Circuit Based Encoding of CNF Formula

Gilles Audemard, Lakhdar Saïs

Circuit Based Encoding of CNF formula

Gilles Audemard and Lakhdar Saïs

CRIL CNRS – Université d'Artois
rue Jean Souvraz SP-18
F-62307 Lens Cedex France
{audemard,sais}@cril.univ-artois.fr

Abstract. In this paper a new circuit sat based encoding of boolean formula is proposed. It makes an original use of the concept of restrictive models introduced by Boufkhad to polynomially translate any formula in conjunctive normal form (CNF) to a circuit sat representation (a conjunction of gates and clauses). Our proposed encoding preserves the satisfiability of the original formula. The set of models of the obtained circuit w.r.t. the original set of variables is a subset of the models (with special characteristics) of the original formula. Each gate represents both a subset of clauses from the original CNF formula and a set of new additional clauses which constrains the set of models to those with a special structure. Using two variant of restrictive models, our circuit sat based encoding leads to a conjunction of two sub-formulas: a set of gates and a horn formula. We also provided a connection between our encoding and the satisfiability of the original formula i.e. when the input formula is satisfiable, our proposed translation delivers a full circuit formula. A new incremental preprocessing process is designed leading to interesting experimental improvements of Minisat a state-of-the-art satisfiability solver. Finally, using our circuit encoding, on many SAT instances interesting results are also obtained wrt. the backdoor set computation problem.

1 Introduction

Propositional satisfiability (SAT) is the problem of deciding whether a boolean formula in conjunctive normal form (CNF) is satisfiable. SAT is one of the most studied NP-Complete problems because of its theoretical and practical importance. Encouraged by the impressive progress in practical solving of SAT, various applications ranging from formal verification to planning are encoded and solved using SAT. Most of the more successful complete solvers are based on the backtrack search algorithm called Davis Putnam Logemann Loveland (DPLL) procedure. Such basic algorithm is enhanced with many important pruning techniques such as learning, extended use of boolean constraint propagation, preprocessing, symmetries breaking etc. The impact of these different improvements depends on the kind of instances to be solved. For example, learning is more useful when solving instances encoding real world problems than on random generated ones. Another important aspect for efficient SAT solving concerns the problem encoding. Traditionally, most solvers work on a formula encoded in conjunctive normal form (CNF). However, encoding knowledge under CNF can flatten some structural

knowledge that would be more apparent in more expressive propositional logic representation formalisms, and that could prove useful in the resolution step [14, 9, 8, 17]. To take benefit from such structural knowledge, recent works have addressed this issue following two different paths of research. The first one use extended boolean formula (nonCNF [17], boolean functions [14], pseudo boolean constraints [3]) for problem encoding. Whereas the second one, try to recover and/or to deduce structural knowledge from CNF encoding (symmetries [1], functional dependencies[8], equivalence [10]). Clearly, it is generally agreed that using CNF is convenient for efficient design of SAT solver. The same arguments are used in the constraint satisfaction problem (CSP) to justify the translation of non-binary instances to binary ones. More recently, different works in both SAT and CSP propose to model and to solve problems using more general representation language. At the same time encoding either the CSP (resp. SAT) in SAT (resp. CSP) is also a subject of many interesting research works (see [4]).

These two different approaches for exploiting structural properties have their own justification and advantages. Obviously, when some useful structures are given during the specification phase of the problem, it is more convenient to encode these informations and to exploit them in both approaches. On the contrary, when some structures are not given and need to be detected automatically, the approach to be used for problem encoding depends on the efficiency of the techniques that one can design for their detection and on the benefit that we can obtain in the resolution phase.

In this paper, we follow the second approach which consists in detecting hidden structures of CNF formula. More precisely, based on two previous related works proposed by Purdom [13] (complementary search) to avoid search redundancies and by Boufkhad [2] on exploiting the restrictive solution (solution that has special characteristics), we propose a new and original encoding of any formula in conjunctive normal as a conjunction of boolean functions (gates) and clauses. Each gate represents both a subset of clauses from the original CNF formula and a set of new additional clauses which constrains the set of models to those with special characteristics. Using auxiliary variables, we obtain a polynomial circuit sat based encoding which preserves the satisfiability of the original formula. Using two restrictive variants of models, we obtain a circuit sat formula where the remaining clauses belong to a tractable class (horn or reverse-horn). We also prove that a formula is satisfiable iff our proposed circuit sat encoding delivers a full circuit formula. Consequently, covering the hole CNF formula with a set of gates is intractable in the general case.

As mentioned in [15], many classes of instances derived from logic circuit (e.g. property checking) may contain a portion of circuit-derived clauses that describe the hardware part and a non circuit part made of clauses that describe more general properties. Interestingly enough, our proposed translation delivers a circuit sat formula where the circuit part contains a set of covered gates (i.e. the clauses representing such gates appears in the original formula) and a set of derived gates from the interaction between different part of the formula.

The circuit sat formula obtained by our encoding can be exploited in different ways. First as proposed recently, particularly when dealing with instances encoding EDA applications, one can exploit promising circuit SAT solver as in [11, 17] or hierarchical

SAT solving as in [12]. Secondly, SAT solvers can be used on the new CNF formula obtained from the circuit sat formula. We can also exploit the derived circuit sat formula to compute a strong backdoor set of variables [19, 8].

