Solving Max-SAT as weighted CSP *

Simon de Givry[1], Javier Larrosa[2], Pedro Meseguer[3], and Thomas Schiex[1]

[1] INRA, Toulouse, France
{degivry|tschiex}@toulouse.inra.fr
[2] Dep. LSI, UPC, Barcelona, Spain
larrosa@lsi.upc.es
[3] IIIA-CSIC, Campus UAB, Bellaterra, Spain
pedro@iiia.csic.es

Abstract. For the last ten years, a significant amount of work in the constraint community has been devoted to the improvement of complete methods for solving soft constraints networks. We wanted to see how recent progress in the weighted CSP (WCSP) field could compete with other approaches in related fields. One of these fields is propositional logic and the well-known Max-SAT problem. In this paper, we show how Max-SAT can be encoded as a weighted constraint network, either directly or using a dual encoding. We then solve Max-SAT instances using state-of-the-art algorithms for weighted Max-CSP, dedicated Max-SAT solvers and the state-of-the-art MIP solver CPLEX. The results show that, despite a limited adaptation to CNF structure, WCSP-solver based methods are competitive with existing methods and can even outperform them, especially on the hardest, most over-constrained problems.

1 Introduction

Since the eighties, both constraint satisfaction and boolean satisfiability have been the topic of intense algorithmic research. In both areas, the main problem is to assign values to variables in such a way that no forbidden combination of values appears in the solution.

Using closely related techniques such as backtrack search, local consistency enforcing (aka constraint propagation), and constraint learning, both areas have produced generic complete solvers which have been applied to a large range of problems. In the SAT domain, one major area of application is electronic design automation (EDA) with problems that range from formal validation to routing.

Quite early in the history of constraint satisfaction, the issue of infeasible problems has been addressed [18, 4, 7]. Most of the recent algorithmic work has focused on the so-called WCSP (weighted constraint satisfaction problem) where the aim is to find an assignment that minimizes the sum of weights associated with the constraints violated by the assignment. Complete algorithms

* This research is partially supported by the French-Spanish collaboration PICASSO 05158SM - Integrated Action HF02-69. The second and third authors are also supported by the REPLI project TIC-2002-04470-C03.

that address these problems rely on variants of depth-first branch and bound search using dedicated lower bounds. Since the early algorithms of [6], huge improvements have been obtained using increasingly sophisticated lower bounds. Recently [20, 13, 15], it has been possible to simplify and strengthen the definition of these lower bounds by expressing them as a result of the enforcing of a local consistency property.

In the SAT area, the similar issue of infeasible problems has been considered more recently, leading to increasing interest in the (weighted) Max-SAT problem. In Max-SAT, the problem is to assign values to boolean variables in order to maximize the number of satisfied clauses in a CNF formula. Max-SAT has applications in routing problems [26] and is also closely related to the Max-CUT problem (other applications are described in [10]). When turned into a "yes-no" problem by adding a goal k representing the number of clauses to be satisfied, Max-SAT and even Max-2SAT (where clauses only involve 2 variables) are NP-complete and more precisely MAX-SNP-complete. Both problems have been intensively studied on the theoretical side.

The problem hardness has also been studied empirically in [27]. This phase transition analysis of random Max-3SAT problems shows that using the usual fixed length random SAT model, the Max-3SAT problem does not show an easy/hard/easy pattern as the clauses/variables ratio increases but an easy/hard pattern: the empirical complexity of Max-3SAT increases as this ratio increases.

As usual for solving NP-complete problems, either complete or incomplete algorithms can be used to tackle the problem. There is a long list of incomplete algorithms for Max-SAT. In this paper, we only deal with complete algorithms, that identify provenly optimal solutions in finite time. Two main classes of complete algorithms have been proposed based either on variations on the Davis-Putnam-Logemann-Loveland (DPLL) approach for satisfiability or on 0/1 linear programming models. Along the DPLL line, current solvers use pseudo-boolean formulae to model Max-SAT [2, 25, 5, 1]. A pseudo-boolean (PB) formula is a linear inequality on boolean variables which can model clauses but also more complex constraints such as cardinality constraints [24]. One of the first algorithm in this line is OPBDP [2]. More recently, PBS (Pseudo Boolean Solver) [1] was designed based on the Chaff SAT solver [16].

Also based on the DPLL algorithm, a more theoretical line of research has tried to define complete algorithms that would provide non naive guaranteed worst-case upper bounds on time complexity based on the overall length L of the input formula or the number K of its clauses. While most of this work is essentially theoretical and never reaches the level of actually implementing the algorithms presented, one exception is [9] which implemented a Max-2SAT solver that achieves worst case upper bounds of $O(1.0970^L)$ and $O(1.2035^K)$[1].

Another natural approach to solve the Max-SAT problem is to model it as a mixed integer linear program (MIP). This linear program can then be solved

[1] These theoretical results have been very slightly improved since in [8], but no corresponding implementation is available.

directly by a dedicated MIP solver such as ILOG CPLEX. Note that dedicated branch and cut algorithms above MINTO have also been defined [3].

In this paper, we model the Max-SAT problem as a weighted CSP. Because most of the existing work on WCSP has been done on binary WCSP, we consider two possible approaches: *i*) a direct conversion of clauses into constraints, which produces non-binary problems and requires the solver to be adapted to deal with them, and *ii*) a dual (binary) formulation as proposed in [14].

