A New Formula Rewriting by Reasoning on a Graphical Representation of SAT Instances

Philippe Jégou and **Lionel Paris**
LSIS - UMR CNRS 6168
Université Paul Cézanne (Aix-Marseille 3)
Avenue Escadrille Normandie-Niemen
13397 Marseille Cedex 20 (France)
{philippe.jegou, lionel.paris}@univ-cezanne.fr

Abstract

In this paper, we propose a new approach for solving the SAT problem. This approach consists in representing SAT instances thanks to an undirected graph issued from a polynomial transformation from SAT to the CLIQUE problem. Considering this graph, we exploit well known properties of chordal graphs to manipulate the SAT instance. Firstly, these properties allow us to define a new class of SAT polynomial instances. Moreover, they allow us to rewrite SAT instances in disjunctions of smaller instances which could be significantly easier to solve.

Introduction

To solve the SAT problem, classical approaches have shown their interest as *minisat* (en and orensson 2003), but also their bounds. So, we propose here a new approach which is really different from classical approaches, hoping that this new viewpoint could open a new way for solving of SAT instances. This approach considers a SAT instance given by a CNF F and will rewrite it in a formula $F_1 \vee F_2 \vee \ldots \vee F_N$ which is satisfiable if and only if F is satisfiable. Here, we guarantee that each sub-formula F_i will be easier to solve (smaller) than the original CNF F. The process used here to rewrite F in $F_1 \vee F_2 \vee \ldots \vee F_N$ can be applied recursively. This rewriting is obtained in exploiting a graph induced by the CNF F: vertices corresponds to the literals of clauses while edges are defined in joining compatible literals (neither opposite literals nor literals issued from the same clause). This graph defines a way for a polynomial transformation from SAT to the CLIQUE problem of graph theory. Based on this transformation, we can exploit recent results obtained on *Perfect Graphs* to present a new class of SAT polynomial instances. Moreover, we show that the restriction of *Perfect Graphs* to one of its sub-classes, namely, *Chordal Graphs* (aslo called *Triangulated Graphs*) allows efficient handling of SAT formulas by means of their graphical representation. Precisely, the rewriting of F in $F_1 \vee F_2 \vee \ldots \vee F_N$ is obtained in exploiting methods realizing triangulation of graphs. This field of graphs algorithms has been intensively studied during the last fifty years and then we know for it efficient algorithms. Note that our approach which is based on graph triangulation is different from the work already realized in the neighbouring domain of CSPs (as for example with the tree-clustering method (Dechter and Pearl 1989)), and then it is not an adaptation of this kind of works for the SAT problem. Our approach offers then a new and original way for solving SAT.

The rest of this paper is organized as follows: the first section recalls notations on SAT and introduces the relation between SAT and the CLIQUE problem. The following section presents recent results on *Perfect Graphs* and classical results on *Chordal Graphs* which are usable for the CLIQUE problem and then for SAT. The next section introduces the approach for rewriting formulas. Finally, the last section concludes this paper.

From SAT to CLIQUE

Preliminaries

We consider the SAT problem which is defined by a particular Boolean formula given by a pair $F = (X, C)$. F is a Boolean formula over X, and is defined by a conjunction of clauses, that is, $F = C_1 \wedge C_2 \wedge \ldots C_m$. Here, X is a finite set of Boolean variables $\{x_1, x_2, \ldots x_n\}$. A *truth assignment* for X is a function $A : X \rightarrow \{0, 1\}^n$ (0 for *False* and 1 for *True*). If x is a variable, x and $\neg x$ are *literals* over X. Moreover, we say that x and $\neg x$ are *opposite* literals. The literal $\neg x$ is true if and only if x is false. C is a set of clauses $\{C_1, C_2, \ldots C_m\}$. A *clause* C_i over X is a disjunction $l_{i_1} \vee l_{i_2} \vee \ldots l_{i_{n_i}}$ of literals. A truth assignment satisfies a clause C_i if at least, one literal l_{i_j} is true. So, a truth assignment A satisfies a formula F if and only if A satisfies each clause in F. In this case, we say that F is *satisfiable*. Such a truth assignment is generally called *model* of F. We can consider partial truth assignments, that is functions which are partially defined on X. By extension, we call also *partial model* a partial truth assignment which satisfies all clauses. Given a Boolean formula F, in the sequel, L_F will denote the set of literals appearing in F.

Example 1 Consider the formula $\mathcal{F} = C_1 \wedge C_2 \wedge C_3 \wedge C_4$ defined over $X = \{a, b, c\}$ where :

- $C_1 = (a \vee b \vee c) = (l_{1_1} \vee l_{1_2} \vee l_{1_3})$ where $l_{1_1} = a$, $l_{1_2} = b$ and $l_{1_3} = c$.

- $C_2 = (\neg a \vee \neg b) = (l_{2_1} \vee l_{2_2})$ where $l_{2_1} = \neg a$ and $l_{2_2} = \neg b$.
- $C_3 = (\neg a \vee \neg c) = (l_{3_1} \vee l_{3_2})$ where $l_{3_1} = \neg a$ and $l_{3_2} = \neg c$
- $C_4 = (a \vee \neg b \vee c) = (l_{4_1} \vee l_{4_2} \vee l_{4_3})$ where $l_{4_1} = a$, $l_{4_2} = \neg b$ and $l_{4_3} = c$.

