ELSEVIER

Operations Research Letters 23 (1998) 81–88

operations research letters

A two-phase algorithm for solving a class of hard satisfiability problems [1]

Joost P. Warners[a, b, *], Hans van Maaren[b]

[a] SEN2, CWI, P.O. Box 94079, 1090 GB Amsterdam, The Netherlands

[b] Department of Technical Mathematics and Informatics, Faculty of Information Technology and Systems, Delft University of Technology, P.O. Box 5031, 2600 GA Delft, The Netherlands

Received 1 April 1998; received in revised form 1 September 1998

Abstract

The DIMACS suite of satisfiability (SAT) benchmarks contains a set of instances that are very hard for existing algorithms. These instances arise from learning the parity function on 32 bits. In this paper we develop a two-phase algorithm that is capable of solving these instances. In the first phase, a polynomially solvable subproblem is identified and solved. Using the solution to this problem, we can considerably restrict the size of the search space in the second phase of the algorithm, which is an extension of the well-known Davis–Putnam–Logemann–Loveland algorithm. We conclude with reporting on our computational results on the parity instances. © 1998 Elsevier Science B.V. All rights reserved.

Keywords: Satisfiability; Polynomial algorithm; Davis–Putnam algorithm

1. Introduction

In a recent paper by Selman et al. [9] 10 challenges in propositional reasoning are formulated. One of these is to develop an efficient algorithm for solving instances arising from the parity learning problem on 32 bits [3]. Several instances of this problem are available in the DIMACS suite of SAT benchmarks [10]. None of the currently known algorithms appear to be capable of solving these instances in reasonable time. Incomplete algorithms do not succeed in finding models, while it seems that for systematic search procedures the search space is too large [9].

We develop a two-phase algorithm for the parity problems that is capable of finding models in less than five minutes. In the first phase of the algorithm a polynomially solvable subproblem is isolated and solved. The subproblem can be identified using linear programming; it has a *balanced polynomial representation* [11], and can be shown to be equivalent to a formula that is a conjunction of (nested) equivalencies (CoE). Such formulas are also known as XOR–SAT formulas, which were shown to be polynomially solvable by Schaefer [8]. First solving the CoE subformula allows us to reduce the search–space in the second phase considerably. In that phase we apply a DPLL-type algorithm [4] to a conjunction of a

* Corresponding author. E-mail: j.p.warners@twi.tudelft.nl.

[1] Supported by the Dutch Organization for Scientific Research (NWO) under grant SION 612-33-001.

formula in conjunctive normal form (CNF) and a CoE formula.

This paper is organized as follows. In Section 2 we discuss the necessary preliminaries. Subsequently, we introduce the concept of balanced polynomial representations (BPR) and show that a formula with BPR is equivalent to a CoE formula. We briefly review a polynomial-time algorithm for CoE formulas. Section 4 is concerned with the recognition of CoE subformulas, and in Section 5 we extend the DPLL algorithm to solve conjunctions of CNF and CoE formulas. We conclude with computational results.

2. Preliminaries and notation

A propositional formula Φ in conjunctive normal form (CNF) is the conjunction of n clauses, where each clause is a disjunction of literals $(\neg)p_i$. Each literal is an *atomic proposition* (or *variable*) or its negation ($\neg$). Let m be the number of atomic propositions. Thus each clause C_k is of the form

$$C_k = \bigvee_{i\in I_k} p_i \vee \bigvee_{j\in J_k} \neg p_j$$

with $I_k, J_k \subseteq \{1,\dots,m\}$ disjoint. The satisfiability problem of propositional logic is to assign truth values to the variables, such that each clause evaluates to true (i.e. one of its literals is true) and so the whole formula evaluates to true, or it must be proved that no such assignment exists.

We define the matrix $A \in \mathbb{R}^{n\times m}$ to be the *clause–variable* matrix. Each row corresponds to a clause and each column is associated with a variable. It holds that $a_{ki}=1$ if $i \in I_k$, $a_{ki}=-1$ if $i\in J_k$, while $a_{ki}=0$ for any $i \notin I_k \cup J_k$. Note that, associating a $\{-1, 1\}$ variable x_i with each proposition letter p_i, the integer linear programming formulation of the satisfiability problem can be stated as finding a vector $x \in \{-1,1\}^m$ such that $Ax \geqslant b$, where $b\in\mathbb{R}^n$, with $b_k = 2 - |I_k \cup J_k|$.

