Toward a Generalization and a Reformulation of Goods in SAT Preliminary Report[1]

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

Abstract

Learning useful information when solving SAT or CSP problems to speed up a tree-search approaches, is one of the main explored tracks in various works. Such information are known as goods and nogoods and they aim to forbid to repetitively visit the same parts of the search space. Unfortunately and unlike nogoods, the exploitation of goods is limited to tree-search approaches based on the structural properties of the problem. In this paper, we propose to generalize and reformulate structural goods under SAT. We also propose a learning scheme of general goods and show their integration in a DPLL-like procedure.

Introduction

The notion of nogoods is well known and has shown its interest for solving efficiently both the satisfiability problem (SAT) and the Constraint Satisfaction Problems (CSPs). A nogood is an assignment of some variables of the problem which cannot be extended to a solution (or a model). For the modern SAT solvers, such as MiniSat (Eén and Sörensson 2003), the exploitation of nogoods is one of their principle aspects and it is usually known as CDCL (Conflict Driven Clause Learning). When a conflict is reached (an empty clause is produced), it is analyzed and its reason (a clause) is learned and recorded, to avoid the occurrence of the same conflict latter during the search. This learnt nogood allows to prune the search tree. Moreover, this analysis allows to realize a non chronological backtracking.

The dual notion of a nogood is a good. A good is a partial truth assignment which can be extended to solve a part of the problem and to realize consequently a forward-jump during the search . This notion has firstly been introduced in the field of CSPs in (Bayardo and Miranker 1994) to solve binary tree-structured constraint networks. It has been extended in (Jégou and Terrioux 2003b) to solve constraint networks which are neither necessary binary nor tree-structured, and then has also been extended to Valued CSPs in (Jégou and Terrioux 2003a). This approach has allowed to solve really hard structured instances (de Givry, Schiex, and Verfaillie 2006). Recently, it has been reformulated in the field of SAT in (Habet, Paris, and Terrioux 2009).

Nevertheless, these goods have only been defined using structural criteria - they are called structural goods - which is really a particular and a limited use case. In this paper, we aim to reformulate and generalize the notion of good, independently of structural criteria and show how they can be exploited for solving SAT instances. Particularly, we show how can such goods be extracted (learnt) and easily integrated in a classical DPLL algorithm, with the hope to enhance its efficiency. This preliminary work is a first track to generalize the notion of goods.

The paper is organized as follow. It starts by a recall of elements about SAT and the DPLL procedure. We will then give and recall the definition of structural (no)goods. Then we present our main contribution by generalizing the notion of goods in SAT by defining this notion and then explaining their detection and integration in a DPLL search. After that, we discuss some related work then conclude and give some perspectives[1].

Basic Notions

This section is dedicated to the definition of the SAT problem and to a reminder of the DPLL algorithm which are necessary for the rest of the paper.

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. If $l = \neg v$, then $\neg l$ will be used to denote v. 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

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

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 clause c is satisfied by a partial truth assignment I if $\exists l \in I$ such that $l \in c$. A clause c is reduced by a partial truth assignment I if $\exists l \in I$ such that $\neg l \in c$. A clause c is falsified by a partial truth assignment I if $\forall l \in c, \neg l \in I$. 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.

About DPLL Algorithm

Here we recall the original DPLL prcoedure, which will be modified in the incoming sections according to the goods concept.

The Davis-Putnam-Logemann-Loveland procedure (DPLL) (Davis, Logemann, and Loveland 1962) is one of the best complete procedures for SAT. It 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).

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

```
1 Unit-propagation ($\mathcal{C}, I$)
2 if $\square \in \mathcal{C}$ 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 DPLL ($\mathcal{C} \cup \{v\}, I \cup \{v\}$) then return $true$
8         else return DPLL ($\mathcal{C} \cup \{\neg v\}, I \cup \{\neg v\}$)
```

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

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

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 $\square$). From a practical point of view, Minisat like solvers are actually considered as the most powerful complete ones when solving large real-life SAT instances. Such solvers are based on recording during the search some nogoods issued from CDCL mechanisms (conflict analysis).

