Polynomial restrictions of SAT: What can be done with an efficient implementation of the Davis and Putnam's procedure ?*

Antoine Rauzy

LaBRI – CNRS URA 1304 – Université Bordeaux I
51, cours de la Libération, F-33405 Talence (France)
email: rauzy@labri.u-bordeaux.fr

Abstract. Constraint Solving Problems are NP-Complete and thus computationaly intractable. Two approaches have been used to tackle this intractability: the improvement of general purpose solvers and the research of polynomial time restrictions. An interesting question follows: what is the behavior of the former solvers on the latter restrictions ?
In this paper, we examplify this problem by studying both theoretical and practical complexities of the Davis and Putnam's procedure on the two main polynomial restrictions of SAT, namely Horn-SAT and 2-SAT. We propose an efficient implementation and an improvement that make it quadratic in the worst case on these sub-classes. We show that this complexity is never reached in practice where linear times are observed, making the Davis and Putnam's as efficient as specialized algorithms.

1 Introduction

Let $\phi = C_1 \wedge C_2 \wedge \ldots \wedge C_m$ be a Boolean expression in conjunctive normal form, i.e. where each *clause* C_i is a disjunction of literals and each *literal* is either a variable p_i or its negation $\neg p_i$ $(1 \leq i \leq n)$.

The SAT(isfiability) problem consists in determining whether such an expression ϕ is true for some assignment of Boolean values to the variables $p_1, \ldots, p_n$. Cook [13] has shown that SAT is NP-complete and it is now the reference NP-complete problem [25]. A k-SAT instance is a SAT instance wherein all of the clauses have the same length k. For $k \geq 3$, k-SAT is NP-complete [13].

As the other Constraint Solving Problems and unless $P = NP$, SAT is thus computationally intractable. Two approaches have been used to tackle this intractability: the improvement of general purpose solvers and the research of restrictions for which there exists polynomial time algorithms. It must be pointed out that solving a restriction requires *a priori* two algorithms: the first one to recognize that a given instance belongs to the restriction, the second one to decide whether instances belonging to this restriction are satisfiable or not. Both must be polynomial to establish that the restriction is so.

* This work is supported by the french inter-PRC project "Classes Polynomiales"

SAT admits some (few) restrictions for which such algorithms are known. These restrictions are mainly Horn-SAT (satisfiability of sets of Horn clauses) and 2-SAT (satisfiability of sets of binary clauses) for which there exist linear time algorithms. These algorithms work on specialized data-structures (graphs or hypergraphs) and cannot be extended into general solvers for SAT. This raises an interesting question: what about the behavior of the general purpose solvers on these restrictions ?

In this paper, we aswer this question for what concerns the good old Davis and Putnam's procedure [17, 16], Horn-SAT and 2-SAT. This procedure is of a particular interest because, despite of its simplicity, it has never been shown inferior to any other complete algorithm. In particular, it outperforms the most efficient resolution based methods [6].

We first propose an efficient implementation and discuss how unit resolution and monotone literal propagation can be achieved in linear time. Then, we establish that this implementation is quadratic on Horn clauses and that an improvement we have proposed in [33] makes it quadratic on binary clauses. We give experimental results obtained on randomly generated sets of clauses. This provides evidence that the quadratic worst case complexity is never reached in practice where linear times are observed. We give elements to explain why. We examine also some related problems such as unique-satisfiability or truth of quantified boolean formulae.

The main interest of these results is that given any SAT instance, it is not necessary to test whether it belongs to one of the cited polynomial classes, in order to apply a specialized decision algorithm. The Davis and Putnam's procedure, that is a complete solver for SAT, ensures good practical results. This show a clear separation between hard instances, that are indeed hard for any algorithm, and easy instances that are symmetrically easy for well implemented general purpose solvers.

The remaining of this paper is organized as follows. We describe how the Davis and Putnam's procedure can be efficiently implemented at section 1. Then, sections 2 and 3 are respectively devoted to Horn instances and 2-SAT instances.

The paper by Davis and Putnam [17], that is cited by most of the authors, does not contain the procedure under its nowadays recursive presentation in terms of assignments of Boolean values to variables. Such a presentation has appeared for the first time in [16]. Nevertheless, even the latter paper does not give any detail about the underlying data structure (nor does the very often referenced [41]). However, this point is significant since linear time simplifications discussed below can only be achieved by choosing the appropriate data structure.

1.1 Data structure

We use a "sparse matrix" to store the clauses. Each row of the matrix encodes a clause (i.e. a list of literals). Each column encodes the list of occurrences of a variable. The design of an enumerative algorithm (i.e. one whose principle is to assign values to variables), on such a matrix requires a single basic instruction: the assignment of a value belonging to {*true, false, neutral*} to a variable. This

operation is performed by visiting each occurrence of the variable and updating various counters that are used for multiple purposes, including the design of algorithms and heuristics.

- For each clause, counts of satisfied, falsified and neutral literals it contains in the current assignment are maintained, as well as a status belonging to {*unchanged, satisfied, shortened, falsified*}. A clause is *unchanged* when all its literals have the value *neutral*. It is *satisfied* when at least one of its literals is satisfied. It is *falsified* when all its literals are falsified. Finally, it is *shortened* otherwise, i.e. when it contains no satisfied literal and at least a falsified literal and a neutral literal. A clause is *active* if it is either unchanged or shortened. A clause is *unit* when it is active and it contains only one unassigned literal.
- Global counts of clauses of each of the fourth categories are maintained.
- The value of each variable is maintained, as well as counts of its positive and negative occurrences in the clauses of each category and counts of its positive and negative occurrences in unit clauses. Note that if a variable occurs both positively and negatively in unit clauses the current set of clauses is unsatisfiable.
- A list of unit literals and a list of monotone literals are maintained. A neutral literal is said to be *monotone*, if its opposite does not occur in active clauses. It is said to be an *unit-literal* if it is the only neutral literal of an unit clause. These lists are doubly chained ones, in order to allow constant time insertion and removal of items.