The paper is organized as follows. After some preliminary definitions, works related to our proposed approach are discussed. The circuit sat based translation of CNF formula is then presented (section 4). In section 5, interesting refinements are described. Our circuit encoding of SAT instances is exploited in two different ways defined in section 6 and evaluated in section 7.

2 Technical background

Let $\mathcal{B}$ be a boolean (i.e. propositional) language of formulas built in the standard way, using usual connectives ($\vee$, $\wedge$, $\neg$, $\Rightarrow$, $\Leftrightarrow$) and a set of propositional variables. A *CNF formula* Σ is a set (interpreted as a conjunction) of *clauses*, where a clause is a set (interpreted as a disjunction) of *literals*. A literal is a positive or negated propositional variable. Let us recall that any boolean formula can be translated to CNF using linear Tseitin encoding [18]. A *unit* (resp. *binary*) clause is a clause of size 1 (resp. 2). A *unit literal* is the unique literal of a unit clause. We note n (resp. m) the number of variables (resp. clauses) of Σ. $\mathcal{V}(\Sigma)$ (resp. $\mathcal{L}(\Sigma)$) is the set of variables (resp. literals) occurring in Σ. The set $\mathcal{L}(\Sigma)$ is the union of positive literals $\mathcal{L}^+(\Sigma)$ and negative literals $\mathcal{L}^-(\Sigma)$. A set of literals $S \subset \mathcal{L}(\Sigma)$ is consistent iff $\forall l \in S, \neg l \notin S$. We denote $\overline{S}$ as $\{\neg l | l \in S\}$ the complement of S. For a given formula Σ and a literal $l \in \mathcal{L}(\Sigma)$, we can rewrite Σ as $(l \vee \alpha(l)) \wedge (\neg l \vee \alpha(\neg l)) \wedge \Gamma$, where $\alpha(l) = \forall_{c \in \Sigma | l \in c} \cup\, c - \{l\}$ (resp. $\alpha(\neg l) = \forall_{c \in \Sigma | \neg l \in c} \cup\, c - \{\neg l\}$) and $\Gamma = \{c | c \in \Sigma, c \cap \{l, \neg l\} = \emptyset\}$. We define $\Sigma \wedge x$ noted $\Sigma(x)$ as a formula obtained from Σ by assigning x the truth-value *true*. Formally $\Sigma(x) = \{C | C \in \Sigma, \{x, \neg x\} \cap C = \emptyset\} \cup \{C \backslash \{\neg x\} | C \in \Sigma, \neg x \in C\}$.

An *interpretation* of a boolean formula is an assignment of truth values $\{true, false\}$ to its variables. A *model* (*solution*) of a formula is an interpretation that satisfies the formula. Accordingly, SAT consists in finding a model of a CNF formula when such a model does exist or in proving that such a model does not exist.

Let us now introduce some definitions and notations on circuit representation (or gates).

A *(boolean) gate* is an expression of the form $y = f(x_1, \ldots, x_k)$, where f is a standard connective among $\{\vee, \wedge\}$ and where y and x_i are propositional literals, that is defined as follows :

- $y = \wedge(x_1, \ldots, x_k)$ represents the set of clauses $\{y \vee \neg x_1 \vee \ldots \vee \neg x_k, \neg y \vee x_1, \ldots, \neg y \vee x_k\}$, translating the requirement that the truth value of y is determined by the conjunction of the truth values of x_i s.t. $i \in [1..k]$;
- $y = \vee(x_1, \ldots, x_k)$ represents the set of clauses $\{\neg y \vee x_1 \vee \ldots \vee x_k, y \vee \neg x_1, \ldots, y \vee \neg x_k\}$;

Moreover, a gate $\neg y = \wedge(x_1, \ldots, x_k)$ (resp. $\neg y = \vee(x_1, \ldots, x_k)$) is equivalent to $y = \vee(\neg x_1, \ldots, \neg x_k)$ (resp. $y = \wedge(\neg x_1, \ldots, \neg x_k)$).

For a given gate g, we define $CNF(g)$ as the set of clauses encoding g. A propositional variable y (resp. $x_1, \ldots, x_k$) is an *output variable* (resp. are *input variables*) of a gate of the form $y = f(x'_1, \ldots, x'_k)$, where $x'_i \in \{x_i, \neg x_i\}$.

A propositional variable z is an *output (dependent) variable of a set of gates* iff z is an output variable of at least one gate in the set. An *input (independent) variable of a set of gates* is an input variable of a gate which is not an output variable of the set of gates. Let $\mathcal{G}$ be a set of gates, we define $|\mathcal{G}| = \sum_{\forall g \in \mathcal{G}} |g|$ st. $|g|$ is the number of its input variables.

A gate is satisfied under a given boolean interpretation iff the left and right hand sides of the gate are simultaneously $true$ or $false$ under this interpretation. An interpretation satisfies a set of gates iff each gate is satisfied under this interpretation. Such an interpretation is called a model of this set of gates.

Finally, we define a *circuit sat* formula as a conjunction of gates ($\mathcal{G}$) and clauses (C). It is called a full circuit, when $C = \emptyset$.

3 Related Works

Our approach is inspired by two related works of Purdom [13] and Boufkhad [2]. In [13], P. Purdom has proposed an original branching criterion (called complementary search) to avoid redundancy during search.