Structural Nogoods and Goods

As introduced previously, the known exploitation of goods is limited to a structural based resolution of the SAT or CSP. Hence and before generalizing the definition of goods, we recall here their definition (including nogoods), properties and use in the context of a tree-decomposition based approach for SAT.

At first, the following definition introduces the tree-decomposition, as defined in (Robertson and Seymour 1986), which uses the primal graph representation of a SAT instance. For more details about the tree-decomposition based-approach for SATplease refer to (Habet, Paris, and Terrioux 2009) (to find for example the definition of $\mathcal{E}_H$) used below).

Definition 1 *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 set $Desc(E_j)$ defines the 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 ($E_{par(1)} = \emptyset$: E_1 is the root cluster). Moreover, the set $E_{par(j)} \cap E_j$ defines the *separator* between E_j and its parent. Finally, the set $\mathcal{C}[E_i]$ contains the clauses belonging exclusively to the cluster E_i ($\mathcal{C}[E_i] = \{c \in \mathcal{C} | var(c) \subseteq E_i \ and \ var(c) \nsubseteq E_{par(i)}\}$).

Based on the structural properties of the graph representation of a SAT instance, the tree-decomposition of the initial instance divides it into independent parts (clusters) but which are still linked by the separators: fixing the variables of a separator disconnects the initial problem into two independent parts.

Accordingly, 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 treated separately. If all the clauses restricted to this cluster are satisfied then one of its children E_j (if exists) is treated by extending the truth assignment of the variables of E_i to those of E_j. If this process fails then it is not possible to extend the truth assignment of the variables of the separator, between E_i and E_j, to a model and a nogood is detected. Storing this nogood can 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 variables of the separator.

In the same way and for the same reasons, if the clauses of the cluster E_i and those of clusters of its descendant 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 and a good is detected. Hence, such (no)goods allow pruning the search space as formalized in follow:

Definition 2 *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 this definition, $\mathcal{F}_{E_i,I}$ is the subproblem 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. According to this definition, 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 1 *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 $I[E_i \cap E_{par(i)}] \subseteq e_g$.*

This 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 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 nogood ng of E_i, no truth assignment I on Y such that $ng \subseteq I$ can be extended to a model of $\mathcal{F}_{E_i,I[E_i \cap E_{par(i)}]}$.*

This achieves the recall of structural (no)goods definition and exploitation. The next sections are dedicated to the reformulation of goods in a more general scope.

Reformulation of Goods

A More General Definition of Goods

A structural good is a partial truth assignment which satisfies a part of (SAT or CSP) problem and which can be extended consistently to satisfy the rest of the problem. This partial truth assignment is located in a separator of the (hyper)graph representing the topology of the problem and such assignment disconnects the rest of the problem from the whole one. This remaining (rest) part (or subproblem) can be solved independently.

In a more general definition, a good is not necessarily located in a separator. Nevertheless, the principle remains similar since we need to preserve the independancy of the associated subproblem for which the solution is independent from the solved part of problem.

We present goods by means of two compatible definitions where the first one is intuitive. Here, a good is a 3-tuple (I_g, X_g, C_g) where I_g is a partial truth assignment which can be extended on the variables of $X_g \subset X$ while satisfying a set of clauses C_g independently from the other clauses of the considered SAT formula. Formally:

Definition 3 ***First definition.*** *Let $\mathcal{F} = (\mathcal{X}, \mathcal{C})$ be a SAT instance, I_g a partial truth assignment, X_g a subset of variables and C_g a subset of clauses. The tuple $g = (I_g, X_g, C_g)$ is a good iff:*

1. *no clause of $\mathcal{C}$ is falsified by I_g*
2. *some clauses of $\mathcal{C}$ can be reduced by I_g*
3. *$var(I_g) \cap X_g = \emptyset$*
4. *there is a partial truth assignment I_{X_g} such that:*
   - *$C_g = \{c \in \mathcal{C} :$ c is satisfied by I_{X_g} and $\forall l \in I_g, var(l) \notin var(c)\}$*
   - *if $c \notin C_g$ is reduced by I_{X_g}, then c is satisfied by I_g*