We denote by $E_{p_1 \leftarrow v_1, \ldots, p_k \leftarrow v_k}$ the matrix E in which values $v_1, \ldots, v_k$ belonging to {*true, false*} are assigned to variables $p_1, \ldots, p_k$. The other variables are assumed to have the value *neutral*. By extension, we denote E_q the matrix E in which the literal q has been satisfied. We denote by $1 - v$ the opposite value of v, i.e. $1 - v = false$ when $v = true$ and vice versa, and by $\neg q$ the opposite of a literal q. Note that the presented way of carrying out assignments is not the one described for instance in the Loveland's book [41]. In general, authors assume that the set of clauses is updated each time an assignment is performed. Of course, the set of clauses encoded by $E_{p_1 \leftarrow v_1, \ldots, p_k \leftarrow v_k}$ is obtained from the set of clauses encoded by E by deleting clauses that contain a satisfied literal and suppressing the occurrences of falsified literals from the other clauses. In our experience, our way of doing is not only simpler to program but also more efficient. In the following, we always say "set of clauses" but the underlying coding by means of a sparse matrix is implicitly assumed.

Lemma 1 (Complexity of the basic operation). *The worst case complexity of the assignment of a value to a variable is in $\mathcal{O}(L \times K)$, where L denotes the maximum number of occurrences of a variable and K denotes the maximal size of a clause.*

In what follows, we keep the same meaning to L and K. We denote by V the number of variables, by C the number of clauses and by S the total number of literals occurring in the clauses. K and L are indeed majored by V and C.

1.2 Linear Time Unit and Monotone Propagation

A very interesting property of the proposed data structure is that it allows linear time unit and monotone propagations. We call *unit and monotone propagation* the process that consists in peeking repeatedly an unit or monotone literal and satisfying it until there is no more unit or monotone literals or a variable occurs both positively and negatively in unit clauses.

Lemma 2 (Linear Time Unit and Monotone Propagation). *Unit and Monotone Propagation is in* $\mathcal{O}(S)$.

Since lists of unit and monotone literals are maintained, detection of such literals is done in constant time. Let us consider the maximum number of times a literal is visited. A clause may change at most two times of status during the propagation process (from *unchanged* to *shortened* then from *shortened* to *falsified* or to *satisfied*), thus its literals are visited at most two times in order to update counters of their owning variables. An additional visit may be necessary to detect the only unassigned literal of the clause (when this clause becomes unit). The list of occurrences of a variable is visited at most once, when a value is assigned to this variable. It follows that each literal is visited at most four times. □

This simple complexity analysis is not essentially new since Minoux in [44] have already remarked that unit resolution can be performed in linear time. We extend it here to monotone literals and to the updating of various counters. Moreover, the algorithm discussed here is far more natural and general than graph based algorithms proposed by Dowling and Gallier [18] (and corrected in [50]), Minoux [44] or Ghallab and Escalada-Imaz [27].

1.3 Algorithm

The Davis and Putnam's procedure, as it is implemented on a sparse matrix, can be described in an informal way as follows: let E the set of clauses to be tested.

- If the satisfiability of E is not immediately decidable (i.e. if neither E is empty nor it contains an empty clause or a variable occurring both positively and negatively in unit clauses), then there are two cases:
- E contains a monotone or a unit literal q. In this case, the value v that satisfies q is assigned to the corresponding variable, and the algorithm is called recursively. It is sound since in both cases, E is satisfiable if and only if E_q is satisfiable.
- E does not contain monotone or unit literal. In this case, a variable p and a value v are chosen and the algorithm is called recursively first on $E_{p \leftarrow v}$ and second on $E_{p \leftarrow 1-v}$, if $E_{p \leftarrow v}$ is not satisfiable.

Heuristics. In practice, a good heuristic for choosing the next literal to assign improves dramatically the performances of the algorithm. Of course, a compromise must be found between the cost of an heuristic and its efficiency. For this study, we used two simple heuristics.

The first one so-called `ffis` (for First Fail in Shortened Clauses) that consists in choosing the variable that has first the maximum number of occurrences in shortened clauses and second the maximum number of occurrences in unchanged clauses.The chosen value is the one satisfying the most frequent literal. `ffis` is not the best heuristic we know, however it is a very interesting tradeoff and seems — as far as we have tested them on a sufficiently large benchmark — as efficient as many other proposed ones [56, 34, 42, 20]. Its cost is in $\mathcal{O}(V)$ (a traversal of the table of variables).

The second heuristics so-called `bimo` (for Best In Matrix Order) is even simpler and consists in choosing the first unassigned variable in the matrix order. The chosen value is the one satisfying the most frequent literal. Its cost is in $\mathcal{O}(1)$ (it suffices to increment a pointer each time a variable is assigned).

On-Line Algorithms. Several authors have examined the case where new clauses are added on-line to the current set [4, 32, 54]. It is clear that the Davis and Putnam's procedure can be easily adapted without any additional cost to this case: it suffices to maintain the current execution stack, i.e. the current assignment plus a number of information to handle backtracking. In what follows, we will do not mention this point anymore, but it will be implicitly understood.

2 Horn Clauses

2.1 Worst Case Complexity

A *Horn clause* is a clause that contains at most one positive literal. Horn-clauses are used extensively in Artificial Intelligence (see e.g. [31]) A set of clauses E is said *Horn-renamable* if it exists a renaming transforming E into a set of Horn clauses (a *renaming* substitutes in the whole set and for some variables p, the occurrences of p for $\neg p$ and vice versa).

Several linear time algorithms have been proposed to test the satisfiability of Horn clauses [18, 44, 27]. These algorithms do not consider the recognition problem (i.e. they take sets of Horn clauses in input). Lewis showed in [39] that in order to decide whether a set of clauses is Horn-renamable or not it suffices to check the satisfiability of an associated set of binary clauses (as it will be discussed at the next section, it is possible the perform this test in linear time). The basic idea is to associate a binary clause $(\neg p \vee \neg q)$ with each pair (p, q) of literals occurring simultaneously in a clause. Each assignment satisfying the set of binary clauses defines a Horn-renaming: it suffices to rename the variables taking the value *false* in the assignment. Conversely, each Horn-renaming defines a satisfying assignment: the value *false* is assigned to renamed variables, the value *true* is assigned to the others. Roughly done this construction may generate a quadratic number of binary clauses. Many authors have proposed linear explicit

or implicit constructions of the set of binary clauses [2, 43, 1, 40, 10, 30]. The following result is well known:

Theorem 1 (Unit-Resolution). *The unit-resolution is complete for Horn clauses.*

It means that if a Horn-renamable set of clauses is unsatisfiable, the Davis and Putnam's procedure demonstrates it just by applying the unit-literal selection rule. Therefore, the following results holds:

Lemma 3 (Unsatisfiable Horn-SAT). *The Davis and Putnam's procedure is in $\mathcal{O}(S)$ on unsatisfiable Horn-renamable instances.*

This result is already known and has been implicitly pointed out for instance in [24]. Unit literal propagation is sufficient to determine whether a Horn set is satisfiable or not. But this requires to know that the tested instance is Horn renamable. What we want to ensure is that our general decision procedure, here the Davis and Putnam's procedure, is polynomial on polynomial classes of SAT without testing before whether the tested instance belongs to a polynomial class or not. It thus remains to look at the complexity of the algorithm on Horn-renamable sets clauses that do not contain unit literal and that are thus satisfiable. Let E be such a set. We can assume without a loss of generality that E does not contain any monotone literal.