Now, let us derive a different formulation of SAT problems, based on a multiplicative rather than additive representation of clauses. Formulations of this type have been used by Gu [6] to obtain effective approximation algorithms for large-scale satisfiability problems. In the following section we need this type of formulation to characterize a particular class of polynomially solvable formulas.

A clause C_k is satisfied, if and only if $x\in\{-1,1\}^m$ satisfies

$$P_k(x) = \prod_{i\in I_k}(1-x_i)\prod_{j\in J_k}(1+x_j) = \prod_{i=1}^{m}(1-a_{ki}x_i) = 0. \qquad (1)$$

Observe that $P_k(x)$ remains a valid representation of clause C_k when multiplying it with a (strictly) positive weight w_k. Let $M=\{1,\dots,m\}$. In general, $x\in\{-1,1\}^m$ is a satisfiable assignment of a formula Φ, if and only if

$$\mathscr{P}(x) = \sum_{k=1}^{n} w_k P_k(x) = \sum_{k=1}^{n} w_k + \sum_{I\subseteq M}(-1)^{|I|}\sum_{k=1}^{n} w_k \prod_{i\in I} a_{ki}x_i = 0,$$

where in principal I runs through all possible subsets of M $(I\neq\emptyset)$ and w is a strictly positive weight vector. Note that the number of subsets that has to be taken into account can be restricted substantially, since in fact only subsets $I\subseteq M$ for which $I\subseteq I_k\cup J_k$ for some $k=1,\dots,n$ need to be considered. In general, for a clause with length ℓ, $2^\ell - 1$ coefficients need to be computed.

We use the notation

$$c_I = (-1)^{|I|}\sum_{k=1}^{n} w_k \prod_{i\in I} a_{ki}, \qquad (2)$$

where $I\subseteq M=\{1,\dots,m\}$. The satisfiability problem has the following *polynomial representation*:

(PR) find $x\in\{-1,1\}^m$ such that

$$\mathscr{P}(x) = \sum_{k=1}^{n} w_k + \sum_{I\subseteq M} c_I \prod_{i\in I} x_i = 0.$$

Observe that by construction $\mathscr{P}(x)\geqslant 0$ for any $x\in\{-1,1\}^m$. Strict inequality implies that the corresponding CNF formula is unsatisfiable. Note that the coefficients c_I are functions of the weights w_k; thus (PR) changes when the weights are adjusted. In the next sections it is shown that this allows us to look for a set of weights such that a formula or subformula can be concluded to be polynomially solvable or even unsatisfiable.

In this paper we also make use of propositional formulas in conjunction of equivalencies form (CoEs).

Such formulas are also known as XOR–SAT formulas, which were shown to be solvable in polynomial time by Schaefer [8] (see also [11]), as opposed to formulas in CNF which are in general NP-complete [2]. An XOR–SAT formula can be represented making use of additive representations in $\mathbb{Z}_2$; however, the multiplicative representation allows us to recognize CoE c.q. XOR–SAT formulas by their CNF representation in a natural way. In the next section we briefly review a polynomial-time algorithm for CoE formulas.

A CoE formula is a conjunction of *equivalency clauses*. An equivalency clause $\boldsymbol{Q}_k$ is defined as a (nested) equivalency of literals or its negation. We denote this as

$$\boldsymbol{Q}_k = [\neg]_k \biguplus_{i \in I_k} p_i, \tag{3}$$

where the square brackets denote the optionality of the negation operator. Observe that the polynomial representation of $\boldsymbol{Q}_k$ is very short:

$$Q_k(x) = \delta_k \prod_{i \in I_k} x_i = 1, \tag{4}$$

where $\delta_k = -1$ if the negation operator is present in Eq. (3), otherwise $\delta_k = 1$. This representation is obtained by directly considering Eq. (3); an equivalent representation is obtained by first translating $\boldsymbol{Q}_k$ to CNF, and then summing the $2^{|I_k|-1}$ associated polynomial representations (1). Conversely, it is easy to see that to any equation of type (4) an equivalency clause is associated. For example, if $\boldsymbol{Q}_k = \neg(p_1 \leftrightarrow p_4 \leftrightarrow p_8)$, then $Q_k(x) = -x_1x_4x_8 = 1$ and vice versa. Note that the CNF representation of $\boldsymbol{Q}_k$ is given by

$$(\neg p_1 \vee \neg p_4 \vee \neg p_8) \wedge (p_1 \vee p_4 \vee \neg p_8)$$
$$\wedge (p_1 \vee \neg p_4 \vee p_8) \wedge (\neg p_1 \vee p_4 \vee p_8). \tag{5}$$

The reader may want to verify that by taking the sum of the polynomial representations of these clauses, indeed a representation equivalent to $Q_k(x)$ is obtained.

3. Balanced polynomial representations

In this section we discuss a notion of balancedness for SAT formulas, based on the polynomial representation (PR). The notions discussed here were earlier introduced in [11]. Let us start with a definition.

Definition 1. Consider the polynomial representation (*PR*). We call the polynomial function $\mathscr{P}(x)$ *balanced* if

$$\sum_{I \subseteq M} |c_I| = \sum_{k=1}^{n} w_k.$$

Furthermore, $\mathscr{P}(x)$ is called *(strictly) positive* if

$$\sum_{I \subseteq M} |c_I| < \sum_{k=1}^{n} w_k.$$

Now assume we are given a SAT formula Φ and its polynomial representation (PR). If $\mathscr{P}(x)$ is balanced, we say that Φ has a *balanced polynomial representation* (BPR). Similarly, if $\mathscr{P}(x)$ is positive, we say that Φ has a *positive polynomial representation* (PPR). In the latter case Φ is unsatisfiable [11].

We have the following lemma.

Lemma 1. If Φ has a balanced polynomial representation, it is equivalent to a CoE formula.

Proof. Observe that if $\mathscr{P}(x)$ is balanced, then for any feasible vector $x \in \{-1,1\}^m$ it must hold that

$$c_I \prod_{i \in I} x_i = -|c_I|,$$

for all $I \subseteq M$. This implies that we may set c_I to $\operatorname{sgn}(c_I)$, thus obtaining an equation of the form (4). □

Let us now review a polynomial time algorithm for solving CoE formulas, which (implicitly) yields all satisfiable solutions. It may be noted that this algorithm is equivalent to Gaussian elimination in $\mathbb{Z}_2$ [8]. We only give the outline here, for a more detailed description the reader is referred to [11].

Consider an equivalency-clause $\boldsymbol{Q}_k$ and its polynomial representation $Q_k(x)$ (4). Obviously, for any feasible solution $x \in \{-1,1\}^m$ it holds that

$$x_j = \delta_k \prod_{i \in I_k \setminus j} x_i \quad \text{for all } j \in I_k.$$

Choosing an index $j \in I_k$ we can substitute the above expression in all equivalency-clauses $\boldsymbol{Q}_l$ $(l \neq k)$ in which x_j occurs, using that $x_i^2 = 1$. Thus all but one occurrence of x_j are eliminated. Now the algorithm runs as follows. We initialize the set $\mathscr{I} = \{x_1, \ldots, x_m\}$, the set of *independent* variables. We loop through the equivalency clauses once, choosing a variable x_j in each one to eliminate from all other equivalency clauses. Subsequently we remove x_j from $\mathscr{I}$, and call it a *dependent* variable. Thus we end up with a set of equivalency clauses for which all satisfiable assignments can be constructed by assigning all possible combinations of truth values to the independent variables. The values of the dependent variables are uniquely determined by an assignment to the independent variables. Note that during the elimination process the equality $-1=1$ might be derived; obviously, this implies that the formula under consideration is a contradiction. Here is a small example.