A model for $\mathcal{F}$ is then the truth assignment $A = \{(a,1),(b,0),(c,0)\}$. A partial model can be $A' = \{(a,0),(c,1)\}$. In this case, all extensions of A' to other variables in X are models of $\mathcal{F}$. Note that we can assimilate sets of literals to truth assignments. For example, $A = \{(a,1),(b,0),(c,0)\}$ is similar to the ordered set of literals $(a, \neg b, \neg c)$, which corresponds to the set $\{l_{1_1} = a, l_{2_2} = \neg b, l_{3_3} = \neg c, l_{4_1} = a\}$ or the set $\{l_{1_1} = a, l_{2_2} = \neg b, l_{3_2} = \neg c, l_{4_2} = \neg b\}$; this assignment is also denoted $(l_{1_1}, l_{2_2}, l_{3_2}, l_{4_2})$. In that case, we can observe that a set of literals is a model if and only if at least one literal of this set appears in each clause. Note that in this example, $L_{\mathcal{F}} = \{l_{1_1}, l_{1_2}, l_{1_3}, l_{2_1}, l_{2_2}, l_{3_1}, l_{3_2}, l_{4_1}, l_{4_2}, l_{4_3}\}$. A literal can occur many times: once for each clause containing it (example: $l_{1_1} = l_{4_1} = a$). So we consider several literals (one per clauses) when a literal appears in several clauses.

Now, we define a sub-formula induced by a subset of literals.

Definition 1 *Given a Boolean formula $F = (X,C)$ and $L \subseteq L_F$, the sub-formula of F induced by L, denoted $L(F)$, is the formula F restricted to literals belonging to L.*

Consider the formula $\mathcal{F}$ of Example 1, and $L = \{l_{1_1}, l_{1_3}, l_{2_1}, l_{2_2}, l_{3_2}, l_{4_2}, l_{4_3}\}$. We obtain the sub-formula $L(\mathcal{F}) = (a \vee c) \wedge (\neg a \vee \neg b) \wedge (\neg c) \wedge (\neg b \vee c)$. One can remark that if in a subset L of $L(\mathcal{F})$, no literal of a clause appears, then the sub-formula $L(\mathcal{F})$ includes the empty clause and then $L(\mathcal{F})$ is not satisfiable. E.g. with $L' = \{l_{1_1}, l_{1_3}, l_{2_1}, l_{2_2}, l_{4_2}, l_{4_3}\}$, $L'(\mathcal{F}) = (a \vee c) \wedge (\neg a \vee \neg b) \wedge \square \wedge (\neg b \vee c)$ (where $\square$ denotes the empty clause) because no literal of the clause C_3 appears in L'.

More generally, it is easy to see that if $L(F)$ is satisfiable, then necessarily, F is satisfiable. Formally:

Theorem 1 *Given Boolean formula $F = (X,C)$ and $L \subseteq L_F$, if $L(F)$ is satisfiable with a model $(l_1, l_2, \ldots l_m)$, then F is satisfiable by the same model $(l_1, l_2, \ldots l_m)$.*

Proof: If $L(F)$ is satisfied by the set of literals $\{l_1, l_2, \ldots l_m\}$, at least one literal of this set appears in each clause of $L(F)$. Since, each clause of $L(F)$ is a sub-clause of a clause in F, necessarily, there is at least one literal of L which appears in each clause of F, and consequently, $(l_1, l_2, \ldots l_m)$ is also a model for F. $\square$

In Example 1, with $L = \{l_{1_1}, l_{1_3}, l_{2_1}, l_{2_2}, l_{3_2}, l_{4_2}, l_{4_3}\}$, the sub-formula $L(\mathcal{F}) = (a \vee c) \wedge (\neg a \vee \neg b) \wedge (\neg c) \wedge (\neg b \vee c)$ is satisfied with the model $A = \{(a,1),(b,0),(c,0)\}$ corresponding to $(l_{1_1} = a, l_{2_2} = \neg b, l_{3_2} = \neg c, l_{4_2} = \neg b)$, which is also a model for $\mathcal{F}$.

Polynomial transformation from SAT to CLIQUE

We define here a graphical representation of SAT instances called *CL-Graphs* (for *Clauses-Literals-Graphs*). This representation is based on a polynomial transformation from SAT to the CLIQUE problem which is well known. Nevertheless, we need here to describe it in detail.

Definition 2 *Given a SAT formula defined by $F = (X,C)$, the CL-Graph of F is an undirected graph $G(F) = (V_F, E_F)$ where :*

- $V_F = \{v_{i_j} : l_{i_j}$ *is a litteral of the clause* $C_i \in C\}$
- $E_F = \{\{v_{i_a}, v_{j_b}\} : i \neq j$, *and* l_{i_a} *and* l_{j_b} *are not opposite literals* $\}$

In other words, V_F is exactly the set of literals L_F, that is each literal of each clause corresponds to a vertex and two vertices are connected by an edge, if they do not appear in the same clause and if they are not opposite literals. So, $n_F = |V_F|$ of vertices in $G(F)$ is equal to the sum of the size of the clauses in the formula F. Consequently, the number of edges $e = |E_F|$ is bounded by $n_F(n_F - 1)/2$. So, the size of the graph $G(F)$ is polynomial w.r.t. the size of F. Moreover, given a formula F, the time cost to compute $G(F)$ is polynomial. Note that this transformation corresponds to the *micro-structure* (Jégou 1993) of the *Literal Encoding* of SAT instances proposed in (Walsh 2000). Moreover, this transformation is different from the one used in (Surynek 2007).