Let q be the literal chosen by the used heuristics. There are two cases:

- E_q is satisfiable. In this case, the algorithm does not backtrack on the satisfaction of q.
- E_q is unsatisfiable, and from the lemma 3, the Davis and Putnam's procedure detects this unsatisfiability in $\mathcal{O}(S)$. Thus it backtracks (with the same complexity), and then it falsifies q. The set $E_{\neg q}$ is a Horn-renamable set of clauses with at least one variable less than E.

Since the process "selection of variable / unit-literal propagation / detection of a falsified clause / backtrack" cannot be repeated more than V times, the following result holds:

Theorem 2 (Horn-SAT). *The Davis and Putnam's procedure is in $\mathcal{O}(V \times S)$ on Horn-renamable instances.*

On the one hand, this shows that the worst case complexity of the Davis and Putnam's procedure is not as good as the worst complexity of specialized algorithms that first decide in $\mathcal{O}(S)$ whether a set of clauses is Horn-renamable then test in $\mathcal{O}(S)$ whether it is satisfiable. On the other hand, the obtained complexity is low and we will see that the practical complexity is actually linear. Note that the key point to obtain a linear worst case complexity stands in the proof of Horn-renamability and that this proof is in turn equivalent to a satisfiability test of a set of binary clauses.

2.2 Experimental Complexity

Fig. 1 shows the results obtained on randomly generated Horn-SAT instances. An instance consists of C non-tautological clauses of size 3 built independently over 1000 and 2000 variables, each variable having the same probability to be drawn. One of the literals of a clause is positive, the two others are negative (to be sure not to introduce biases, instances are randomly renamed). Presented curves have been obtained by interpolating points corresponding to the draw of 100 instances for the value 0.5, 0.6 and so on of ratio C/V. Fig. 1 shows that:
– Below the value 2.3 of the ratio C/V, curves of running times with both heuristics are quasi-identical.
– Beyond this value, running times grows linearly with the ratio C/V for `bimo`, and more than linearly with `ffis`.

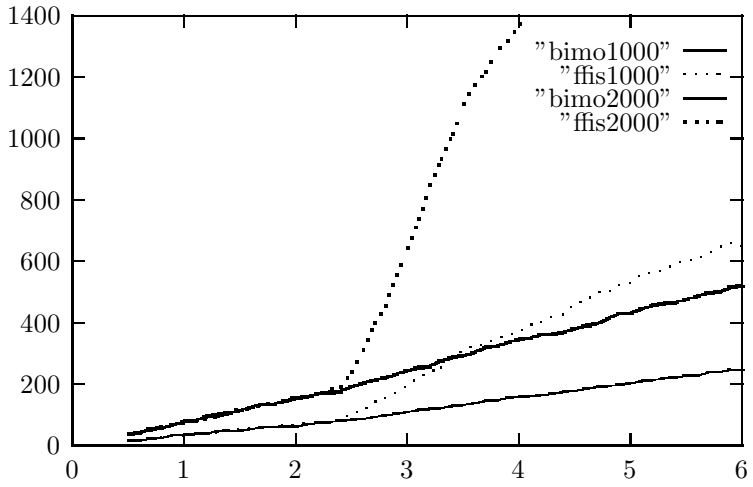

Fig. 1. Horn-SAT: Relation between sizes of instances and running times

This is explained as follows. For any value of the ratio C/V and for both heuristics, the procedure never backtracks. This is due to the choice of the most frequent literal (once the variable chosen). The number of assignments required to satisfy all of the clauses grows as C/V grows. Below the ratio 2.3, the satisfiability of instances is proved only by means of monotone literal propagation. No choice is performed, and thus the two curves are identical. Beyond this ratio, monotone literal propagation does not suffice. Some choices must be performed. The number of required choices increases as C/V increases. Conversely, the number of monotone literals decreases as C/V increases, even if it remains high for a long time. For instance, about 60% of assigned literals are monotone at $C/V = 6.5$. With `bimo`, each choice is performed in constant time and thus, even if a linear number of such choices must be done, the Davis and Putnam's procedure remains linear. With `ffis`, each choice is performed in linear time. The Davis and Putnam's procedure tends thus to be quadratic, since it must

perform ρV choices, where ρ is a constant between 0 and 1 that depends on C/V.

These observations justify the way we draw instances: the easiness of a Horn-renamable instance depends on the number of monotone literals it contains, that depends on turn on the proportion of "positive" literals. Statistically, the more there are such literals, the less there are monotone ones. By compelling at least one literal per clause to be "positive", we ensure that there is a quite high proportion of such literals, namely about 1/3. Note that drawing instances whose clauses are longer than 3 would decrease this proportion. On the other hand, introducing binary clauses makes the problem easier, as we will see at the next section.

