A Max-SAT Inference-Based Pre-processing for Max-Clique

Federico Heras and Javier Larrosa*

Universitat Politècnica de Catalunya,
Gran Capità 1-3,
08034 Barcelona, Spain

Abstract. In this paper we propose the use of two resolution-based rules for the Max-SAT encoding of the Maximum Clique Problem. These rules simplify the problem instance in such a way that a lower bound of the optimum becomes explicit. Then, we present a pre-processing procedure that applies such rules. Empirical results show evidence that the lower bound obtained with the pre-processing outperforms previous approaches. Finally, we show that a branch-and-bound Max-SAT solver fed with the simplified problem can be boosted several orders of magnitude.

Keywords: Max-SAT, Max-clique, Inference.

1 Introduction

Given an undirected graph, the *maximum clique Problem* (*Max-Clique*) calls for finding a maximum-sized complete subgraph, that is, a subgraph whose vertices are pairwise adjacent. The *Max-Clique* is a prominent combinatorial optimization problem with many applications such as bioinformatics [10, 23, 14] and computer vision [3] to name a few. From the recent literature, there are two types of algorithms to handle the Max-Clique problem. The first one is formed by *branch and bound* algorithms that solve the problem to optimality [11, 25, 22]. The second one is formed by *stochastic local search* solvers that cannot prove optimality, but empirical results show that they return quite accurate upper bounds [24, 5]. Both types of algorithms have a graph as input and they apply techniques that exploit the structure of such graph.

In this paper, we focus on the Max-SAT encoding of the Max-Clique problem and we exploit its properties. We introduce two simplification rules for the Max-Clique problem based on the resolution rule for Max-SAT [16] and we apply them in a preprocessing procedure. The result of the pre-process is an equivalent Max-SAT formula with an explicit lower bound of the optimum. Afterwards, we give the pre-processed instance to the state-of-the-art Max-SAT solver MINIMAXSAT [13]. Empirical results indicate that our pre-processing generates very powerful initial lower bounds. Besides, fetching the Max-SAT solver with the simplified formula can boost the search process in several problem instances.

* Research funded by project TIN2006-15387-C03-0.

H. Kleine Büning and X. Zhao (Eds.): SAT 2008, LNCS 4996, pp. 139–152, 2008.

The structure of this paper is the following. Section 2 introduces all the preliminary notation and concepts about Max-SAT and how to encode the Max-Clique problem as Max-SAT. Then, Section 3 presents the two simplification rules that are used in the pre-processing introduced in Section 4. Section 5 includes the experimental investigation and the related work can be found in Section 6. Finally, Section 7 presents some concluding remarks and points out our future work.

2 Preliminaries

2.1 The Max-SAT Framework

The following notation and terminology has been borrowed from [16]. In the sequel X is a set of boolean variables taking values over the set $\{\mathbf{t}, \mathbf{f}\}$, which stands for *true* and *false*, respectively. A *literal* is either a variable (e.g. x) or its negation (e.g. $\bar{x}$). We will use $l_1, l_2, l_3, \ldots$ to denote literals and $var(l)$ to denote the variable related to l (namely, $var(x) = var(\bar{x}) = x$). A *clause* $C = l_1 \vee l_2 \vee \ldots \vee l_k$ is a disjunction of literals such that $\forall_{1 \leq i,j \leq k,\ i \neq j}\ var(l_i) \neq var(l_j)$. The size of a clause, noted $|C|$, is the number of literals that it has. $var(C)$ is the set of variables that appear in C (namely, $var(C) = \{var(l) | l \in C\}$). We refer to a clause as *positive (negative)* if all its literals appear in the positive (negative) polarity. An assignment satisfies a clause iff it satisfies one or more of its literals. If variable x is instantiated to $\mathbf{t}$, literal x is satisfied and literal $\bar{x}$ is falsified. Similarly, if variable x is instantiated to $\mathbf{f}$, literal $\bar{x}$ is satisfied and literal x is falsified. The *empty clause*, noted $\square$, cannot be satisfied. Sometimes it is convenient to think of clause C as its equivalent $C \vee \square$. An *assignment* is an instantiation of a subset of X. The assignment is *complete* if it instantiates all the variables (otherwise it is partial).

A *weighted clause* is a pair (C, w) such that C is a classical clause and w is the cost of its falsification. In this paper we assume costs being natural numbers. A *weighted* formula in conjunctive normal form (CNF) is a set of weighted clauses. The cost of an assignment is the sum of weights of all the clauses that it falsifies.

As shown in [16], the De Morgan rule cannot be used in Max-SAT. Instead, the following rule should be repeatedly used until CNF is achieved:

$$(A \vee \overline{l \vee C}, w) \equiv \{(A \vee \bar{C}, w), (A \vee \bar{l} \vee C, w)\}$$

Following [16], we assume without loss of generality the existence of a known upper bound $\top$ of the optimal solution ($\top$ is a strictly positive natural number). A *model* is a complete assignment with cost less than $\top$. A Max-SAT instance is a pair $(\mathcal{F}, \top)$ and the task of interest is to find a model of minimum cost, if there is any. We say that two weighted formulas are equivalent, $\mathcal{F} \equiv \mathcal{F}'$, if the cost of their optimal assignment is the same or if neither of them has a model.

Observe that any weight $w \geq \top$ indicates that the associated clause *must be necessarily satisfied*. Thus, we can replace w by $\top$ without changing the problem. Consequently, we can assume all costs in the interval $[0..\top]$. A clause with weight $\top$ is called *mandatory* (or *hard*), otherwise it is called *non-mandatory* (or *soft*).

