D. Du, J. Gu, and P. M. Pardalos.

Resende, M. G. C., L. S. Pitsoulis, and P. M. Pardalos. (1997). "Approximate solution of weighted MAX-SAT problems using GRASP". In: **Satisfiability problem: Theory and Applications**. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Vol. 35. AMS. Eds.: D. Du, J. Gu, and P.M. Pardalos, pp. 393-405.

Resende, M. G. C. and C. C. Ribeiro. (2003). "GRASP with path-relinking: Recent advances and applications". Submitted to **Metaheuristics: Progress as Real Problem Solvers**, Kluwer Academic Publishers. Eds.: T. Ibaraki, K. Nonobe and M. Yagiura.

Shang, Y. (1997). "Global Search Methods for Solving Nonlinear Optimization Problems". Ph.D. Thesis, Dept. of Computer Science, Univ. of Illinois, USA.