To solve converted Max-SAT instances, we use adapted versions of the WCSP solvers defined in [15] which are depth-first branch and bound algorithms that maintain some level of local consistency during search.

For comparison purposes, we also solve the original Max-SAT problems using two dedicated solvers (OPBDP and PBS), a pure Max-2SAT dedicated solver (`max2sat` by J. Gramm) and a general MIP solver (CPLEX). The results of our experiments show that despite the fact that our generic WCSP code ignores most clauses properties, uses classical CSP data-structures instead of specialized clauses data-structures and relies on simple variable ordering, it can outperform existing pseudo-boolean solvers, commercial MIP solvers and is even competitive with a code restricted to Max-2SAT. The good performances of our algorithm are especially obvious on problems with high clauses/variables ratio which is probably related to the strength of the lower bound induced by (full directional) soft arc consistency. The results we get are consistent with what has been observed in classical CSP when comparing arc consistency maintenance to eg. forward-checking: the overhead for enforcing higher level of consistencies may slow down the algorithm on relatively simple problems but provides both highly increased performances and limited variability in the cpu-times on hard problems.

2 Notation and definitions

2.1 Sat and (weighted) Max-SAT

In propositional logic a variable v_i may take values 0 (for false) or 1 (for true). A literal ℓ_i is a variable v_i or its negation $\bar{v}_i$. A clause C_j is a disjunction of literals. A logical formula in conjunctive normal form (CNF) is a conjunction of clauses. Given a logical formula in CNF, the SAT problem considers finding an assignment of the variables that satisfies the formula, or getting a proof that no such assignment exists.

When a logical formula is unsatisfiable, the Max-SAT problem tries to find the assignment that satisfies as many clauses as possible. In the rest of the paper, we assume that each clause C_j is associated with a positive weight w_j. In this case, the weighted Max-SAT problem looks for the assignment that maximizes the sum of weights of satisfied clauses.

2.2 Weighted CSP

A *constraint satisfaction problem* (CSP) is a triple $(\mathcal{X}, \mathcal{D}, \mathcal{C})$, where $\mathcal{X}$ is a set of variables $\{x_1, \dots, x_n\}$, $\mathcal{D}$ is a collection of domains $\{D_1, \dots, D_n\}$ and $\mathcal{C}$ is a set of constraints $\{c_1, \ldots, c_e\}$. Each variable $x_i \in \mathcal{X}$ takes values in the finite domain D_i. A constraint c_i is defined over a subset of variables $var(c_i)$, and $rel(c_i) \subset \prod_{j \in var(c_i)} D_j$ specifies the value tuples permitted by c_i. $var(c_i)$ is called the *scope* of the constraint and $|var(c_i)|$ is its *arity*. A tuple t is an ordered set of values assigned to the ordered set of variables $\mathcal{X}_t \subseteq \mathcal{X}$. For a subset B of $\mathcal{X}_t$, the projection of t over B is noted $t \downarrow_B$. A *solution* is a tuple involving all variables that satisfies every constraint.

Following [13], we define Weighted CSP (WCSP) as a specific subclass of valued CSP [21], where constraint costs can take their values in the set $\{0, 1, \ldots, k\}$ and k represents a maximum acceptable cost, $k \in \{1, \ldots, \infty\}$. The combination of two costs is done using bounded addition denoted $\oplus$ and defined as $a \oplus b = \min\{k, a + b\}$.

A WCSP is then a tuple $(k, \mathcal{X}, \mathcal{D}, \mathcal{C})$. $\mathcal{X}$ and $\mathcal{D}$ are variables and domains, as in standard CSP. $\mathcal{C}$ is the set of constraints as cost functions. A constraint c_i assigns costs to assignments to variables $var(c_i)$ (namely, $c_i : \prod_{j \in var(c_i)} D_j \rightarrow \{0, \ldots, k\}$). In the rest of the paper, we assume the existence of a unary constraint for every variable and also a zero-arity constraint $c_\varnothing$ (if no such constraint is defined, we can always define *dummy* ones $c_i(a) = 0, \forall a \in D_i$ and $c_\varnothing = 0$).

When a constraint c assigns cost k or above to a tuple t, it means that c forbids t, otherwise t is permitted by c with the corresponding cost. The *cost* of a tuple t, noted $\mathcal{V}(t)$, is the bounded sum over all applicable costs,

$$\mathcal{V}(t) = \bigoplus_{c_i \in \mathcal{C},\ var(c_i) \subseteq \mathcal{X}_t} c_i(t \downarrow_{var(c_i)})$$

Tuple t is *consistent* if $\mathcal{V}(t) < k$. The usual task of interest is to find a complete consistent assignment with minimum cost, which is NP-hard.

3 Modeling and solving the Max-SAT problem

3.1 As a Pseudo-Boolean Problem

A *pseudo-boolean* (PB) problem is a special case of CSP where all variables share a bi-valued domain $D = \{0, 1\}$ and constraints are linear inequalities. A PB constraint takes the form,

$$\sum_{i=1}^{n} a_{ij} v_i \quad \begin{matrix} \leq \\ \geq \\ = \end{matrix} \quad b_j, \qquad a_{ij}, b_j \in \boldsymbol{Z}$$

A Max-SAT instance with r clauses and n variables can be translated into a PB problem as follows. We first introduce r extra variables y_j (one per clause) and replace clause C_j by the relaxed formula $\neg C_j \rightarrow y_j$ which forces y_j to 1 when C_j is violated. This formula can directly be represented by a clause and translated to a pseudo-boolean formula denoted $RPB(C_j)$ by replacing each occurrence of $\bar{v_i}$ by $(1 - v_i)$ and the $\vee$ operator by $+$.

