ELSEVIER

Discrete Applied Mathematics 99 (2000) 229–244

DISCRETE APPLIED MATHEMATICS



# balanced polynomial representations

Joost P. Warners[a,b,*,1], Hans van Maaren[a]

[a] *Department of Technical Mathematics and Informatics, Faculty of Information Technology and Systems, Delft University of Technology, P.O. Box 5031, 2600 GA Delft, Netherlands*
[b] *SEN2, CWI, P.O. Box 94079, 1090 GB, Amsterdam, Netherlands*

Received 10 September 1997; revised 30 June 1998; accepted 9 March 1999

**Abstract**

We consider a specific class of satisfiability (SAT) problems, the conjunctions of (nested) equivalencies (CoE). It is well known that CNF (conjunctive normal form) translations of CoE formulas are hard for branching and resolution algorithms. Tseitin proved that regular resolution requires a running time exponential in the size of the input. We review a polynomial time algorithm for solving CoE formulas, and address the problem of recognizing a CoE formula by its CNF representation. Making use of elliptic approximations of 3SAT problems, the so-called doubly balanced 3SAT formulas can be seen to be equivalent to CoE formulas. Subsequently, the notion of doubly balancedness is generalized by using polynomial representations of satisfiability problems, to obtain a general characterization of CoE formulas. We briefly address the problem of finding CoE subformulas, and finally the application of the developed theory to several DIMACS benchmarks is discussed. 

*Keywords*: Satisfiability; Conjunctive normal form; Polynomial functions; Polynomial-time algorithm

## 1. Introduction

The satisfiability problem of propositional logic (SAT) is the original NP-complete problem [6]. Thus, no algorithms are known that can solve each instance of SAT in polynomial time. However, there are various classes of specific SAT formulas for which polynomial time algorithms exist. For example, 2SAT formulas (i.e. formulas in which each clause contains at most two literals), and Horn formulas (in which each clause contains at most one positive literal) are solvable in linear time [1,7], while various generalizations of these classes are polynomially solvable as well [3–5,10].

* Corresponding author.
*E-mail address:* j.p.warners@twi.tudelft.nl (J.P. Warners)

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

0166-218X/00/$ - see front matter 
PII: S0166-218X(99)00135-3

In this paper we discuss another class of polynomially solvable SAT problems, the *conjunctions of* (*nested*) *equivalencies* (CoE), that arise from so-called *balanced* (*CNF*) *formulas* [8]. In the literature, CoE formulas are also known as XOR ('exclusive or') SAT formulas, which were shown to be polynomially solvable by Schaefer [15]. Balanced formulas are balanced with respect to the number of variable occurrences, and positive and negative occurrences of individual variables. The notion of *doubly balancedness* [13] is related to occurrences of pairs of variables. Doubly balanced formulas can be seen to be equivalent to CoE formulas by means of their *elliptic approximation*. Elliptic approximations of 3SAT formulas were introduced in [13,14,18]. By making use of polynomial representations of satisfiability problems such as used by Gu [11], a general characterization of CNF translations of CoE formulas is obtained. This involves *balanced polynomial representations*, a concept that we define later on.

Tseitin [16] introduced a specific kind of CoE formulas (in which each variable occurs exactly twice) to prove an exponential lower bound on the running time of regular resolution. In general, balanced formulas are hard for branching and other resolution-like algorithms [8]. We review a special purpose algorithm for solving CoE formulas. It turns out that several of the well-known DIMACS benchmarks [12] are in fact equivalent to CoE formulas and thus can be solved efficiently using this algorithm.

It may be noted that specific CoE formulas, namely those arising from a particular kind of doubly balanced formulas, can also be solved efficiently by making use of the notion of *symmetry* as introduced by Benhamou and Sais [2,9]. A CNF formula is said to contain symmetries if it remains invariant under a permutation of variable names. However, in general the CoE formulas need not be symmetric in this sense.

This paper is organized as follows. In the next section we introduce some notation and discuss the preliminaries. Subsequently, we review an algorithm for solving CoE formulas. In Section 4 we consider doubly balanced formulas and show that by their elliptic approximation these can be seen to be equivalent to CoE formulas. Section 5 is concerned with generalizing the notion of doubly balancedness by making use of polynomial representations of satisfiability problems. In the subsequent section we briefly address the problem of finding CoE (or unsatisfiable) *sub*formulas, and in the final section preliminary computational results on DIMACS benchmarks are given.

## 2. Preliminaries and notation

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

$$\Phi = \boldsymbol{C}_1 \wedge \boldsymbol{C}_2 \wedge \cdots \wedge \boldsymbol{C}_n = \bigwedge_{k=1}^{n} \boldsymbol{C}_k,$$

where each clause $\boldsymbol{C}_k$ is of the form

$$\boldsymbol{C}_k = \bigvee_{i \in I_k} p_i \vee \bigvee_{j \in J_k} \neg p_j$$

with $I_k, J_k \subseteq \{1, \ldots, m\}$ 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.