Property 1 (Purdom [13]). Let Σ be a CNF formula, l be a branching literal then Σ is satisfiable iff $\Sigma(l)$ is satisfiable or $\Sigma(\neg l) \wedge \neg\alpha(\neg l)$ is satisfiable.

As noted by Purdom, the exploitation of the property 1 requires additional clauses that can be derived by translating the formula $\neg\alpha(\neg l)$ in Disjunctive Normal Form (DNF) to a CNF formula. This drawback was also noted by Gallo and Urbani [6] :"*Purdom's branching criterion succeeds in reducing the size of the search tree but a price must be paid. In fact, the formula must be transformed into the standard form of set of clauses, which might be quite costly*". For this reason, the property above is only exploited when $\alpha(\neg l)$ is reduced to a single clause (the literal $\neg l$ occurs only once in Σ). The negation of such a clause is a set of unit clauses.

In [2], Boufkhad has defined a concept of restrictive solution. This kind of solution has special characteristics that can be checked in polynomial time and each satisfiable formula has at least one of these special solution. Three variant of solution has been proposed : Negative Prime Solution (NPS), Positive Prime Solution (PPS) and Locally Optimized Solution (LOS) (see definition 1). Using such restrictive models, Boufkhad and Dubois obtained a new theoretical upper bound of the threshold of random 3-SAT formula [5]. Similary to Purdom, another use proposed by Boufkhad [2] is to add new

clauses to the formula in order to restrict its set of models to only those with special characteristics.

Definition 1 (NPS, PPS [2]). *An NPS (resp. PPS) is a solution such that variables assigned the value false (resp. true) cannot be individually inverted to* $true$ *(resp.* $false$*) without contradicting the formula.*

Furthermore, Boufkhad [2] introduced the notion of Locally Optimized Solution (LOS) relative to a truth assignment S. It is called optimized in the sense that no better solution can be found by just inverting the value of a variable. It is said locally optimized relative to a truth assignment S because the value assigned to any variable x in S (called the reference value of x in S) is preferred to the opposite one. Any satisfiable formula has at least one LOS relative to any truth assignment S [2].

Property 2 (Boufkhad [2]). Let Σ be a CNF, $S \in \mathcal{L}(\Sigma)$ a consistent set of literals and $C = \bigwedge_{l \in S}(l \vee \neg\alpha(\neg l))$. Σ is satisfiable if and only if $\Sigma \wedge C$ is satisfiable.

From the proof of the property [2], it follows that any solution to $\Sigma \wedge C$ is a LOS relative to S of Σ.

Obviously, the two properties 1 and 2 are very similar. For the same reasons as in Purdom, only literals that occur at most twice are considered in [2]. Then, the size of the additional clauses is less than 3.

4 Circuit based encoding

In this section, we present our circuit based encoding of CNF formula. Our proposed approach is based on the results presented in the previous section. Our main goal is to avoid the drawback behind the approaches proposed by Purdom and Boufkhad.

The results presented in this section can be summarized as follows. First, using auxiliary variables, we avoid the main drawback of Purdom and Boufkhad approaches i.e. the additional constraints can be obtained using linear time approach. Second, the conjunction of the original formula and the additional constraint lead to a circuit sat formula. A connection between full circuit encoding and the satisfiability of the original formula is established.

Property 3. Let $\Sigma = (l \vee \alpha(l)) \wedge (\neg l \vee \alpha(\neg l)) \wedge \Gamma$ be a CNF. Σ is satisfiable iff $(l = \alpha(\neg l)) \wedge (l \vee \alpha(l)) \wedge \Gamma$ is satisfiable

Proof. The proof is straightforward. Indeed, the conjunction of the two formula $(\neg l \vee \alpha(\neg l)) \in \Sigma$ and the added formula $(l \vee \neg\alpha(\neg l))$ can be characterized using a an equation of the form $(l = \alpha(\neg l))$, where $\alpha(\neg l)$ is a conjunction of clauses. The rest of the proof is a consequence of the property 2. □

The following property shows how the equation $l = \alpha(\neg l)$ (see. property 3) can be translated in linear time to a set of boolean gates using auxiliary variables.

Property 4. Let $l = \alpha(\neg l) = \wedge(l_1, \ldots, l_m, c_1 \ldots, c_k)$ be a boolean equation, where $|c_i| > 1$. Let y_i be an auxiliary variable representing a clause c_i. The gate $(l = \alpha(\neg l))$ and $\{l = \wedge(l_1, \ldots, l_m, y_1, \ldots, y_k), y_1 = \vee(c_1), \ldots, y_k = \vee(c_k)\}$ are equivalent for SAT.

Obviously, if $\alpha(\neg l)$ is reduced to a unique clause c_i, no auxiliary variable is introduced i.e. $l = \vee(c_i)$.

In the following examples, we illustrate the application of properties 3 and 4 to different formulas.