Let u and w be two costs. Their sum is defined as,

$$u \oplus w = min\{u + w, \top\}$$

in order to keep the result within the interval $[0..\top]$. If $u \geq w$, their subtraction is defined as,

$$u \ominus w = \begin{cases} u - w & : \quad u \neq \top \\ \top & : \quad u = \top \end{cases}$$

Essentially, $\ominus$ behaves like the usual subtraction except in that $\top$ is an absorbing element.

The identification of mandatory clauses with the $\top$ symbol allows to extend some well-known simplification rules from SAT to Max-SAT such as *addition* $\{(A, u), (A, w)\} \equiv \{(A, u \oplus w)\}$ or *subsumption* $\{(A, \top), (A \vee B, w)\} \equiv \{(A, \top)\}$.

A weighted CNF formula may contain $(\square, w)$. Since $\square$ cannot be satisfied, w is added to the cost of any assignment. Therefore, w is an explicit *lower bound* of the optimal model. When the lower bound and the upper bound have the same value (i.e., $(\square, \top) \in \mathcal{F}$) the formula does not have any model and we call this situation an *explicit contradiction*.

The notion of resolution can be extended to weighted formulas as follows,

$$\{(x \vee A, u), (\bar{x} \vee B, w)\} \equiv \left\{ \begin{array}{l} (A \vee B, m), \\ (x \vee A, u \ominus m), \\ (\bar{x} \vee B, w \ominus m), \\ (x \vee A \vee \bar{B}, m), \\ (\bar{x} \vee \bar{A} \vee B, m) \end{array} \right\}$$

where A and B are arbitrary disjunctions of literals and $m = \min\{u, w\}$.

$(x \vee A, u)$ and $(\bar{x} \vee B, w)$ are called the *prior clashing clauses*. $(A \vee B, m)$ is called the *resolvent*. $(x \vee A, u \ominus m)$ and $(\bar{x} \vee B, w \ominus m)$ are called the *posterior clashing clauses*. $(x \vee A \vee \bar{B}, m)$ and $(\bar{x} \vee \bar{A} \vee B, m)$ are called the *compensation clauses*.

Example 1. *If we apply resolution to the following clauses* $\{(x_1 \vee x_2, 3), (\bar{x}_1 \vee x_2 \vee x_3, 4)\}$ *(with* $\top = 5$*) we obtain* $\{(x_2 \vee x_2 \vee x_3, 3), (x_1 \vee x_2, 3 \ominus 3), (\bar{x}_1 \vee x_2 \vee x_3, 4 \ominus 3), (x_1 \vee x_2 \vee \overline{(x_2 \vee x_3)}, 3), (\bar{x}_1 \vee \bar{x}_2 \vee x_2 \vee x_3, 3)\}$. *The first and fourth clauses can be simplified. The second clause can be omitted because its weight is zero. The fifth clause can be omitted because it is a tautology. Therefore, we obtain the equivalent formula* $\{(x_2 \vee x_3, 3), (\bar{x}_1 \vee x_2 \vee x_3, 1), (x_1 \vee x_2 \vee \bar{x}_3, 3)\}$.

2.2 Inference-Based Simplification Rules

A Max-SAT problem can be solved to optimality with a pure inference algorithm, namely, an algorithm that only applies the resolution rule [4, 16]. However, such an algorithm has exponential space requirements and it is not used in practice. A natural alternative is to use only restricted forms of resolution that simplify the formula and use search afterwards. The application of a *simplification rule* is simply the application of a limited number of resolution steps. Current Max-SAT solvers apply simplification rules at each node of a search tree. Their main objective is to simplify the problem instance

and to make explicit a lower bound (i.e. create new empty clauses). The following example shows the application of two steps of resolution that lead to increase the lower bound.

Example 2. *Consider a weighted formula* $\{(x_1 \vee x_2, 3), (\bar{x}_1 \vee x_2, 2), (\bar{x}_2, 1)\}$ *(with* $\top = 5$*). Suppose we apply the resolution rule between the first and the second clause. We obtain* $\{(x_1 \vee x_2, 1), (x_2, 2), (\bar{x}_2, 1)\}$*. Now, we apply the resolution rule between the second and the third clause so that the lower bound is increased* $\{(x_1 \vee x_2, 1), (x_2, 1), (\square, 1)\}$*. Observe that the three formulas are equivalent, but the last one is more explicit and presumably simpler.*

2.3 Encoding the Min-Vertex-Covering and Max-Clique as Max-SAT

Definition 1. *Given a graph* $G = (V, E)$*, a* vertex covering *is a set* $U \subseteq V$ *such that for every edge* (v_i, v_j) *either* $v_i \in U$ *or* $v_j \in U$*. The size of a vertex covering is* $|U|$*. The* minimum vertex covering *(Min-Vertex-Covering) problem consists in finding a covering of minimal size.*

The *minimum vertex covering* problem is a well-known NP-Hard problem and it is well-known that it can be naturally formulated as (weighted) Max-SAT. We associate one variable x_i to each graph vertex. Value *true* (respectively, *false*) indicates that vertex x_i belongs to U (respectively, to $V - U$). There is a binary weighted clause $(x_i \vee x_j, \top)$ for each edge $(v_i, v_j) \in E$. It specifies that at least one of these vertices must be in the covering because there is an edge connecting them. There is a unary clause $(\bar{x}_i, 1)$ for each variable x_i, in order to specify that it is preferred not to add vertices to U. $\top$ must be set to a sufficiently large number. Note that different weights in unary and binary clauses are required to express the relative importance of each type of clauses.