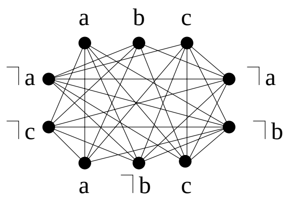

Figure 1: The CL-Graph $G(\mathcal{F})$ associated to the Boolean formula $\mathcal{F}$ of example 1.

We now recall the definition of the CLIQUE problem. A *clique* of a graph $G = (V,E)$ is a set $V' \subseteq V$ of vertices such that two vertices in V' are joined by an edge in E.

- CLIQUE Problem
- Instance: An undirected graph $G = (V,E)$ and a positive integer $k \leq |V|$.
- Question: Does G contain a clique V' of size k or more?

This problem has been shown to be NP-complete by Karp in 1972 (Karp 1972). Note that the *clique number* of a graph G, denoted $\omega(G)$ is the size of the largest clique in G. The next theorem indicates that the transformation from a Boolean formula F into its CL-Graph $G(F)$ defines a transformation from SAT to CLIQUE.

Theorem 2 *A Boolean formula $F = (X,C)$ is satisfiable if and only if its CL-Graph $G(F)$ contains a clique of size $m = |C|$.*

Proof: $F = (X, C)$ is satisfiable if and only if there is a truth assignment A which satisfies each clause. Consider such an assignment and the associated ordered set of literals $(l_1, l_2, \ldots l_m)$ where l_i is a literal of the clause C_i which satisfies this clause and m is the number of clauses in F. Necessarily, there are no opposite literals in $(l_1, l_2, \ldots l_m)$. So, in $G(F)$, all the vertices corresponding to literals in $(l_1, l_2, \ldots l_m)$ form a clique $\{l_1, l_2, \ldots l_m\}$ of size m because they are not opposite and they cannot appear in the same clause. Now, consider a clique of size m in $G(F)$. Necessarily, there is at least one vertex issued from each clause (because vertices issued from a same clause are not connected). Moreover, since all vertices are connected, it is not possible to have two opposite literals in the clique. So, this clique corresponds to a set of compatible literals which appear in all the clauses. Then, the assignment associated to this set of literals is a model for F. $\square$

Figure 2 shows an illustration of such an association between a clique of the CL-Graph of the formula of example 1 and a model of this formula.

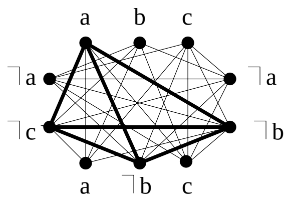

Figure 2: A clique of size 4 in the CL-Graph associated to a model of the formula $\mathcal{F}$.

In the next section, we show how this polynomial transformation from SAT to CLIQUE can be exploited using results from graph theory.

Perfect CL-Graphs: New SAT Polynomial Instances

Perfect Graphs

The *Perfect Graphs* were discovered by Claude Berge at the begining of the sixties (see (Golumbic 1980) for an introduction to perfect graphs). To define perfect graphs, we must recall the notion of *chromatic number* of a graph G, denoted $\chi(G)$. This number is defined as the smallest number of colors that can be assigned to vertices of G in such a way that every two adjacent vertices receive two distinct colors. Trivially, the chromatic number of every graph is at least its clique number, that is $\omega(G) \leq \chi(G)$. Finally, a graph G is *perfect* if, for each of its induced subgraphs G', $\chi(G') = \omega(G')$. The interest for perfect graphs is today really important because these graphs possess nice mathematical properties that could be exploited from a practical viewpoint (e.g. CLIQUE can be solved in polynomial time on these graphs (Grotschel, Lovasz, and Schrijver 1988)). Some important questions were asked during the sixties by C. Berge, as for example the ”Strong Perfect Graph Conjecture”. To define this conjecture, we recall that a *hole* is a chordless cycle of length at least four while an *antihole* is the complement of such a cycle. Moreover, holes and antiholes are *odd* or *even* according to the parity of their number of vertices. Finally, a graph is a *Berge Graph* if it contains no odd hole and no odd antihole. ”Strong Perfect Graph Conjecture” has been recently solved and can be now defined as a theorem:

Theorem 3 *(Chudnovsky et al. 2006) Berge Graphs are exactly Perfect Graphs.*

Independantly from this work, (Chudnovsky et al. 2005) have shown that Berge Graphs can be recognized in polynomial time. Its running time is $O(n^9)$ where n is the number of vertices in the considered graph. Since finding a maximal clique size in a perfect graph is possible in polynomial time (Grotschel, Lovasz, and Schrijver 1988), then the CLIQUE problem can be now polynomialy solved on perfect graphs: you test if the considered graph is a Berge graph (a perfect graph by theorem 3), and if it is ok, you find one of its maximal cliques. For us, this kind of result has consequences which are formalized by the next theorem:

Theorem 4 *If the CL-Graph $G(F)$ of a Boolean formula $F = (X, C)$ is a Perfect Graph, then recognizing and checking the satisfiability of F is polynomial.*

The practical application of this result is of limited interest. Firstly, the CL-Graph $G(F)$ of a given formula F is not necessarily a perfect graph (unless $P = NP$). Moreover, exploiting perfect graphs from a practical viewpoint is not obvious because at present, we do not have efficient tools to manipulate them (recall the complexity of recognizing Berge graphs!). So, to exploit their properties, we can restrict our study on a well known sub-class of perfect graphs called *Chordal Graphs*, which was one of the first classes of graphs to be recognized as being perfect and because numerous efficient algorithms has been developped for them during the last forty years. Finaly, note that using Perfect Graphs to define sets of polynomial instances has been recently proposed in the field of CSP in (Salamon and Jeavons 2008) who extends to perfect graphs a work introduced on Chordal graphs in (Jégou 1993).

Chordal Graphs

Chordal graphs (also called *Trianguled graphs*) have been introduced in graph theory and were shown to be an interesting sub-class of perfect graphs ((Golumbic 1980) offers also an easy introduction to chordal graphs). They possess combinatorial properties which can be exploited from an algorithmic viewpoint.

Definition 3 *An undirected graph $G = (V, E)$ is chordal (or triangulated) if each cycle of length at least 4 has a chord, that is an edge joining two non-consecutive vertices in the cycle.*

There is another definition of chordal graphs which is based on the notion of ordering of vertices.

Definition 4 *An undirected graph $G = (V, E)$ is chordal (or triangulated) if it has a perfect elimination order (p.e.o), i.e. a vertex order $\sigma = (v_1, \ldots, v_n)$ such that, for any vertex v_i, the vertices in the neighborhood of v_i which follow v_i in σ form a clique.*

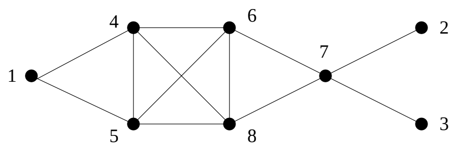

Figure 3: A chordal graph. The labeling of the vertices corresponds to a possible p.e.o.

Numerous works were realized during the last fifty years around chordal graphs. We recall some of them, related to the basic properties of chordal graphs. The first one is related to the number of maximal cliques, where a maximal clique is a clique which is not included in a larger clique. Note that for the general case, this number can be exponential in the number of vertices.

Theorem 5 *(Fulkerson and Gross 1965) The number of maximal cliques in a chordal graph is at most equal to its number of vertices.*

Another interesting property is related to the fact that when CLIQUE is restricted to chordal graphs, then the complexity of this problem is polynomial.

Theorem 6 *(Gavril 1972) Finding all maximal cliques in a chordal graph is linear w.r.t. its size. So, the restriction of the CLIQUE problem to chordal graphs can be solved in linear time.*

Finally, chordal graphs can be recognized in polynomial time, and the most efficient algorithm called MCS is due to Tarjan and Yannakakis (Tarjan and Yannakakis 1984).

Theorem 7 *The cost of recognizing chordal graphs is linear w.r.t. their size.*

By exploiting the polynomial transformation from SAT to CLIQUE, we can easily define a new set of SAT instances which can be solved in polynomial time.

Theorem 8 *If the CL-Graph $G(F)$ of a Boolean formula $F = (X, C)$ is chordal, then recognizing and checking the satisfiability of F is polynomial.*

Proof: Given a Boolean formula $F = (X, C)$, we compute its CL-Graph $G(F)$. Then we verify if this graph is chordal using a polynomial time algorithm as MCS. Finally, we use theorem 6 in computing its maximal cliques and checking if there is at least one clique of size $|C|$. The total cost is at most $|L_F|^2 = |V_F|^2 = n_F^2$ that is polynomial w.r.t the size of the formula F. $\square$

From Any Graphs to Chordal Graphs

Unfortunately, the CL-Graph $G(F)$ of a given formula F is not necessarily chordal (unless $P = NP$). E.g. for example 1, the sequence $(a = l_{1_1}, a = l_{4_1}, c = l_{1_3}, c = l_{4_3}, a = l_{1_1})$ is a cycle of length 4 without chord in $G(\mathcal{F})$. So the graph in figure 1 is not chordal. Nevertheless, even if a graph is not chordal, we can exploit properties of chordal graphs. This kind of work has already been realized in the field of constraint networks, that is CSPs. For example, the decomposition of constraint networks is generally based on the exploitation of chordal graphs. A well known approach, called tree-clustering (Dechter and Pearl 1989) considers a constraint network, and realizes a clustering of variables. In that work, vertices of the graphs correspond to Boolean variables while edges are given by clauses. Two variables are joined by an edge if they appear simultaneously in the same clause. Our approach here is closer to the one of (Jégou 1993) who considers as vertices values of domains of CSPs variables while edges correspond to the compatibility between values of different domains. Although these approaches do not consider the same graph, they apply the notion of *triangulation* of graph.