Example 1. Let us consider the following CNF formula $\Sigma = (x_1 \vee x_3) \wedge (\neg x_1 \vee x_5) \wedge (\neg x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee \neg x_4) \wedge (x_2 \vee \neg x_4 \vee x_5)$. Applying the properties 3 and 4 to the literal x_1 we obtain 3 gates: $x_1 = \wedge(x_5, y_1, y_2)$, $y_1 = \vee(\neg x_2, x_3)$ and $y_2 = \vee(x_2, \neg x_4)$

Example 2. Let $\Sigma = \Gamma \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3 \vee x_4) \wedge (\neg x_4 \vee x_1) \wedge (\neg x_4 \vee x_2) \wedge (\neg x_4 \vee x_3)$ be a CNF formula encoding the gate g ($x_4 = \wedge(x_1, x_2, x_3)$). To illustrate the detection of such explicit gate, we consider two distinct case. First, if $\neg x_4 \notin \mathcal{L}(\Gamma)$, applying the properties 3 and 4 to the literal x_4 we detect the same gate g. In the second case, where $\neg x_4 \in \mathcal{L}(\Gamma)$ i.e. $\{(\neg x_4 \vee \gamma(\neg x_4))\} \subset \Gamma$, we detect a gate g' ($x_4 = \wedge(x_1, x_2, x_3, \ldots)$), the gate g' include the gate g and other auxiliary variables introduced to represent the clauses in $\gamma(\neg x_4)$.

The example 2 shows that when some clauses of the original formula express a gate (explicit gate), our approach can recover such gates in a very simple way. The question of recovering explicit gates has been subject of interesting works by [8, 15]. Properties 3 and 4 describe one step in our encoding. Given a consistent set of literals S, our proposed translation iterates the application of the above properties on each literal of S. The algorithm 1 describes the encoding of any CNF as a circuit sat formula. It produces a set of gates and a set of clauses. Combination of both sets is equivalent wrt. SAT to the original formula.

Definition 2. *Let Σ be a CNF formula, S a consistent set of literals and $\mathcal{G}$ a set of gates obtained by applying $CircuitSat(\Sigma, S)$. We define $cov(\Sigma, \mathcal{G})$ the set of clauses of Σ covered by $\mathcal{G}$ as $\{c | c \in (\neg l \vee \alpha(\neg l) and l \in S\}$ i.e. clauses considered in the for loop (line 5, algothme 1). Dually, we define $uncov(\Sigma, \mathcal{G}) = \Sigma - cov(\Sigma, \mathcal{G})$.*

Let us note that the set of uncovered clauses of Σ can be obtained from $\mathcal{G}$ and Σ i.e. $uncov(\Sigma, \mathcal{G})$.

Theorem 1. *Algorithm 1 is a correct circuit sat encoding and its worst case time complexity is $O(n \times m)$.*

Proof. At each step (while loop) a literal is selected from S, processed by the application of the property 3 and 4 and then removed from S. Consequently, the while loop is run

Algorithm 1 CircuitSat(in Σ : CNF, in S : set of literals, out $\mathcal{G}$: set of gates)

```
1: G = ∅
2: while S ≠ ∅ do
3:    select a literal l from S
4:    In = ∅
5:    for c_i ∈ α(¬l) do
6:       if |c_i| < 2 then
7:          In = In ∪ {c_i}
8:       else
9:          G = G ∪ {y_i = ∨(c_i)}
10:         In = In ∪ {y_i}
11:      end if
12:   end for
13:   G = G ∪ {l = ∧(In)}
14:   S = S − {l}
15: end while
```

exactly $|S|$ times. The obtained formula ($\mathcal{G} \wedge uncov(\Sigma, \mathcal{G})$) is circuit sat formula which is equivalent wrt. SAT to Σ (property 2).

Complexity Let consider the case where $|S| = n$. The while loop is run at most n times (line 3 and line 14). The for loop is run at most m times which is an upper bound of the number of clauses in $\alpha(\neg l)$. Consequently, we obtain a worst case time complexity of $n \times m$. □

Property 5. Let Σ be a CNF. S is a model of Σ iff the set of gates $\mathcal{G}$ obtained by $CircuitSat(\Sigma, \overline{S})$ is a full circuit encoding i.e. $uncov(\Sigma, \mathcal{G}) = \emptyset$

Proof. First, as S is a model of Σ, then $\forall c \in \Sigma, S \cap c \neq \emptyset$. Using $\overline{S}$ in $CircuitSat$ algorithm at each iteration of the while loop, we consider a literal $l \in \overline{S}$ (line 3). As $\neg l \in S$, at each iteration the set of clauses $\neg l \vee \alpha(\neg l)$ (line 5) are covered by the added gates. These clauses correspond to the clauses satisfied by $l \in S$. Iterating such a process, $CircuiSat$ covers all the clauses of Σ. The converse is also true. From the set S of consistent literals considered by $CircuitSat$ for covering all the clauses of Σ, one can construct a model $S' = \overline{S}$ of Σ. Indeed, each time a literal $l \in S$ is processed (line 3), we cover all the clauses containing $\neg l$. These clauses are satisfied by $\neg l$ ($\neg l$ is added to S'). As all the clauses of Σ are covered, the constructed set S' is a model of Σ. □

From the Property 5, we can deduce several interesting results :

- for unsatisfiable formula, our circuit sat encoding can not lead to a full circuit encoding.
- if a formula is entirely covered using $CircuitSat$, then the formula is satisfiable.
- in the general case, finding a full circuit encoding is intractable.

Consequently, the Algorithm 1 can be seen as an incomplete Satisfiability solver. Indeed, if the input formula Σ is entirely covered by a set of gates, then Σ is satisfiable (Property 5). However, if the formula Σ is not totally covered, then we can not conclude on the satisfiability of Σ. In this last case, a circuit sat formula with interesting features is obtained. Indeed, the circuit formula encode additional clauses i.e. any model of the circuit sat formula is a restrictive model (eg. NPS, PPS, LOS) of the original formula. Interestingly enough, all the auxiliary variables introduced by the circuit encoding are output variables, then their value can be determined (implied) after assigning truth values to the variables of the original formula. Consequently, any SAT solver can be used on the CNF representation of the circuit sat formula. In such a case, one can limit branching to the variables of the original formula or to the strong backdoor set as proposed in [8].