How to reach the worst case complexity ? It is possible to build instances for which the worst case complexity is reached (even in terms of the number of assignments). This is exemplified by the following family.

$$\begin{aligned} E_n = \quad & \textstyle\bigwedge_{i=1..n}(\neg p_i \vee q_i) \\ & \wedge \textstyle\bigwedge_{i=2..n}(\neg q_{i-1} \vee p_i) \\ & \wedge (\textstyle\bigwedge_{i=1..n}(\neg p_i \vee r_i) \wedge (p_i \vee \neg r_i)) \\ & \wedge (\textstyle\bigwedge_{i=1..n}(q_i \vee \neg s_i) \wedge (q_i \vee \neg t_i)) \\ & \wedge (\textstyle\bigwedge_{i=1..n}(\neg q_i \vee u_i) \wedge (\neg q_i \vee v_i)) \\ & \wedge (\textstyle\bigwedge_{i=1..n}(s_i \vee \neg t_i) \wedge (\neg s_i \vee t_i)) \\ & \wedge \big(\textstyle\bigwedge_{i=1..n-1}(u_i \vee \neg v_i) \wedge (\neg u_i \vee v_i)\big) \\ & \wedge (u_n \vee v_n) \wedge (\neg u_n \vee \neg v_n) \end{aligned}$$

E_n is horn-renamable (it suffices to rename v_n to obtain a Horn set). The p_i's have 4 occurrences (2 positives and 2 negative), excepted p_1 that has a positive occurrence less. The q_i's have 6 occurrences (3 positives and 3 negatives). The r_i's have 2 occurrences (1 positive and 1 negative). The s_i's and the t_i's have 3 occurrences (1 positive and 2 negative). The u_i's and the v_i's have 3 occurrences (2 positive and 1 negative) Thus, E_n contains $24 \times V - 1$ literal occurrences and thus has a size linear in V. There is no unit and no monotone literal in E_n. `ffis` may choose to assign the value 1 to q_1, since q_1 is one of the most frequent variables and has as many positive and negative occurrences. The set $E_{n_{q1\leftarrow 1}}$ is shown unsatisfiable in at least $2n$ unit-literal assignments:

$$q_1 \Rightarrow p_2 \Rightarrow q_2 \Rightarrow p_3 \Rightarrow \ldots \Rightarrow q_n \Rightarrow (u_n \wedge v_n)$$

but the last two clauses constrain u_n and v_n to take different values. Thus, the algorithm backtracks and assigns the value 0 to q_1. It is easy to verify that

$$E_{n_{q_1\leftarrow 0}} \equiv (E_{n-1} \wedge \neg p_1 \wedge \neg r_1 \wedge \neg s_1 \wedge \neg t_1 \wedge (u_1 \Leftrightarrow v_1))$$

The next chosen literal may be q_2, and so on. It follows that at least $n(n+1)/2$ assignments are necessary to find a model of E_n. The same result holds for `bimo` (with the appropriate variable ordering). Note that the above instances are compound of binary clauses.

2.3 Unique Horn Satisfiability

The unique satisfiability problem (unique-SAT) asks whether there exists a unique solution to a given SAT instance. Unique-SAT is known to be co-NP-complete [5]. In [48], D. Pretolani proposes linear time algorithm to solve unique-SAT on Horn clauses. This algorithm involves rather complex techniques. By going on after the first model was found (indeed in the case of satisfiable instances), the Davis and Putnam's procedure will check whether there exists a second model within the same complexity than for the first one: it will explore remaining alternatives, and on each unexplored right branch, either the corresponding set is unsatisfiable and it will be demonstrated it in $\mathcal{O}(S)$ in the worst case, or it is satisfiable and the procedure finds the second model in $\mathcal{O}(V \times S)$. Since the number of unexplored alternatives is majored by V, we have:

Theorem 3 (unique-Horn-SAT). *The Davis and Putnam's procedure is in $\mathcal{O}(V \times S)$ on unique-Horn-SAT.*

In practice, determining whether a Horn-instance is unique satisfiable is not more costly than finding a solution: experiments reported above show that even for very large ratios C/V (50 for instance), there remain variables not valued in the found solution. Thus this solution is in fact a set of 2^k solutions, where k is the number of free variables.

3 Binary Clauses

Binary clauses play an important role in circuit design (see e.g. [37]). Satisfiability of quadratic clauses is the other well known polynomial time restriction of the SAT problem. As the matter of fact, there are a quadratic number of binary clauses (exactly $2n(n-1)$). Thus, simply by saturating a 2-SAT instance for the resolution principle, one obtains a polynomial decision algorithm. There exist linear time algorithms for solving 2-SAT [3, 22, 47]. Two different ways have been proposed to achieve linearity:

- By using the Tarjan's (linear) algorithm for exhibiting and sorting strongly connected components of a graph [52], as it is proposed by Apsvall, Plass and Tarjan in [3]. This implies a graph based formulation of the problem.
- By performing assignments of Boolean values to variables in "parallel" as suggested by Even, Itai and Shamir in [22].

However, as it is shown in [47], it appears that algorithms with a quadratic worst case complexity are more efficient in practice than those with a linear one. We will discuss this point latter.

3.1 A Frequent Error

Conversely to what many authors have written (including Petreschi and Simeone in [47]), the worst case complexity of the Davis and Putnam's procedure is not

polynomial on binary clauses. To be convinced of that, it suffices to consider the following family of instances:

$$E_n = \left(\bigwedge_{i=1..n} (p_i \vee \neg q_i) \wedge (\neg p_i \vee q_i) \right) \wedge (r \vee s) \wedge (\neg r \vee s) \wedge (r \vee \neg s) \wedge (\neg r \vee \neg s)$$

The four last clauses form an unsatisfiable subset. It is easy to verify that the first ones, that encode the relations $p_i = q_i$ for $i = 1..n$, admit 2^n models. With a heuristics such as `bimo` presented at section 1, that chooses variables in the order $p_1, \ldots, p_n, r$, the Davis and Putnam's procedure goes through a complete binary tree on height n. Each leaf of this tree consists of the small search tree showing that the subset of the four last clauses is unsatisfiable.

This way of building sets of clauses – a big subset admitting a lot of solutions and a small unsatisfiable subset – often misleads enumerative methods. Note that it is always possible to mislead any reasonable heuristics (such as `ffis`), by duplicating some clauses, for these heuristics are based on counting arguments.

3.2 An improvement

Nevertheless, an improvement that we have introduced in [33] ensures the polynomiality of the algorithm on binary clauses (no complexity study was done in the cited paper). This improvement generalizes the notion of monotone literal to sets of literals. It uses the following property – so-called model separation:

Lemma 4 (Model separation). *Let E be a set of clauses and σ a partial assignment of the variables of E. If E_σ does not contain any shortened (nor falsified) clause, then E is satisfiable if and only if E_σ is satisfiable.*

Proof it suffices to see that E_σ is a subset of E. Thus, if it is unsatisfiable, E is unsatisfiable too. Otherwise, σ extends any model of E_σ into a model of E. □

This property is used in order to prune the search tree:

Assume the values $v_1, \ldots, v_k$ assigned to the variables $p_1, \ldots, p_k$ in this order. Assume too that the set $E_{p_1 \leftarrow v_1, \ldots p_k \leftarrow v_k}$ has been shown unsatisfiable (a complete subtree under the node labeled with p_k has been explored). Assume finally that $E_{p_1 \leftarrow v_1, \ldots p_k \leftarrow v_k}$ does not contain any clause shortened by the assignment of the variables $p_i, p_{i+1}, \ldots, p_k$ $(1 \leq i \leq k)$.