Example. A balanced polynomial representation is given by

$$\mathscr{P}(x) = 7 - 2x_1x_2x_3 + x_1x_3x_5 - 3x_2x_4x_5 - x_1x_4x_5.$$

After executing the algorithm an equivalent representation is obtained:

$$\mathscr{P}^*(x) = 4 + x_1x_5 + x_2x_5 + x_4 - x_3,$$

with $\mathscr{I} = \{x_5\}$. Thus two distinct solutions can be constructed.

If a formula has a CoE *sub*formula, solving this first may be of help in solving the full formula, since it allows us to take dependencies into account in a systematic way. When solving the full formula the search can possibly be restricted to the independent variables. Moreover, the CoE subformula might be a contradiction, implying that the full formula is also unsatisfiable.

4. Polynomial time recognition of CoE subformulas

Let us now address the problem of recognizing a CoE subformula. We can make use of a linear programming (LP) formulation to find a CoE subformula of maximal weight. Since the construction of the LP can be done in polynomial time (assuming that the maximal clause length is bounded and fixed), and LP problems are polynomially solvable [7], the recognition problem can be solved in polynomial time.

In the formulation the weights w_k occurring in the polynomial representation (PR) are the main decision variables. Essentially, we want to find a set of nonnegative weights w_k and a *slack* $s \geqslant 0$ such that (see Definition 1 and Eq. (2))

$$\sum_{I \subseteq M} \left| \sum_{k=1}^{n} \left(\prod_{i \in I} a_{ki} \right) w_k \right| + s = \sum_{k=1}^{n} w_k. \tag{6}$$

We allow the weights to be equal to zero; if $w_k = 0$ for some k, this implies that clause k is not in the subformula, while if $w_k > 0$ clause k is in the subformula. Our first goal should be to find a solution with s strictly positive (since then the associated subformula has PPR and is unsatisfiable); if no such solution exists, the goal is to identify a subformula of maximal weight with BPR. To check whether solutions with the desired properties exist, we first solve an LP with the objective of maximizing s, and if the optimal value of this LP is equal to zero, a second LP must be solved with the objective to maximize the sum of the weights. Consider the following LP.

$$\text{(LP)} \quad \begin{aligned} &\max \ \alpha s + \beta \sum_{k=1}^{n} w_k \\ &\text{s.t.} \ \sum_{I \subseteq M} (z_I^+ + z_I^-) - \sum_{k=1}^{n} w_k + s = 0, \\ &\qquad \sum_{k=1}^{n} \left(\prod_{i \in I} a_{ki} \right) w_k - z_I^+ + z_I^- = 0, \ I \subseteq M, \\ &\qquad 0 \leqslant w_k \leqslant 1, \quad 1 \leqslant k \leqslant n, \\ &\qquad z_I^+,\ z_I^- \geqslant 0, \quad I \subseteq M, \\ &\qquad s \geqslant 0. \end{aligned}$$

The two separate LPs are obtained by setting $\beta = 0$ and $\alpha = 0$, $s = 0$, respectively. The first constraint evaluates expression (6) and in the subsequent set of constraints the c_I are computed (see Eq. (2)). The auxiliary variables z_I^+ and z_I^- associated with the (nonempty) set I are used to eliminate the absolute values in Eq. (6) in the usual way. For a formula in which the clauses have a maximum length ℓ, the numbers of variables and constraints are bounded by $(2^{\ell+1} - 1)n + 1$ and $(2^{\ell} - 1)n + 1$, respectively.

Observe that if the optimal value of the first LP is equal to zero, no subformula with PPR exists. If the optimal value is positive, the subformula induced by the positive weights in the optimal solution is unsatisfiable. Obviously, the existence of a subformula with PPR is merely a sufficient condition for a formula to be contradictory. If the optimal value of the second

LP equals zero, no CoE subformula exists. For random instances this will usually be the case. On the other hand, instances that stem from some practical application often have a lot of structure that can be utilized via this LP approach. If the LP has a positive optimal value, the CoE subformula consists of the equivalency clauses associated with the sets I for which $c_I = z_I^+ - z_I^- \neq 0$, with $\delta_I = \text{sgn}(-c_I)$.