As for structural goods, I_g corresponds to the partial truth assignment associated to the good. The subset of variables X_g and the subset of clauses C_g correspond to the associated subproblem. The first and the second conditions indicate that a good cannot be inconsistent. The third condition stipulates that the good and the subproblem are defined on different sets of variables. The fourth condition specifies that there is a model for the subproblem and that this solution has no consequence on the whole problem, since the truth assignment corresponding to this model doesn't add constraint for solving the remaining problem (none of its clauses is reduced by I_{X_g}), assuming that the subproblem defined by X_g and C_g can be solved consistently and independently.

For example, consider the SAT instance $\mathcal{F} = (\mathcal{X}, \mathcal{C})$, where $\mathcal{X} = \{x_1, x_2, \ldots x_{14}\}$, $\mathcal{C} = \{c_1, c_2, \ldots c_{12}\}$ and:

$c_1 = (x_1 \vee x_2 \vee x_3 \vee x_4)$

$c_2 = (\neg x_3 \vee x_5 \vee x_6)$

$c_3 = (\neg x_4 \vee x_7 \vee x_8)$

$c_4 = (\neg x_5 \vee x_8)$

$c_5 = (x_6 \vee \neg x_7)$

$c_6 = (\neg x_6 \vee x_9 \vee x_{10})$

$c_7 = (\neg x_8 \vee x_{11} \vee x_{12})$

$c_8 = (\neg x_1 \vee x_{13} \vee x_{14})$

$c_9 = (x_2 \vee x_{13} \vee x_{14})$

$c_{10} = (x_3 \vee x_4 \vee x_{13} \vee x_{14})$

$c_{11} = (x_9 \vee x_{11} \vee x_{13})$

$c_{12} = (x_{10} \vee x_{12} \vee x_{14})$

Consider the partial truth assignment $\{\neg x_5, \neg x_6, \neg x_7, \neg x_8\}$. This assignment satisfies the clauses c_4, c_5, c_6 and c_7 while it reduces the clauses c_2 and c_3. Now, if we consider the partial truth assignment $\{\neg x_3, \neg x_4\}$, the clauses c_2 and c_3 are satisfied, while the clauses c_1 and c_{10} are reduced. Thus, we have a good $g = (I_g, I_{X_g})$ such that $I_g = \{\neg x_3, \neg x_4\}$, $X_g = \{x_5, x_6, x_7, x_8\}$ (then $I_{X_g} = \{\neg x_5, \neg x_6, \neg x_7, \neg x_8\}$) and $C_g = \{c_4, c_5, c_6, c_7\}$. It is possible to see that the set of variables $\{x_3, x_4\}$ is not a separator of the set of clauses because their deletion doesn't disconnect the set of clauses. So, we have a good on variables which cannot constitute a structural good.

Goods can be used to avoid revisiting some parts of the search space during search. If during search, a partial truth assignment I includes I_g, and if not any variable of X_g is assigned, then we know that I_g (and I too), can be extended to the variables of X_g to satisfy the clauses appearing in C_g. So, these clauses can be deleted, and the next variable to assign will be a variable that hasn't already be assigned and then which doesn't belongs to X_g. This jump in the search space is called a *forwardjump* in BTD (Jégou and Terrioux 2003b).

The second definition of goods replaces X_g and C_g by I_{X_g} which is a partial truth assignment on variables of X_g which satisfies clauses of C_g. Note that the set of satisfied clauses C_g can be computed by selecting the clauses in $\mathcal{C}$ which are satisfied by I_{X_g}. So, the size of the good g introduced in this definition will be smaller than that introduced in the previous one.