There is an extra constraint $\sum_{j=1}^{r} w_j y_j \leq K$, where $K \in [W, \ldots, 0]$ and $W = \sum_{j=1, j \neq m}^{r} w_j$ such that $w_m = \max_j \{w_j\}, j = 1, \ldots, r$. This constraint bounds the maximum violation cost.

As example, the set of clauses $\{\bar{v}_1, \bar{v}_2, v_1 \vee v_2\}$ (all with the same unit weight) generates a PB problem with five variables and four constraints,

$$(1-v_1)+y_1 \geq 1 \qquad (1-v_2)+y_2 \geq 1 \qquad v_1+v_2+y_3 \geq 1 \qquad y_1+y_2+y_3 \leq 2$$

This translation is the most compact encoding we could think of. In their papers, the authors of PBS [1] use a less compact encoding where a stronger formulation $\neg C_j \leftrightarrow y_j$ is used instead of $\neg C_j \rightarrow y_j$. This encoding was also tested with PBS but provided similar results and it is therefore ignored in the rest of this paper.

The PB problem is solved combining DPLL and constraint propagation. DPLL is used on the r constraints $RPB(C_j)$ which have a clausal structure. When a y variable becomes instantiated by DPLL, this is propagated through remaining constraints as follows. Assuming that $\{y_1, \ldots, y_p\}$ is the subset of y variables instantiated, if $\sum_{j=1}^{p} w_j y_j > K$ then this constraint is violated. Otherwise, all unassigned y_i such that $w_i > K - \sum_{j=1}^{p} w_j y_j$ must be fixed to 0 (otherwise the constraint would be violated). This propagation may generate new unary clauses, which are again propagated by DPLL, etc. In this way, for a given K the problem is solved or is detected as unsolvable.

To find the minimum weight of unsatisfied clauses K should be minimized. Initially, K takes value W. Then, either a depth-first branch and bound approach (OPBDP) or an iterative approach (PBS) can be used. With iterative resolving, clause learning can naturally speedup the solving process.

3.2 As a Mixed ILP

An *integer linear problem* (ILP) considers the minimization of a linear function of integer variables under linear constraints. Mixed ILP involve continuous and integer variables.

Given a Max-SAT instance with n variables and r clauses, it is translated into a Mixed ILP as follows. We use r extra *continuous* variables y_j, one per clause. Each clause C_j is encoded as the linear constraint $RPB(C_j)$ as in the previous case. Note that integrality constraints on y_i are useless since they only appear in one constraint, where all other variables are integer. The function to minimize is the weighted sum,

$$\min \sum_{j=1}^{r} w_j y_j \qquad (1)$$

As example, the set of clauses $\{\bar{v}_1, \bar{v}_2, v_1 \vee v_2\}$ generates the following ILP,

$$\begin{array}{lllll} \min & & y_1 + y_2 + y_3 & & \\ 1 - v_1 & & + y_1 & & \geq 1 \\ & 1 - v_2 & & + y_2 & \geq 1 \\ v_1 + & v_2 & & + y_3 & \geq 1 \end{array}$$

where $v_i \in \{0,1\}, i = 1,2, y_j \in [0,1], j = 1,2,3$.

The MIP is solved by computing its linear relaxation, obtained by replacing the integrality requirements by simple bounds, $0 \leq v_i \leq 1, \quad i = 1, \ldots, n$. If the solution of the linear relaxation has integer v variables, it is compared with the best solution found so far. If the solution has fractional v variables, one v_i is chosen for branching, generating two subproblems (one with $v_i = 0$, the other with $v_i = 1$), which are solved by the same method. A number of other sophisticated techniques can be involved in this process [3].

3.3 As a WCSP

Primal encoding. A weighted Max-SAT instance is directly expressed as a WCSP as follows. WCSP variables are the logical variables of the Max-SAT instance, with the domain $\{0,1\}$. Each clause C_j with weight w_j generates a cost function, which assigns cost 0 to those tuples satisfying C_j, and assigns cost w_j to the only tuple violating C_j. When two cost functions involve the same variables, they can be added together. The WCSP solution, the total assignment with minimum cost, corresponds to the solution of Max-SAT.

The algorithms used to solve the WCSP are specific depth-first branch and bound algorithms. Such algorithms rely on an upper bound ub on the cost of the optimal solution and a lower bound lb on the cost of the optimal extension of the current assignment. The cost of the currently best known solution provides ub. An ad-hoc mechanism provides lb. The current branch is pruned as soon as $lb \geq ub$.

Given the current assignment, we have an associated WCSP subproblem where $S(ub)$ is the valuation structure, $c_\varnothing$ is the current lower bound, and current constraints are the constraints inherited from the parent node projected according with the last assigned variable. To process this subproblem, a given soft local consistency property is enforced at each node. As in the classical CSP case, local consistency enforcing performs local computations that preserve the semantics of the problem, prune infeasible values (whose use would provenly lead to cost greater than or equal to ub) and may increase $c_\varnothing$ (see [19, 13, 15]).