Definition 5 *Given an undirected graph $G = (V, E)$ which is not chordal, a triangulation Tr of G consists in adding a set E' of edges such that the graph $Tr(G) = (V, E \cup E') = (V, T)$ where $T = E \cup E'$ is chordal.*

Figure 4 shows a possible triangulation of the graph of Figure 1.

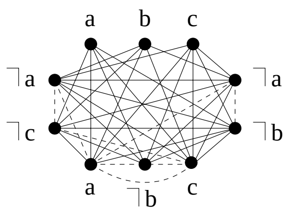

Figure 4: The triangulation (doted lines represent added edges) of the CL-Graph given in figure 1.

After a triangulation, it is easy to find maximal cliques in $Tr(G)$ and we know that all maximal cliques in $G = (V, E)$ appear in at least one maximal clique of $Tr(G)$. This approach allows us to simplify the problem of finding maximal cliques of G since we can limit the search to sub-problems induced by maximal cliques of $Tr(G)$ while initially, the whole graph G must be considered. Intuitively, we can see that intially, the complexity was $exp(|V|)$ while now, the complexity is limited to $exp(M)$ where M is the maximum size of maximal cliques in $Tr(G)$, where $M \leq |V|$. Note that the value M verifies $M \leq w + 1$ where w is the *tree-width* of the graph G (Robertson and Seymour 1986). Unfortunately computing an optimal triangulation, that is a triangulation which minimizes the value M to get $M = w+1$, is known to be a NP-hard problem (Arnborg, Corneil, and Proskuroswki 1987). So, many works deal with this problem and then several approaches and algorithms were proposed for triangulation. In any case, they aim at minimizing either the number of added edges, or the size of the cliques in the triangulated graph. We can distinguish four classes of approaches:

1. **Optimal triangulations.** As the problem is NP-hard, no polynomial algorithm is known yet. Hence, the proposed algorithms have an exponential time complexity. Unfortunately, their implementations do not have much interest

from a practical viewpoint. For instance, the algorithm described in (Fomin, Kratsch, and Todinca 2004), whose time complexity is $O(n^4.(1.9601^n))$ (where $n = |V|$), has never been implemented due to the weak expected interest in practice (Todinca 2005). Moreover, a recent work (Gogate and Dechter 2004) has shown that the algorithm proposed in (Shoikhet and Geiger 1997) cannot solve small graphs (50 vertices and 100 edges).

2. **Approximation algorithms.** These algorithms approximate the optimum by a constant factor. Their complexity is often polynomial in the tree-width (Amir 2001). However, this approach seems unusable. Indeed, the last algorithm proposed in (Amir 2001) has a time complexity in $O(n^3.\log^4(n).k^5.\log(k))$ with a hidden constant greater than 850 (Bouchitté et al. 2004). Moreover, according to (Amir to appear), it requires a runtime between 6 minutes and 6 days depending on the considered instance while a naive heuristic triangulation obtains better results (w.r.t. the tree-width) in at most two minutes.
3. **Minimal triangulation.** A minimal triangulation computes a set E' such that $G' = (X, E \cup E')$ is triangulated and for every subset $E" \subsetneq E'$, the graph $G" = (X, E \cup E")$ is not triangulated. Note that a minimal triangulation is not necessarily optimal (minimum). The main interest of this approach is related to the existence of polynomial algorithms. For instance, the algorithms LEX-M (Rose, Tarjan, and Lueker 1976) and LB (Berry 1999) have a time complexity in $O(ne')$ where e' is the number of edges in G'.
4. **Heuristic triangulation.** These methods generally build a perfect elimination order by adding some edges to the initial graph. They often achieve this work in polynomial time (generally between $O(n+e')$ and $O(n(n+e'))$) but they do not provide any minimality warranty. Nonetheless, in practice, they can be easily implemented and their interest seems justified (Kjaerulff 1990). Actually, Kjærulff has observed that these heuristics compute triangulations reasonably close to the optimum. The two most interesting heuristics are finally MCS and min-fill. MCS relies on the order computed by the algorithm of (Tarjan and Yannakakis 1984) which recognizes the triangulated graphs. Min-fill orders the vertices from 1 to n by choosing as next vertex one which leads to add a minimum of edges when completing the subgraph induced by its unnumbered neighbors.

Based on these results, in this paper, we will only use heuristic triangulations. In the next section, we analyze the triangulation of CL-Graphs to show how this operation can be exploited to rewrite SAT formulas.

Rewriting SAT instances

Consider a Boolean formula F, its CL-Graphs $G(F) = (V_F, E_F)$ and $Tr(G(F)) = (V_F, T)$ a triangulation of $G(F)$. We define $K = \{K_1, K_2, \dots K_N\}$, the set of maximal cliques of (V_F, T) where $N \leq n_F = |V_F|$ (theorem 5).