Definition 4 ***Second definition.*** *Let $\mathcal{F} = (\mathcal{X}, \mathcal{C})$ be a SAT instance. Let I_g be a partial truth assignment. Let I_{X_g} be a partial truth assignment. The pair $g = (I_g, I_{X_g})$ is a good iff:*

- *no clause of $\mathcal{C}$ is falsified by $I_g \cup I_{X_g}$*
- *some clauses of $\mathcal{C}$ can be reduced by I_g*
- $var(I_g) \cap var(I_{X_g}) = \emptyset$
- *if a clause c is reduced by I_{X_g}, then c is satisfied by I_g*

Once the assignment I_g is constructed and if there is a good $g = (I_g, I_{X_g})$ that has already been recorded, we can exploit g by removing clauses (the unrecorded set of clauses C_g) which are satisfied by I_{X_g}, and then we can realize a forwardjump by deleting X_g from the set on unassigned variables. The remaining problem can be solved avoiding to look for a partial truth assignment which satisfies clauses that appear in C_g. Thus, the remaining problem to solve will be the whole current one without the variables belonging to X_g and without the clauses appearing in C_g. In the rest of the paper, we will use this second definition.

Integrating Goods in DPLL

The DPLL algorithm must be modified to integrate our general definition of goods. This is done in DPLL-Good algorithm. Firstly, we assume that the formula to solve is defined by an initial set of clauses $\mathcal{C}_0$, that is defined on a set of variables $\mathcal{X}_0$. These sets will not be modified during search. The first call to DPPL-Good will be realized with $\mathcal{X} = \mathcal{X}_0$, $\mathcal{C} = \mathcal{C}_0$ and $I = \emptyset$. We assume that a database will be used to memorize goods. It is denoted $\mathcal{G}$ and initially we have $\mathcal{G} = \emptyset$. $\mathcal{G}$ will be updated during search by adding new goods. So, before assigning a new variable we decide if the current partial and consistent truth assignment is a possible good to add to $\mathcal{G}$. This is realized by the procedure AddGood() which can modify the database $\mathcal{G}$. This procedure is described in details after the algorithm DPPL-Good.

After this first step, we try to exploit goods. The function Good() is called to find goods. If a good $g = (I_g, I_{X_g})$ is found, the function returns **true**. In this case, we have $I_g \subseteq I$ and $I \cap I_{X_g} = \emptyset$. After, a forwardjump is realized. Otherwise, the function Good() returns **false**. A forwardjump consists in eliminating X_g from the set of non-assigned variables and in deleting the set of clauses C_g. The function FindClauses() is called to compute the set C_g. This task can be easily realized in selecting the clauses which are satisfied by the truth assignment I_{X_g}.

If we consider the example, with the partial truth assignment $\{x_1, \neg x_2, \neg x_3, \neg x_4, \neg x_5, \neg x_6, \neg x_7, \neg x_8\}$. The good $g = (I_g, I_{X_g})$ where $I_g = \{\neg x_3, \neg x_4\}$ and $I_{X_g} = \{\neg x_5, \neg x_6, \neg x_7, \neg x_8\}$ will be recorded. Later during the search, assume that we have a new partial truth assignment $\{\neg x_1, x_2, \neg x_3, \neg x_4\}$. The good $g = (I_g, I_{X_g})$ can be exploited. Indeed, considering $I_g = \{\neg x_3, \neg x_4\}$, the variables belonging to $X_g = \{x_5, x_6, x_7, x_8\}$ haven't to be assigned because we know that a partial truth assignment on X_g can be obtained. So, the clauses belonging to $C_g = \{c_4, c_5, c_6, c_7\}$ can be deleted. Then, a forwardjump can be done and the search continues by the assignment of a new variable (eg. x_9).

Algorithm 3: DPLL-Good(in: $\mathcal{C}, \mathcal{X}, I$)

```
Unit-propagation ($\mathcal{C}, \mathcal{X}, I$)
if $\square \in \mathcal{C}$ then return $false$
else
    if $\mathcal{C} = \emptyset$ then return $true$
    else
        /* possible updating of the good database $\mathcal{G}$ */
        AddGood ($I, \mathcal{C}, \mathcal{X}$)
        /* possible use of goods */
        if Good ($I, \mathcal{G}, g$) then
            /* We consider a good $g = (I_g, I_{X_g})$ */
            if $X_g \subseteq \mathcal{X}$ then
                $C_g \leftarrow FindClauses(\mathcal{C}, I_{X_g})$
                $\mathcal{C} \leftarrow \mathcal{C} - C_g$
                $\mathcal{X} \leftarrow \mathcal{X} - X_g$
        Choose a variable $v \in \mathcal{X}$ (branching heuristic)
        if DPLL-Good($\mathcal{C} \cup \{v\}, \mathcal{X} \cup \{v\}, I \cup \{v\}$) then return $true$
        else return DPLL-Good ($\mathcal{C} \cup \{\neg v\}, \mathcal{X} \cup \{v\}, I \cup \{\neg v\}$)
```

