A Tree Decomposition Based Approach to Solve Structured SAT Instances*

Djamal Habet   Lionel Paris   Cyril Terrioux
LSIS – UMR CNRS 6168
Université Paul Cézanne, Marseille, France
{Djamal.Habet, Lionel.Paris, Cyril.Terrioux}@lsis.org

Abstract

The main purpose of the paper is to solve structured instances of the satisfiability problem. The structure of a SAT instance is represented by an hypergraph, whose vertices correspond to the variables and the hyper-edges to the clauses. The proposed method is based on a tree decomposition of this hyper-graph which guides the enumeration process of a DPLL-like method. During the search, the method makes explicit some information which is recorded as structural goods and nogoods. By exploiting this information, the method avoids some redundancies in the search, and so it guarantees a bounded theoretical time complexity which is related to the tree-decomposition. Finally, the method is assessed on structured SAT benchmarks.

1 Introduction

Propositional satisfiability (SAT) is the problem of deciding whether a Boolean formula in the Conjunctive Normal Form (CNF) is satisfiable. SAT is one of the most studied NP-Complete problems because of its theoretical and practical importances. Encouraged by the impressive progress in the practical solving of SAT, various applications ranging from formal verification to planning are encoded and solved using SAT. Most of the successful complete SAT solvers are based on a backtrack search algorithm called Davis-Putnam-Logemann-Loveland (DPLL) procedure [1]. Such basic algorithms are enhanced with many important pruning techniques like learning, extended use of Boolean constraint propagation, preprocessing, symmetries breaking, etc. The impact of these different improvements depends on the kind of the treated instances. For example, learning is more relevant when solving instances encoding real world problems than the randomly generated ones.

Improving the efficiency of the solving methods by exploiting the problem structure have been widely studied in CSP (Constraint Satisfaction Problem) (for example, see [2]) and less in SAT. Here, the structure of a problem corresponds to its structural properties which can be represented and captured by the theoretical properties of the (hyper)graph representing that problem. Concerning SAT, the structure of the instances modeling real-world applications is the result of the modular design of these problems, such that the models have a minimal interconnectivity and each one has its distinct variables [3]. Hence, decomposing the initial SAT problem into a set of subproblems may facilitate its solving. Accordingly, the main purpose of the work presented in this paper is to provide a new approach based on the tree-decomposition of the graph representing a SAT instance. This decomposition offers an order on the problem variables which will be exploited by a DPLL-like method. Moreover, during the search, many information which correspond either to some fails or to some successes when attempting to solve the (sub)problem(s) are learned as goods and nogoods and such learning can permit pruning the search space. Hence, the proposed approach theoretically bounds the time complexity of the solving.

The paper is organized as follows. Section 2 introduces the necessary notions and notations. Section 3 presents our tree-decomposition based approach (named DPLL-TD) by giving its theoretical basis and detailing its algorithmic description. It also gives the proof of its soundness, completeness and termination, then exhibits a proof of its complexity in time and space. Section 4 presents some implementation details of DPLL-TD and gives the results obtained on structured SAT benchmarks issued from the previous SAT competitions. Section 5 presents some existing works on the use of the tree-decomposition for solving SAT. Finally, Section 6 discusses and concludes the paper.

2 Basic Notions

This section is dedicated to the definition of the SAT problem, to a reminder of the DPLL algorithm and to the introduction of some concepts of the graph theory which are all necessary to understand the rest of the paper.

*This work is supported by ANR UNLOC French project (ANR-08-BLAN-0289-04).

2.1 About SAT

A satisfiability instance $\mathcal{F}$ is defined by $\mathcal{F} = (\mathcal{X}, \mathcal{C})$, where $\mathcal{X}$ is a set of boolean variables (taking their values from the set $\{true,\ false\}$) and $\mathcal{C}$ is a set of clauses. A clause is a finite disjunction of literals and a literal is either a variable or its negation. For a given literal l, $var(l) = \{v | l = v \ or \ l = \neg v\}$ is the singleton-set of the variable which concerns l. Furthermore, a literal is viewed as a clause with only one literal which matches with the definition of a unit clause. Moreover, for a given clause c, the set $var(c) = \cup_{l \ in \ c}\, var(l)$ defines all the variables that are involved in c ($l \ in \ c$ means that the literal l appears in c). For example, if $c = x_1 \vee \neg x_2 \vee \neg x_3$ then we have $var(\neg x_2) = \{x_2\}$ and $var(c) = \{x_1,\ x_2,\ x_3\}$.

A truth assignment I of the variables of $\mathcal{F}$ is represented by a set of literals that verifies the condition $\forall (l_1,\ l_2) \in I^2$ such that $l_1 \neq l_2$, we have $var(l_1) \neq var(l_2)$. A variable that appears positively (resp. negatively) in I means that it is fixed to the value $true$ (resp. $false$). Besides, a truth assignment I of the variables $\mathcal{X}$ is said to be partial if $|I| < |\mathcal{X}|$ and complete if $|I| = |\mathcal{X}|$ (all the variables are fixed). Moreover, given a truth assignment I of a set of a variables $Y \subseteq \mathcal{X}$ and the subset $Z \subseteq Y$, $I[Z]$ is the projection of I on the variables of Z. A model for $\mathcal{F}$ is a truth assignment which satisfies all the clauses of $\mathcal{F}$. Finally, $Sol(\mathcal{F})$ denotes the set of all the models of $\mathcal{F}$. Accordingly, the satisfiability problem (SAT) consists in determining whether a CNF formula $\mathcal{F}$ admits a model ($Sol(\mathcal{F}) \neq \emptyset$). If it is the case, $\mathcal{F}$ is said satisfiable, otherwise $\mathcal{F}$ is unsatisfiable.

2.2 About DPLL Algorithm

Despite its simplicity and seniority, the Davis-Putnam-Logemann-Loveland procedure (DPLL) [1] is one of the best complete procedures for SAT. DPLL procedure is a backtracking algorithm: at each step, it chooses a variable according to some branching heuristic (line 6 in Algorithm 1), satisfies this variable, simplifies the SAT instance and then recursively checks if the simplified instance can be satisfied (line 7). If this is the case then the initial SAT instance is satisfiable. Else, an identical recursive call is achieved assuming the opposite truth value for the current branching variable (line 8). The simplification step essentially deletes the satisfied clauses and reduces the size of the clauses containing falsified literals.