Definition 2. *Given a graph* $G = (V, E)$*, a* clique *is a set* $U \subseteq V$ *such that for every vertex* $v \in U$*,* v *is connected to all the vertices in* U*. The size of a clique is* $|U|$*. The* maximum clique *problem (Max-Clique) consists in finding a clique of maximal size.*

The *maximum clique* problem is a well-known NP-Hard problem. As noted in [11], finding the maximum clique of a graph $G = (V, E)$ is equivalent to finding a minimum vertex covering of the complementary graph $\bar{G}$. Given a graph $G = (V, E)$, its complementary graph is denoted by $\bar{G} = (V, \bar{E})$. It is constructed with the same set of vertices V and $(v_i, v_j) \in \bar{E}$ iff $(v_i, v_j) \notin E$. Hence, we can model Max-Clique problems as Minimum Vertex Covering problems over the complementary graph. Observe that the maximum size of the maximum clique is equivalent to $|V| - s$, where s is the size of the minimum vertex covering.

Note that the Max-SAT encoding of the Max-Clique problem only contains negative unit soft clauses and positive binary hard clauses.

3 Two Simplification Rules

In this section we present two simplification rules that can be executed frequently in the Max-SAT encoding of a Max-Clique problem. The correctness of both rules can be easily established with a sequence of resolution steps.

3.1 Star Rule

The *Star Rule* [21] can be used to create new empty clauses from a long clause and a set of appropriate unit clauses.

$$\{(\overline{x_{i1}} \vee \overline{x_{i2}} \vee \ldots \vee \overline{x_{ik}}, w_0), (x_{i1}, w_1), (x_{i2}, w_2), \ldots, (x_{ik}, w_k)\} \equiv$$

$$\left\{ \begin{array}{l} (\square, m), (\overline{x_{i1}} \vee \overline{x_{i2}} \vee \ldots \vee \overline{x_{ik}}, w_0 \ominus m), \\ (x_{i1}, w_1 \ominus m), (x_{i2}, w_2 \ominus m), \ldots, (x_{ik}, w_k \ominus m), \\ (x_{i1} \vee \overline{\overline{x_{i2}} \vee \overline{x_{i3}} \vee \ldots \vee \overline{x_{ik}}}, m), \\ (x_{i2} \vee \overline{\overline{x_{i3}} \vee \overline{x_{i4}} \vee \ldots \vee \overline{x_{ik}}}, m), \\ (x_{i3} \vee \overline{\overline{x_{i4}} \vee \overline{x_{i5}} \vee \ldots \vee \overline{x_{ik}}}, m), \\ \ldots, \\ (x_{ik-1} \vee x_{ik}, m) \end{array} \right\}$$

where $m = min\{w_0, w_1, \ldots, w_k\}$.

The new empty clause is added to the possibly existing one, which produces a lower bound increment. It is clear that the Star Rule may be effective when a large number of unit clauses are available.

Example 3. *Consider the initial formula* $\{(\bar{x}_1 \vee \bar{x}_2, 1), (x_1, 1), (x_2, 1)\}$. *In this example, we show each step of resolution needed to obtain the same result provided by the Star Rule. First, we apply the resolution rule between the first and the third clauses and we obtain* $\{(x_1 \vee x_2, 1), (x_1, 1), (\bar{x}_1, 1)\}$. *Then, we apply the resolution rule between the second and the third clauses to obtain* $\{(x_1 \vee x_2, 1), (\square, 1)\}$.

3.2 Unit Rule

The original *Unit Rule* can be used to create new unit clauses from a long clause and a set of appropriate binary hard clauses. Given a subset of variables $\{x_{i1}, x_{i2}, \ldots, x_{ik}, x_j\} \subseteq X$, consider the following subset of binary hard clauses:

$$Bin(x_{i1}, x_{i2}, \ldots, x_{ik}, x_j) = \left\{ (x_{i1} \vee x_j, \top), (x_{i2} \vee x_j, \top), \ldots, (x_{ik} \vee x_j, \top) \right\}$$

The *Unit Rule* has the form,

$$\{(\overline{x_{i1}} \vee \overline{x_{i2}} \vee \ldots \vee \overline{x_{ik}}, w), Bin(x_{i1}, x_{i2}, \ldots, x_{ik}, x_j)\} \equiv$$

$$\left\{ (\overline{x_{i1}} \vee \overline{x_{i2}} \vee \ldots \vee \overline{x_{ik}} \vee \overline{x_j}, w), Bin(x_{i1}, x_{i2}, \ldots, x_{ik}, x_j), (x_j, w) \right\}$$

Example 4. *Consider the initial formula* $\{(\bar{x}_1 \vee \bar{x}_2, 1), (x_1 \vee x_3, \top), (x_2 \vee x_3, \top)\}$. *In this example, we show each step of resolution needed to obtain the same result provided by the Unit Rule. First, we apply the resolution rule between the first and the second clauses and we obtain* $\{(\bar{x}_2 \vee x_3, 1), (\bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_3, 1), (x_1 \vee x_3, \top), (x_2 \vee x_3, \top)\}$. *Then, we apply the resolution rule between the first and the last clauses to obtain* $\{(x_3, 1), (\bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_3, 1), (x_1 \vee x_3, \top), (x_2 \vee x_3, \top)\}$.

4 Pre-processing

In this section we show a pre-process that exploits the synergy between the Unit and the Star rules. The unit rule generates unit positive clauses from negative clauses and binary positive hard clauses. This unit clauses are used by the star rule which transforms them into empty clauses, which means an increment of the lower bound. The pre-process works in a on-demand manner: it triggers the unit rule only if it can guarantee that it will allow the subsequent execution of the star rule.