Algorithm 4: UnitPropagationGood(in/out: $\mathcal{C}, \mathcal{X}, I$)

```
while there is no empty clause and a unit clause $l$ exists in $\mathcal{C}$ do
    $I \leftarrow I \cup \{l\}$ (satisfy $l$)
    $\mathcal{X} \leftarrow \mathcal{X} - var(l)$
    simplify $\mathcal{C}$
```

Theorem 1 *DPLL-Good is sound, complete and finishes.*

Proof: We only give here the sketch of the proof. DPLL-Good differs from DPLL in recording goods and in using 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-Good do not alter these properties. We assume that DPLL-Good($\mathcal{C}, \mathcal{X}, I$) is the current call and we want to check the satisfiability of the subformula ($\mathcal{X}$,$\mathcal{C}$). If I doesn't contains a good, then DPLL-Good runs exactly as DPLL does. If I contains a good and ($\mathcal{X}$,$\mathcal{C}$) admits a model, then this formula will be simplified, exploiting a good $g = (I_g, I_{X_g})$, or more precisely $g = (I_g, X_g, C_g)$. DPLL-Good() will be called with I and the subset of clauses $\mathcal{C} - C_g$. We know that the clauses appearing in C_g admit a model, the partial truth assignment I_{X_g}, while this model doesn't reduce any clause of $\mathcal{C} - C_g$. So, ($\mathcal{X}$,$\mathcal{C}$) will be satisfiable iff ($\mathcal{X} - X_g$,$\mathcal{C} - C_g$) is also satisfiable. By induction, assuming that DPLL-Good() is sound and complete for strictly smaller instances, DPLL-Good() is then sound and complete for ($\mathcal{X}$,$\mathcal{C}$). Moreover, by the same reasons as DPLL, DPLL-Good() terminates.$\square$

We now describe the procedure AddGood() which computes a good $g = (I_g, I_{X_g})$ and which updates the database $\mathcal{G}$ by adding it. Note that different strategies can be used and the number of potential goods can be large. Here we present a basic algorithm which could be easily modified.

This first proposition of good computation is driven by time complexity considerations and practical efficiency. Firstly, we must find the set of variables X_g. This set corresponds to the set of assigned variables such that the clauses where they appear are satisfied. Note that the function $Clauses(x)$ returns the set of clauses where the variable x appears. The partial truth assignment I_{X_g} is computed using X_g and I.

In the example, assume that the current assignment is $I = \{x_1, \neg x_2, \neg x_3, \neg x_4, \neg x_5, \neg x_6, \neg x_7, \neg x_8\}$. The set of satisfied clauses is $\{c_1, c_2, c_3, c_4, c_5, c_6, c_7\}$ while clauses c_8, c_9 and c_{10} are reduced. So, the algorithm finds $X_g = \{x_5, x_6, x_7, x_8\}$ because all the clauses where these variables appear are satisfied. Moreover, we have $I_{X_g} = \{\neg x_5, \neg x_6, \neg x_7, \neg x_8\}$

In the second step, we compute the partial truth assignment I_g which is defined by literals which satisfy clauses which are reduced by literals that appear in I_{X_g}, assuming that these clauses are not also satisfied by I_{X_g}. Finally, a pair (I_g, I_{X_g}) can be added in $\mathcal{G}$ if the good (I_g, I_{X_g}) doesn't already appears in $\mathcal{G}$.

In the example, the algorithm finds $I_g = \{\neg x_3, \neg x_4\}$ because the clauses which are reduced by $I_{X_g} = \{\neg x_5, \neg x_6, \neg x_7, \neg x_8\}$ are satisfied by $I_g = \{\neg x_3, \neg x_4\}$.