Then, from the model separation lemma, the search tree can be pruned from the node labeled with p_{i-1} to the node labeled with p_k.

Let, for instance, E be the following set of clauses:

$$\begin{aligned} E = \quad & (p_1 \vee \neg q_1) \wedge (\neg p_1 \vee q_1) \wedge (p_2 \vee \neg q_2) \wedge (\neg p_2 \vee q_2) \\ & \wedge (r \vee s) \wedge (\neg r \vee s) \wedge (r \vee \neg s) \wedge (\neg r \vee \neg s) \end{aligned}$$

Assume that the procedure assigns first the value 1 to p_1 (and thus the value 1 to q_1 by unit literal propagation), then second the value 1 to p_2 (and thus the value 1 to q_2 by unit literal propagation). $E_{p_1 \leftarrow 1, q_1 \leftarrow 1, p_2 \leftarrow 1, q_2 \leftarrow 1}$ is unsatisfiable and

contains no shortened clause. Thus, E is also unsatisfiable and it is not necessary to explore the assignments $p_1 \leftarrow 1, q_1 \leftarrow 1, p_2 \leftarrow 0, q_2 \leftarrow 0$ and $p_1 \leftarrow 0, q_1 \leftarrow 0$.

Note that the complexity of the detection of the model separation is linear w.r.t. the number of pruned branches, thanks to the counters described in the first section.

The improved Davis and Putnam's procedure, when applied to sets of binary clauses, is very similar to an algorithm proposed in [22] to solve instances of the timetable problem (which is reducible to 2-SAT). Model separation lemma also generalizes the notion of "autarch" proposed by Monien and Speckenmeyer in [46]. The key idea is that when a pair variable/value is chosen, either this assignment is demonstrate unsatisfiable by unit literal propagation, or it can be definitely kept.

3.3 Polynomiality of the improved Davis and Putnam's procedure

Let E be a set of binary clauses. We can assume without a loss of generality that E does not contain unit-clause, nor monotone literal.

Let $< p, v >$ be the pair variable/value chosen by the heuristic. The assignment of v to p creates eventually some unit-literals. These literals are chosen, thanks to the unit-literal selection rule. Their assignments may create new unit-literals that are themselves chosen and so on. At the end of this process, a set E_σ is obtained. There are two cases:

- At least one of the clauses of E is falsified by σ. Then, the algorithm backtracks and assigns the value $1 - v$ to p.
- None of the clauses of E_σ is falsified and E_σ does not contain any shortened clause, since a shortened binary clause is unit. In this case, the model separation lemma holds and E is satisfiable if and only if E_σ is satisfiable too. Thus, the partial assignment σ is definitive.

Since the process "selection of variable / unit-literal propagation / detection of a falsified clause / backtrack" cannot be repeated more than V times, the following result holds:

Theorem 4 (2-SAT). *The improved Davis et Putnam's procedure is in $\mathcal{O}(V \times S)$ on binary clauses.*

3.4 Experimental Complexity

Before presenting experimental results of the Davis and Putnam's procedure on binary clauses, we must discuss recent works about threeshold phenomena and randomly generated k-SAT.

The interest in randomly generated k-SAT instances has been increased recently by an experimental result due to several authors independently [21, 45, 38, 15] that remarked that k-SAT instances obey a 0/1 law. Below a ratio C/V that depends on k, the probability to draw a satisfiable instance tends to 1 as

V tends to infinity. Beyond this ratio it tends (quickly) to 0. For $k = 2$ the threshold has been actually established equal to 1 by several authors [12, 28]. For $k = 3$ and $k = 4$ experimental values have been found (respectively 4.25 for $k = 3$ and 9.8 for $k = 4$). Researchers work hard to find (theoretical) lower and upper bounds for these values and Dubois in [20] gives a general equation $ln(2) - V.2^k - exp(kV/(2^k - 1) = 0$, for which the upper zeros would be very close to the values of the thresholds of k-SAT instances (for k=3 and beyond). Threshold phenomena occur also in sets of clauses with different lengths [26].

In a practical point of view, one can observe that the hardest randomly generated k-SAT instances for Davis and Putnam's procedure (and for other algorithms, even incomplete ones such as Selman's GSAT [51]) are those around the threshold [11, 45, 20, 49] that is where about 50% of the drawn instances are satisfiable. Even for small numbers of variables (say 50) running times suddenly increase near the threshold and quickly decrease after. Satisfiable instances are significantly easier than unsatisfiable ones, but they are hard too, i.e. that running times quickly increase as V increases.

The Fig. 2 shows the results obtained on randomly generated 2-SAT instances (heuristics: `bimo`). An instance is compound of C different and non tautological clauses, all literals having the same probability. For 1000, 1500, 2000 and 2500 variables, 100 instances have been generated for the values 0.5, 0.55, 0.60, ..., 1.95 and 2 of the ratio C/V. This study enlighten the following phenomena:

– For a given the number of variables, the average running time reaches a maximum near the value 1 of the ratio C/V, i.e. near the threshold. The maximum is closer to 1 as V increases.

– At least for the ratio C/V around 1 (i.e. near the difficulty peek), running times grows linearly with V.