Note that a CoE subformula of maximal weight is not guaranteed to be a subformula of maximal *size*. In particular, if a CNF formula contains only clause–disjoint CoE subformulas, the LP approach will identify the maximal size CoE subformula (i.e. the union of the clause disjoint CoE subformulas). If however some of the subformulas are not clause disjoint, then the maximal weight CoE subformula does not necessarily coincide with the maximal size subformula. In this respect using an interior point method for solving (LP) might be better than the simplex method, since an IPM yields an optimal solution with a maximal number of nonzero variables.

In practice, heuristics that look for particular structures may often succeed in identifying CoE subformulas. Indeed, for the parity formulas solved in this paper such heuristics suffice. The heuristic we used was simply to look for 'blocks of clauses' with a structure similar to that of Eq. (5) (see for the outline of a local search approach [11]). However, if a subformula is 'well hidden', or does not conform this standard structure, using the LP approach described above will succeed in identifying it, whereas the heuristic methods are likely to fail.

5. A DPLL algorithm for solving mixed CNF/CoE formulas

One of the best known exact algorithms for solving CNF formulas is the variant of the Davis–Putnam algorithm [5] introduced by Davis et al. [4], which is known as the DPLL algorithm. The DPLL algorithm implicitly enumerates all 2^m distinct solutions by setting up a binary search tree. We can easily extend this algorithm to solve conjunctions of CNF and CoE formulas. In Fig. 1 the extension of the algorithm is summarized.

Let us look a bit more closely at the algorithm. First we consider the unit resolution phase. When a unit literal is propagated through the formula, some clauses become true, while others reduce in length by one. For equivalency clauses it holds that each in which the current unit literal occurs simply reduces in length by one. As usual, unit resolution is applied until no unit clauses remain, where it is noted that an equivalency clause of length one can be regarded as a unit clause in the usual sense. After the unit resolution phase it is checked whether the current formula can be declared either satisfiable or contradictory. If not, a *branching* or *splitting* variable l is chosen in some pre-specified way and the DPLL procedure is recursively called with this variable set to true and false, respectively. Note that if a set $\mathcal{I}$ of independent variables is specified, it appears to be sensible to restrict the set of candidate branching variables to $\mathcal{I}$; then the dependent variables are only considered in the unit resolution phase.

```
procedure DPLL ($\Phi = \Phi_{CNF} \cup \Phi_{CoE}$, depth);
  $\Phi$:=unit_resolution($\Phi$);
  if $\Phi = \emptyset$ then
    $\Phi$ is satisfiable: return(satisfiable)
  if $C_k = \emptyset$ for a $C_k \in \Phi_{CNF}$ then
    $\Phi$ is contradictory: backtrack.
  if $Q_k = false$ for a $Q_k \in \Phi_{CoE}$ then
    $\Phi$ is contradictory: backtrack.
  $l$:=branch_rule($\Phi$);
  DPLL($\Phi \cup \{l\}$, depth+1);
  DPLL($\Phi \cup \{\neg l\}$, depth+1);
return(unsatisfiable)
```

Fig. 1. The DPLL algorithm extended for CNF/CoE formulas.

6. Solving the DIMACS parity instances

We apply the techniques that we discussed previously to solve the DIMACS par*-*-c.cnf instances. These instances all contain a subformula with balanced polynomial representation. This subformula is a CNF translation of a CoE formula in which all equivalency clauses have length three. It is not strictly necessary to apply the LP approach to identify this formula, since it can be easily found by inspection. For completeness, we list the required time for constructing and solving the LPs in Table 1. These tests were run on

Table 1
Results of using the LP approach for identifying CoE subformulas

Instance	n	Row	Col	Time	Opt
par8-1-c.cnf	254	282	818	0.16	224
par8-2-c.cnf	270	301	872	0.18	240
par8-3-c.cnf	298	338	974	0.19	268
par8-4-c.cnf	266	297	860	0.17	236
par8-5-c.cnf	298	335	968	0.19	268
par16-1-c.cnf	1264	1537	4338	1.37	1080
par16-2-c.cnf	1392	1692	4776	1.60	1208
par16-3-c.cnf	1332	1619	4570	1.75	1148
par16-4-c.cnf	1292	1567	4426	1.51	1108
par16-5-c.cnf	1360	1653	4666	1.81	1176
par32-1-c.cnf	5254	6524	18302	16.84	4632
par32-2-c.cnf	5206	6466	18138	16.08	4584
par32-3-c.cnf	5294	6574	18442	17.20	4672
par32-4-c.cnf	5326	6618	18562	15.12	4704
par32-5-c.cnf	5350	6648	18646	16.18	4728