Let us associate a $\{-1, 1\}$-variable $x_i$ with each proposition letter $p_i$. Then a clause $\boldsymbol{C}_k$ may be written as a linear inequality in the following way:

$$C_k(x) = \sum_{i \in I_k} x_i - \sum_{j \in J_k} x_j \geqslant 2 - l(\boldsymbol{C}_k), \tag{1}$$

where $l(\boldsymbol{C}_k)$ denotes the length of clause $k$, i.e. $l(\boldsymbol{C}_k) = |I_k \cup J_k|$. Using matrix notation, the integer linear programming formulation of the satisfiability problem can be stated as

$$(IP_{\mathrm{SAT}}) \quad \text{find } x \in \{-1, 1\}^m \quad \text{such that } Ax \geqslant b,$$

where $A \in \mathbb{R}^{n \times m}$ and $b \in \mathbb{R}^n$. We have that $a_k^{\mathrm{T}} x = C_k(x)$, where by $a_k^{\mathrm{T}}$ we denote the $k$th row of the matrix $A$. Obviously, $a_{ki} = 1$ if $i \in I_k$, $a_{ki} = -1$ if $i \in J_k$, while $a_{ki} = 0$ for any $i \notin I_k \cup J_k$. Furthermore, $b_k = 2 - l(\boldsymbol{C}_k)$.

In an instance of 3-*satisfiability* (3SAT) all clauses have length equal to 3, so $b = -e$. By $e$ we denote the all-one vector of length $n$.

Let us now introduce the concept of CoE formulas, which are also known as XOR SAT formulas. Such formulas are solvable in polynomial time [15] as opposed to CNF formulas which are in general NP-complete [6]. In the next section a polynomial-time algorithm for CoE formulas is reviewed. A CoE formula $\Psi$ is the conjunction of $t$ equivalency-clauses, where each equivalency-clause $\boldsymbol{Q}_k$ is a (nested) equivalency of literals or its negation. In the first case we refer to $\boldsymbol{Q}_k$ as a *positive equivalency-clause*, otherwise it is called a *negative equivalency-clause*. An example of a (negative) equivalency-clause is $\neg(p_2 \leftrightarrow p_4 \leftrightarrow p_7)$ which is true if either one or all three of its variables are false. An equivalency-clause is denoted as

$$\boldsymbol{Q}_k = [\neg] \leftrightarrow\!\!\!\!\!\bigcup_{i \in S_k} p_i, \tag{2}$$

where the square brackets indicate the optionality of the negation operator. Note that for a positive equivalency-clause to be true, an *even* number of variables must be false, while for a negative one an *odd* number of variables must be false. In the following we associate an indicator $\delta(k)$ with an equivalency-clause $\boldsymbol{Q}_k$, and let $\delta(k) = 1(-1)$ if $\boldsymbol{Q}_k$ is a positive (negative) equivalency-clause.

## 3. Solving conjunctions of equivalencies

Let us now review an algorithm for solving CoE formulas, which is in fact Gaussian elimination in $\mathbb{Z}_2$ [15]. Suppose we are given a CoE formula $\Psi$. Obviously, for any

satisfying assignment it holds that

$$p_j \leftrightarrow \left( \underset{i \in S_k \setminus \{j\}}{\leftrightarrow} p_i \right), \tag{3}$$

for a positive equivalency-clause $\boldsymbol{Q}_k$, $1 \leqslant k \leqslant t$, and any $j \in S_k$. If $\boldsymbol{Q}_k$ is a negative equivalency-clause, the right-hand side of (3) must be negated. This observation can be used to obtain a polynomial-time algorithm. All but one occurrences of $p_j$ can be eliminated by making use of equivalency (3), to obtain a formula $\Psi'$ that is equivalent to formula $\Psi$. Note that the value of $p_j$ is uniquely determined by the values of the other variables in $\boldsymbol{Q}_k$. We call such a variable $p_j$ a *dependent* variable. Initially, all the variables are said to be independent and contained in the set of independent variables $\mathscr{I}$. Let us use the notation

$$S_k \oplus S_l = (S_k \cup S_l) \setminus (S_k \cap S_l).$$

This is convenient, since when we use expression (3) to eliminate $p_j$ from equivalency-clause $\boldsymbol{Q}_l$ we obtain

$$p_j \leftrightarrow \left( \underset{i \in S_l \setminus \{j\}}{\leftrightarrow} p_i \right) \equiv \left( \underset{i \in S_k \setminus \{j\}}{\leftrightarrow} p_i \right) \leftrightarrow \left( \underset{i \in S_l \setminus \{j\}}{\leftrightarrow} p_i \right) \equiv \underset{i \in S_k \oplus S_l}{\leftrightarrow} p_i.$$

Note that if $S_k = \{j\}$, this substitution performs unit resolution; the length of equivalency-clause $\boldsymbol{Q}_l$ then reduces by one. If $S_k \oplus S_l = \emptyset$, while $\delta(k) = -\delta(l)$ an inconsistency is detected, implying that the formula under consideration is unsatisfiable. If no inconsistency is detected, the algorithm terminates when the CoE formula is rewritten to a form where each equivalency-clause contains exactly one dependent variable. After termination of the algorithm, *all* satisfiable solutions can be constructed by assigning all possible combinations of truth values to the independent variables. Algorithm SOLVE_CoE is summarized in Fig. 1. In the outer loop each equivalency-clause is considered at most once; after it is considered it is labelled. Note that at the end of each outer loop, unlabelled clauses consist of only independent variables, while any labelled clause either contains a dependent variable or is empty. The algorithm returns that $\Psi$ is a contradiction, or the rewritten formula $\Psi$ and the set of independent variables $\mathscr{I}$ using which all satisfying assignments can be constructed.