This version of the algorithm isn't necessarily the most efficient considering some criteria. Firstly, the size of the goods, more precisely, the size of I_g. In the work around BTD in CSPs (Jégou and Terrioux 2003b), the authors has observed that the more the size of goods is small, the more they can frequently be exploited. Moreover, another consequence is related to the size and then the management of the database $\mathcal{G}$. This management can be easily prohibitive

Algorithm 5: AddGood(in: $I, \mathcal{C}, \mathcal{X}$; in/out: $\mathcal{G}$)

```
1  /* finding $X_g$ and $I_{X_g}$ */
2  $X_g \leftarrow \emptyset$; $I_{X_g} \leftarrow \emptyset$
3  for $l \in I$ do
4      if $\forall c \in Clauses(var(l))$, c is satisfied then
5          $X_g \leftarrow X_g \cup \{var(l)\}$
6          $I_{X_g} \leftarrow I_{X_g} \cup \{l\}$
7  /* finding $I_g$ */
8  $I_g \leftarrow \emptyset$
9  for $l \in I_{X_g}$ do
10     for $c \in \mathcal{C}_0$ such that $l$ reduces $c$ do
11         if $\nexists l' \in I_{X_g} : l'$ satisfies $c$ then
12             Choose $l'' \in I$ such that $l''$ satisfies $c$
13             $I_g \leftarrow I_g \cup \{l''\}$
14 /* conditionnal updating of $\mathcal{G}$ */
15 if $(I_g, I_{X_g}) \notin \mathcal{G}$ then $\mathcal{G} \leftarrow \mathcal{G} \cup \{(I_g, I_{X_g})\}$
```

from a practical viewpoint. With our algorithm, no limitation is given for this size. So, we could easily introduce a parameter to limit the maximal size of I_g.

Secondly, the procedure AddGood() must run quickly because the addition of goods will be checked after each assignment of variable in DPLL-Good. So, more other restrictions can be considered. For example, we can consider the current assignment I, and then look for variables of X_g traversing in reverse order the current assignment. More precisely, for a current assignment $I = \{l_1, l_2, \dots l_i, l_{i+1}, \dots, l_k\}$ such that all the clauses where variables associated to literals $\{l_{i+1}, \dots, l_k\}$ appear are satisfied, while there is an unsatisfied clause which is reduced by l_i while isn't satisfied by $\{l_{i+1}, \dots, l_k\}$, we can stop the search for X_g to the variables belonging to $\{l_{i+1}, \dots, l_k\}$. In this case, $I_{X_g} = \{l_{i+1}, \dots, l_k\}$ and $X_g = \{var(l_{i+1}), \dots, var(l_k)\}$. Considering the ordering of variables can speed up the computation of goods while limiting their number. Moreover, this kind of limitation is similar to the one realized in BTD which is based on exploitation of compatible orderings in tree-decomposition.

In this section, we have introduced a preliminary presentation of goods. We have pointed potential limitations which seem necessary for an efficient exploitation of goods. This kind of limitations will be considered after the implementation of goods and their experimentation which is out of the scope of this preliminary paper.

Related Work

The notion of generalized good is naturally related to the notion of structural good. This relation can be formulated by a property which links these two notions. It can be done by giving another (and equivalent) definition of structural good. This new one will be expressed using the notion of separator in (hyper)graphs, that is, separator in networks.

Definition 5 ***New formulation of structural goods.*** *Given a SAT instance $\mathcal{F} = (\mathcal{X}, \mathcal{C})$ and a separator S of the network representing $\mathcal{F}$, a truth assignment I_s on a subset of S is a structural good if any extension of I_s on S can be extended* *to a model of $(\mathcal{X}_S, \mathcal{C}_S)$ which is one of the separate parts on the network.*

As $(\mathcal{X}_S, \mathcal{C}_S)$ is disconnected from the rest of the formula $\mathcal{F}$ under the assumption that S is deleted (the involved variables are assigned), there is no clause of $(\mathcal{X}_S, \mathcal{C}_S)$ that links int to the rest of the formula. Therefore, no assignment of variables $\mathcal{X}_S$ will reduce or satisfy clauses belonging to disconnected regions. We can therefore state the following property.