As our proposed algorithm is based on the two properties 2 and 3, if we consider S as a complete assignment of the variables of Σ, it is important to note that any model of circuit sat formula is a LOS of Σ with respect to S. Obviously, if we consider $S = \mathcal{L}^+$ (resp. $S = \mathcal{L}^-$), then any model of the circuit formula is an NPS (resp. PPS) of the original formula Σ. In these last two cases, the obtained circuit sat is described by the following property.

Property 6. Let Σ be a CNF. If $S = \mathcal{L}^+$ (resp. $S = \mathcal{L}^-$) then $CircuitSat(\Sigma, S)$ delivers a set of gates $\mathcal{G}$ such that all clauses of Σ not covered by $\mathcal{G}$ are positive (resp. negative).

Proof. The proof is straightforward. If we consider S as the set of all positive literals (resp. negatives literals), all clauses of Σ containing a negative literal (resp. a positive literal) are covered (line 5). Consequently, when Σ is a satisfiable horn (resp. reverse horn) formula, using $S = \mathcal{L}^+(\Sigma)$ (resp. $S = \mathcal{L}^+(\Sigma)$), we obtain a full circuit formula.

In the following example, we show that on structured SAT instances, our algorithm $CircuitSAT(\Sigma, S)$ can deliver interesting new constraints.

Example 3 (Pigeon hole problem). Let us consider the well known pigeon hole problem. The problem PH(n) consists in putting all the n pigeons into $n-1$ different holes such that each hole contain at most one pigeon. To encode this problem in CNF formula we need $n \times (n-1)$ propositional variables p_i^j with $i \in \{1, \ldots, n\}$, $j \in \{1, \ldots, n-1\}$. Each variable p_i^j expresses that the pigeon i is in the hole j. The CNF formula $PH(n)$ contains two kind of clauses. (i) $\bigwedge(p_i^1 \vee p_i^2 \ldots p_i^{n-1}), 1 \leq i \leq n$ encoding that the pigeon i is not left free (must be put in a hole) and $\bigwedge(\neg p_i^j \vee \neg p_k^j), 1 \leq j \leq n-1, 1 \leq i < k \leq n$ expressing that two different pigeons (i and k) can not be put in the same hole j. Applying the algorithm 1 using positive literals ($\mathcal{L}^+(\Sigma)$), we obtain $n \times (n-1)$ gates. Each gate g is of the form $p_j^i = \wedge(\neg p_1^i, \neg p_2^i \ldots p_{j-1}^i, p_{j+1}^i, \ldots \neg p_n^i)$ which expresses a new interesting implicit information (see the $CNF(g)$): "each hole contains exactly one pigeon". On real world problem, we expect that our approach might deduce interesting and meaningful knowledge.

5 circuit sat encoding: refinements

In this section, we present interesting refinements of the circuit encoding. Let us motivate them with the following example.

Example 4. Let $\Sigma = \{c_1, c_2, c_3, c_4, c_5\}$ be a CNF st. $c_1 = (x_1 \vee x_2 \vee \neg x_3)$, $c2 = (x_4 \vee x_2 \vee \neg x_3)$, $c_3 = (x_3 \vee \neg x_4 \vee x_2)$, $c_4 = (x_1 \vee \neg x_2 \vee x_3)$, $c_5 = (x_4 \vee \neg x_1)$ and $S = \{\neg x_1, x_2, \neg x_3, \neg x_4)$. The set of gates obtained by $CircuitSat(\Sigma, S)$ is $\mathcal{G} = \mathcal{G}_1 \cup \mathcal{G}_2 \cup \mathcal{G}_3$ such that :

- $\mathcal{G}_1 = \{\neg x1 = \wedge(y_1, y_2), y_1 = \vee(x_2, \neg x_3), y_2 = \vee(\neg x_2, x_3)\}$.
- $\mathcal{G}_2 = \{x_2 = \vee(x_1, x_3)\}$.
- $\mathcal{G}_3 = \{\neg x_3 = \wedge(y_3, y_4), y_3 = \vee(\neg x_4, x_2), y_4 = \vee(x_1, \neg x_2)\}$.
- $\mathcal{G}_4 = \{\neg x_4 = \wedge(y_5, \neg x_1), y_5 = \vee(x_2, \neg x_3)\}$.

Remark 1. In the example 4, we can remark that :

1. $\overline{S}$ is a model of Σ and $uncov(\Sigma, \mathcal{G}) = \emptyset$ (Property 5),
2. the gates $y_1 = \vee(x_2, \neg x_3)$ and $y_5 = \vee(x_2, \neg x_3)$ represent the same boolean function $\vee(x_2, \neg x_3)$,
3. the clause c_4 is covered three times by: $\mathcal{G}_1$, $\mathcal{G}_2$ and $\mathcal{G}_3$.