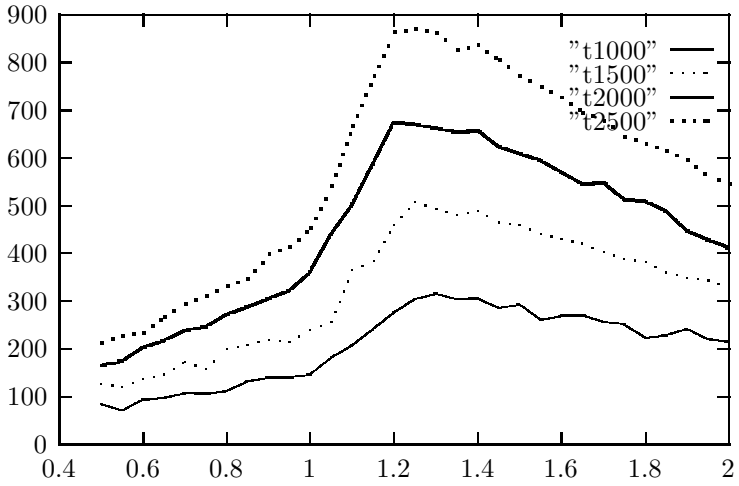

Fig. 2. 2-SAT: Relation between sizes and running times

This is explained as follows. Below the value 1 of the ratio C/V, instances are almost always satisfiable, and thus are Horn instances (since a 2-SAT instance is satisfiable if and only if it is Horn). The results obtained in the previous section apply here too, i.e. the procedure never bactracks. The difference stands in that here only one choice is necessary in average. The number of assignments increases as V increases, which explains that the running times (slowly) increases. Beyond the value 1 of the ratio C/V, instances tends to be unsatisfiable. For both heuristics (`bimo` and `ffis`), the procedure backtracks exactly once (in average). Thus unsatisfiablity is shown by means of two unit literal propagations. This has been already remarked by Petreshi and Simeone [47]. The "length" of each unit literal propagation depends on the ratio C/V: the more it is close to 1, the more propagations tend to be long. This explains why there is a difficulty peak.

How to reach the worst case complexity ? The example of the previous section that shows that the Davis and Putnam's procedure can actually reach its worst case quadratic complexity on Horn clauses is also a 2-SAT instance. Moreover, it is easy to verify that the model separation property never applies on this example.

Fig. 3, a Davis and Putnam's tree is pictured that sketches how the worst case complexity is reached on both Horn clauses and binary clauses. This tree clarifies the idea behind the Even, Itai and Shamir's algorithm. If the two unit literal propagations are performed in "parallel" (and fairly), one can stop immediatly after the first one is achieved. Thus, left branches of the pictured tree are "cut" at the same "length" than right branches. This trick ensures the linearity.

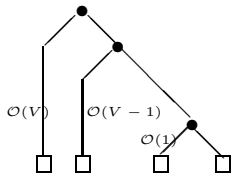

Fig. 3. 2-SAT: A worst case Davis and Putnam's tree

3.5 Related Problems

Unique Satisfiability A linear algorithm that determine whether set of binary clauses is uniquely satisfiable has been proposed in [29]. It is derived from the already cited algorithm proposed in [3]. The argumentation we have developed for Horn clauses applies here too. Thus, we have the following result:

Theorem 5 (unique-2-SAT). *The improved Davis and Putnam's procedure is in $\mathcal{O}(S \times V)$ on unique-2-SAT.*

Note that, for the same reasons than for Horn instances, the Davis and Putnam's procedure is not more costly in practice to find the first solution than to

determine whether it is unique (and thus there are good reasons to think that it is more efficient in practice than the algorithm of [29]).

Quantified Boolean Formulae Let $\psi = Q_1 x_1 Q_2 x_2 \dots Q_n x_n \phi$, where each Q_i is either $\exists$ or $\forall$ and ϕ is a formula in conjunctive normal form built over the variables $x_1, \dots, x_n$. The quantified Boolean formulae problem (QBF) consists in determining whether such a formula ψ is true. QBF is P-SPACE complete and limitations on the position and the number of quantifier alternations define the polynomial hierarchy (see [35] for a survey on complexity classes).

In [3], a linear algorithm for solving QBF for quadratic formulae is proposed. What we do here uses a similar trick (but translated from a graph based algorithm to a assignment based algorithm).

Let ψ a QBF instance in which all the clauses are at most quadratic. The quantifier alternation defines an order on variables: $x_1 < x_2 < \dots < x_n$.

Consider the following adaptation of the Davis and Putnam's procedure:
– The terminal cases are those of the standard procedure (all the clauses are satisfied or a clause is falsified), plus the following one: if a universally quantified variable occurs into an unit-clause, then ψ is false (since the variable cannot take two values).
– The monotone literal selection rule is removed.
– The chosen variable, when there are no unit literal, is the least unassigned one. Let x_i be this variable, then examine successively $\phi_{x_i \leftarrow 0}$ and $\phi_{x_i \leftarrow 1}$ by propagating these assignments with the unit literal selection rule. Now, there are two cases : If x_i is existentially quantified, then either both assignments drive to failure cases and the formula ψ is false, or the last assignment is kept and the algorithm is called recursively on the corresponding formula. If x_i is universally quantified, then either one of the assignments drives to a failure case and the formula ψ is false, or the last assignment is kept and the algorithm is called recursively on the corresponding formula.

The soundness of the algorithm comes from the following lemma:

Lemma 5 (Extended model separation). *Let $\psi = Q_1 x_1 Q_2 x_2 \dots Q_n x_n \phi$ be quadratic QBF instance with no unit-clause, and such that the unit propagations of the assignments $x_1 \leftarrow 0$ and $x_1 \leftarrow 1$ don't drive to failure cases. Let ψ_0 and ψ_1 be the two obtained formulae (ϕ_0 and ϕ_1 being their respective sets of clauses).*

1. *ϕ_0 is satisfiable if and only if ϕ_1 is satisfiable.*
2. *For any variable y occurring in one of these formulae and that is not assigned in ϕ_0 and ϕ_1, $\phi_v \models y$ if and only if $\phi_{1-v} \models y$ ($v \in \{0, 1\}$).*

Proof. The first point comes from the model separation lemma. The second one is demonstrated as follows: Assume ϕ_0 and ϕ_1 satisfiable and $\phi_v \models y$. Since ϕ_v is a proper subset of ϕ, $\phi \models y$. Assume that there exists a model of ϕ_{1-v} that assigns the value 0 to y. Then by model separation lemma, this model can be extended into a model of ϕ. A contradiction. □

The above lemma explains why it is not necessary to explore the two branches of the alternative when a choice point occurs: one suffices to make sure that either

a universally quantified variable is implied and thus the whole formula is false, or that no universally quantified variable is implied and the formula is true.