The different levels of local consistencies we have considered are node consistency (NC*), arc consistency (AC*), directional arc consistency (DAC*) and full DAC (FDAC*), for binary problems as defined in [15]. These local consistencies can be enforced in time $O(nd)$ (NC*), $O(n^2d^3)$ (AC*), $O(ed^2)$ (DAC*) and $O(end^3)$ (FDAC*), where e is the number of constraints, n the number of variables and d the maximum domain size.

Among these local consistencies, NC* is the weakest and FDAC* is the strongest. DAC* and AC* are incomparable between them, both are stronger than NC* but weaker than FDAC* [15].

Each form of local consistency defines a solver which maintains the corresponding property. For instance, MFDAC is the branch and bound algorithm that maintains FDAC* during search. Since the Max-SAT translation produces non-binary constraints, we straightforwardly extend the previous local consistencies to the non-binary case as follows: a problem is considered as locally consistent iff it is locally consistent with respect to unary and binary constraints (other constraints are delayed until their arity is reduced by further assignments).

Dual encoding. An alternative modeling is the dual formulation [14]. There is a variable x_i for each clause C_i. The domain of x_i is the set of possible assignments to the logical variables in C_i. When x_i takes one of its domain values, it represents the fact that the logical variables of C_i have been assigned accordingly. There is a unary constraint on each variable x_i. This constraint assigns cost 0 to each domain value satisfying clause C_i, and assigns cost w_i to the only domain value violating C_i (namely, the assignment which dissatisfies every literal in C_i). There is a binary constraint between every two variables x_i and x_j corresponding to clauses C_i and C_j sharing logical variables. This constraint gives infinite cost to pairs formed by domain values which assign different logical values to the shared logical variables, and cost 0 to every other pair. The solution of the dual problem corresponds to the solution of the primal problem, which produces a solution for Max-SAT. This formulation produces a binary encoding, so that existing WCSP algorithm implementations can be directly applied.

Heuristics. Each time a new variable has to be assigned, the algorithm looks for variables with one feasible value and selects one of them first. If all variables have two values, a variable selection heuristic must be used. Since all domains have the same cardinality, a smallest-domain criterion may not be used.

We denote $T_j = \prod_{i \in var(c_j)} D_i$ the set of tuples valuated by constraint c_j. W_j is the average cost given by c_j, defined as $W_j = \frac{1}{|T_j|} \sum_{t \in T_j} c_j(t)$. We define $Z_i = \sum_{j \in \mathcal{C}, i \in var(c_j)} W_j$. It measures the average cost in which variable x_i is involved.

A natural heuristic would be to select the variable with the highest Z_i, since the assignment of such a variable is likely to produce high costs and, consequently, anticipate pruning. The problem of this heuristic is its computational cost. Unless we can exploit the semantics of the constraints to compute W_j efficiently, its cost is $O(e \times d^r)$ where e is the number of constraints, d is the largest domain size (2 in MaxSAT) and r is the problem arity. We found this heuristic very informative but it was not cost effective. Thus, we made an approximation.

Let Z_i^k be the contribution of k-arity constraints to Z_i. The approximate heuristic selects the variable with highest $Z_i^1 + Z_i^2$, which has cost $O(e_1 d + e_2 d^2)$ with e_1 and e_2 being the number of unary and binary constraints. Only when all variables have $Z_i^1 + Z_i^2$ equal to zero, we discriminate using Z_i^3, which has cost $O(e_3 d^3)$ (this is rarely needed, typically at nodes near to the root). The heuristics is used dynamically, all values are computed at each node according to the current subproblem. Once the variable has been selected, the value with the lowest unary cost is assigned first.

4 Empirical results

Here we report the results of an empirical evaluation of WCSP techniques compared to state-of-the-art pseudo-boolean and ILP solvers on a set of benchmarks.

4.1 Benchmarks

The benchmarks are composed of:

- unsatisfiable instances of the 2^{nd} DIMACS Implementation Challenge [12]: random 3-SAT instances (*aim* and *dubois*), pigeon hole problem (*hole*), 2-coloring problems (*pret*) and random SAT instances (*jnh*) with variable length clauses (2-14 literals per clause).
- extended *jnh* instances weighted using uniformly distributed integer weights between 1 and 1,000 [17].
- random 2-SAT and 3-SAT instances created by Allen van Gelder *mkcnf* generator [23]. The generation parameters are the clause length l, the number of variables n and the number of clauses r. We generated a set of instances with $(l, n, r) \in \{2, 3\} \times \{40, 80\} \times \{100, 200, \cdots, 3000\}^2$. For each parameter configuration, 10 instances were generated. Note that this generator prevents duplicate or opposite literals in clauses but not duplicate clauses.

We assume unit clause weights for all instances, except for the extended *jnh* instances.

We experimented with the 4 types of local consistency (NC*, AC*, DAC* and FDAC*) and 2 problem encodings (primal and dual). Among the 8 alternatives, maintaining FDAC* with the primal encoding was the obvious best choice (it was typically much better than any of the others, and never much worse). For clarity in the analysis, we essentially report results on MFDAC. Our implementation of MFDAC [15] (C code) is compared to four solvers:

- Pseudo-boolean optimization solver OPBDP v1.1 [2] (C++ code).
- Pseudo-boolean solver PBS v0.2 [1] (Sun binary).
- Max-2SAT solver `max2sat` [9] (Java code), *only* for 2-SAT problems.
- Commercial ILP solver CPLEX v8.1.0 [11] (Sun binary).