Dealing with multiple definitions
To reduce the number of auxiliary variables, one needs to avoid such multiple functional definition (case 2). It is achieved by a simple modification of the Algorithm 1. Indeed, in the for loop before processing a clause c_i, one needs to search in the current set of gates $\mathcal{G}$ for a gate $y_j = \vee(c_i)$ st. $j < i$. In such a case we use y_j instead of introducing a new auxiliary variable y_i. The variable y_j is then added to the current set of input variables In. The complexity of searching for multiple definition is in $O(|\mathcal{G}|)$.

Reducing multiple clauses covering
It is obvious that the circuit sat formula obtained by the algorithm 1 depends on the set S of literals and on the heuristic used to select the next literal (line 3). To reduce the number of times each clause is covered, different heuristics can be designed. In this paper, the next literal l which appears in the maximum number of uncovered clauses is selected. If $S \cap \mathcal{L}(uncov(\Sigma, \mathcal{G})) = \emptyset$, then no new clause can be covered and the process is ended. The while condition (line 2) is substituted with $(S \cap \mathcal{L}(uncov(\Sigma, \mathcal{G})) \neq \emptyset)$.

Partial encoding of CNF
Our approach is also dependent on the chosen set S of literals (see property 6). One can be interested in considering a specific set of literals to cover only a sub-formula of Σ. In our approach, one can specify some preferences on the set of literals or on the clauses to be covered. For example, preferences on literals can be defined using an upper bound k on the number of occurrences in Σ. This parameter can be used to make the approach incremental and adaptable to a given problem. Given a set of preferred clauses, the

second approach tries to cover in priority such preferred clauses. For clases, preference can be given to those of the intractable part of the formula. As an example, covering the non horn part of the formula might lead circuit-horn formula. This can be very useful, if we are interested in computing the bakdoor sets (see section 6). As unit propagation is complete on horn formula, one can use a SAT or Circuit Solver on the circuit part of the formula, while the horn part is only determined by boolean constraint propagation.

All the refinements presented above are integrated in our implementation of the CircuitSAT encoding.

Finally, as shown in Property 5, the task of encoding a CNF formula as full circuit one is intractable in the general case. Let us mention another possible way (not considered in this paper) to minimize the size of the uncovered set of clauses of a given formula, concern the use stochastic local search to find an approximate solution i.e a solution which satisfy a maximum number of clauses. Using such best computed solution a better covering of the formula might be achieved.

6 Handling circuit sat formula

In this section, we briefly discuss the usefulness of our proposed approach. Different paths are sketched in order to take advantage of the circuit sat representation.

Circuit sat formula & backdoor sets

The first circuit sat formula can be exploited for deriving useful hidden structure of a given problem instance. Following the recent approach proposed in [8] (LSAT) for computing backdoor sets, we show that our circuit sat representation is suitable for computing such structure. The notion of (strong) Backdoor introduced by Williams-etal in [19] is an active research topic because of its connection to problem hardness. A set of variables forms a backdoor for a given formula if there exists an assignment to these variables such that the simplified formula can be solved in polynomial time. Such a set of variables is called a strong backdoor if any assignment to these variables leads to a tractable sub-formula. This kind of structure is related to the notion of independent variables (see section 2) [16, 7]. In the general case, the set of input variables is not a strong backdoor set, because of the possible recursive definitions in the set of gates. In [8], using graph representation, the strong backdoor set is defined as the union of the set of input variables and the set of variables from the cycle cutset of the graph.

Let us remind that computing the smallest strong backdoor is an NP-hard problem. In practice, approximating (in polynomial time) a strong backdoor of “reasonable” size is an interesting and important issue. The main drawback of the approach proposed in [8] is that on many SAT instances the set of recovered gates (using unit propagation) is either small or empty. Consequently, the CNF part covers a large number of the original clauses, then the set of independent variables tends to cover all the variables of the original formula.

As we have shown in section 4, our proposed encoding can deliver a set of gates and a horn CNF part (see. property 6). Consequently, only the set of gates are considered in

the computation of the strong backdoor. Interestingly enough, our approach is adaptive, i.e. can be parametrized in order to cover the intractable part of the formula (see section 5).

Circuit SAT based preprocessing

Our circuit encoding deliver an interesting polynomial preprocessing technique. Indeed, the circuit formula, encoding a given instance is more constrained than the original one. Our preprocessing technique is made of two different steps :

1. generating a circuit formula using partial circuit encoding as defined in section 5. More precisely, the generated circuit formula is obtained by processing a set of literals S with a number of occurrences bounded by a given constant k.
2. translating the circuit formula (obtained in the first step) to CNF.

7 Experiments

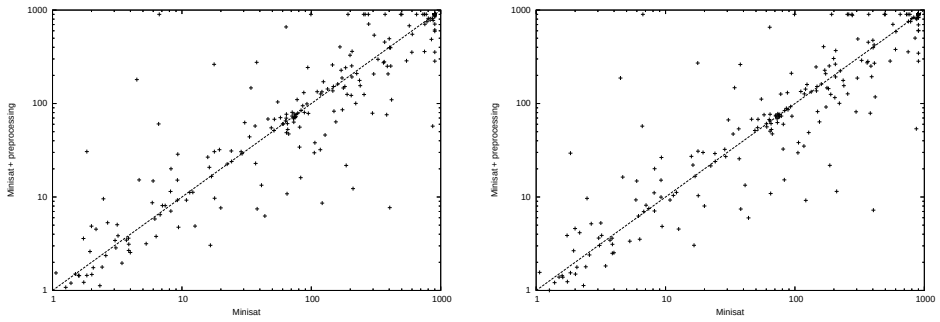
We have experimented the aforementioned CircuitSAT encoding on the last SAT competitions benchmarks (`http://www.satcompetition.org`). To show the usefulness of our circuit sat encoding, two kind of experiments are conducted. First, following the approach by Ostrowski-etal [8], strong backdoor sets are computed using as input the circuit formula obtained by our proposed circuit encoding. Secondly, our circuit sat encoding is used as a preprocessing i.e. a new CNF formula is generated from its circuit encoding leading to more constrained instances (see section 6). Minisat-2.0 is then used to solve the instance with and without our preprocessing. All experiments have been conducted on Pentium IV, 3Ghz under linux Fedora Core 4.