Hence, the DPLL algorithm constructs a binary search tree where its nodes are results of the recursive calls. While a solution is not found, all leaves represent a dead-end corresponding to a contradiction (an empty clause denoted by $\Box$). DPLL procedure performance is closely related to the selection of the branching variable which affects the search tree size and consequently the required time to solve $\mathcal{F}$.

Algorithm 1: DPLL(in: $\mathcal{C}$, I)

```
1 Unit-propagation ($\mathcal{C}$, $I$)
2 if $\Box \in \mathcal{C}$ then return $false$
3 else
4     if $\mathcal{C} = \emptyset$ then return $true$
5     else
6         Choose an unassigned variable $v$ (branching heuristic)
7         if DP ($\mathcal{C} \cup \{v\}, I \cup \{v\}$) then return $true$
8         else return DP ($\mathcal{C} \cup \{\neg v\}, I \cup \{\neg v\}$)
```

Algorithm 2: UnitPropagation(in/out: $\mathcal{C}$, I)

```
1 while there is no empty clause and a unit clause $l$ exists in $\mathcal{F}$ do
2     $I \leftarrow I \cup \{l\}$ (satisfy $l$)
3     simplify $\mathcal{C}$
```

2.3 About Graph Theory

As it was announced in the introduction, our aim is to exploit the structural properties of the SAT instances expressed by their graph representation which will be decomposed accordingly. Hence, we should formulate and recall some definitions related to these points. At first, let us define the graph representation of a SAT instance that we use in the hope to capture its structure.

Definition 1 *Let $\mathcal{F} = (\mathcal{X}, \mathcal{C})$ be a SAT instance. The hypergraph $H = (\mathcal{V}, \mathcal{E})$ that characterizes $\mathcal{F}$ is defined such that each variable in $\mathcal{X}$ is represented by a vertex in $\mathcal{V}$. Moreover, each clause $c \in \mathcal{C}$ is represented by an hyperedge $e \in \mathcal{E}$ which is a subset of $\mathcal{V}$, such that $var(c) = e$.*

For example, consider the SAT instance $\mathcal{F} = (\{x_1, x_2, x_3\}, \{x_1 \vee \neg x_2 \vee x_3,\ \neg x_1 \vee \neg x_2 \vee x_3,\ x_1 \vee \neg x_3,\ \ x_2 \vee x_3,\ \ x_1 \vee x_4\})$ then its corresponding hypergraph is defined by $H = (\{x_1, x_2, x_3\}, \{\{x_1, x_2, x_3\},\ \{x_1, x_3\},\ \{x_2, x_3\},\ \{x_1, x_4\}\})$.

However, this hypergraph representation cannot be directly exploited in terms of the tree decomposition (defined below) which is defined on graphs and not on hypergraphs. Hence, we need to define the primal graph associated to the hypergraph as follows.

Definition 2 *The primal graph of an hypergraph $H = (\mathcal{V}, \mathcal{E})$ is the graph $G_H = (\mathcal{V}, \mathcal{E}_H)$ such that $\mathcal{E}_H = \{\{x, y\} | x, y \in \mathcal{V} \ and \ \exists\ e \in \mathcal{E}, \{x, y\} \subseteq e\}$. We denote by $G_{(\mathcal{X}, \mathcal{C})}$ the primal graph associated to the SAT instance $\mathcal{F} = (\mathcal{X}, \mathcal{C})$.*

Let us consider the above SAT formula $\mathcal{F}$ and its hypergraph representation H. According to the last definition, the primal graph associated to H is $G_H = (\{x_1, x_2, x_3\}, \{\{x_1, x_2\},\ \{x_1, x_3\},\ \{x_2, x_3\},\ \{x_1, x_4\}\})$.

Now, we can introduce the tree-decomposition (as defined in [4]) which uses this primal graph representation of a SAT instance. This decomposition might allow a judicious exploitation of the structural characteristics of the instance.

Definition 3 *Let $G_{(\mathcal{X},\mathcal{C})} = (\mathcal{V}, \mathcal{E}_H)$ be the (primal) graph associated to a SAT instance $\mathcal{F} = (\mathcal{X}, \mathcal{C})$. A tree-decomposition of $G_{(\mathcal{X},\mathcal{C})}$ is a pair $(E, \mathcal{T})$ where $\mathcal{T} = (J, F)$ is a tree with nodes J and edges F and $E = \{E_i : i \in J\}$ a family of subsets of $\mathcal{V}$, such that each subset (called a cluster) E_i is a node of $\mathcal{T}$ and verifies: (i) $\cup_{i \in J} E_i = \mathcal{V}$, (ii) for each edge $\{x, y\} \in \mathcal{E}_H$, there exists $i \in J$ with $\{x, y\} \subseteq E_i$, and (iii) for all $i, j, k \in J$, if k is in a path from i to j in $\mathcal{T}$, then $E_i \cap E_j \subseteq E_k$. The width of a tree-decomposition $(E, \mathcal{T})$ is equal to $max_{i \in J}|E_i| - 1$. The tree-width w of $G_{(\mathcal{X},\mathcal{C})}$ is the minimal width over all the tree-decompositions of $G_{(\mathcal{X},\mathcal{C})}$.*

We denote by $Desc(E_j)$ the set of variables belonging to E_j or to a descendant E_k of E_j. Also, let E_i be a cluster and E_j one of its children, $E_{par(j)}$ is the parent cluster E_i of E_j and we assume that $E_{par(1)} = \emptyset$ (the root cluster). Moreover, the set $\mathcal{C}[E_i]$ contains the clauses belonging exclusively to the cluster E_i. In other words we have $\mathcal{C}[E_i] = \{c \in \mathcal{C} | var(c) \subseteq E_i \text{ and } var(c) \not\subseteq E_{par(i)}\}$.