The most interesting property is now related to the fact that each satisfying assignment of F is covered by a maximal clique of (V_F, T) and then appears at least in one element K_i of K. Formally:

Theorem 9 *Given a Boolean formula $F = (X, C)$, a triangulation $Tr(G(F)) = (V_F, T)$ of $G(F) = (V_F, E_F)$ and the set $K = \{K_1, K_2, \dots K_N\}$ of the maximal cliques of (V_F, T), if $(l_1, l_2, \dots l_m)$ is a model of F, then $\exists K_i \in K$ such that $\{l_1, l_2, \dots l_m\} \subseteq K_i$.*

Proof: Consider $(l_1, l_2, \dots l_m)$ a satisfying assignment of F. By theorem 2, we know that $\{l_1, l_2, \dots l_m\}$ is a clique of size m in $G(F) = (V_F, E_F)$. Since $E_F \subseteq T$, each clique of $G(F)$ is a clique of $Tr(G(F)) = (V_F, T)$. Then, there is at least one maximal clique K_i of (V_F, T) which contains the clique $\{l_1, l_2, \dots l_m\}$. $\square$

For Example 1 and the triangulation of the CL-Graph of formula $\mathcal{F}$ given in figure 4, we obtain 4 maximal cliques :

- $K_1 = \{a = l_{1_1}, \neg b = l_{2_2}, \neg c = l_{3_2}, a = l_{4_1}, \neg b = l_{4_2}, c = l_{4_3}\}$.
- $K_2 = \{b = l_{1_2}, \neg a = l_{2_1}, \neg a = l_{3_1}, \neg c = l_{3_2}, a = l_{4_1}, c = l_{4_3}\}$.
- $K_3 = \{c = l_{1_3}, \neg a = l_{2_1}, \neg b = l_{2_2}, \neg a = l_{3_1}, a = l_{4_1}, \neg b = l_{4_2}, c = l_{4_3}\}$.
- $K_4 = \{\neg a = l_{2_1}, \neg b = l_{2_2}, \neg a = l_{3_1}, \neg c = l_{3_2}, a = l_{4_1}, \neg b = l_{4_2}, c = l_{4_3}\}$.

This formula has exactly 3 models : $A_1 = \{(a,1),(b,0),(c,0)\}$, $A_2 = \{(a,0),(b,0),(c,1)\}$ and $A_3 = \{(a,0),(b,1),(c,1)\}$. We can see that for each model, there is a maximal clique K_i that includes the clique corresponding to the model:

- Two cliques for A_1
  - $\{a = l_{1_1}, \neg b = l_{2_2}, \neg c = l_{3_2}, a = l_{4_1}\} \subset K_1$ and
  - $\{a = l_{1_1}, \neg b = l_{2_2}, \neg c = l_{3_2}, \neg b = l_{4_2}\} \subset K_3$.
- Four cliques for A_2:
  - $\{c = l_{1_3}, \neg a = l_{2_1}, \neg a = l_{3_1}, \neg b = l_{4_2}\} \subset K_3$ and
  - $\{c = l_{1_3}, \neg a = l_{2_1}, \neg a = l_{3_1}, c = l_{4_3}\} \subset K_3$ and
  - $\{c = l_{1_3}, \neg b = l_{2_2}, \neg a = l_{3_1}, \neg b = l_{4_2}\} \subset K_3$ and
  - $\{c = l_{1_3}, \neg b = l_{2_2}, \neg a = l_{3_1}, c = l_{4_3}\} \subset K_3$.
- One clique for A_3 : $\{b = l_{1_2}, \neg a = l_{2_1}, \neg a = l_{3_1}, c = l_{4_3}\} \subset K_2$.

With this example, one can see that the converse of the theorem 9 is not true since there is no model included in the maximal clique K_4.

We can observe that since each K_i is a subset of literals, for each maximal clique K_i, we can define an associated sub-formula denoted F_i such that $F_i = K_i(F)$. Here, $K_i(F)$ is the formula induced by F in considering only literals of clauses appearing in K_i. So by theorem 1, we obtain:

Theorem 10 *Given a Boolean formula $F = (X, C)$, a triangulation $Tr(G(F)) = (V_F, T)$ of $G(F) = (V_F, E_F)$ and the set $K = \{K_1, K_2, \dots K_N\}$ of the maximal cliques of (V_F, T), then $F \models F_1 \vee F_2 \vee \dots \vee F_N$ where $F_i = K_i(F)$ and moreover, $F_1 \vee F_2 \vee \dots \vee F_N \models F$.*

Proof: By theorem 9, each model $(l_1, l_2, \dots l_m)$ of F corresponds to a clique in a subset K_i. So, it is necessarily a model of $F_i = K_i(F)$. Conversely, consider a model $(l_1, l_2, \dots l_m)$ of F_i. Since F_i is a sub-formula of F, by theorem 1, $(l_1, l_2, \dots l_m)$ is a model of F. $\square$

So, this theorem allows us to define an algorithm for rewriting a Boolean formula F in another. Firstly, define the CL-Graph $G(F)$ of F. Consider a triangulation $Tr(G(F))$ of $G(F)$. List the maximal cliques (set K) of $Tr(G(F))$. Finally, define the disjunction of sub-formulas induced by K.