In table 7, experimental evaluation is illustrated by of the number of gates ($\#gates$), number of auxiliary variables ($\#aVar$), number of independent variables ($\#iVar$), number of variables in the cycle cutset $\#cSet$, the size of the strong backdoor set ($\#sB = \#iVar + \#cSet$) and by the cumulated (circuit encoding + strong backdoor set computation) cup-time ($times$) in seconds. As the set S of literals used for encoding a given instance is reduced to the positive literals of the instance ($S = \mathcal{L}^+$), all the non covered clauses are horn. Consequently, the number of initial horn clauses ($\#horn$) and the remaining number of horn clauses ($rHorn$) after the translation are also given. This experimental evaluation shows clearly the efficiency of our proposed encoding, on all instances the cumulated time is less than one second. In term of the backdoor size, our approach is competitive with Ostrowski-etal published results [8]. On many classes of instances, the size of the strong backdoor is about 50 % of the number of variables. Note that for flat200-1, phnf-size10, hole-10 (see example 3) no auxiliary variables are needed. On the these instances all positive literals occurs exactly one time. Furthermore the remaining not covered horn clauses is still close to the initial number of horn clauses. For the pigeon hole instance, the number of horn clauses remain the same. Indeed, as the horn clauses are totally negative, none of them is covered using $S = \mathcal{L}^+$.

$instance$	$\#var$	$\#cla$	$\#horn$	$\#rHorn$	$\#gates$	$\#aVar$	$\#iVar$	$\#cSet$	$\#sB$	$time$
qg1-08	512	148 957	23 608	23 608	236	176	182	56	238	0.16
par32-1	3 176	10 277	3 831	3 330	5 791	4 903	852	843	1 695	0.20
logistics.d	4 713	21 991	19 621	11 216	6 100	4 234	2 473	1 518	3 991	0.13
bmc-galileo9	63 624	326 999	188 648	136 873	40 621	28 790	30 655	8 573	39 228	1.00
flat200-1	600	2 237	2 037	2 037	200	0	400	0	400	0.05
phnf-size10	61 649	3157 120	2361 046	850 086	35 348	0	18 030	25 120	43 150	7.60
hole-10	110	561	550	550	11	0	99	0	99	0.01
uuf250-1	250	1 065	556	288	1 246	1 086	76	108	182	0.01

Table 1. Circuit encoding and strong backdoor set computation

The scatter plots given in figure 1 illustrate the comparative results of the state-of-the-art solver Minisat-2.0 on the original instances and on instance obtained using our proposed preprocessing with different bounds k ranging from 0 to 3. The bound $k = 0$ corresponds to the results obtained on the original instance. We tested all the 739 instances corresponding to the crafted and industrial categories. Each dot with (t_x, t_y) coordinates corresponds to a given SAT instance. Dots above (resp. below) the diagonal indicate instances where the original formula is solved faster i.e. $tx < ty$ (resp. slower i.e. $tx > ty$) than the preprocessing one. For clarity reason, only instances solved in more than one second are presented in figure 1. Indeed, for these easily solved instances the preprocessing is not necessary.



$k = 1$ $k = 3$

Fig. 1. Minisat-2.0 with and without preprocessing

All these results are summarized in table 2. The number of dots below (resp. above) the diagonal corresponding to the number of instances where the preprocessing improve $\#imp$ (resp. decrease $\#dec$) the Minisat performances are reported. The total number of solved instances ($\#solved$) and the total cpu-time in seconds ($time$) are also given. The best results are obtained with bound $k = 3$. Indeed, Minisat on the preprocessed instance is able to solve 5 more instances within the time limit set to 900 seconds.

$bound$	$\#imp$	$\#dec$	$\#solved$	$time(sc.)$
$k=0$	-	-	463	279 368
$k=1$	113	90	461	280 959
$k=2$	108	94	460	281 322
$k=3$	107	104	468	277 308

Table 2. All instances : summary of results

The table 3 gives a detailed picture of these promising results. For $k = 1$ and $k = 3$, time in seconds needed for processing the instance ($pre\text{-}time$) and for solving the instance ($run\text{-}time$) is also reported. We are actually conducting extensive experiments using greater values for the bound k, to determine the maximum value of k leading to better improvements.