We used default configuration parameters for all the solvers, except for PBS which used VSIDS decision heuristic (as advised by the authors) and for CPLEX whose stopping criterion was set to $gub - glb \leq 0.999$ to ensure completeness.

[2] Only 2000 for 80 variables instances.

In order to reduce the search effort for all algorithms and put ourselves in a realistic situation, we used *walksat* [22] with default parameters (10 runs of 100000 flips) to compute a first upper bound. This upper bound was injected in all algorithms using either available configuration parameters or by modifying the `max2sat` code to access an internal parameter. In the case of the DIMACS instances, *walksat* always found the optimum, so the complete solvers had just to prove optimality. In the case of extended *jnh* instances, we used the optimum values from [17]. Because of this preprocessing step, CPLEX focused on optimality proof rather than improving integer solutions (*set mip emphasis 2*). Note that in general, only few Gomory fractional cuts were added by CPLEX. All the experiments, except for CPLEX, ran on a Sun Enterprise 250 (UltraSPARC-II 400MHz, 640 Megabytes at 100 MHz). CPLEX ran on a Sun Blade 1000 (UltraSPARC-III 750MHz, 1 Gigabytes) and a ratio (370/198 from SPEC CPU 2000 results) was applied for time measurements.

4.2 Results

The results for DIMACS benchmarks are shown in Table 1 and 2. For each instance, the table lists the instance name, the number of variables ($|V|$), the number of clauses ($|C|$), the optimum (minimization of the clause violation), and the total cpu time in seconds (rounded downwards) for the various solvers. In the case of Table 2, there are two parts corresponding to the original *jnh* instances and the extended *jnh*. In both tables, the last two lines give the number of instances completely solved in less than 600 seconds and the average time for all the instances (if unsolved, 600 is counted). Note that all these problems have an extremely low optimum value, which means that they are near the satisfiability complexity peak. As observed by [27], these instances are hard as SAT instances but easy as Max-SAT instances (the hardest instances have higher clauses to variables ratio which causes high optimum values)

In Table 1, MFDAC was able to solve almost half of the instances while PBS solved them all. We do not report larger instances ($|V| > 100$) where PBS was the only successful algorithm (except for CPLEX on *hole*10). PBS contains several SAT-solver sophistications like conflict diagnosis and clause recording which make it efficient on instances near the transition phase. In comparison, OPBDP is much simpler. But its specific design for SAT (dedicated SAT rules and data structures) makes the difference with MFDAC: OPBDP can visit up to 3 times more nodes per seconds than MFDAC. CPLEX solved the same number of problems than MFDAC and is the best choice for the structured (highly symmetric) pigeon-hole problems [3].

The original unsatisfiable *jnh* instances are best solved using OPBDP and MFDAC (see table 2, first part) which solved all the instances. MFDAC was 4.6 times slower than OPBDP and explored 6-7 times more nodes than OPBDP. We conjecture that our naive approach for tackling non-binary constraints is

[3] Pigeon-hole problems have very efficient encoding as pseudo-boolean formulae and CPLEX may possibly detect this even if a clausal formulation is used.

Name	$\|V\|$	$\|C\|$	Opt	MFDAC	OPBDP	PBS	CPLEX
aim-100-1_6-no-1	100	160	1	-	595	0	71
aim-100-1_6-no-2	100	160	1	-	92	0	23
aim-100-1_6-no-3	100	160	1	-	-	0	11
aim-100-1_6-no-4	100	160	1	-	-	0	2
aim-100-2_0-no-1	100	200	1	-	0	0	-
aim-100-2_0-no-2	100	200	1	-	54	0	-
aim-100-2_0-no-3	100	200	1	-	60	0	-
aim-100-2_0-no-4	100	200	1	-	33	0	-
aim-50-1_6-no-1	50	80	1	5	0	0	0
aim-50-1_6-no-2	50	80	1	0	0	0	0
aim-50-1_6-no-3	50	80	1	3	0	0	0
aim-50-1_6-no-4	50	80	1	0	0	0	0
aim-50-2_0-no-1	50	100	1	1	0	0	0
aim-50-2_0-no-2	50	100	1	0	0	0	4
aim-50-2_0-no-3	50	100	1	0	0	0	3
aim-50-2_0-no-4	50	100	1	1	0	0	0
dubois20	60	160	1	407	70	0	-
dubois21	63	168	1	-	145	0	-
dubois22	66	176	1	-	298	0	-
dubois23	69	184	1	-	596	0	-
dubois24	72	192	1	-	-	0	-
dubois25	75	200	1	-	-	0	-
dubois26	78	208	1	-	-	0	-
dubois27	81	216	1	-	-	0	-
dubois28	84	224	1	-	-	0	-
dubois29	87	232	1	-	-	0	-
dubois30	90	240	1	-	-	0	-
hole06	42	133	1	0	1	0	0
hole07	56	204	1	7	27	1	0
hole08	72	297	1	123	-	10	0
hole09	90	415	1	-	-	69	0
pret60_25	60	160	1	532	77	0	-
pret60_40	60	160	1	530	76	0	-
pret60_60	60	160	1	531	77	0	-
pret60_75	60	160	1	530	77	0	-
$\|solved\|$	35	35	35	16	24	**35**	16
Average	72.1	172.8	1.0	402.0	253.9	**2.4**	329.2

Table 1. DIMACS unsatisfiable instances. Time in seconds. A "-" means the problem was not solved in less than 600 seconds.