As an example, Figure 1 shows a tree-decomposition corresponding to some SAT instance (not given). $E = \{E_i | i = 1 \cdots 7\}$ is the set of clusters and E_1 is the root one of this tree-decomposition. E_1 has three children E_2, E_3 and E_4, $E_2 = \{x_1, x_2, x_6, x_7\}$, $E_1 \cap E_2 = \{x_1, x_2\}$ (this intersection is called the *separator* between the cluster E_2 and its parent E_1), $E_{par(3)} = E_1$, and finally $Desc(E_3) = \{x_3, x_4, x_8, x_{15}, x_{18}, x_{20}\}$.

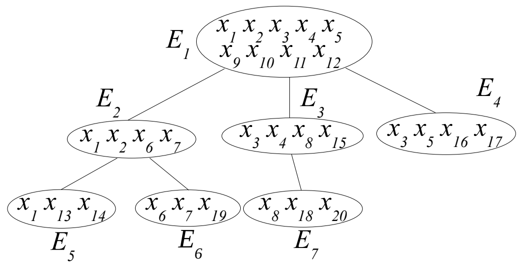

Figure 1. A tree-decomposition example.

Before discussing the theoretical and algorithmic details of our approach, we will first give the basic idea behind it. The tree-decomposition of the initial instance divides it into independent (and smaller) parts which correspond to the clusters, but which are still linked by the separators. Solving the initial instance amounts to solve its various parts (clusters). Starting from the root and using a depth-first-search algorithm, each cluster E_i is attacked separately. If all the clauses restricted to this cluster are satisfied then we will attack one of its children E_j (if exists) by trying to extend the truth assignment of the variables of E_i to those of E_j. To be more precise, it is necessary to satisfy the clauses in E_j with the respect of the constraints expressed by the truth assignment of the variables in the separator between E_i and E_j. In fact, these variables are shared between these two clusters and consequently should be assigned identically. If this process fails then we can deduce that it is never possible to extend the truth assignment of the separator variables to a model. More interesting still, storing this information could be helpful to avoid repeating the same treatment in the case of a backtrack on the cluster E_i and the occurrence of the same truth assignment of the separator variables. In the same way and for the same reasons, if the clauses of the cluster E_i and those of clusters of its descent are satisfied then it is also useful to store the truth assignment of the variables of the separator as a scalable truth assignment for this part of the instance. To summarize, we apply a backtrack search on the tree-decomposition of the SAT instance, while saving any useful information to prune the search space.

3 A Tree Decomposition Based Approach for SAT

This section represents the major contribution of our work. As a first step, it outlines the various theoretical aspects related to the use of the tree-decomposition for SAT. In a second time, it introduces and details DPLL-TD which is the algorithmic translation of these theoretical aspects. Finally, we exhibit evidences on the completeness and soundness of our algorithm and its complexity in time and space.

3.1 Theoretical Foundations

In the following, we consider an instance $\mathcal{F} = (\mathcal{X}, \mathcal{C})$ and a tree-decomposition $(E, \mathcal{T})$ associated to the graph $G_{(\mathcal{X},\mathcal{C})}$. The aim of the first theorem above (easy to prove) is to show that a tree-decomposition of the graph $G_{(\mathcal{X},\mathcal{C})}$ does not change the initial instance (no information is lost) and a clause belongs to a single cluster.

Theorem 1 *The sets $(\mathcal{C}[E_i])_i$ form a partition of $\mathcal{C}$.*

By the use of the definition of $\mathcal{C}[E_i]$, let us introduce now the concept of a subproblem induced by a truth assignment.

Definition 4 *Let E_i be a cluster. The subproblem $\mathcal{F}_{E_i,I}$ rooted in E_i and induced by a truth assignment I on a subset of $E_i \cap E_{par(i)}$ is $(Desc(E_i), \bigcup_{E_k \subseteq Desc(E_i)} \mathcal{C}[E_k] \cup \{l | l \in I \text{ and } var(l) \subseteq E_i \cap E_{par(i)}\})$*

In other words, $\mathcal{F}_{E_i,I}$ is formed by all the clauses and the variables of the cluster E_i and its descendant clusters with additional constraints expressing the current truth assignment of the variables in the separator between E_i and its parent. These constraints are simply formulated by the unit clauses $\{l | l \in I \textit{ and } var(l) \subseteq E_i \cap E_{par(i)}\}$. This definition means also that solving the sub-problem corresponding to the subtree rooted in E_i must respect the truth assignment of the variables that it shares with its parent.

Property 1 *Given two truth assignments I and I' such that $I \subseteq I'$ and a cluster E_i, we have $Sol(\mathcal{F}_{E_i,I'}) \subseteq Sol(\mathcal{F}_{E_i,I})$.*

Proof: $\mathcal{F}_{E_i,I'}$ only differs from $\mathcal{F}_{E_i,I}$ in having some additional clauses, namely the unit clauses l such that $l \in I' - I$ and $var(l) \subseteq E_i \cap E_{par(i)}$. So, we have necessarily $Sol(\mathcal{F}_{E_i,I'}) \subseteq Sol(\mathcal{F}_{E_i,I})$.$\square$
This property characterizes the existing relation between the models of the subproblems rooted in the same cluster but induced by two different truth assignments where one (I') extends the other (I).

Definition 5 *A variable v is an independence variable if there exists a cluster E_i and two of its children E_j and E_k such that $v \in E_i \cap E_j \cap E_k$. We denote $\mathcal{X}_{ind}$ the set of independence variables of $\mathcal{F}$ with respect to the considered tree-decomposition.*

Independence variables have a particular role in the tree-decomposition of a SAT instance. Specifically, in order to decompose in a valid way a SAT instance into a set of independent sub-problems, we must ensure that these variables are assigned with the same value in the various sub-problems in which they operate. For example, according to the Figure 1, there is one independence variable x_3 located on the intersection of E_1 with two of its children E_3 and E_4. Hence $\mathcal{X}_{ind} = \{x_3\}$. If x_3 is not assigned when treating E_1 (recall that it is possible to satisfy a set of clauses without necessary fixing all the variables occurring in these clauses) then x_3 must be assigned identically when treating independently the subproblems rooted in E_3 and E_4. The following theorem formally establishes the independence between the sub-problems in terms of the independence variables.