**Example 1.** Let the formula $\Psi$ be given by

$$\Psi = (p_1 \leftrightarrow p_2 \leftrightarrow p_3) \wedge (p_2 \leftrightarrow p_3 \leftrightarrow p_4) \wedge \neg(p_1 \leftrightarrow p_3 \leftrightarrow p_4).$$

Initializing, we have $\mathscr{I} = \{p_1, p_2, p_3, p_4\}$ and all clauses are unlabelled.

**Iteration 1.** We choose $l = 1$, $j = 1$. For $k = 3$, $j \in S_k$, so we carry out the substitution. We have that $S_1 \oplus S_3 = \{2,4\}$ and $\delta(1)\,\delta(3) = -1$, so

$$\Psi := (p_1 \leftrightarrow p_2 \leftrightarrow p_3) \wedge (p_2 \leftrightarrow p_3 \leftrightarrow p_4) \wedge \neg(p_2 \leftrightarrow p_4)$$

and $\mathscr{I} = \{p_2, p_3, p_4\}$.

```
Initialize the set 𝒥 = {p_1, ..., p_m}.
while not all clauses are labelled do
  choose an unlabelled clause Q_l
  choose a variable p_j ∈ S_l
  𝒥 := 𝒥\{p_j}
  for k = 1 to t do
    if j ∈ S_k and k ≠ l do
      S_k := S_k ⊕ S_l;  δ(k) := δ(k)δ(l)
      if  S_k = ∅ then
        if  δ(k) = −1 return contradiction
        else label clause Q_k
      endif
    endif
  endfor
  label clause Q_l
endwhile
return satisfiable, 𝒥, Ψ
```

Fig. 1. Algorithm SOLVE-CoE.

**Iteration 2.** Now let us choose $l=2$, $j=2$ thus $\mathscr{J}:=\{p_3, p_4\}$. Performing the substitution for $k=1$ and $k=3$ we find

$$\Psi := (p_1 \leftrightarrow p_4) \wedge (p_2 \leftrightarrow p_3 \leftrightarrow p_4) \wedge \neg p_3.$$

**Iteration 3.** Finally, $l=3$, $j=3$, so $\mathscr{J}:=\{p_4\}$. We find

$$\Psi := (p_1 \leftrightarrow p_4) \wedge \neg(p_2 \leftrightarrow p_4) \wedge \neg p_3.$$

The single independent variable $p_4$ can be arbitrarily set, and substituting it in $\Psi$ we can construct two satisfying assignments, corresponding to $p_4$ and $\neg p_4$, respectively, $(p_1 \wedge \neg p_2 \wedge \neg p_3 \wedge p_4)$ and $(\neg p_1 \wedge p_2 \wedge \neg p_3 \wedge \neg p_4)$.

Let us consider the complexity of the algorithm. It requires $\min\{m,t\}$ iterations, since each time the outer loop is executed, the number of independent variables and the number of unlabelled clauses decrease. In each iteration all equivalency-clause are considered once, and the length of these clauses is bounded by $m$. Thus we have the following complexity bound.

**Lemma 2.** *The algorithm runs in* $\mathcal{O}(mt \cdot \min\{m,t\})$ *time.*

We conclude that this algorithm solves CoE formulas in polynomial time. Let us now turn to the issue of recognizing CoE formulas by their CNF translation.

## 4. A special case: doubly balanced formulas

In this section we restrict ourselves to 3SAT formulas. We consider a specific class of 3SAT CNF formulas and show that these are equivalent to CoE formulas. Let us consider the CNF translation of an equivalency-clause of length 3, say $p \leftrightarrow q \leftrightarrow r$. Then

$$\Phi = (p \vee \neg q \vee \neg r) \wedge (\neg p \vee q \vee \neg r) \wedge (\neg p \vee \neg q \vee r) \wedge (p \vee q \vee r).$$

Note the particular structure of this CNF formula. A formula with this structure is called *doubly balanced* [8,13]. Doubly balanced formulas are defined as follows.

**Definition 3.** A 3SAT formula $\Phi$ is called *doubly balanced* if

(i) Of each proposition $p_i$, the number of unnegated and negated occurrences are equal, or equivalently

$$\sum_{k=1}^{n} a_{ki} = 0 \quad \text{for all } i = 1, \ldots, m.$$

(ii) For any two propositions $p_i$ and $p_j$, the number of clauses in which both appear simultaneously with the same sign (i.e. both negated or both unnegated) is equal to the number of clauses in which both appear simultaneously with opposite signs (i.e. one appears negated and the other unnegated), or equivalently

$$\sum_{k=1}^{n} a_{ki} a_{kj} = 0 \quad \text{for all } i, j = 1, \ldots, m, i \neq j.$$

It appears that doubly balanced formulas are equivalent to CoE formulas. That this is indeed the case is the main result of the present section. Let us state it as a theorem.

**Theorem 4.** *A doubly balanced formula is equivalent to a CoE formula and as such can be solved in polynomial time.*

The remainder of this section is concerned with proving this theorem. In the proof we make use of elliptic approximations of satisfiability problems, which were earlier introduced in [13,14,18].