Name	$\|V\|$	$\|C\|$	Opt	MFDAC	OPBDP	PBS	CPLEX	Opt	MFDAC	OPBDP	PBS	CPLEX
jnh04	100	850	1	0	0	0	38	95	2	0	0	112
jnh05	100	850	1	0	0	0	3	183	3	0	0	60
jnh06	100	850	1	0	0	0	39	99	3	0	0	84
jnh08	100	850	2	11	**1**	13	33	462	11	1	0	157
jnh09	100	850	2	5	**1**	18	274	333	89	0	2	-
jnh10	100	850	1	0	0	0	5	85	4	1	0	16
jnh11	100	850	1	0	0	0	32	172	26	0	0	439
jnh13	100	850	2	12	**1**	16	28	109	4	0	0	20
jnh14	100	850	2	10	**1**	19	170	101	11	0	0	79
jnh15	100	850	2	12	**2**	20	86	206	9	1	0	89
jnh16	100	850	1	4	8	0	490	6	23	8	0	190
jnh18	100	850	1	0	1	0	31	130	15	2	0	184
jnh19	100	850	2	14	**1**	40	162	166	12	0	0	97
jnh202	100	800	1	0	0	0	2	68	0	0	0	8
jnh203	100	800	1	0	0	0	13	39	8	0	0	21
jnh208	100	800	1	0	0	0	8	79	7	0	0	35
jnh211	100	800	2	14	0	14	34	259	13	0	0	31
jnh214	100	800	1	0	0	0	7	75	3	0	0	34
jnh215	100	800	1	0	0	0	18	88	15	0	0	46
jnh216	100	800	1	0	1	0	8	12	1	1	0	32
jnh219	100	800	1	0	1	0	34	82	12	1	0	46
jnh302	100	900	4	241	**76**	-	-	395	17	0	1	114
jnh303	100	900	3	247	**37**	-	-	351	35	2	**1**	326
jnh304	100	900	3	31	**7**	207	150	321	3	0	0	92
jnh305	100	900	3	59	**14**	-	183	742	65	**16**	148	-
jnh306	100	900	1	0	2	0	144	16	7	2	0	96
jnh307	100	900	3	25	**11**	121	130	540	34	**1**	3	278
jnh308	100	900	2	124	**1**	17	82	130	3	0	0	60
jnh309	100	900	2	3	0	15	26	276	3	0	0	75
jnh310	100	900	3	69	**7**	295	173	463	18	**3**	14	426
$\|solved\|$	30	30	30	**30**	**30**	27	28	30	**30**	**30**	**30**	28
Average	100.0	851.7	1.7	29.4	**6.3**	86.8	120.6	202.8	15.2	**1.9**	5.9	148.7

Table 2. JNH instances with unit clause weights first and with random clause weights next. Time in seconds. A "-" means the problem was not solved in less than 600 seconds.