Theorem 2 *Given a cluster E_i, E_j and E_k two of its children and a truth assignment I on a subset Y of E_i, if $\mathcal{X}_{ind} \cap E_i \subseteq Y$ then the subproblems $\mathcal{F}_{E_j,I[E_i \cap E_j]}$ and $\mathcal{F}_{E_k,I[E_i \cap E_k]}$ are independent.*

Proof: By definition of a tree-decomposition, $Desc(E_j) \cap Desc(E_k) = E_j \cap E_k \subseteq E_i$. Clearly, we have $E_j \cap E_k \subseteq \mathcal{X}_{ind}$ and so $E_j \cap E_k \subseteq \mathcal{X}_{ind} \cap E_i$. As the variables of $\mathcal{X}_{ind} \cap E_i$ are assigned in I, it ensues that these variables have the same values in any truth assignment which satisfies in $\mathcal{F}_{E_j,I[E_i \cap E_j]}$ and $\mathcal{F}_{E_k,I[E_i \cap E_k]}$ thanks to the unit clauses l s.t. $l \in I$ and $var(l) \subseteq E_i \cap E_j \cap E_k$. Moreover, these unit clauses are the only clauses shared by $\mathcal{F}_{E_j,I[E_i \cap E_j]}$ and $\mathcal{F}_{E_k,I[E_i \cap E_k]}$ (according to Theorem 1). Hence the subproblems $\mathcal{F}_{E_j,I[E_i \cap E_j]}$ and $\mathcal{F}_{E_k,I[E_i \cap E_k]}$ are independent. $\square$
The next corollary states that the satisfiability of the subproblems rooted in brother clusters led to the satisfiability of the subproblem rooted on their parent cluster.

Corollary 1 *Given a cluster E_i and a truth assignment I on a subset Y of E_i, if $\mathcal{X}_{ind} \cap E_i \subseteq Y$, I satisfies the clauses of $\mathcal{C}[E_i]$, and, for each child E_j of E_i, the subproblem $\mathcal{F}_{E_j,I[E_i \cap E_j]}$ is satisfiable, then I can be extended to a model of $\mathcal{F}_{E_i,I[E_i \cap E_{par(i)}]}$.*

Proof: Let M_{E_j} be a model of $\mathcal{F}_{E_j,I[E_i \cap E_j]}$. By definition of $\mathcal{F}_{E_j,I[E_i \cap E_j]}$, there is no variable v such that $I \cup M_{E_j}$ contains two opposite literals of v. According to theorem 2, for any children E_j and E_k of E_i, $\mathcal{F}_{E_j,I[E_i \cap E_j]}$ and $\mathcal{F}_{E_k,I[E_i \cap E_k]}$ are independent. So the truth assignment $I \cup \bigcup_{E_j \in children(E_i)} M_{E_j}$ satisfies both the clauses of $\mathcal{C}[E_i]$ and ones of $\mathcal{F}_{E_j,I[E_i \cap E_j]}$ for each child E_j of E_i. So I can be extended to a model of $\mathcal{F}_{E_i,I[E_i \cap E_{par(i)}]}$.$\square$
As explained just before the beginning of this section, we also want to keep track of any useful information that will allow us to unnecessarily repeat several times the same treatment, namely solving the same subproblems. This corresponds to making cuts in the branches of the tree exploring the search space in the presence of information indicating whether the satisfiability of the sub-problem being processed. Such information are goods and nogoods which we define formally as follows.

Definition 6 *Given a cluster E_i, a truth assignment I on a subset of $E_i \cap E_{par(i)}$ is a structural good (respectively nogood) of E_i if any extension of I on $E_i \cap E_{par(i)}$ can be extended to a model of $\mathcal{F}_{E_i,I}$ (resp. if $Sol(\mathcal{F}_{E_i,I}) = \emptyset$).*

In other words, a structural good (resp. nogood) is a truth assignment I on a subset of $E_i \cap E_{par(i)}$ which can (resp. cannot) be extended to a model of $\mathcal{F}_{I,E_i}$. Remark that these (no)goods are recorded on the separator between E_i and $E_{par(i)}$

Property 2 *Given a cluster E_i and a subset $Y \subseteq \mathcal{X}$ such that $Desc(E_i) \cap Y \subseteq E_i \cap E_{par(i)}$, for any good g of E_i, every truth assignment I on Y can be extended to a model of $\mathcal{F}_{E_i,I[E_i \cap E_{par(i)}]}$ if there exists an extension e_g of g on $E_i \cap E_{par(i)}$ such that $I[E_i \cap E_{par(i)}] \subseteq e_g$.*

Proof: By definition of a good, e_g and so $I[E_i \cap E_{par(i)}]$ (since $I[E_i \cap E_{par(i)}] \subseteq e_g$) can be extended to a model M of $\mathcal{F}_{E_i,g}$. It ensues that M satisfies the clauses of $\bigcup_{E_k \subseteq Desc(E_i)} \mathcal{C}[E_k]$. Moreover, it also satisfies the unit clauses of $\{l | l \in I \textit{ and } var(l) \subseteq E_i \cap E_{par(i)}\}$ since $I[E_i \cap E_{par(i)}] \subseteq e_g \subseteq M$. So M is a model of $\mathcal{F}_{E_i,I[E_i \cap E_{par(i)}]}$ and $I[E_i \cap E_{par(i)}]$ can be extended to a model of $\mathcal{F}_{E_i,I[E_i \cap E_{par(i)}]}$. $\square$

The previous property gives the condition that allows us to make a cut according to the recorded goods. Extending e_g to g is the operation of completing g by interpreting some (or all) of the unassigned variables on the separator $E_i \cap E_{par(i)}$ in g, if necessary to ensure that $I[E_i \cap E_{par(i)}] \subseteq e_g$, otherwise we have $e = e_g$ (and also when $|e| = |E_i \cap E_{par(i)}|$). In a more obvious manner, the next property expresses the cut conditions by the nogoods.