Before introducing the details of the pre-processing, we present a useful definition.

Definition 3. *A negative clause* $(\mathcal{C}, w) = (\overline{x_{i1}} \vee \overline{x_{i2}} \vee \ldots \overline{x_{ik}}, w)$ *is* unit-related *with respect to* x'*, and it is noted* $(\mathcal{C}, w)_{x'}$*, if and only if* $Bin(x_{i1}, x_{i2}, \ldots, x_{ik}, x') \in \mathcal{F}$.

Observe that we can always apply the Unit Rule to a clause $\mathcal{C}$ unit-related with respect to literal x in order to generate a new positive unit soft clause (x, w).

The basic idea of the pre-processing is to generate the appropriate unit clauses with the *Unit Rule* so that we can apply the *Star Rule* later in order to increase the lower bound. The final objective is to increase as much as possible the lower bound.

The pre-processing is shown in Algorithm 1. It iterates over all the negative clauses (line 1). For each negative clause $(\mathcal{C}, w_0)$ the algorithm wants to obtain one unit clause for each literal in $\mathcal{C}$. To do so, for each literal l_i in $\mathcal{C}$ the algorithm seeks a clause $(\mathcal{C}', w_i)$ unit-related with respect to l_i (i.e. $(\mathcal{C}', w_i)_{l_i}$) and store it in the structure S. Note that all the negative clauses inserted in S must be different, and they must be also different from the initial $(\mathcal{C}, w_0)$ (lines 2-5). If it succeeds in finding unit-related clauses for each literal in $\mathcal{C}$ (line 6), then the algorithm applies the two simplification rules. First, for each pair in structure S, it applies the Unit Rule in order to create the corresponding unit clause (lines 7,8). Once all unit clauses have been generated, the algorithm proceeds to apply the Star Rule (line 9).

Recall that this process is applied to Max-clique problems (the original formula contains negative soft units and positive hard binary clauses). Therefore, one can easily see that, at any point of the execution of algorithm 1, each negative clause (C, w) is in $\mathcal{F}$ because either (i) (C, w) is an initial unit soft clause in $\mathcal{F}$ or (ii) (C, w) was generated by some application of the Unit Rule. This observation leads to a nice property:

Lemma 1. *Within the pre-processing algorithm, all the compensation clauses in* $\mathcal{F}$ *generated by the Star Rule are subsumed by binary hard clauses in* $\mathcal{F}$.

Proof-Sketch 1. *Consider the case in which all clauses have weight 1 (as it happens in the Max-Clique Problem). Suppose that the Star Rule is applied to an arbitrary subset of clauses in* $\mathcal{F}$*:*

$$\{(\overline{x_{i1}} \vee \overline{x_{i2}} \vee \ldots \vee \overline{x_{ik}}, 1), (x_{i1}, 1), (x_{i2}, 1), \ldots, (x_{ik}, 1)\}$$

Observe that $\{(\overline{x_{i1}} \vee \overline{x_{i2}} \vee \ldots \vee \overline{x_{ik}}, 1)$ *is in* $\mathcal{F}$ *because the following set of Unit Rules were applied (in reverse order):*

$$\{(\overline{x_{i1}} \vee \overline{x_{i2}} \vee \ldots \vee \overline{x_{ik-1}}, 1), Bin(x_{i1}, x_{i2}, \ldots, x_{ik-1}, x_{ik})\}$$

Algorithm 1. Algorithm to transform the Max-Clique problem into an equivalent but simpler one. Note that each application of the Unit Rule (line 5) generate a new clause to be considered in the main Loop 1

Procedure MC-Preprocessing($\mathcal{F}$)

1 **foreach** $(\mathcal{C}, w_0) = (\overline{l_1} \vee \overline{l_2} \vee \ldots \vee \overline{l_k}, w_0) \in \mathcal{F}$ **do**
2 $S := \emptyset$;
3 **foreach** $l_i \in \mathcal{C}$ **do**
4 **if** $\exists (\mathcal{C}', w_i)$ s.t. $(\mathcal{C}', w_i) \neq (\mathcal{C}, w_0) \wedge (\mathcal{C}', w_i) \notin S \wedge (\mathcal{C}', w_i)_{l_i}$ **then**
5 $S := S \cup ((\mathcal{C}', w_i), l_i)$
6 **if** $|S| = k$ **then**
7 **foreach** $((\mathcal{C}', w_i), l_i) \in S$ s.t. $(\mathcal{C}', w_i) = (\overline{l'_1} \vee \overline{l'_2} \vee \ldots \vee \overline{l'_p}, w_i)$ **do**
8 Apply Unit Rule to $\{(\mathcal{C}', w_i)$,Bin$(l'_1, l'_2, \ldots, l'_p, l_i)\}$;
9 Apply Star Rule to $\{(l_1, w_1), (l_2, w_2), \ldots, (l_k, w_k), (\mathcal{C}, w_0)\}$;

$$\{(\overline{x_{i1}} \vee \overline{x_{i2}} \vee \ldots \vee \overline{x_{ik-2}}, 1), Bin(x_{i1}, x_{i2}, \ldots, x_{ik-2}, x_{ik-1})\}$$

$$\ldots$$

$$\{(\overline{x_{i1}}, 1), Bin(x_{i1}, x_{i2})\}$$

Precisely, the sets of binary clauses used at each application of the Unit Rule are enough to subsume all the compensation clauses produced by the initial Star Rule.

5 Empirical Results

In this section we present the benchmarks and the algorithms we tested in our empirical evaluation.