a HP9000/C200 workstation, 200 MHz. CPLEX was used to solve the LPs, using the barrier algorithm. Since the CoE subformulas are clause disjoint, the maximal size CoE subformula is identified by the LP approach. In Table 1 are listed, for each instance, the number of clauses n, the number of rows row and columns col in the corresponding LP, the time for constructing and solving the LP, and the value of the optimal solution (opt). By construction it holds that $n + 2 * row = col$; furthermore, due to the particular structure of the instances (cf. Eq. (5)), the number of equivalency clauses k induced by the optimal solution is equal to opt/4.

The first and second phase of the algorithm were implemented in C and compiled using gcc with the flag -O2 set. The results reported in Tables 2 and 3 were obtained running the code on a SGI Power Challenge with a 200 MHz R10k processor. All times reported are in seconds. In Table 2 we report on the results of the first phase of the algorithm which consists of isolating (by inspection; this requires less than 0.01 s) and solving the CoE subformulas. The initial numbers of variables and clauses are given by m and n. The number of equivalency clauses in the CoE subformula is denoted by k; note that indeed $k = \text{opt}/4$, while the size of the remaining CNF is $n-$opt clauses. In the table we also indicate the number of independent variables determining the solutions of the CoE formula. The number of satisfying solutions for the CoE subformula equals $2^{|\mathscr{I}|}$.

Note that the CoE formula does not need to be solved separately for the modified DPLL algorithm to be valid. However, if it is solved, and subsequently it turns out that some dependent variable does not occur in the CNF part of the formula, this variable and the equivalency clause it occurs in need not be considered in the DPLL search procedure. So, if we have the choice between two variables p_i and p_j of which only p_i occurs in the CNF subformula as well, we choose to remove p_j from the set of independent variables. This allows us to reduce the problem size for phase two considerably. Moreover, on solving the CoE formula an inconsistency might be detected. For example, the dubois*.cnf and pret*.cnf instances, which are also in the DIMACS suite, are already found to be unsatisfiable in the first phase of our algorithm. These instances are fully equivalent to CoE formulas and thus solved in polynomial time [11].

Before starting the second phase of the algorithm we first remove as many dependent variables and equivalency clauses as possible. It may be noted that on branching strategies considering only the CNF subformula this has no effect as far as the node count is concerned; computation times however will reduce. The remaining numbers of variables, clauses and equivalency clauses are given by m, n and k. Note that $m = k + |\mathscr{I}|$; each dependent variable occurs in exactly one equivalency clause. We tested several branching strategies on the par16* instances, and used the one that appeared to be the best to solve the

Table 2
Results of the first phase of the algorithm

Instance	m	n	k	Time	$\|\mathscr{I}\|$
par8-1-c.cnf	64	254	56	0.01	8
par8-2-c.cnf	68	270	60	0.01	8
par8-3-c.cnf	75	298	67	0.01	8
par8-4-c.cnf	67	266	59	0.01	8
par8-5-c.cnf	75	298	67	0.01	8
par16-1-c.cnf	317	1264	270	0.04	47
par16-2-c.cnf	349	1392	302	0.06	47
par16-3-c.cnf	334	1332	287	0.05	47
par16-4-c.cnf	324	1292	277	0.06	47
par16-5-c.cnf	341	1360	294	0.06	47
par32-1-c.cnf	1315	5254	1158	4.49	157
par32-2-c.cnf	1303	5206	1146	3.80	157
par32-3-c.cnf	1325	5294	1168	4.50	157
par32-4-c.cnf	1333	5326	1176	4.39	157
par32-5-c.cnf	1339	5350	1182	4.62	157