Property 3 *Given a cluster E_i and a subset $Y \subseteq \mathcal{X}$ such that $Desc(E_i) \cap Y \subseteq E_i \cap E_{par(i)}$, for any nogood ng of E_i, no truth assignment I on Y such that $ng \subseteq I$ can be extended to a model of $\mathcal{F}_{E_i, I[E_i \cap E_{par(i)}]}$.*

Proof: We have $Sol(\mathcal{F}_{E_i, I[E_i \cap E_{par(i)}]}) \subseteq Sol(\mathcal{F}_{E_i, ng})$ (property 1). Moreover, by definition of a nogood, $Sol(\mathcal{F}_{E_i, ng}) = \emptyset$. So $Sol(\mathcal{F}_{E_i, I[E_i \cap E_{par(i)}]}) = \emptyset$ and I cannot be extended to a model on $Desc(E_i)$. $\Box$

Now that we made and proved all the theoretical elements related to our approach, we describe their algorithmic exploitation below.

3.2 The DPLL-TD Algorithm

Consider a SAT instance $\mathcal{F} = (\mathcal{X}, \mathcal{C})$ and a tree-decomposition $(E, \mathcal{T})$ of the primal graph obtained from the hypergraph representation of $\mathcal{F}$. Algorithm 3 describes the DPLL-TD (for *DPLL with Tree Decomposition*) method to solve the satisfiability problem. During the search, DPLL-TD attempts to extend the current truth assignment to a model, if it exists, guided by the tree-decomposition of $\mathcal{F}$. Moreover, DPLL-TD exploits the information previously learned (the set of goods $\mathcal{G}$ and nogoods $\mathcal{N}$) in order to prune the search space. Hence, DPLL-TD($\mathcal{C}, I, E_i, \mathcal{G}, \mathcal{N}$) returns $true$ (resp. $false$) if the subproblem $\mathcal{F}_{E_i, I}$ rooted in E_i is satisfiable (resp. unsatisfiable). Before giving more details, some notations and precisions are necessary. The set of goods of a given cluster E_i are represented and recorded in $\mathcal{G}_{E_i}$ and $\mathcal{G}$ is the set defined by $\mathcal{G} = \{\mathcal{G}_{E_i} | E_i \in E\}$. Moreover, the set of nogoods is $\mathcal{N}$ and $\mathcal{N}[E_i]$ is the set of clauses (representing the nogoods as it will be explained below) belonging only to the cluster E_i. So, the definition of $\mathcal{N}[E_i]$ matches with this of $\mathcal{C}[E_i]$ given before.

The first call of DPLL-TD is done with the parameters $(\mathcal{C}, \emptyset, E_1, \mathcal{G}, \mathcal{N})$, where E_1 is the root of the tree-decomposition. In fact, initially there are no assigned variables and no recorded (no)goods ($\mathcal{G}_{E_i} = \emptyset$ and $\mathcal{N} = \emptyset$). The first step of the algorithm (line 1) is the propagation of the unit clauses belonging to $\mathcal{C} \cup \mathcal{N}$. Accordingly, if an empty clause is found then the algorithm returns $false$ which means that $\mathcal{F}$ is unsatisfiable. Else, if $\mathcal{C} \cup \mathcal{N}$ is emptied then $\mathcal{F}$ is satisfiable and the algorithm returns $true$ (line 4).

Algorithm 3: DPLL-TD(in: $\mathcal{C}, I, E_i$, in/out: $\mathcal{G}, \mathcal{N}$)

```
1  Unit-propagation ($\mathcal{C} \cup \mathcal{N}$, $I$)
2  if $\Box \in \mathcal{C} \cup \mathcal{N}$ then return $false$
3  else
4      if $\mathcal{C} \cup \mathcal{N} = \emptyset$ then return $true$
5      else
6          if $(\mathcal{C} \cup \mathcal{N})[E_i] = \emptyset$ and $\mathcal{X}_{ind} \cap E_i = \emptyset$ then
7              $sat \leftarrow true$
8              $S \leftarrow children(E_i)$
9              while $sat$ and $S \neq \emptyset$ do
10                 Choose a cluster $E_j$ in $S$
11                 $S \leftarrow S - \{E_j\}$
12                 if $\exists g \in \mathcal{G}_{E_j}, \exists e_g$ extension of $g$ on $E_i \cap E_j$, $I[E_i \cap E_j] \subseteq e_g$ then
13                     $I \leftarrow I \cup g$
14                 else
15                     $sat \leftarrow$ DPLL-TD ($\mathcal{C}, I, E_j, \mathcal{G}, \mathcal{N}$)
16                     if $sat$ then
17                         Let $g \in \mathcal{G}_{E_j}$ be the last recorded good
18                         $I \leftarrow I \cup g$
19                     else $\mathcal{N} \leftarrow \mathcal{N} \cup \{\bigvee_{l_k \in I[E_i \cap E_j]} \neg l_k\}$
20             if $sat$ then $\mathcal{G}_{E_i} \leftarrow \mathcal{G}_{E_i} \cup \{I[E_i \cap E_{par(i)}]\}$
21             return $sat$
22         else
23             Choose an unassigned variable $v$ in $E_i$
24             if DPLL-TD ($\mathcal{C} \cup \{v\}, I \cup \{v\}, E_i, \mathcal{G}, \mathcal{N}$) then
25                 return $True$
26             else return DPLL-TD ($\mathcal{C} \cup \{\neg v\}, I \cup \{\neg v\}, E_i, \mathcal{G}, \mathcal{N}$)
```

Now, consider the set of clauses ($(\mathcal{C} \cup \mathcal{N})[E_i]$) and the set of independence variables $\mathcal{X}_{ind} \cap E_i$ restricted to the considered cluster E_i. If one of these sets is not empty, Algorithm 3 proceeds to a DPLL enumeration on E_i in order to fix some of its variables (lines 23-26). For example, in Figure 1, $x_3 \in \mathcal{X} \cap E_1$. If x_3 is not assigned then an enumeration on x_3 and possibly on the unfixed variables of $(\mathcal{C} \cup \mathcal{N})[E_1]$ is done to ensure the same truth value for x_3 in both the clusters E_1, E_2 and E_3. Hence, the independence of subproblems rooted in E_3 and E_4 is guaranteed (Corollary 1). Otherwise (the condition in line 6 is checked), DPLL-TD treats the subproblems rooted on the clusters E_j children of E_i (recall that if all the subproblems $\mathcal{F}_{E_j, I[E_i \cap E_j]}$ are satisfied then $\mathcal{F}_{E_i, I}$ is also satisfied).