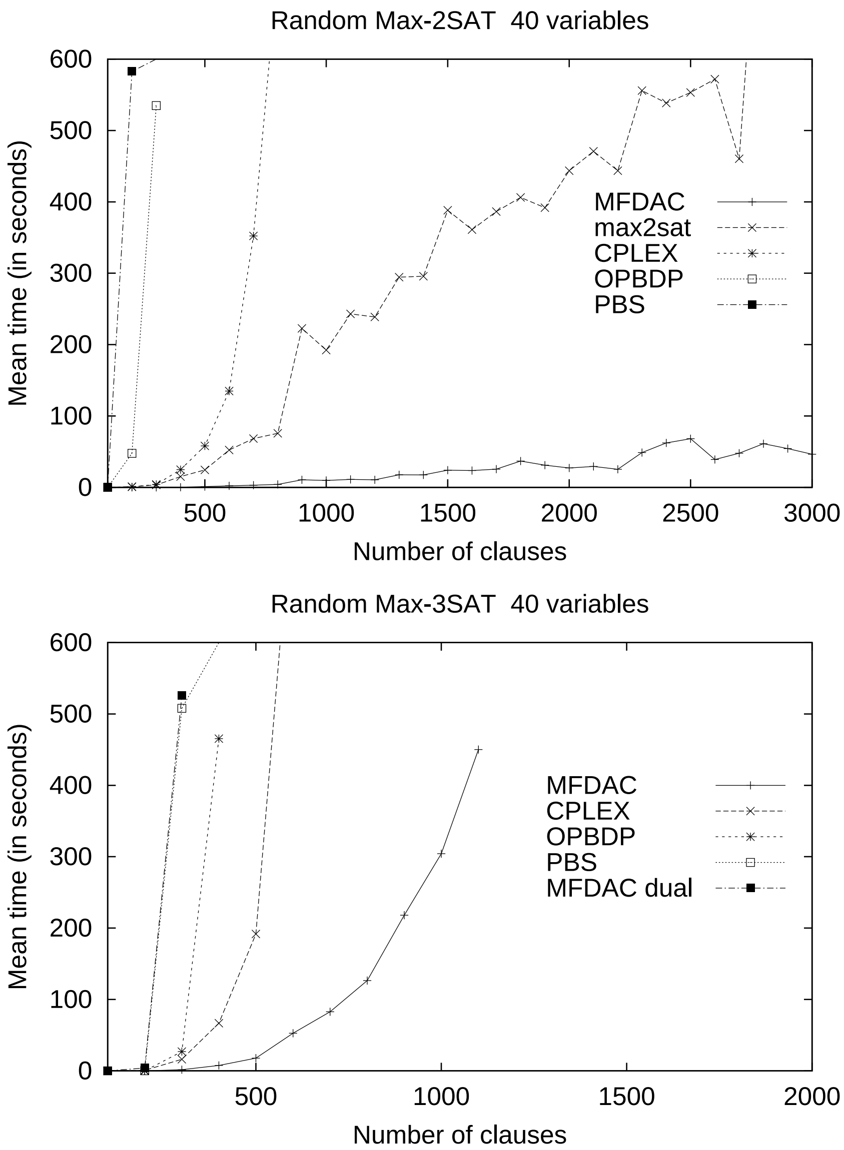

Fig. 1. Randomly-generated Max-2SAT and Max-3SAT instances with 40 variables.

responsible of this poor pruning behavior (recall that mean clause length in *jnh* is equal to 5). PBS is 3 (resp. 14) times slower than MFDAC (resp. OPBDP), mainly due to its bad performances on unsatisfiable instances with 3 or more violated clauses at the optimum. CPLEX was slower than PBS but seems more robust. Adding clause weights boosted all the solvers, except surprisingly for CPLEX ([17] observed exactly the opposite but they were not using an initial bound nor the same configuration parameters as us). OPBDP is still the best choice, but PBS (with equivalences) is now second best and 4.6 times faster than MFDAC.

With randomly-generated Max-kSAT instances and large clauses/variables ratios, MFDAC was by far the best as it is shown in Figure 1. PBS and OPBDP were unable to solve problems with more than 400 clauses. CPLEX exceeded the time limit for Max-2SAT (resp. Max-3SAT) with 40 variables when there are more than 800 (resp. 600) clauses. Considering Max-2SAT (40-variables), MFDAC solved all the 300 instances in less than 156 seconds each. `max2sat` was second best and solved 220 instances in less than 600 seconds each. At a clauses/variables ratio of only 5 (200/40), we got the following numerical results (mean time in seconds and in parenthesis, mean number of nodes and number of problems completely solved): MFDAC 0s(429nd,10), CPLEX 0.7s(89nd,10), `max2sat` 1.1s(257nd,10), OPBDP 47.7s(691887nd,10), PBS 582s(1139115nd,1). At a clauses/variables ratio of 10 (400/40), results were: MFDAC 0.1s(4013nd,10), `max2sat` 15.1s(6002nd,10), CPLEX 24.8s(4839nd,10), OPBDP > 600s(-,0) and PBS > 600s(-,0). For Max-3SAT (40-variables), instances become more difficult, the gap between MFDAC and the other solvers was reduced (CPLEX is 8-9 times slower than MFDAC for a c/v ratio of 10) but the efficiency order between solvers remained the same. With more variables (Max-2SAT 80-variables), CPLEX was faster than MFDAC if there are less than 400 clauses. And with Max-3SAT 80 variables, OPBPD was the winner, and MFDAC second best, for less than 400 clauses. When clauses/variables ratio decreases and when the clause length increases, instances are closer to the satisfiability threshold which is beneficial to SAT-based solvers such as OPDPB. In summary, MFDAC proved its superiority on large clauses/variables ratios. The speed-up obtained was even more important on problems with small length clauses.

5 Conclusion

On the Max-SAT problem, and despite a very limited adaptation of WCSP code to CNF propositional logic formula, we observe that the use of recent local consistency maintenance algorithms defined in [15] allows to reach a level of performance competitive with recent Max-SAT complete solvers and state-of-the art MIP solvers. This is especially true on the hardest problems, with a high clause/variable ratio.

The current MFDAC code used is far from being finely optimized code and is not specifically tuned to Max-SAT problems. For example, it does not specifically exploit the fundamental properties of CNF in propositional logic: the fact that domains are always binary and that dedicated data-structures can be used for CNF representation. The extension of the local consistency to non-binary constraints could also be improved by studying subproblems involving more than 2 variables.

These results show that there is a clear opportunity to study if recent local consistency notions like full directional arc consistency could be adapted to propositional logic and injected in existing Max-SAT solvers. More work is needed to see if these algorithms could be applied to other central combinatorial optimization problems such as Max-CUT or the Maximum Probable Explanation (MPE) problem in Bayesian networks.

Acknowledgements

We would like to thank Michel Correge and Michel Lemaître for letting us use CPLEX, for hosting the associated heavy benchmarking and associated nuisances. We would also like to thank Fadi Aloul and Jens Gramm for making their code available to us.

References

[1] Aloul, F., Ramani, A., Markov, I., and Sakallah, K. Pbs: A backtrack-search pseudo-boolean solver and optimizer. In *Symposium on the Theory and Applications of Satisfiability Testing (SAT)* (Cincinnati, (OH), 2002), pp. 346–353.

[2] Barth, P. A davis-putnam based enumeration algorithm for linear pseudo-boolean optimization. Tech. Rep. MPI-I-95-2-003, Max-Planck Institut Für Informatik, 1995.