**Lemma 5.** *Let $\Phi$ be a 3SAT-formula with integer linear description ($IP_{SAT}$). The ellipsoid*

$$\mathscr{E} = \{x \in \mathbb{R}^m \mid x^{\mathrm{T}} A^{\mathrm{T}} A x - 2e^{\mathrm{T}} A x \leqslant 3n\}$$

*contains all satisfying assignments of $\Phi$.*

**Proof.** For each clause $k$ it holds that $C_k(x) \in \{-3, -1, 1, 3\}$, for all $x \in \{-1, 1\}^m$. The clause is satisfied if and only if $C_k(x) \in \{-1, 1, 3\}$. Thus $x \in \{-1, 1\}^m$ is a satisfiable assignment if and only if $Ax \in \{-1, 1, 3\}^n$. Subtracting $e$ on both sides and taking

squares, we find that for any satisfiable assignment it holds that $(Ax-e)^{\mathrm{T}}(Ax-e)\leqslant 4n$. Expanding this product yields the desired result, noting that $e^{\mathrm{T}}e=n$. □

Considering the ellipsoid, it is clear that certain contradictory assignments $x\in\{-1,1\}^m$ may also be contained in it; therefore we speak of an *approximation*. In general most contradictory assignments are not contained in the ellipsoid, and thus using specific properties of the ellipsoid such as its eigenvalue structure and its center, effective branching rules and satisfiability tests are obtained; see [14,18].

If the ellipsoid is *normalized and centered*, i.e. it is centered at the origin and its axes are parallel to the unit vectors, these heuristics do not distinguish between the variables and thus are of no use. However, the structure of such ellipsoids can be exploited in a different way. Note that for these ellipsoids $A^{\mathrm{T}}A$ is diagonal, and $A^{\mathrm{T}}e\equiv 0$. Let us first establish the following easy lemma.

**Lemma 6.** *$\Phi$ is doubly balanced if and only if $A^{\mathrm{T}}A$ is a diagonal matrix and $A^{\mathrm{T}}e\equiv 0$.*

**Proof.** It follows straightforwardly from Definition 3. □

We conclude that the elliptic approximation of a doubly balanced formula is normalized and centered. Let us denote by $\alpha_i$ the number of occurrences of the proposition $p_i$ in a formula $\Phi$, in other words, $\alpha_i$ denotes the number of clauses in which $p_i$ appears. Normalized, centered ellipsoids have the following property.

**Lemma 7.** *If the ellipsoid $\mathscr{E}$ is normalized and centered, then any vector $x\in\{-1,1\}^m$ lies on its boundary.*

**Proof.** Using that $A^{\mathrm{T}}A$ is diagonal, $A^{\mathrm{T}}e\equiv 0$ and $x_i^2=1$ for $x_i\in\{-1,1\}$, we find that

$$x^{\mathrm{T}}A^{\mathrm{T}}Ax-2e^{\mathrm{T}}Ax=\sum_{i=1}^{m}\alpha_i x_i^2=\sum_{i=1}^{m}\alpha_i=3n.$$

The last equality follows from the fact that the total number of variable occurrences in a 3SAT formula is equal to three times the number of clauses. □

We have the following corollary.

**Corollary 8.** *If all assignments lie on the boundary of $\mathscr{E}$, then for any satisfying assignment $x\in\{-1,1\}^m$, $a_k^{\mathrm{T}}x=C_k(x)$ equals either $-1$ or 3 for each $k=1,\ldots,n$.*

**Proof.** First note that for any satisfying assignment $C_k(x)\in\{-1,1,3\}$. Next, assume that for a satisfying assignment $a_k^{\mathrm{T}}x=C_k(x)=1$, for some $k$. Then $(Ax-e)^{\mathrm{T}}(Ax-e)\leqslant 4(n-1)$, implying that $x^{\mathrm{T}}A^{\mathrm{T}}Ax-2e^{\mathrm{T}}Ax<3n$, thus arriving at a contradiction. □

So for each clause a satisfying assignment either satisfies *exactly one* of its literals or *all three*. This implies that each clause $k$ can be regarded as a nested equivalence of its three literals. Let us state this in a lemma.

Table 1
Proof of Lemma 9

| $p$ | $q$ | $r$ | $p \leftrightarrow q \leftrightarrow r$ | $\neg(p \leftrightarrow q \leftrightarrow r)$ | $\lvert J_k\rvert = 0$ | $\lvert J_k\rvert = 1$ | $\lvert J_k\rvert = 2$ | $\lvert J_k\rvert = 3$ |
|---|---|---|---|---|---|---|---|---|
| T | T | T | T | F | T | F | T | F |
| T | T | F | F | T | F | T | F | T |
| T | F | T | F | T | F | T | F | T |
| T | F | F | T | F | T | F | T | F |
| F | T | T | F | T | F | T | F | T |
| F | T | F | T | F | T | F | T | F |
| F | F | T | T | F | T | F | T | F |
| F | F | F | F | T | F | T | F | T |

**Lemma 9.** *The requirement that of a clause of length three $\boldsymbol{C}_k$ either one or all three literals are satisfied is expressed by the equivalency-clause*

$$\boldsymbol{Q}_k = [\neg] \leftrightarrow\!\!\!\!\!\!\bigcup_{i \in S_k} p_i,$$