When treating the children clusters of E_i, the previously recorded (no)goods can be useful. Indeed, DPLL-TD checks if there exists a good g in $\mathcal{G}_{E_j}$ between E_j (child of E_i) and E_i which verifies the cut condition expressed by Property 2 (line 12). If g is found then $\mathcal{F}_{E_j, I[E_i \cap E_j]}$ is satisfiable and it is unnecessary to address it again. Otherwise, the satisfiability of $\mathcal{F}_{E_j, I[E_i \cap E_j]}$ is unknown and it will be determined by the recursive call DPLL-TD($\mathcal{C}, I, E_j, \mathcal{G}, \mathcal{N}$). According to the last case, $\mathcal{F}_{E_j, I[E_i \cap E_j]}$ is unsatisfiable (the last call returned $false$) means that a nogood is detected between E_j and E_i. This one is represented and recorded as a new clause defined by $\vee_{l_k \in I[E_i \cap E_j]} \neg l_k$ which is added to the set of nogoods $\mathcal{N}$ (line 19). For example, consider

Figure 1, the cluster E_1, its child E_2 and a truth assignment on $E_1 \cap E_2 = \{x_1, x_2\}$ which is $I[\{x_1, x_2\}] = \{\neg x_1, x_2\}$. If $F_{2,\{\neg x_1, x_2\}}$ is unsatisfiable then $\{\neg x_1, x_2\}$ is a nogood, which is recorded by the clause $x_1 \vee \neg x_2$ in $\mathcal{N}$. Hence, DPLL-TD backtracks on the variables of E_i and tests the existence of a new truth assignment which satisfies $(\mathcal{C} \cup \mathcal{N})[E_i]$. Note that even if the learned clauses (nogoods) are saved in a distinct set $\mathcal{N}$, they are considered like any other clause in $\mathcal{C}$. This distinction is made in order to highlight the nogoods learned during the search. Moreover, this nogood representation does not require any particular treatment, mainly testing whether a cut can be achieved through nogoods. Thus, nogood processing is hidden for DPLL-TD and the learned clauses may enhance the effectiveness of the unit propagations (filtering).

Finally, the lines 13 and 18 correspond to the extension of the current truth assignment according to the used good in the case of a cut (by Property 2). Hence, I is extended by adding the literals of the good g to those of I ensuring the correctness of the good recording on E_i if $\mathcal{F}_{E_i,I}$ is satisfiable (line 20) where a good corresponds to the obtained truth assignment of the variables of $E_i \cap E_{par(i)}$. Note that, because the independence variables are set in advance, no contradiction can be found during the extension of the current truth assignment. For example, reconsider Figure 1, the cluster E_2, its child E_6 and a truth assignment on $E_2 \cap E_6 = \{x_6, x_7\}$ which is $I[\{x_6, x_7\}] = \{x_6\}$. Also, suppose that there exists a good $g = \{x_6, \neg x_7\} \in \mathcal{G}_{E_6}$. According to Property 2, $F_{E_6,\{x_6\}}$ is satisfiable and the current truth assignment is extended by $\{\neg x_7\}$.

3.3 DPLL-TD Properties

Now, wee are interested in the complexity of DPLL-TD, to its completeness, soundness and termination.

Theorem 3 *DPLL-TD is sound, complete and finishes.*

Proof: DPLL-TD differs from DPLL in recording structural (no)goods and uses these nogoods as new clauses and these goods to avoid redundancies in the search. As DPLL is sound, complete and finishes, we have to prove that the additional treatments achieved by DPLL-TD do not alter these properties. We assume that DPLL-TD$(\mathcal{C}, I, E_i, \mathcal{G}, \mathcal{N})$ is the current call and we want to check the satisfiability of the subproblem $\mathcal{F}_{E_i, I[E_i \cap E_{par(i)}]}$.

For this purpose, let us consider the case where I satisfies all the clauses of $(\mathcal{C} \cup \mathcal{N})[E_i]$. DPLL-TD tries to extend I on each child of E_i. Let E_j be such a child. If the condition of line 12 holds then $\mathcal{F}_{E_j, I[E_j \cap E_i]}$ is satisfiable (according to Property 2 expressing the cut conditions by goods). Otherwise, the satisfiability of $\mathcal{F}_{E_j, I[E_j \cap E_i]}$ remains unknown, a recursive call DPLL-TD$(\mathcal{C}, I, E_j, \mathcal{G}, \mathcal{N})$ is necessary. If this call returns false (unsatisfiable) this means that $\mathcal{F}_{E_j, I[E_j \cap E_i]}$ has no model and recording a nogood in the form of a clause $\bigvee_{l_k \in I[E_i \cap E_j]} \neg l_k$ is a valid operation. As a structural nogood is a particular case of nogood then its use as any clause of C is also valid. At the end of the **while** loop, if I can be extended to a model on all its children then $I[E_i \cap E_{par(i)}]$ can be extended to a model of $\mathcal{F}_{E_i, I[E_i \cap E_{par(i)}]}$ (according to Corollary 1). Moreover, since I has been extended with the value of each used good on its children, recording the good $I[E_i \cap E_{par(i)}]$ of E_i is valid. Thus, recording and exploiting (no)goods are valid we conclude that DPLL-TD is sound, complete and finishes.□

Theorem 4 *DPLL-TD has a time complexity in $O((|E| + m).2^{w+s})$ and a space complexity in $O(|E|.s.2^s)$ for an instance having n variables and m clauses. The associated tree-decomposition to this instance is $(E, \mathcal{T})$ whose width is w and whose largest intersection between two clusters is s.*