Theorem 2 *If I_s is a structural good, according to a separator S, and an associated subproblem$(\mathcal{X}_S, \mathcal{C}_S)$, then the pair $g = (I_{s+}, I_{\mathcal{X}_S})$ where I_{s+} is an extension of I_s on S, and $I_{\mathcal{X}_S}$ a model of $(\mathcal{X}_S, \mathcal{C}_S)$, is a good.*

Note that as indicated in the example of previous section, the converse is false because goods can be defined on subsets of variables which aren't necessary separators of the network.

There are also connexions between generalized goods and Autarks (Monien and Speckenmeyer 1985). So, we recall briefly the definition of aurtaky. Given a SAT formula $\mathcal{F} = (\mathcal{X}, \mathcal{C})$, an *autarky* is a partial truth assignment I such that $\forall c \in \mathcal{C}, I$ satisfies c or I doesn't reduce c. A partial truth assignment I such that $I' \subsetneq I$, is called a *local autarky* if the set of clauses C simplified by I (denoted $\mathcal{C}_I$) is included in the set of clauses $\mathcal{C}$ simplified by I' (denoted $\mathcal{C}_{I'}$). So, every model of $\mathcal{C}_{I'}$ is a model for $\mathcal{C}_I$. This property can be used to prune the search of a DPLL algorithm because $\mathcal{C}_{I'}$ is satisfiable iff $\mathcal{C}_I$ is satisfiable. Indeed, if $\mathcal{C}_I$ is proved unsatisfiable, then $\mathcal{C}_{I'}$ is also proved unsatisfiable, and then an unexplored branch between I' and I has not to be explored during search, after proving unsatifiability of $\mathcal{C}_I$.

One can see that autarkies are used to avoid to explore unsatisfiable regions of the search tree. Thus, the use of aurtakies is different from the use of goods, since they allow to do non-chronological backtracking while goods allow forwardjumping. But, aurtakies are defined using a similar principle of independence between parts of the search space, or more precisely, subsets of clauses which are independent w.r.t. partial truth assignments. Nevertheless, the techniques used to the detection of autarkies could probably be used to the detection of goods.

Conclusion

We have proposed a new formulation of goods under SAT by generalizing the definition of the structural ones and giving their algorithmic aspects (regarding to their detection, recording then integration) in order to include them in a classical DPLL procedure. Thus, we wanted to step out of the restrictive framework of structural goods, while displaying links between them, including the reformulation of these structural goods in terms of these *general* ones. Moreover, our generalization of goods can be exploited in a local search algorithm since a good definition just need a first truth assignment (which can be produced by a local search algorithm). Also, work on aurtakies should be studied closely because of their proximity to goods, although the two concepts are quite different. As was said before, this work is preliminary and will be enriched by including the implementation of algorithms designed here. Then we try to move to other formalisms such as (V) CSP and find an adequate expression of what is introduced here.

References

Bayardo, R. J., and Miranker, D. P. 1994. An optimal backtrack algorithm for tree-structured constraint satisfaction problems. *Artificial Intelligence* 71(1):159–181.

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

de Givry, S.; Schiex, T.; and Verfaillie, G. 2006. Exploiting tree decomposition and soft local consistency in weighted csp. In *Proceedings of the 21st national conference on Artificial intelligence*, 22–27.

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

Habet, D.; Paris, L.; and Terrioux, C. 2009. A tree decomposition based approach to solve structured sat instances. In *Proceedings of the 21th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2009)*, 115–122.

Jégou, P., and Terrioux, C. 2003a. Bounded backtracking for the valued constraint satisfaction problems. In *Proceedings of the Ninth International Conference on Principles and Practice of Constraint Programming (CP-2003)*, 709–723.

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

Monien, B., and Speckenmeyer, E. 1985. Solving satisfiability in less than 2^n steps. *Discrete Applied Mathematics* 10:287 – 295.

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