		$k=0$	$k=1$		$k=3$	
$instance$	SAT	$run\text{-}time$	$run\text{-}time$	$pre\text{-}time$	$run\text{-}time$	$prep\text{-}time$
sgp_6-6-10	Y	296.6	74.8	3.9	847.7	4.7
mod2-rand3bip-sat-220-3	Y	105.4	29.7	0.1	31.86	0.1
strips-gripper-14t27	Y	-	-	0.4	61.4	0.4
dead-dnd009	N	327.2	297.0	0.1	733.7	0.1
pbl-00100	N	-	354.4	0.1	106.5	0.1
grid-pbl-0070	N	-	-	0.1	150.1	0.1
QG6.gensys-ukn001	N	728.1	-	0.1	298.8	0.1
clauses-6	Y	749.3	-	8.3	-	14.1
gensys-icl005	N	543.1	286.0	0.1	596.9	0.1

Table 3. Minisat-2.0 with and without preprocessing : a detailed picture

8 Conclusion

In this paper, we have proposed an original new circuit sat based encoding of CNF formula. It makes an original use of the concept of restrictive models introduced by Boufkhad to polynomially translate any formula in conjunctive normal form (CNF) to a circuit sat representation (a conjunction of gates and clauses). The derived circuit sat formula is equivalent with respect to SAT to the original CNF. Depending on the characteristics of the considered restrictive solution, two variants have been proposed. We proved that when they are used, a formula made of a conjunction of gates and horn (or reverse horn) clauses can be obtained. We also provided a connection between our encoding and the satisfiability of the original formula. Several interesting refinements and different ways to handle the circuit sat formula are proposed. Last but not least, our encoding can also be used to recover explicit gates and other meaningful structural knowledge. Our formal and informal analysis demonstrate that one can obtain several advantage by exploiting our circuit sat encoding. The experimental results show two

possible use of the circuit sat encoded formula. First, results on the strong backdoor set computation problem are very encouraging. Secondly, an incremental preprocessing (using bounds on the occurrences number of literals) leads to interesting improvements with respect to Minisat-2.0 a state-of-the-art Satisfiability solver.

References

1. Fadi A. Aloul, Arathi Ramani, Igor L. Markov, and Karem A. Sakallah. Solving difficult instances of boolean satisfiability in the presence of symmetry. *Transactions on Computer Aided Design*, 2003.
2. Yacine Boufkhad. *Aspects probabilistes et algorithmiques du problème de satisfiabilité*. phd thesis, Université de Paris 6, Laboratoire d' Informatique de Paris 6 (LIP6), December 1996.
3. H. Dixon, M. Ginsberg, and A. Parkes. Generalizing boolean satisfiability I: Background and survey of existing work. *Journal of Artificial Intelligence Research*, 21:193–243, 2004.
4. L. Drake, A. Frisch, I. Gent, and T. Walsh. Automatically reformulating SAT-encoded CSP. In *Proceedings of workshop on reformulating CSP (CP)*, 2002.
5. Olivier Dubois and Yacine Boufkhad. A general upper bound for the satisfiability threshold of random r-SAT formulae. *Journal of Algorithms*, 24(2):395–420, August 1997.
6. G. Gallo and G. Urbani. Algorithms for testing the satisfiability of propositional formulae. *journal of logic programming*, 7(1):45–61, July 1989.
7. E. Giunchiglia, M. Maratea, and A. Tacchella. Dependent and independent variables in propositional satisfiability. In *proceedings of European Conference on Logics in Artificial Intelligence*, volume 2424 of *LNCS*, pages 296–307, 2002.
8. E. Gregoire, B. Mazure, R. Ostrowski, and L. Sais. Automatic extraction of functional dependencies. In *proceedings of SAT*, volume 3542 of *LNCS*, pages 122–132, 2005.
9. Henry A. Kautz, David McAllester, and Bart Selman. Exploiting variable dependency in local search. In *"Abstracts of the Poster Sessions of IJCAI-97"*, 1997.
10. Chu Min Li. Equivalency reasoning to solve a class of hard sat problems. *Information Processing Letters*, 76:75–81, 2000.
11. F. Lu, L. Wang, K. Cheng, and R. Huang. A circuit sat solver with signal correlation guided learning. In *proceedings of international conference DATE*, pages 892–897, 2003.
12. Y. Novikov and R. Brinkmann. Foundations of hierarchical SAT solving. In *Proceedings of International Workshop on Boolean Problems*, pages 103–142, 2004.
13. P. W. Purdom. Solving satisfiability with less searching. *IEEE transactions on pattern analysis ans machine intelligence*, PAMI-6(4):510–513, July 1984.
14. Antoine Rauzy, Lakhdar Saïs, and Laure Brisoux. Calcul propositionnel : vers une extension du formalisme. In *Actes des Cinquièmes Journèes Nationales sur la Résolution Pratique de Problèmes NP-complets (JNPC'99)*, pages 189–198, Lyon, 1999.
15. J Roy, I. Markov, and V. Bertacco. Restoring circuit structure from SAT instances. In *proceedings of international workshop on Logic and synthesis*, 2004.
16. B. Selman, H. Kautz, and D. McAllester. Ten challenges in propositional reasoning and search. In *proceedings of IJCAI*, 1997.
17. C. Thiffault, F. Bacchus, and T. Walsh. Solving non-clausal formulas with DPLL search. In *proceedings of international conference CP*, pages 663–678, 2004.
18. G. Tseitin. On the complexity of proofs in propositional logics. *Automation of Reasoning: Classical Papers in Computational Logic*, 2, 1967.
19. R. Williams, C. Gomes, and B. Selman. Backdoors to typical case complexity. In *proceedings of IJCAI*, pages 1173–1178, 2003.