5.1 Benchmarks

In our experiments we consider instances that have been used in several works related with the Max-Clique problem.

- Random graph instances for which we solved the Max-Clique problem [16]. They were submitted to Max-SAT Evaluations 2006 and 2007 [2]. A *random graph* is defined by two parameters $\langle n, e \rangle$ where n is the number of nodes and e is the number of edges. Edges are randomly decided using a uniform probability distribution. Those random instances have a fixed number of nodes (150) and the graph density is varied.
- The 66 Max-Clique instances from the DIMACS challenge [15] [1] [11, 25, 24, 5]. They were also submitted to Max-SAT Evaluations 2006 and 2007 [2].

[1] `ftp://dimacs.rutgers.edu/pub/challenge/graph/benchmarks/clique`

- Ke Xu's Max-Clique instances with hidden optimum solutions [30, 29] which are advocated to be very difficult to solve. We considered the following publicly available sets of instances: frb10, frb15, frb20, frb25 and frb30.
- 11 Max-Clique real instances, provided by J.S. Sokol, corresponding to the protein structure alignment problem transformed into the maximum clique problem as described in [10, 16]. In this problem, the goal is to compute a score of similarity between two proteins based on a particular knowledge of their respective tri-dimensional structure.

5.2 Experiments Considered

First, we compare the lower bound obtained with our new pre-processing with respect to previous lower bounds. Second, we study the effect of feeding a Max-SAT solver with the pre-processed instance. The results are presented in plots and tables. In most of the tables, the common columns are: Problem, Nodes and Density that refer to the name of the instance, the number of nodes and the density of the graph, respectively. In tables and plots, OPT refers to value of the optimal solution. For each experiment, additional information is presented. Execution times are presented in seconds. All the experiments were performed on a 3.2 Ghz Intel Pentium with 1 GB and Linux.

5.3 Comparison of the New Lower Bound

Let LB-NEW be the lower bound obtained with the new pre-processing. Our first aim is to compare LB-NEW with respect to previous lower bounds for *Max-SAT* and *WCSP* which are deeply related to our approach. The best current lower bound in Max-SAT solvers is LB-UB [18]. The best current lower bounds for WCSP solvers are EDAC* [8] and OSAC [7]. We did preliminary experiments and observed that the lower bounds computed with LB-UB and EDAC* were similar. Hence, we simply present the results for LB-UB. We also tested WCSP solvers with OSAC [7] but we discarded it for the final experiments because most of its executions were aborted due to a time limit of 600 seconds. See Section 6 for a more detailed explanation of these lower bounds and their relationships. In this experiment we will also report the time needed by the novel pre-processing and we will refer to it as Pre-Time.

Figure 1 reports the results of applying our preprocessing to the max-clique problem of random graphs with 150 nodes and varying number of edges. Note that instances with low graph density have an associated Max-SAT encoding containing a large number of binary hard clauses. The figure shows three values which are average results of 30 instances per point. Two main observations can be extracted. First, the lower bound of the novel preprocessing LB-NEW is quite powerful when the graph density is low, and it is quite near to the optimal solution value. However, it loses accuracy as the graph density increases. The second observation is that LB-NEW is much more accurate than LB-UB.

Figure 2 shows results for the 66 DIMACS instances. The plot reports lower bound gains (as (LB-NEW - LB-UB)/ LB-UB) versus problem density. It can be seen that LB-NEW is typically $60 - 80\%$ higher than LB-UB. The effect of the problem density in this small sample of instances is not very clear, but again it seems that the benefits of LB-NEW are more important in low density graphs.

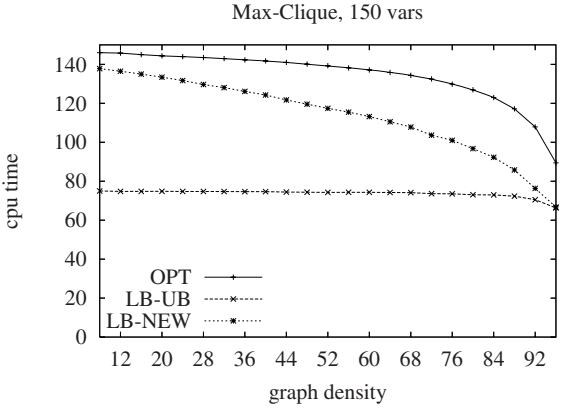

Fig. 1. Lower bounds computed as a pre-processing compared with optimal solutions for random graphs

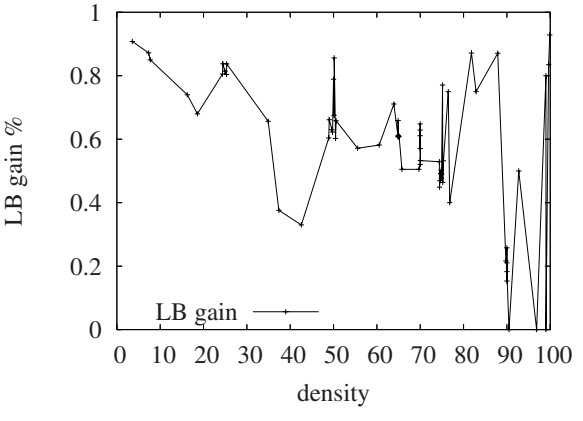

Fig. 2. Lower bound increment as (LB-NEW - LB-UB)/ LB-UB for the 66 Dimacs instances

Observe the time of the pre-processing Time-Pre, the new lower bound LB-NEW, the lower bound LB-UB and the value of the optimal solution OPT in Figures 4 and 5. For the instances with hidden optimum solutions (Figure 4), the time required by the pre-processing is negligible and LB-NEW clearly improves LB-UB. For the protein alignment instances (Figure 5) our pre-processing can be computed in less than 5 seconds for all the instances and LB-NEW is very close to the optimal solution OPT. It is worth to observe that LB-NEW almost doubles LB-UB.