Proof: Let us consider a cluster E_j and a partial truth assignment I on $E_j \cap E_{par(j)}$. For sake of simplicity, we try to extend I on E_j by assuming that the variable ordering is static. If $E_j = E_1$, $I = \emptyset$ and DPLL-TD will compute at most $2^{|E_1|+1}$ partial truth assignments which extend I. Otherwise, there are at most $2^{|E_j|-|I|+1}$ partial truth assignments extending I. As there is at most $2^{|I|}$ possible partial truth assignments of size $|I|$ and the subproblem $\mathcal{F}_{E_j,I}$ is solved only once thanks to recorded (no)goods, the whole number of partial truth assignments studied by DPLL-TD for the cluster E_j is at most $\sum_{|I|=0}^{s} 2^{|E_j|-|I|+1}.2^{|I|} = s.2^{|E_j|+1}$. As the cluster E_j is independent of its parent cluster as soon as the variables of $\mathcal{X}_{ind} \cap E_j \cap E_{par(j)}$ are assigned, DPLL-TD will study at most $O(2^{w+2})$ truth assignments. Moreover, for each partial truth assignments, DPLL-TD performs unit propagations in $O(m + |\mathcal{N}|)$. For a given separator of size k, the number of partial truth assignments is bounded by 2^{k+1}. So the whole number of recorded (no)goods is $|E|.2^{s+1}$. Finally, recording a (no)good can be performed in $O(s)$ and the check of line 12 can be achieved in $O(s.2^{s+1})$. So, the time complexity of DPLL-TD is $O((m + |E|.2^{s+1}).2^{w+1} + s.2^{s+1}.2^{w+1}$, i.e. $O((m + |E|).2^{w+s})$.

Concerning the space complexity, it only depends on the (no)good recording. So, as the space required for recording a (no)good is $O(s)$, the space complexity of DPLL-TD is $O(|E|.s.2^s)$. □

In simple terms, the complexity of DPLL-TD depends on the characteristics of the tree-decomposition , i.e. its width and the size of the largest separator. Compared to a classical DPLL procedure, the complexity of this last one is mainly related to the number of the variables of the treated instance.

		DPLL-TD		Minisat		Rsat		Zchaff		Satz		DPLL-TD
Benchs.	# inst.	#s	t.	#s	t.	#s	t.	#s	t.	#s	t.	position
linvrinv	8	3	1	3	1	3	2	3	9	3	1	1^*
mod2-3cage-unsat	23	6	2370	**18**	2579	1	540	0	-	0	-	2
mod2-rand3bip-sat	33	1	351	**17**	2534	6	1112	7	1646	8	1923	5
mod2-rand3bip-unsat	15	**11**	793	9	1062	6	872	6	1124	0	-	1
mod2c-3cage-unsat	6	**4**	115	3	563	0	-	0	-	0	-	1
mod2c-rand3bip-sat	33	0	-	17	2559	11	1410	**18**	3156	0	-	5
mod2c-rand3bip-unsat	15	**15**	676	10	1810	8	1337	6	1125	0	-	1
clqcolor	16	7	181	6	171	**9**	36	**9**	23	0	-	3
fclqcolor	16	9	136	9	169	9	22	9	11	0	-	1^*
fphp	42	16	617	**17**	165	16	49	16	28	**17**	555	3
php	42	16	123	6	258	16	134	12	231	**18**	721	2
sorge05	104	36	1752	**69**	5571	59	4347	54	2748	33	1316	4
driverLog	36	32	1	**36**	< 1	**36**	< 1	**36**	< 1	**36**	2	5
Ferry	36	20	496	**36**	6	**36**	4	**36**	22	17	299	4
rovers	22	22	3	22	< 1	22	< 1	22	< 1	22	1	1^*
satellite	20	20	22	20	< 1	20	< 1	20	< 1	20	259	1^*
difficult-contest05-jarvisalo	10	**3**	323	2	659	0	-	0	-	0	-	1
medium-contest05-jarvisalo	10	2	782	**9**	1955	1	470	3	850	3	497	4
spence-medium	10	3	220	**9**	1031	4	294	4	111	8	877	5
grieu	10	**5**	1222	1	71	**5**	347	3	525	**5**	378	1^*
industrial-jarvisalo	7	1	16	3	112	3	400	2	99	**4**	448	5

Table 1. DPLL-TD results and comparisons

4 Experimental Results

This section presents some implementation details of DPLL-TD and compares the results that it obtains to those of other powerful SAT solvers: Minisat [5], Rsat [6], Satz [7] and zChaff [8]. DPLL-TD is coded on the basis of Satz which is used as an enumeration algorithm in the clusters. We have selected Satz for our good knowledge of its source code which we have extensively modified (for example, to consider the nogood recording as clauses). DPLL-TD is tested on almost 1800 SAT instances (pretreated by SatElite [9]) issued from the SAT competitions ranging from 2002 to 2007 (`www.satcompetition.org`). The experiments are carried on a P4 3.2 Ghz PC with 1 GB of RAM, with 600 seconds. for the CPU time-out per instance. The tree-decomposition is obtained by the min-fill algorithm [10] and the root cluster contains the largest number of variables. To evaluate our approach, we have selected the most relevant instances in Table 1 (instances which present an interesting structure according to the ratio between the number of the variables and the width of the decomposition, and reflect the global observation over the 1800 instances). The first column corresponds to the benchmark name and the second one to the number of tested instances per benchmark. The rest of the table gives for each solver the number of solved instances (#s) and the sum of needed times of successful solving (t). The times reported for DPLL-TD include the used time to construct the tree decomposition. The last column give the position of DPLL-TD regarding to the number of solved instances (1^* means that DPLL-TD is not the only one to solve most instances). Regarding to this last column, it can be announced that our approach is competitive and can outperform two very powerful solvers, Minisat and Rsat, on some benchmarks.