If we consider the formula $\mathcal{F}$ given in Example 1, we obtain a formula $\mathcal{F}_1 \vee \mathcal{F}_2 \vee \mathcal{F}_3 \vee \mathcal{F}_4$ semantically equivalent to $\mathcal{F}$, defined by the disjunction of four sub-formulas, one by maximal clique:

- $\mathcal{F}_1 = a \wedge \neg b \wedge \neg c \wedge (a \vee \neg b \vee c)$.
- $\mathcal{F}_2 = b \wedge \neg a \wedge (\neg a \vee \neg c) \wedge (a \vee c)$.
- $\mathcal{F}_3 = c \wedge (\neg a \vee \neg b) \wedge \neg a \wedge (a \vee \neg b \vee c)$.
- $\mathcal{F}_4 = \square \wedge (\neg a \vee \neg b) \wedge (\neg a \vee \neg c) \wedge (a \vee \neg b \vee c)$ ($\square$ denotes the empty clause)

The interest of such a rewriting is motivated by some hopes:

- Each formula F_i of the disjunction can be significantly easier than F (numerous clauses are significantly reduced, with sometimes unit clauses or empty clauses). In example 1, the sub-formula $\mathcal{F}_4$ can be suppressed since it contains the empty clause. Moreover, other sub-formulas contain several unit clauses.
- This approach can be exploited using a parallel approach, in solving each sub-formula.

Moreover, after a first application of theorem 10, we obtain a semantically equivalent formula. We can simplify the obtained disjunctive formula by applying a unit propagation on each sub-formula F_i to obtain then a simpler formula which preserves satisfiability of F as indicated in the next theorem:

Theorem 11 *Given a Boolean formula $F = (X, C)$, a triangulation $Tr(G(F)) = (V_F, T)$ of $G(F) = (V_F, E_F)$ and the set $K = \{K_1, K_2, \dots K_N\}$ of the maximal cliques of (V_F, T), then:*

[F is satisfiable] iff [$UP(F_1) \vee \dots \vee UP(F_N(F))$ is satisfiable]
where $F_i = K_i(F)$ and $UP(F_i)$ is the Boolean formula obtained by applying unit propagation to the formula F_i.

The proof of this theorem is a straightforward application of theorem 10 and basic properties of unit propagation.

On the formula given in Example 1, we obtain a formula semantically equivalent based on the disjunction of three sub-formulas, one for each maximal clique which do not induced an empty clause (as $\mathcal{F}_4$ does), that is $\mathcal{F}_1 \vee \mathcal{F}_2 \vee \mathcal{F}_3$:

- $UP(\mathcal{F}_1)$ produces the empty formula (not the empty clause) satisfied by the assignment $(a = 1, b = 0, c = 0)$
- $UP(\mathcal{F}_2)$ produces the empty formula satisfied by the assignment $(a = 0, b = 1, c = 1)$
- $UP(\mathcal{F}_3)$ produces the empty formula satisfied by the partial assignment $(a = 0, c = 1)$

Now, we can summarize the approach:

1. Compute the CL-Graph $G(F)$ of F
2. Triangulate $G(F)$ to obtain $Tr(G(F))$
3. List the maximal cliques $K = \{K_1, K_2, \dots K_N\}$ of $Tr(G(F))$
4. Define the rewritten formula $K_1(F) \vee K_2(F) \vee \dots \vee K_N(F)$
5. Run unit propagation on each subformula $F_i = K_i(F)$ to get each $UP(F_i)$

We show that the time complexity of the rewriting scheme is polynomial, assuming that we apply a polynomial time heuristic triangulation:

Theorem 12 *The time cost of the rewriting scheme is $O(n_F^2)$ if we consider a heuristic triangulation.*

Proof: Actually, Step 1 is computable linearly in the size of F for the vertices, that is $O(n_F)$ while creating edges is then bounded by $O(n_F^2) = O(n(n-1)/2)$ since $n_F(n_F - 1)/2$ is the maximum number of edges. The second step can be realized in linear time w.r.t. the size of the CL-Graph, that is $O(n_F^2)$ if we consider an heuristic algorithm as MCS. Step 3 is also computable linearly in the size of the CL-Graph, that is in $O(n_F^2)$, finding then $N \leq n_F$ maximal cliques. The fourth step can be realized in $O(n_F.N)$ since we define N sub-formulas each one containing at most n_F literals (as the input formula F). Finally, it is well known that unit propagation can be linearly realized. So the last step which consists in realizing N unit propagations can be realized in $O(n_F.N)$.$\square$

We defined a first scheme of rewriting where each part $FP_i = UP(F_i)$ in the rewritten formula is a sub-formula of F. If no sub-formula is trivially satisfiable (this can be stated by unit propagation), the process we described can easily be applied recursively to each sub-formula FP_i. After one level of rewriting, we obtain $FP = FP_1 \vee FP_2 \vee \dots \vee FP_{N_1}$ where $N_1 \leq N$ if we delete sub-formula FP_i containing at least one empty clause. At the second level, we consider independently each FP_i and then we apply the same rewriting done for the first step. This recursive application can be continued on several levels. Theorems 10 and 11 can be considered to prove the validity of this recursive approach. By this way, we have the guarantee that each sub-formula will be smaller than the one from which this sub-formula was obtained. From a complexity viewpoint, the number of sub-formulas for a given formula is bounded by its size (at most n_F). So, for a rewriting running on l levels (rewriting is recursively appllied l times), we obtain at most n_F^l sub-formulas, which is a roughly upper bound. Since the time cost for rewriting a sub-formula is $O(n_F^2)$, the total cost is then bounded by $O(n_F^2.n_F^l)$.