5.4 Feeding a Max-SAT Solver with the Pre-processed Instance

In our second experiment we analyze how the execution time of a Max-SAT solver can be reduced by feeding it with the pre-processed instance. Then we also compare it to

specific Max-Clique solvers. However most of the Max-Clique solvers in the literature are not publicly available. The only Max-Clique solver available is DF-MAX [6, 15] that has been used in most of the comparisons of previous works. While DF-MAX is an old algorithm, it is still very efficient on sparse graphs [27]. In that context, we used MINIMAXSAT [13], the overall best branch and bound Max-SAT solver in the 2007 Max-SAT Evaluation [2] and we will refer to it simply as MS. When we feed MS with the pre-processed instance we will refer to it as MS+Pre.

First, we compared the execution times of MS and MS+Pre in the random graphs of Figure 1 but no significant differences were found in execution time.

In Figure 3 we compare the number of solved instances by the Max-SAT solver MS with respect to the rest of specific Max-Clique solvers in the 66 DIMACS instances. For this experiment, we considered a time limit of 2.5 hours so that we can compare with other solvers via a normalization process. The results indicate that the Max-SAT solver MS lays in the middle, being the constraint programming approach [25] and the coloring approach [11] the best current solvers for the DIMACS instances. But, we observed that giving a larger time limit of 8 hours 4 more instances can be solved with MS and MS+Pre, precisely the group of brock400*. We observed that MS+Pre was able to solve the instances san200_0.7_1 and san200_0.7_2 in 0.14 seconds and 116.31 seconds, while MS required 363.47 seconds and 6046.06 seconds, respectively. However, MS was able to solve the instance san400_0.9_1 within 5 minutes, while MS+Pre was not able to solve such an instance. In the other 63 instances, both approaches performed quite similar.

Solver	Solved Instances
Regin [25]	52
Fahle [11]	45
MS	39
MS + Pre	38
Wood [28]	38
Ostergard [22]	36
DF-MAX	31

Fig. 3. Solved instances for the 66 graphs from the Second DIMACS challenge [15] within a time limit of 2.5 hours

From the previous results, it seems that our new pre-processing generates a powerful initial lower bound but it does not allow important speed-ups for a Max-SAT solver. In what follows, we show that such speed-ups occur in some specific instances.

Observe the results in Figures 4 and 5. They include the execution time of MS, MS+Pre and DF-MAX. The time limit was set to 4 hours.

Regarding the instances with hidden optimum solutions (Figure 4), MS and DF-MAX performed quite similar and solved exactly the same number of instances. Differently, MS+Pre is about two orders of magnitude faster than the other approaches. Furthermore, it is able to solve all the instances of the frb25 set, and one instance of the frb30 set.

Problem	nodes	density	Time-Pre	MS	DF-MAX	MS+NEW	LB-UB	LB-NEW	OPT
frb10-6-1	60	65.71	0.00	0.00	0.00	0.00	30	45	50
frb10-6-2	60	64.63	0.00	0.00	0.00	0.00	30	43	50
frb10-6-3	60	66.05	0.00	0.00	0.00	0.00	30	43	50
frb10-6-4	60	63.95	0.00	0.00	0.00	0.00	30	44	50
frb10-6-5	60	64.24	0.00	0.00	0.00	0.00	30	43	50
frb15-9-1	135	72.22	0.00	1.65	1.00	0.10	67	106	120
frb15-9-2	135	72.21	0.00	1.11	1.00	0.22	67	102	120
frb15-9-3	135	72.40	0.00	1.31	1.00	0.14	67	105	120
frb15-9-4	135	72.32	0.00	1.05	1.00	0.25	67	105	120
frb15-9-5	135	71.69	0.00	1.58	1.00	0.37	67	105	120
frb20-11-1	220	76.61	0.00	295.94	418.00	2.40	110	173	200
frb20-11-2	220	76.86	0.01	267.54	412.00	29.39	110	172	200
frb20-11-3	220	76.70	0.00	411.83	483.00	4.84	110	175	200
frb20-11-4	220	76.87	0.00	490.59	450.00	5.58	109	174	200
frb20-11-5	220	76.75	0.00	556.99	698.00	16.72	110	173	200
frb25-13-1	325	79.87	0.02	-	-	2212.94	162	260	300
frb25-13-2	325	79.83	0.01	-	-	583.69	162	261	300
frb25-13-3	325	79.72	0.01	-	-	247.34	162	256	300
frb25-13-4	325	80.18	0.01	-	-	432.19	162	260	300
frb25-13-5	325	80.04	0.02	-	-	930.17	161	259	300
frb30-15-1	450	82.28	0.02	-	-	-	225	360	-
frb30-15-2	450	82.24	0.02	-	-	-	224	364	-
frb30-15-3	450	82.28	0.02	-	-	-	225	363	-
frb30-15-4	450	82.28	0.02	-	-	-	225	366	-
frb30-15-5	450	82.31	0.03	-	-	5772.32	225	366	420

Fig. 4. Instances with hidden optimum solutions. 4 hours of time limit.