*where $S_k = I_k \cup J_k$ and the negation operator is present (and $\delta(k) = -1$) if and only if $|J_k|$ is odd.*

**Proof.** The correctness of the lemma is verified by the truth table in Table 1. The first three columns show all possible truth valuations of the three variables involved (denoted by $p$, $q$, $r$). The subsequent two columns indicate whether the associated equivalency-clauses under the given assignments evaluate to true (T) or false (F). The last four columns indicate whether the associated clauses with $|J_k| = 0, 1, 2, 3$ negative literals and the additional requirement that exactly one or all three literals are true, evaluate to true (T) or false (F). □

So we conclude that a CoE formula can be constructed that is fully equivalent to the original doubly balanced CNF formula, thus completing the proof of Theorem 4. Let us return for a moment to the example of the beginning of this section. The CNF formula $\Phi$ is doubly balanced and so (by Lemma 9) it is equivalent to

$$\Psi = (p \leftrightarrow q \leftrightarrow r) \wedge (p \leftrightarrow q \leftrightarrow r) \wedge (p \leftrightarrow q \leftrightarrow r) \wedge (p \leftrightarrow q \leftrightarrow r).$$

Thus, indeed, $\Psi = p \leftrightarrow q \leftrightarrow r$.

Note that the complexity of constructing the elliptic approximation is linear in the number of clauses, and thus checking whether a formula is doubly balanced can be done efficiently. In the next section we address the issue of recognizing CoE formulas in CNF formulas in general (i.e. not restricted to 3SAT and doubly balancedness).

## 5. A general characterization of CoE formulas

In the previous section we have only considered 3SAT and its elliptic approximation, as a special class of CNF formulas that are equivalent to CoE formulas. The

ellipsoid is in fact the second-order Taylor truncation of a full representation [14] of satisfiability problems. We now investigate full polynomial representations of general CNF formulas which enable us to give a *general characterization* of CNF translations of CoE formulas. Similar polynomial representations are used by (among others) Gu [11], to obtain efficient approximation algorithms for large satisfiability problems.

Let us first consider a polynomial representation of a CoE formula. Recall that with each proposition letter $p_i$ a $\{-1,1\}$-variable $x_i$ is associated. It holds that the equivalency-clause $\boldsymbol{Q}_k$ is satisfied if and only if (see (2))

$$Q_k(x) = \delta(k) \prod_{i \in S_k} x_i = 1, \tag{4}$$

since this equation is satisfied if and only if $\delta(k)=1$ $(-1)$ and an even (odd) number of $x_i$, $i \in S_k$, variables is equal to $-1$. Note that for any assignment $x \in \{-1,1\}^m$, $Q_k(x)= \pm 1$. Thus we have a concise alternative formulation of $\Psi$.

$$\text{(CoE)} \quad \text{find } x \in \{-1,1\}^m \quad \text{such that } \sum_{k=1}^{t} Q_k(x) = \sum_{k=1}^{t} \delta(k) \prod_{i \in S_k} x_i = t.$$

Conversely, it is easy to see that any problem of the form (CoE) can be directly translated into a CoE formula. Let us now derive a condition under which a CNF formula $\Phi$ can be reduced to the form (CoE) (and thus is polynomially solvable).

Let $\Phi$ be a SAT formula consisting of $n$ clauses and $m$ variables. Consider a clause and associated linear inequality as given by (1), and the integer linear description ($IP_{\mathrm{SAT}}$). A $\{-1,1\}$-vector $x$ satisfies (1) if and only if

$$P_k(x) = \prod_{i \in I_k} (1 - x_i) \prod_{j \in J_k} (1 + x_j) = \prod_{i=1}^{m} (1 - a_{ki} x_i) = 0. \tag{5}$$

If $x$ is not a satisfying assignment it holds that $P_k(x) = 2^{|I_k \cup J_k|} > 0$. Denote $M = \{1, \ldots, m\}$. In general, $x \in \{-1,1\}^m$ is a satisfiable assignment of a formula $\Phi$, if and only if

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

where in principal $I$ runs through all possible subsets of $M$ $(I \neq \emptyset)$. 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 $k = 1, \ldots, n$ need to be considered. In general, for a clause with length $l(\boldsymbol{C}_k)$, $2^{l(\boldsymbol{C}_k)} - 1$ coefficients need to be computed.

Let us use the notation

$$c_I = (-1)^{|I|} \sum_{k=1}^{n} \prod_{i \in I} a_{ki}, \tag{6}$$

where $I \subseteq M$. Then the satisfiability problem has the following *polynomial representation*.

$$\text{(PR)} \quad \text{find } x \in \{-1,1\}^m \quad \text{such that } \mathscr{P}(x) = n + \sum_{I \subseteq M} c_I \prod_{i \in I} x_i = 0.$$

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.

Now we can generalize the notion of doubly balancedness (that is restricted to 3SAT) to a notion of balancedness for general SAT formulas. Let us give a definition.