Conclusion

In this paper, we exploited the relation between SAT and the CLIQUE problem, studying a polynomial transformation from SAT to CLIQUE. Particularly, we have seen that thanks to such a transformation, we can exploit old and recent results issued from graph theory, related to Perfect Graphs. For example, we defined new sets of polynomal SAT instances. We have focussed our study on a particular class of Perfect Graphs, called Chordal Graphs, which can be used for practical cases. Precisely, we have introduced a new approach for rewriting CNF formulas in easier SAT formulas, using graphs algorithms.

We have different ways to extend this work. The most important is related to study the practical interest of such an approach. For example, we must show the classes of benchmarks which are well adapted to this kind of approach. Moreover, we must estimate the practical efficiency of such an approach on different classes of benchmarks. For that, we need to adapt triangulation algorithms to the particular structures and properties of CL-Graphs. For example, CL-Graphs are generally dense graphs the size of which can be important. So, it could be interesting to implement these graphs and their algorithms in manipulating their (sparse) complementary graphs. Moreover, literals issued from the same clause form an *independent set* (sub-graph without edge), and this basic property could be exploited to obtain better triangulations.

Acknowledgment

This work was partially supported by the ANR UNLOC French project (ANR-08-BLAN-0289-04).

References

Amir, E. 2001. Approximation for triangulation of minimum treewidth. In *Proceedings of UAI*, 7–15.

Amir, E. to appear. Approximation algorithms for treewidth. *Algorithmica* 28.

Arnborg, S.; Corneil, D.; and Proskuroswki, A. 1987. Complexity of finding embeddings in a k-tree. *SIAM Journal of Discrete Mathematics* 8:277–284.

Berry, A. 1999. A Wide-Range Efficient Algorithm for Minimal Triangulation. In *Proceedings of SODA'99 SIAM Conference*.

Bouchitté, V.; Kratsch, D.; Muller, H.; and Todinca, I. 2004. On treewidth approximations. *Discrete Appl. Math.* 136(2-3):183–196.

Chudnovsky, M.; Cornuejols, G.; Seymour, X. L. P.; and Vuskovic, K. 2005. Recognizing berge graphs. *Combinatorica* 25:143–186.

Chudnovsky, M.; Robertson, N.; Seymour, P.; and Thomas, R. 2006. The strong perfect graph theorem. *Ann. Math.* 164:51–229.

Dechter, R., and Pearl, J. 1989. Tree-Clustering for Constraint Networks. *Artificial Intelligence* 38:353–366.

en, N. E., and orensson, N. S. 2003. An extensible sat-solver. In *Proceedings of SAT 2003*, 402–518.

Fomin, F.; Kratsch, D.; and Todinca, I. 2004. Exact (exponential) algorithms for treewidth and minimum fill-in. In *Proceedings of ICALP*, 568–580.

Fulkerson, D., and Gross, O. 1965. Incidence matrices and interval graphs. *Pacific J. Math.* 15:835–855.

Gavril, F. 1972. Algorithms for minimum coloring, maximum clique, minimum covering by cliques, and maximum independent set of a chordal graph. *SIAM Journal on Computing* 1 (2):180–187.

Gogate, V., and Dechter, R. 2004. A complete anytime algorithm for treewidth. In *Proceedings of UAI*, 201–208.

Golumbic, M. 1980. *Algorithmic Graph Theory and Perfect Graphs*. Academic Press. New-York.

Grotschel, M.; Lovasz, L.; and Schrijver, A. 1988. *Geometric Algorithms and Combinatorial Optimization*. Springer-Verlag, See chapter 9, Stable Sets in Graphs, pp. 273-303.

Jégou, P. 1993. Decomposition of Domains Based on the Micro-Structure of Finite Constraint Satisfaction Problems. In *Proceedings of AAAI 93*, 731–736.

Karp, R. 1972. *reducibility among combinatorial problems*. In *Complexity of Computer Computations* of R. Miller and J. Thatcher, ed. Plenum Press. 85–103.

Kjaerulff, U. 1990. Triangulation of Graphs - Algorithms Giving Small Total State Space. Technical report, Judex R.R. Aalborg., Denmark.

Robertson, N., and Seymour, P. 1986. Graph minors II: Algorithmic aspects of treewidth. *Algorithms* 7:309–322.

Rose, D.; Tarjan, R.; and Lueker, G. 1976. Algorithmic Aspects of Vertex Elimination on Graphs. *SIAM Journal on computing* 5:266–283.

Salamon, A. Z., and Jeavons, P. G. 2008. Perfect constraints are tractable. In *Proceedings of CP*, 524–528.

Shoikhet, K., and Geiger, D. 1997. A practical algorithm for finding optimal triangulation. In *Proceedings of AAAI*, 185–190.

Surynek, P. 2007. Solving difficult sat instances using greedy clique decomposition. In *Proceedings of the 7th Symposium on Abstraction, Reformulation, and Approximation (SARA 2007)*, 359–374. Whistler, Canada, Springer-Verlag LNAI 4612.

Tarjan, R., and Yannakakis, M. 1984. Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs. *SIAM Journal on Computing* 13 (3):566–579.

Todinca, I. 2005. Private communication.

Walsh, T. 2000. Sat v csp. In *Proceedings of the Sixth International Conference on Principles and Practice of Constraint Programming (CP-2000)*, 441–456. Springer-Verlag LNCS-1894.