Problem	nodes	density	Time-Pre	MS	DF-MAX	MS+NEW	LB-UB	LB-NEW	OPT
1bpi–2knt	2436	15.06	3.80	-	528.00	564.99	1218	2372	2407
1bpi–5pti	3016	15.36	5.62	-	2525.00	456.18	1508	2945	2974
1knt–1bpi	2494	14.86	3.80	-	430.00	375.45	1247	2429	2464
1knt–2knt	1806	14.76	2.02	358.93	19.00	87.22	903	1751	1767
1knt–5pti	2236	15.15	3.19	-	226.00	13441.60	1118	2179	2208
1vii–1cph	171	10.88	0.01	0.03	0.00	0.19	85	158	165
2knt–5pti	2184	15.28	2.96	-	238.00	327.04	1092	2124	2156
3ebx–1era	2548	14.72	4.04	-	886.00	865.05	1274	2483	2517
3ebx–6ebx	1768	14.45	1.99	1532.70	88.00	94.56	884	1717	1740
6ebx–1era	1666	14.35	1.89	2705.69	45.00	104.35	833	1616	1646
sandiaprot	2279	14.83	3.30	-	189.00	7710.05	1139	2220	2248

Fig. 5. Protein structure alignment problem transformed into Max-Clique. 4 hours of time limit.

Regarding the protein alignment instances (Figure 5) the novel lower bound LB-NEW is very close to the optimal solution OPT. MS can solve only four instances within the time limit, while MS+Pre is able to solve all the instances and 9 of them in less than fifteen minutes. The performances of DF-MAX and MS+Pre are quite similar in most of the instances. Recall that DF-MAX is still the best specific solver for low density instances like those. Observe that in [16] a Max-SAT solver called MAX-DPLL was able to solve 10 of the 11 instances. MAX-DPLL applies at each node of the search tree the *cycle rule* which can be seen as a very limited version of our more general approach. However, the current implementation of the cycle rule has severe memory limitations [16].

6 Related Work

There exists a handful of specific Max-Clique *branch and bound* solvers [6, 15, 22, 11, 25, 27]. They mainly differ in their bounds. The best current upper bounds are based on constraint programming techniques [25] and on approximate graph coloring techniques [11, 27].

Besides specific algorithms, Max-Clique can also be solved with generic solvers. In the following, we review Max-SAT and *WCSP* approaches.

Current complete algorithms for Max-SAT are also branch and bound algorithms. In that context, the upper bound ub is the cost of the best complete assignment found so far and the lower bound (lb) is the sum of the weights of the clauses in the original formula violated by the current partial assignment plus an *underestimation* of the cost of extending the current partial assignment. lb and ub are used to avoid visiting useless parts of the search tree when $lb \geq ub$. Most of them compute underestimations based on detecting *inconsistent subsets*: Given a WCNF formula $\mathcal{F}$, an *inconsistent subset* is a subset of clauses of $\mathcal{F}$ such that at least one of the clauses is always unsatisfied by any assignment of the variables. When an inconsistent subset is detected, two approaches are possible:

- Relaxation: Remove the clauses involved in the inconsistent subset from the formula and increase the underestimation [18].
- Inference: Apply the resolution rule to create an equivalent formula with new empty clauses [16].

Best current Max-SAT solvers use unit propagation (UP) to detect inconsistent subsets and then they apply a mixture of the previous two approaches [13, 20].

The *star rule* [21] captures the following inconsistent subset that can be also detected via UP:

$$\{(\overline{x_1} \vee \overline{x_2} \vee \ldots \vee \overline{x_k}, w_0), (x_1, w_1), (x_2, w_2), \ldots, (x_k, w_k)\}$$

If we relax the formula, the underestimation can be increased by $min\{w_0, w_1, \ldots, w_k\}$. This was applied during search in [26, 1] restricted to $k = 2$ and in general in [18, 20].

Following the inference-based approach, we have precisely the same transformation presented in Section 3. It was applied in [12, 20] restricted to $k = 2$ and in general in [13].

Let S be the largest subset of hard binary clauses of the Max-Clique problem with no literals in common among them. For each clause $(x_i \vee x_j, \top)$ in S and their respective unit clauses $(x_i, 1)$ and $(x_j, 1)$, we can relax the formula by removing them and increasing the underestimation by 1. Hence, we can obtain a resulting lower bound of $LB_{\mathcal{P}} = |S|$. It is precisely the best lower bound that can be computed via UP and then relaxing the formula like LB-UB used in Section 5. A lower bound based on detecting inconsistent subsets via UP and then transforming the formula may be greater than $LB_{\mathcal{P}}$. However, in practice it is always quite near to $LB_{\mathcal{P}}$. The main observation of our work is that the binary hard clauses of the Maximum Clique problem lead to generate lots of unit clauses that can be used later by the star rule once and again.

The Max-SAT problem was reformulated as a *Weighted Constraint Satisfaction Problem* (*WCSP*) [17] in [9, 8] with boolean variables and weighted constraints. Current WCSP solvers apply an inference-based method called *Existential Directional Arc Consistency* (*EDAC**) [8]. The application of EDAC* in a WCNF formula only affects to unit and binary clauses that are replaced by other unary and binary clauses and a weighted empty clause. A lower bound based on EDAC* is always equal or lower than $LB_{\mathcal{P}}$. The most relevant difference of our approach is that we allow the creation of larger arity clauses.

7 Conclusions and Future Work

In this paper we present a pre-processing based on the Max-SAT resolution rule for the Maximum Clique problem (and other related problems). We show empirically how it can be used to obtain a powerful initial lower bound and, in some cases, how the search process of a complete systematic search algorithm can be boosted several orders of magnitude. The following step of our research plan will consist in powering a branch and bound algorithm with our new algorithm at each node of the search tree. This may lead to solve the Maximum Clique problem in a very efficient way. To do so, we are currently working on the necessary data-structures so that the algorithm can be performed in a very fast way.