Since the examination of least unassigned variable is in $\mathcal{O}(S)$, the following result holds:

Theorem 6 (QBF restricted to quadratic clauses). *The extended improved Davis et Putnam's procedure is in $\mathcal{O}(V \times S)$ on the QBF problem restricted to sets of binary clauses.*

Note however that, on the contrary to what happens for 2-SAT and unique 2-SAT, this result requires to know that the given instance contains only quadratic clauses. Once again there are good reasons to think that the improved Davis and Putnam's procedure is more efficient in practice than the algorithm of [3].

4 Conclusion

What is done : In this paper, we provided worst case complexity bounds for the Davis and Putnam's procedure applied to Horn clauses and binary clauses. In both cases, this complexity is quadratic. In practice, the Davis and Putnam's procedure never reaches this complexity on randomly generated instances where linear times are observed.

We designed, on the same basis, algorithms to solve related problems such as the unique satisfiability or the truth of a quantified Boolean formula that have a good complexity on known polynomial restrictions of these problems.

What cannot be done : The variations studied here are designed without adding any functionality to the standard data structure used to implement the Davis and Putnam's procedure. They use two principles:
– A filtering principle : the unit literal propagation rule.
– An intelligent backtracking principle : the model separation property.

There exist polynomial classes that cannot be captured with these simple mechanisms. These classes are those defined by means of structural properties of the underlying graph of constraints. The Constraint Solving Problem community has extensively studied this kind of properties. It is well known, for instance, that if this graph is a tree, the problem is solvable in polynomial time. But this requires to assign values to variables in an order depending on the graph. Such an order cannot be induced by counting arguments (what do heuristics associated with the Davis and Putnam's procedure). In the case of SAT, several such classes have been exhibited [55, 1, 23, 36]. But, conversely to what happens for Horn-SAT and 2-SAT, it is doubtful that these classes have any practical interest, due to their very unnatural look. Nevertheless it could be interesting to randomly generate instances of these classes and see what happens.

What could be done : Other polynomial restrictions of SAT could be handled as we have done in this paper. It is the case of $r, r-SAT$ instances of Tovey [53] (improved by Dubois [19]) that are trivial to solve for they contain less clauses

than variables. More interesting is the class of q-Horn clauses introduced in [7] that generalizes both Horn clauses and binary clauses. A linear algorithm has been proposed recently to recognize instances of this class in [9] (once recognized, solving them could be easily done in linear time). A complexity index of SAT instances based on this class has been proposed in [8]. In [49], we have proposed a variation on the Davis and Putnam's procedure that is polynomial on this class and once again linear in practice. However, our variation has a quite high complexity that can be surely improved.

More generally, it could be interesting to examine which simple filterings and/or pruning principles make enumerative algorithms (designed to solve constraint problems) polynomial on known polynomial restrictions. A first step in this direction has been done for CSPs in [14].

References

1. V. Arvind and S. Biswas. An $\mathcal{O}(n^2)$ algorithm for the satisfiability problem of a subset of propositional sentences in CNF that includes horn sentences. *Information Processing Letters*, 24:67–69, 1987.
2. B. Aspvall. Recognizing Disguised NR(1) Instances of the Satisfiability Problem. *Journal of Algorithms*, 1:97–103, 1980.
3. B. Aspvall, M. Plass, and R. Tarjan. A Linear Time Algorithm for Testing the Truth of Certain Quantified Boolean Formulae. *Information Processing Letters*, 8(3):121–123, 1979.
4. G. Ausiello and G. Italiano. On-Line Algorithms for Polynomially Solvable Satisfiability Problems. *Journal of Logic Programming*, 10:69–90, 1990.
5. A. Blass and Y. Gurevitch. On the unique satisfiability problem. *Information and Control*, 55:80–88, 1982.
6. J.M. Boï and A. Rauzy. Two algorithms for constraints system solving in propositional calculus and their implementation in prologIII. In P. Jorrand and V. Sugrev, editors, *Proceedings Artificial Intelligence IV Methodology, Systems, Applications (AIMSA'90)*, pages 139–148. North-Holand, September 1990. Alba-Varna bulgarie.
7. E. Boros, Y. Crama, and P.L. Hammer. Polynomial-time inference of all implications for Horn and related formulae. *Annals of Mathematics and Artificial Intelligence*, 1:21–32, 1990.
8. E. Boros, Y. Crama, P.L. Hammer, and M. Saks. A complexity index for satisfiability problems. *SIAM Journ. Comp.*, 23:45–49, 1994.
9. E. Boros, P.L. Hammer, and X. Sun. Recognition of q-Horn formulae in linear time. *Discrete Applied Mathematics*, 55:1–13, 1994.
10. V. Chandru, C.R. Coulard, P.L. Hammer, M. Montanez, and X. Sun. On renamable Horn and generalized Horn functions. In *Annals of Mathematics and Artificial Intelligence*, volume 1. J.C. Baltzer AG, Scientific Publishing Company, Basel Switzerland, 1990.
11. P. Cheeseman, B. Kanefsky, and W.M. Taylor. Where the Really Hard Problems Are. In *Proceedings of the International Joint Conference of Artificial Intelligence, IJCAI'91*, 1991.
12. V. Chvátal and B. Reed. Miks gets some (the odds are on his side). In *Proceedings of the 33rd IEEE Symp. on Foundations of Computer Science*, pages 620–627, 1992.