**Definition 10.** Consider the polynomial representation (PR). We call the polynomial function $\mathscr{P}(x)$ *balanced* if

$$\sum_{I \subseteq M} |c_I| = n. \tag{7}$$

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

$$\sum_{I \subseteq M} |c_I| < n. \tag{8}$$

Assume we are given a SAT formula $\Phi$ and its polynomial representation (PR). If $\mathscr{P}(x)$ is balanced, we say that $\Phi$ has a *balanced polynomial representation*. Similarly, if $\mathscr{P}(x)$ is positive, we say that $\Phi$ has a *positive polynomial representation*.

We have the following lemma.

**Lemma 11.** *If $\Phi$ has a balanced polynomial representation, it is equivalent to a conjunction of equivalencies.*

**Proof.** We need to show that for balanced polynomials $\mathscr{P}(x)$, (PR) (and hence $\Phi$) can be reduced to the form (CoE). Note 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 for all subsets $I \subseteq M$ for which $c_I \neq 0$,

$$\prod_{i \in I} x_i = \begin{cases} 1 & \text{if } c_I < 0, \\ -1 & \text{if } c_I > 0. \end{cases}$$

Enumerating the $k = 1, \ldots, t \leqslant n$ sets $I \subseteq M$ for which $c_I \neq 0$, we take $\delta(k) = -\operatorname{sgn}(c_I)$ and $S_k = I$ to prove the lemma. □

Let us now state a theorem.

**Theorem 12.** *Given are a propositional formula $\Phi$ and its polynomial representation (PR).*

- *If $\Phi$ has a positive polynomial representation, it is unsatisfiable.*
- *If $\Phi$ has a balanced polynomial representation, it is equivalent to a CoE formula and can be solved in $\mathcal{O}(mn \cdot \min\{m,n\})$ time, using algorithm* SOLVE_CoE.

- *More generally, using a modified version of algorithm* SOLVE_COE, *if*

$$\sum_{I \subseteq M} |c_I| = n + 2z, \tag{9}$$

*a satisfying solution, or proof that no solution exists, can be found in*

$$\mathcal{O}\left(\binom{n+2z}{z} mn \cdot \min\{m,n\}\right)$$

*time.*

**Proof.** The first statement follows from the fact that $\mathscr{P}(x) > 0$ for any $x \in \{-1,1\}^m$; the second statement is clear from Lemma 11. Let us now consider the third statement. Assume that all nonzero coefficients $c_I$ equal $\pm 1$, and consider (PR). For any satisfying assignment exactly $z$ of the terms $c_I \prod_{i \in I} x_i$ must equal '1' instead of '$-1$' to imply that $\mathscr{P}(x) = 0$. So $z$ out of $n + 2z$ sets $I \subseteq M$ (with $c_I \neq 0$) need to be chosen to contribute '$+1$' to $\mathscr{P}(x)$, and subsequently algorithm SOLVE_COE can be applied. This concludes the proof. □

Obviously, $\Phi$ having a positive polynomial representation is merely a *sufficient* condition for its unsatisfiability. Furthermore, concerning the third statement of this theorem, it may be interesting to mention that in specific cases, by making use of the particular numerical values of the coefficients $c_I$, the complexity bound given might turn out to be somewhat pessimistic. We do not pursue this here.

For a moment, let us consider 3SAT formulas. Then

$$\mathscr{P}(x) = n - \sum_{k=1}^{n} \left[ \sum_{i=1}^{m} a_{ki} x_i - \sum_{i=1}^{m} \sum_{j \neq i} a_{ki} a_{kj} x_i x_j + \sum_{i=1}^{m} \sum_{j \neq i} \sum_{l \neq i,\, j} a_{ki}\, a_{kj}\, a_{kl}\, x_i\, x_j\, x_l \right].$$

From this it is clear that doubly balanced formulas either have a balanced or positive polynomial representation; the linear and bilinear terms vanish, while at most $n$ (trilinear) terms remain (see Definition 3). However, 3SAT formulas need not be doubly balanced to have such a representation. Consider the following example.

**Example 13.** Given is the formula $\Phi$.

$$\Phi = (p \vee q \vee r) \wedge (p \vee q \vee \neg r) \wedge (\neg p \vee \neg q \vee \neg s) \wedge (\neg p \vee \neg q \vee s).$$

Obviously, $\Phi$ is not doubly balanced in the sense of Definition 3, but

$$\mathscr{P}_\Phi(x) = 4 + 4x_p x_q,$$

so condition (7) is satisfied implying that $\mathscr{P}_\Phi(x)$ is balanced.

Thus the notion of balanced polynomial representations is stronger than that of doubly balancedness. Obviously, checking the first is, in general, also computationally more involved. If the maximum clause length is bounded by $l$ it requires computing (at most) $(2^l - 1)n$ coefficients; doubly balancedness can be checked by computing $4n$ coefficients.

## 6. Detecting CoE subformulas

To help solving or to solve satisfiability problems, it may be helpful to detect CoE subformulas, or even unsatisfiable subformulas. If an unsatisfiable subformula is isolated, obviously the full formula is also unsatisfiable.

It may be noted that in random formulas the occurrence of CoE subformulas is highly unlikely; in SAT translations of practical problems on the other hand, often structure is present that might be detected in this way.