Table 3
Results of the second phase of the algorithm

Instance	m	n	k	Nodes	Time	Nodes	Time
par8-1-c.cnf	31	30	23	3	0.00	1	0.00
par8-2-c.cnf	31	30	23	3	0.00	1	0.00
par8-3-c.cnf	31	30	23	2	0.00	1	0.00
par8-4-c.cnf	31	30	23	3	0.00	3	0.00
par8-5-c.cnf	31	30	23	4	0.00	4	0.00
par16-1-c.cnf	124	184	77	82	0.02	67	0.02
par16-2-c.cnf	124	184	77	58	0.01	144	0.03
par16-3-c.cnf	124	184	77	55	0.01	137	0.03
par16-4-c.cnf	124	184	77	51	0.01	131	0.03
par16-5-c.cnf	124	184	77	49	0.01	85	0.02
par32-1-c.cnf	375	622	218	410634	193	130258	62
par32-2-c.cnf	375	622	218	201699	90	335988	160
par32-3-c.cnf	375	622	218	502747	248	6712	3
par32-4-c.cnf	375	622	218	218021	101	267032	135
par32-5-c.cnf	375	622	218	179325	84	328253	164

larger instances. In Table 3 we report on the results. The branching strategy we arrived at is simply the *maximal occurrence in shortest clause* rule, with a lexicographic tie break, where the candidate branching variables are restricted to the set of independent variables. Note that for determining a branching variable the equivalency clauses are not considered. We report on the node counts obtained by first branching to l and $\neg l$, respectively. The node count gives the number of times that a branching variable was chosen. A typical phenomenon of DPLL algorithms that we also encountered here is that using different branching strategies the computation times and node counts may vary heavily.

Examining the tables we conclude that the smaller instances are solved in fractions of seconds, while the largest take at most about four minutes. To the best of our knowledge, none of the current state-of-the-art implementations of the DPLL procedure are capable of solving the par32∗ instances in less than 24 h, and often they require several days of computation time. Recently, it came to our attention that the instances were solved by an unspecified algorithm (‘GT6’) in 2–4 h [1].

The application of the techniques and notions described in this paper to more general SAT problems is the subject of further research.

Acknowledgements

We thank the anonymous referee whose suggestions contributed to the clarity and completeness of this paper.

References

[1] GT6 algorithm solves the extended DIMACS 32-bit parity problem. Note available from `http://www.research.att.com/~kautz/challenge/`, 1998.

[2] S.A. Cook, The complexity of theorem proving procedures, in: Proc. 3rd Ann. ACM Symp. on the Theory of Computing, 1971, pp. 151–158.

[3] J.M. Crawford, M.J. Kearns, R.E. Schapire, The minimal disagreement parity problem as a hard satisfiability problem, Draft version, 1995.

[4] M. Davis, M. Logemann, D. Loveland, A machine program for theorem proving, Commun. ACM 5 (1962) 394–397.

[5] M. Davis, H. Putnam, A computing procedure for quantification theory, J. ACM 7 (1960) 210–215.

[6] J. Gu, Global optimization for satisfiability (SAT) problem, IEEE Trans. Knowl. Data Eng. 6 (3) (1994) 361–381.

[7] N. Karmarkar, A new polynomial-time algorithm for linear programming, Combinatorica 4 (1984) 373–395.

[8] T.J. Schaefer, The complexity of satisfiability problems, Proc. 10th Symp. on the Theory of Computing, 1978, pp. 216–226.

[9] B. Selman, H. Kautz, D. McAllester, Ten challenges in propositional reasoning and search, Proc.15th Int. Joint Conf. on Artificial Intelligence, IJCAI-97, Nagoya, Aichi, Japan, 1997.

[10] M.A. Trick, Second DIMACS challenge test problems, in; D.S. Johnson, M.A. Trick (Eds.), Cliques, Coloring and Satisfiability: Second DIMACS implementation challenge, vol. 26 of DIMACS series in Discrete Mathematics and Computer Science, American Mathematical Society, 1996, Providence, RI, pp. 653–657.

[11] J.P. Warners, H. Van Maaren, Recognition of tractable satisfiability problems through balanced polynomial representations, Discrete Appl. Math., accepted for publication.