13. S.A. Cook. The Complexity of Theorem Proving Procedures. In *Proceedings of the 3rd Ann. Symp. on Theory of Computing, ACM*, pages 151–158, 1971.
14. M.C. Cooper, D.A. Cohen, and P.G. Jeavons. Characterizing Tractable Constraints. *Artificial Intelligence*, 65:347–361, 1994.
15. J.M. Crawford and L.D. Auton. Experimental results on the crossover point in satisfiability problems. In *Proceedings of the Eleventh National Conference on Artificial Intelligence (Washington, D.C., AAAI'1993)*, pages 21–27, 1993.
16. M. Davis, G. Logemann, and D. Loveland. A Machine Program for Theorem Proving. *CACM*, 5:394–397, 1962.
17. M. Davis and H. Putnam. A Computing Procedure for Quantification Theory. *JACM*, 7:201–215, 1960.
18. W.F. Dowling and J.H. Gallier. Linear-time Algorithms for Testing the Satisfiablity of Propositional Horn Formulae. *J. Logic Programming*, 3:267–284, 1984.
19. O. Dubois. On the r,s-SAT satisfiability problem and a conjecture of Tovey. *Discrete Applied Mathematics*, 26:51–60, 1990.
20. O. Dubois, P. André, Y. Boufkhad, and J. Carlier. SAT versus UNSAT. In *SAT Chalenge*, volume 26, pages 415–436. AMS, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, 1996.
21. O. Dubois and J. Carlier. Sur le problème de satisfiabilité. Communication at the Barbizon Workshop on SAT, october 1991.
22. S. Even, A. Itai, and A. Shamir. On the Complexity of Timetable and Multicommodity Flow Problems. *SIAM J. Comput.*, 5:691–703, 1976.
23. G. Gallo and M.G. Scutellà. Polynomially Solvable Satisfiability Problems. *Information Processing Letters*, 29:221–227, 1988.
24. G. Gallo and G. Urbani. Algorithms for Testing the Satisfiablity of Propositional Formulae. *Journal of Logic Programming*, 7:45–61, 1989.
25. M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, San Fransisco, 1979.
26. I.P. Gent and T. Walsh. The SAT Phase Transition. In A.G. Cohn, editor, *Proceedings of 11th European Conference on Artificial Intelligence, ECAI'94*, pages 105–109. Wiley, 1994.
27. M. Ghallab and E. Escalada-Imaz. A linear control algorithm for a class of rule-based systems. *Journal of Logic Programming*, 11:117–132, 1991.
28. A. Goerdt. A threshold for unsatifiability. In I.M. Havel and V. Koubek, editors, *Proceedings of Mathematical Foundations of Computer Science, MFCS'92*, pages 264–272, August 1992.
29. P. Hansen and B. Jaumard. Uniquely solvable quadratic boolean equations. *Discrete Applied Mathematics*, 12:147–154, 1985.
30. J.-J. Hébrard. A linear algorithm for renaming a set of clauses as a Horn set. *Theoretical Computer Science*, 124:343–350, 1994.
31. L. Henschen and L. Wos. Unit refutations and Horn sets. *JACM*, 21(4):590–605, October 1974.
32. J.N. Hooker. Solving the incremental satisfiability problem. *Journal of Logic Programming*, 15:177–186, 1993.
33. S. Jeannicot, L. Oxusoff, and A. Rauzy. Évaluation Sémantique en Calcul Propositionnel. *Revue d'Intelligence Artificielle*, 2:41–60, 1988.
34. R.J. Jeroslow and J. Wang. Solving Propositional Satisfiability Problems. *Annals of Mathematics and Artificial Intelligence*, 1:167–188, 1990.
35. D.S. Johnson. A Catalog of Complexity Classes. In J. Van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume A, pages 67–162. Elsevier, 1990.

36. D.E. Knuth. Nested Satisfiability. *Acta Informatica*, 28:1–6, 1990.
37. T. Larrabee. Test Pattern Generation Using Boolean Satisfiability. *IEEE Transactions on Computer-Aided Design*, 11(1):4–15, January 1992.
38. T. Larrabee and Y. Tsuji. Evidence for a Satisfiability Threshold for Random 3CNF Formulas. In H. Hirsh and al., editors, *Proceedings of Spring Symposium on Artificial Intelligence and NP-Hard Problems (Stanford CA 1993)*, pages 112–118, 1993.
39. H.R. Lewis. Renaming a Set of Clauses as a Horn Set. *JACM*, 25(1):134–135, 1978.
40. G. Lindhorst and F. Shahrokhi. On renaming a set of clauses as a Horn set. *Information Processing Letters*, 30:289–293, 1989.
41. D. Loveland. *Automated Theorem Proving: A Logical Basis*. North Holland, 1978.
42. E.L. Lozinskii. A simple test improves checking satisfiability. *Journal of Logic Programming*, 15:99–111, 1993.
43. H. Mannila and K. Mehlorn. A fast algorithm for renaming a set of clauses as a Horn set. *Information Processing Letters*, 21:269–272, 1985.
44. M. Minoux. LTUR: A Simplified Linear-Time Unit Resolution Algorithm for Horn Formulae and its Computer Implementation. *Information Processing Letters*, 29:1–12, 1988.
45. D. Mitchell, B. Selman, and H. Levesque. Hard and Easy Distributions of SAT Problems. In *Proceedings Tenth National Conference on Artificial Intelligence (AAAI'92)*, 1992.
46. B. Monien and E. Speckenmeyer. Solving Satisfiability in Less than 2^n Steps. *Discrete Applied Math.*, 10:287–295, 1985.
47. R. Petreschi and B. Simeone. Experimental Comparison on 2-Satisfiability Algorithms. *RAIRO Recherche Opérationelle*, 25:241–264, 8 1991.
48. D. Pretolani. A linear time algorithm for unique Horn satisfiability. *Information Processing Letters*, 48:61–66, 1993.
49. A. Rauzy. On the Complexity of the Davis and Putnam's Procedure on Some Polynomial Sub-Classes of SAT. Technical Report 806-94, LaBRI, URA CNRS 1304, Université BordeauxI, 9 1994.
50. M.G. Scutellà. A Note on Dowling and Gallier's Top-Down Algorithm for Propositional Horn Satisfiability. *Journal of Logic Programming*, 8:265–273, 1990.
51. B. Selman, H. Levesque, and D. Mitchell. A New Method for Solving Hard Satisfiability Problems. In *Proceedings of the* 10^{th} *National Conference on Artificial Intelligence, AAAI'92*, 1992.
52. R.E. Tarjan. Depth First Search and Linear Graph Algorithms. *SIAM J. Comput.*, 1:146–160, 1972.
53. C.A. Tovey. A Simplified NP-complete Satisfiability Problem. *Discrete Applied Mathematics*, 8:85–89, 1984.
54. A. van Gelder. Linear Time Unit Resolution for Propositional Formulas - in Prolog, Yet. submitted to the Journal of Logic Programming, 1994.
55. S. Yamasaki and S. Doshita. The satisfiability problem for a class consisting of Horn sentences and some non-Horn sentences in propositional logic. *Information and Computation*, 59:1–12, 1983.
56. R. Zabih and D. Mac Allester. A rearrangement search strategy for determining propositional satisfiability. In *Proceedings of the National Conference on Artificial Intelligence, AAAI'88*, pages 155–160, 1988.