Let us consider the problem of finding an unsatisfiable or CoE subformula $\Phi'$ of $\Phi$. This can be formulated as an absolute value integer program in the following way. Introduced are the $\{0,1\}$ decision variables $y_k$, $1 \leqslant k \leqslant n$,

$$y_k = \begin{cases} 1 & \text{if clause } k \text{ is in } \Phi', \\ 0 & \text{if clause } k \text{ is in } \Phi \setminus \Phi'. \end{cases}$$

We have the following optimization problem.

$$\text{(AIP)} \qquad \begin{aligned} \min \quad & \sum_{I \subseteq M} \left| \sum_{k=1}^{n} \left( \prod_{i \in I} a_{ki} \right) y_k \right| - \sum_{k=1}^{n} y_k \\ \text{s.t.} \quad & \sum_{k=1}^{n} y_k \geqslant 1, \\ & y_k \in \{0,1\}^n. \end{aligned}$$

Note that a constraint is added to ensure the optimal solution does not induce the empty subformula. Let us denote the optimal value of (AIP) by $v^*$ and the optimal solution vector by $y^*$.

**Lemma 14.** *If $v^* < 0$ the subformula $\Phi' \subseteq \Phi$ induced by $y^*$ is unsatisfiable. If $v^* = 0$ the subformula $\Phi' \subseteq \Phi$ is solvable in polynomial time.*

**Proof.** Let us assume that $n^*$ variables $y_k^*$ are nonzero, hence subformula $\Phi'$ contains $n^*$ clauses. Note that

$$\sum_{I \subseteq M} \left| \sum_{y_k^* = 1} \left( \prod_{i \in I} a_{ki} \right) \right| = v^* + n^*.$$

So by Theorem 12 it follows that $\Phi'$ is unsatisfiable if $v^* < 0$ (since then it has a positive polynomial representation), and that $\Phi'$ has a balanced polynomial representation if $v^* = 0$ (thus allowing it to be solved in polynomial time). □

Unfortunately, in general it is hard to solve (AIP) exactly; if the absolute values are eliminated from the objective function using auxiliary variables, the constraint matrix will in general no longer be totally unimodular. Solving it to optimality using a mathematical programming procedure does not make sense, since that is as difficult as

solving the SAT instance itself (at least theoretically; in practice it might even be much more difficult). Therefore, we resort to using a very simple local search procedure to try to find CoE or unsatisfiable subformulas.

The idea is to start from the full formula (i.e. $y_k = 1$ for all $k$). The effect on the objective function of removing clause $k$ from the formula (setting variable $y_k$ to '0') is computed for all clauses $k$, and subsequently the best choice in terms of objective value is taken. This process continues until either the empty formula remains, or until the objective value is equal to or smaller than zero; then a CoE or unsatisfiable subformula is isolated. Even though this procedure is very simple, and many possible improvements could be incorporated (backtracking, randomization, etc.) it was successful on a particular set of benchmarks (see next section).

If an unsatisfiable subformula is found, we are done since the full formula must also be unsatisfiable. If a CoE subformula is obtained, it can be solved efficiently by algorithm SOLVE_CoE. In this way a number of dependent variables are identified; subsequently, these variables do not need to be considered explicitly when setting up a search tree. Thus the number of variables is reduced considerably. Let us try to illustrate with an example the possible merits of this approach.

**Example 15.** Assume that we are given a propositional formula $\Phi$ and by applying the simple heuristic procedure it is discovered that $\Phi$ is equivalent to

$$\Psi = (p_1 \leftrightarrow p_2 \leftrightarrow p_3) \wedge (p_1 \leftrightarrow p_4 \leftrightarrow p_5) \wedge (p_2 \leftrightarrow p_6 \leftrightarrow p_7) \wedge \neg(p_2 \leftrightarrow p_4 \leftrightarrow p_6) \wedge (\Phi \setminus \Phi').$$

Solving (or rather, rewriting) the CoE subformula we find

$$\Psi := \neg(p_1 \leftrightarrow p_5 \leftrightarrow p_7) \wedge (p_2 \leftrightarrow p_6 \leftrightarrow p_7) \wedge \neg(p_3 \leftrightarrow p_5 \leftrightarrow p_6) \wedge \neg(p_4 \leftrightarrow p_7) \wedge (\Phi \setminus \Phi'),$$

and $\mathscr{I} = \{p_5, p_6, p_7\}$. So there are three independent variables left to branch on (assuming no additional independent variables occur in $\Phi$), implying that the search tree needs to contain at most $2^3 - 1$ nodes; if the CoE subformula had not been solved at first, the tree might be considerably larger. Note that one can completely remove all dependent variables from the problem; for example, suppose that $\Phi$ contains a clause $p_3 \vee p_4$. This clause is equivalent to $\neg(p_5 \leftrightarrow p_6) \vee \neg p_7$. Observe that the (equivalency-) clausal structure is destroyed in this way; such a clause can be transformed to CNF again, but in general this cannot be done efficiently without auxiliary variables.

In the next section we consider some actual benchmarks.

## 7. Application to DIMACS benchmarks

In the set of DIMACS benchmarks[2] [12] there are several formulas that are both doubly balanced and have a balanced polynomial representation. It turns out that the `dubois*.cnf` and the `pret*_*.cnf` (which are all 3SAT formulas) can be proved unsatisfiable by Lemma 16. Also if this lemma is not used, these instances are solved within seconds by algorithm SOLVE_CoE.