[3] Borchers, B., Mitchell, J., and Joy, S. A branch-and-cut algorithm for MAX-SAT and weighted MAX-SAT. 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*. AMS, 1997, pp. 519–536.

[4] Borning, A., Mahert, M., Martindale, A., and Wilson, M. Constraint hierarchies and logic programming. In *Int. conf. on logic programming* (1989), pp. 149–164.

[5] Dixon, H., and Ginsberg, M. Inference methods for a pseudo-boolean satisfiability solver. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence (AAAI-02)* (2002), pp. 635–640.

[6] Freuder, E., and Wallace, R. Partial constraint satisfaction. *Artificial Intelligence 58* (Dec. 1992), 21–70.

[7] Freuder, E. C. Partial constraint satisfaction. In *Proc. of the 11^{th} IJCAI* (Detroit, MI, 1989), pp. 278–283.

[8] Gramm, J., Hirsch, E. A., Niedermeier, R., and Rossmanith, P. New worst-case upper bounds for MAX-2-SAT with application to MAX-CUT. Tech. Rep. TR00-037, Electronic Colloquium on Computational Complexity, 2000.

[9] Gramm, J., and Niedermeier, R. Faster exact solutions for max2Sat. In *Proceedings of the 4th Italian Conference on Algorithms and Complexity* (Rome, Italy, Mar. 2000), G. Bongiovanni, G. Gambosi, and R. Petreschi, Eds., vol. 1767 of *LNCS*, Springer, pp. 174–186.

[10] Hansen, P., and Jaumard, B. Algorithms for the maximum satisfiability problem. *Computing 44* (1990), 279–303.

[11] ILOG. Cplex solver 8.1.0. www.ilog.com/products/cplex, 2002.

[12] Johnson, D. S., and Trick, M. A., Eds. *Second DIMACS implementation challenge:cliques, coloring and satisfiability*, vol. 26 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*. AMS, 1996.

[13] Larrosa, J. On arc and node consistency in weighted CSP. In *Proc. AAAI'02* (Edmondton, (CA), 2002).

[14] Larrosa, J., and Dechter, R. On the dual representation of non-binary semiring-based CSPs. CP'2000 Workshop on Soft Constraints, Oct. 2000.

[15] Larrosa, J., and Schiex, T. In the quest of the best form of local consistency for weighted CSP. In *Proc. of the 18^{th} IJCAI* (Acapulco, Mexico, Aug. 2003). see www.inra.fr/bia/T/schiex/Export/ijcai03.pdf.

[16] Moskewicz, M., Madigan, C., Zhao, Y., Zhang, L., and Malik, S. Chaff: Engineering an efficient sat solver. In *38^{th} Design Automation Conference (DAC'01)* (June 2001), pp. 530–535.

[17] Resende, M., Pitsoulis, L., and Pardalos, P. Approximate solution of weighted max-SAT problems using GRASP. In *Satisfiability problem: Theory and Applications*, D. Du, J. Gu, and P. Pardalos, Eds., vol. 35 of *DIMACS Series on Discrete Mathematics and Theoretical Computer Science*. AMS, 1997, pp. 393–405.

[18] Rosenfeld, A., Hummel, R., and Zucker, S. Scene labeling by relaxation operations. *IEEE Trans. on Systems, Man, and Cybernetics 6*, 6 (1976), 173–184.

[19] Schiex, T. Arc cohérence pour contraintes molles. In *Actes de JNPC'00* (Marseille, June 2000).

[20] Schiex, T. Arc consistency for soft constraints. In *Principles and Practice of Constraint Programming - CP 2000* (Singapore, Sept. 2000), vol. 1894 of *LNCS*, pp. 411–424.

[21] Schiex, T., Fargier, H., and Verfaillie, G. Valued constraint satisfaction problems: hard and easy problems. In *Proc. of the 14^{th} IJCAI* (Montréal, Canada, Aug. 1995), pp. 631–637.

[22] Selman, B., Kautz, H., and Cohen, B. Noise strategies for improving local search. In *Proc. of AAAI'94* (Seattle, WA, 1994), pp. 337–343.

[23] van Gelder, A. Cnfgen formula generator. ftp://dimacs.rutgers.edu/pub/challenge/satisfiability/contributed/UCSC/instances, 1993.

[24] van Hentenryck, P., and Deville, Y. The cardinality operator: A new logical connective for constraint logic programming. In *Proc. of the 8^{th} international conference on logic programming* (Paris, France, June 1991).

[25] Whittemore, J., Kim, J., and Sakallah, K. SATIRE: A new incremental satisfiability engine. In *Proceedings of the 38th conference on Design automation* (Las Vegas, NV, June 2001), ACM, pp. 542–545.

[26] Xu, H., Rutenbar, R. A., and Sakallah, K. sub-SAT: A formulation for relaxed boolean satisfiability with applications in routing. In *Proc. Int. Symp. on Physical Design* (San Diego (CA), Apr. 2002).

[27] Zhang, W. Phase transitions and backbones of 3-SAT and maximum 3-SAT. In *Proc. of the 7^th International Conference on Principles and Practice of Constraint Programming (CP-01)* (Paphos, Cyprus, Nov. 2001), vol. 2239 of *LNCS*, Springer, pp. 153–167.