References

1. Alsinet, T., Manyà, F., Planes, J.: Improved exact solvers for weighted Max-SAT. In: Bacchus, F., Walsh, T. (eds.) SAT 2005. LNCS, vol. 3569, pp. 371–377. Springer, Heidelberg (2005)
2. Argelich, J., Li, C.M., Manyá, F., Planes, J.: The first and second max-sat evaluations. Journal on Satisfiability, Boolean Modeling and Computation (to appear, 2008)
3. Balas, E., Yu, C.S.: Finding a maximum clique in an arbitrary graph. SIAM Journal of Computation 15(4), 1054–1068 (1986)
4. Bonet, M.L., Levy, J., Manyà, F.: Resolution for max-sat. Artificial Intelligence 171(8-9), 606–618 (2007)
5. S.B.: A new trust region technique for the maximum weight clique problem. Discrete Applied Mathematics 154(15), 2080–2096 (2006)
6. Carraghan, R., Pardalos, P.: An exact algorithm for the maximum clique problem. Operations Research Letters 9, 375–382 (1990)

7. Cooper, M.C., de Givry, S., Schiex, T.: Optimal soft arc consistency. In: Proceedings of IJCAI, pp. 68–73 (2007)
8. de Givry, S., Heras, F., Larrosa, J., Zytnicki, M.: Existential arc consistency: getting closer to full arc consistency in weighted csps. In: Proceedings of IJCAI (2005)
9. de Givry, S., Larrosa, J., Meseguer, P., Schiex, T.: Solving max-sat as weighted csp. In: Rossi, F. (ed.) CP 2003. LNCS, vol. 2833, pp. 363–376. Springer, Heidelberg (2003)
10. Barnes, E., Strickland, D.M., Sokol, J.S.: Optimal protein structure alignment using maximum cliques. Operations Research 53, 389–402 (2005)
11. Fahle, T.: Simple and fast: Improving a branch-and-bound algorithm for maximum clique. In: Möhring, R.H., Raman, R. (eds.) ESA 2002. LNCS, vol. 2461, pp. 485–498. Springer, Heidelberg (2002)
12. Heras, F., Larrosa, J.: New inference rules for efficient max-sat solving. In: AAAI (2006)
13. Heras, F., Larrosa, J., Oliveras, A.: Minimaxsat: A new weighted max-sat solver. In: Marques-Silva, J., Sakallah, K.A. (eds.) SAT 2007. LNCS, vol. 4501, pp. 41–55. Springer, Heidelberg (2007)
14. Ji, Y., Xu, X., Stormo, G.D.: A graph theoretical approach for predicting common RNA secondary structure motifs including pseudoknots in unaligned sequences. Bioinformatics 20(10), 1603–1611 (2004)
15. Johnson, D.S., Trick, M.: Second DIMACS implementation challenge: cliques, coloring and satisfiability. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol. 26, AMS (1996)
16. Larrosa, J., Heras, F., de Givry, S.: A logical approach to efficient Max-SAT solving. Artificial Intelligence, an international journal 172, 204–233
17. Larrosa, J., Schiex, T.: Solving weighted csp by maintaining arc-consistency. Artificial Intelligence 159(1-2), 1–26 (2004)
18. Li, C.M., Manyà, F., Planes, J.: Exploiting unit propagation to compute lower bounds in branch and bound Max-SAT solvers. In: van Beek, P. (ed.) CP 2005. LNCS, vol. 3709, pp. 403–414. Springer, Heidelberg (2005)
19. Li, C.M., Manya, F., Planes, J.: Detecting disjoint inconsistent subformulas for computing lower bounds for Max-SAT. In: Proceedings of AAAI (2006)
20. Li, C.M., Manyà, F., Planes, J.: New inference rules for max-sat. Journal of Artificial Intelligence Research 30, 321–359 (2007)
21. Niedermeier, R., Rossmanith, P.: New upper bounds for maximum satisfiability. Journal of Algorithms 36(1), 63–88 (2000)
22. Ostergard, P.R.J.: A fast algorithm for the maximum clique problem. Discrete Applied Mathematics 120, 197–207 (2002)
23. Pevzner, P.A., Sze, S.: Combinatorial approaches to finding subtle signals in DNA sequences. In: ISMB, pp. 269–278 (2000)
24. Pullan, W.J., Hoos, H.H.: Dynamic local search for the maximum clique problem. J. Artif. Intell. Res (JAIR) 25, 159–185 (2006)
25. Régin, J.-C.: Using constraint programming to solve the maximum clique problem. In: Rossi, F. (ed.) CP 2003. LNCS, vol. 2833, pp. 634–648. Springer, Heidelberg (2003)
26. Shen, H., Zhang, H.: Study of lower bounds for Max-2-SAT. In: AAAI (2004)
27. Tomita, E., Seki, T.: An efficient branch-and-bound algorithm for finding a maximum clique. In: Calude, C.S., Dinneen, M.J., Vajnovszki, V. (eds.) DMTCS 2003. LNCS, vol. 2731, pp. 278–289. Springer, Heidelberg (2003)
28. Wood, D.: An algorithm for finding maximum cliques in a graph. Operations Research Letters 21, 211–217 (1997)
29. Xu, K., Boussemart, F., Hemery, F., Lecoutre, C.: A simple model to generate hard satisfiable instances. In: Proceedings of IJCAI, pp. 337–342 (2005)
30. Xu, K., Li, W.: Many hard examples in exact phase transitions with application to generating hard satisfiable instances. CoRR, cs.CC/0302001 (2003)