Ranking according to the number of solved instances is used to provide a first reading and interpretation of the results. In fact, it is hard to present the results obtained on each instance. Besides, this comparison can be enriched regarding to the resolution time. However and for a given benchmark, the tested solvers do not solve necessary the same instances, which make difficult a strict comparison regarding the time columns. Concerning the comparison to the others approaches based on the tree decomposition, it is necessary to know that there are a few practicable algorithms (see Section 5). We have also tried to obtain some of the existing ones but unfortunately they are not maintained (the used libraries are obsolete ...) then it is impossible to run them. Furthermore and regarding the results presented in the papers presenting such approaches (for example [3, 11]), these ones are not competitive regarding to the recent high challenging solvers (Minisat, Rsat...). On the basis of these very encouraging results, we can argue that our approach is relevant and we believe that these results can be improved, by studying other points including the selecting heuristics of the root cluster, a dynamic cluster selection, the quality of the tree-decomposition, . . . Also, DPLL-TD improves significantly Satz, which is used as the basis of our implementation. Consequently, it is conceivable that we can obtain better results if we use more powerful solvers such as Minisat and Rsat.

5 Related Work

Some works try to enhance the performances of SAT solvers by guiding the variable ordering heuristic by the tree-decompositions of the treated instances. For examples, a variable ordering heuristic presented in [11] uses Dtree [12] (a static binary tree-decomposition) to compute the variable group ordering. A dynamic variable clustering is presented in [13] without showing any experimental results to demonstrate if such approach is more promising than a static variable clustering one. In [14], the authors provide a heuristic to solve the most constrained subproblem first. But again, no experimental evaluation is performed. A static global variable ordering based on recursive min-cut bisection of hypergraphs was proposed in [15]. In [3], the authors present dynamic decomposition methods based on the hypergraph (corresponding to the SAT instance) separator. A constrained partitioning scheme that can be exploited to derive a variable order that can be used to guide and speedup SAT solvers is described [16]. Opposed to the minimum tree-width decomposition schemes, this approach performs constraint partitioning on a hypergraph by analyzing both the number of the occurrences of the variables among all the clauses of the SAT instance and the connectivity of these clauses. Finally, the method DPLL-TD has a clear relationship with the BTD method [2] proposed for CSP but they are sufficiently different. On the one hand, the resolution methods for SAT and CSP do not employ exactly the same techniques and differ on their specificities. For example, if all the variables must be instantiated to produce a solution for a CSP, it is not necessary the case in SAT. This simple difference, apparently insignificant, imposes the need to define a formal framework specific to SAT when designing the DPLL-TD algorithm (the introduction of the independence variables is a direct consequence). On the other hand, the (no)goods in DPLL-TD can have a variable size and do not affect all the variables of a separator as in BTD. In this case, such (no)goods are *generic* and may permit a more powerful cuts in DPLL-TD than BTD.

6 Conclusion and Perspectives

We have proposed a new approach (DPLL-TD) based on the tree decomposition to solve structured satisfiability instances and we have supported our work by both theoretical and practical evidences. The obtained variable order is static on the clusters and a dynamic over the variables of these clusters. Also, (no)goods are learnt to prune the search space and the spatial and temporal complexities of DPLL-TD are bounded according to the tree-decomposition characteristics. Even if some works dealing with such approach exist for SAT, it is necessary to emphasis the fact that they are rarely supported by an empirical evaluation that may give a practical impact. Other aspects of solving SAT by tree-decomposition will be studied, mainly the triangulation, the choice of the root cluster, the definition of heuristics to choose the next cluster to treat, . . . We will also work on the use of the conflict analysis to derive new nogoods as it is done on the CDCL-based solvers and the use of restarts (on the basis of the information learned in previous executions) while changing either the root cluster or the decomposition itself (for example, building a new tree-decomposition based of the most conflicting variables). This information can also be used to determine dynamically the cluster orders based on the most constrained first.

References

[1] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem-proving. *Commun. ACM*, 5(7):394–397, 1962.

[2] P. Jégou and C. Terrioux. Hybrid backtracking bounded by tree-decomposition of constraint networks. *Artificial Intelligence*, 146:43–75, 2003.

[3] W. Li and P. van Beek. Guiding real-world sat solving with dynamic hypergraph separator decomposition. In *Proceedings of ICTAI 2004*, pages 542–548, 2004.

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

[5] N. Eén and N. Sörensson. An extensible sat-solver. In *Proceedings of SAT 2003*, pages 402–518, 2003.

[6] K. Pipatsrisawat and A. Darwiche. Rsat 2.0: Sat solver description. Technical Report D–153, Automated Reasoning Group, Computer Science Department, UCLA, 2007.

[7] C.M. Li. Exploiting Yet More the Power of Unit Clause Propagation to solve 3-SAT Problem. In *ECAI'96 Workshop on Advances in Propositional Deduction*, pages 11–16, 1996.

[8] L. Zhang and C.F. Madigan. Efficient conflict driven learning in a boolean satisfiability solver. In *Proceedings of ICCAD 2001*, pages 279–285, 2001.

[9] N. Eén and N. Sörensson. Effective preprocessing in sat through variable and clause elimination. In *Proceedings of SAT 2005*, pages 61–75, 2005.

[10] U. Kjærulff. Triangulation of graphs: Algorithms giving small total state space. Technical report, University of Aalborg, 1990.

[11] J. Huang and A. Darwiche. A structure-based variable ordering heuristic for sat. In *Proceedings of IJCAI 2003*, pages 1167–1172, 2003.

[12] A. Darwiche. A compiler for deterministic, decomposable negation normal form. In *Proceedings of AAAI 2002*, pages 627–634, 2002.

[13] P. Bjesse, J. Kukula, R. Damiano, T. Stanion, and Y. Zhu. Guiding sat diagnosis with tree decompositions. In *Proceedings of SAT 2004*, pages 315–329, 2004.

[14] E. Amir and S. Mcilraith. Solving satisfiability using decomposition and the most constrained subproblem. In *LICS workshop on Theory and Applications of Satisfiability Testing*, 2001.

[15] F.A. Aloul, I.L. Markov, and K.A. Sakallah. MINCE: A Static Global Variable-Ordering Heuristic for SAT Search and BDD Manipulation. *Journal of Universal Computer Science (JUCS)*, 10(12):1562–1596, 2004.

[16] V. Durairaj and P. Kalla. Guiding CNF-SAT search via efficient constraint partitioning. In *Proceedings ICCAD 2004*, pages 498–501, 2004.