**Lemma 16.** *Let $\Psi$ be a conjunction of equivalencies. If*
- *each variable occurs an* even *number of times in $\Psi$, and*
- *the number of negative equivalency-clauses is* odd,

*$\Psi$ is unsatisfiable.*

**Proof.** Consider problem (CoE). A relaxation of this is, to find $x \in \{-1,1\}^m$ that satisfies

$$\prod_{k=1}^{t} \delta(k) \prod_{i \in S_k} x_i = 1.$$

If the number of occurrences of each variable is even, this reduces to the product over all $\delta(k)$'s. If the number of negative equivalency-clauses is odd, a contradiction follows, so the formula under consideration is not satisfiable. □

In fact these formulas can be considered as special cases of the propositional formulas associated with *Tseitin graphs* [16], which Tseitin used to prove an exponential lower bound on the running time of regular resolution. For branching and other resolution-like algorithms these formulas are very hard, and the number of nodes required in the tree grows exponentially with the size of the formula.

Yet another interpretation of these formulas is, that they are 3SAT translations of formulas of the form

$$([\neg]p_n \leftrightarrow (\cdots([\neg]p_2 \leftrightarrow ([\neg]p_1 \leftrightarrow ([\neg]p_n \leftrightarrow (\cdots([\neg]p_2 \leftrightarrow [\neg]p_1))\cdots).$$

Here '$[\neg]$' denotes that the negation operator is optional. Such formulas without any negations are known as Urquhart formulas [17]. To obtain `dubois*.cnf`- and `pret*.cnf`-like formulas, an *odd* number of proposition letters must be negated, and subsequently the formula must be translated to 3SAT using auxiliary variables. Since the number of negations is odd, the resulting formula is clearly unsatisfiable.

Finally, let us turn to the `par*-c.cnf` instances. These are not doubly balanced, nor do they have a balanced polynomial representation. However, they contain doubly balanced subformulas that are detected by applying the local search procedure described in the previous section. We solve the instances by first solving the CoE subformulas and subsequently setting up a search tree over the remaining variables. In Table 2 the results are reported. It may be stressed that all computational results are obtained

[2] `ftp://dimacs.rutgers.edu/pub/challenge/satisfiability/benchmarks/`.

Table 2
Solution times for the par*-c.cnf instances. Given is the initial size of the formula and the time $t_{db}$ required to detect and solve its CoE subformula; the size of the remaining formula, and the time $t_{br}$ and number of nodes to solve that. All instances are satisfiable

| Name | $m$ | $n$ | $t_{db}$ | $\bar{m}$ | $\bar{n}$ | $t_{br}$ | nodes |
|---|---|---|---|---|---|---|---|
| par8-1-c | 64 | 254 | 1.1 | 8 | 30 | .8 | 39 |
| par8-2-c | 68 | 270 | 1.2 | 8 | 30 | 1.0 | 51 |
| par8-3-c | 75 | 298 | 1.3 | 8 | 30 | .9 | 45 |
| par8-4-c | 67 | 266 | 1.2 | 8 | 30 | .2 | 8 |
| par8-5-c | 75 | 298 | 1.3 | 8 | 30 | .2 | 8 |
| par16-1-c | 317 | 1264 | 11.6 | 47 | 184 | 732 | 5913 |
| par16-2-c | 349 | 1392 | 13.2 | 47 | 184 | 9.6 | 72 |
| par16-3-c | 334 | 1332 | 12.5 | 47 | 184 | 25.5 | 198 |
| par16-4-c | 324 | 1292 | 11.6 | 47 | 184 | 28.5 | 227 |
| par16-5-c | 341 | 1360 | 12.9 | 47 | 184 | 71 | 573 |

with an ad hoc MATLAB$^{TM}$ implementation; computation times could be improved substantially using more sophisticated implementations.

We also tried to solve the larger par32-*-c.cnf instances, that contain large doubly balanced subformulas as well. These instances have over 5000 clauses and more than 1300 variables. After solving the CoE subformula, 622 clauses and 157 variables remain. Unfortunately, this remaining problem still appeared to be too large to solve using the simple branching algorithm that was used to solve the smaller instances.

In a more recent paper we managed to solve the {\tt par 32-×-c.cnf} instances as well [19].

## Acknowledgements

We thank Jan Friso Groote, Oliver Kullmann and Michiel Odijk for some helpful discussions while writing this paper. Also thanks go to the anonymous referee, whose suggestions helped improve the presentation of the paper.

## References

[1] B. Aspvall, M.F. Plass, R.E. Tarjan, A linear-time algorithm for testing the truth of certain quantified boolean formulas, Inform. Process. Lett. 8 (3) (1979) 121–123.
[2] B. Benhamou, L. Sais, Theoretical study of symmetries in propositional calculus and applications, in: Proceedings of the 11th Conference on Automated Deduction, 1992, pp. 281–294.
[3] E. Boros, Y. Crama, P.L. Hammer, M. Saks, A complexity index for satisfiability problems, SIAM J. Comput. 23 (1992) 45–49.
[4] V. Chandru, C.R. Coullard, P.L. Hammer, M. Montanez, X. Sun, On renamable Horn and generalized Horn functions, Ann. Math. Artif. Intell. 1 (1990) 33–47.
[5] V. Chandru, J.N. Hooker, Extended Horn sets in propositional logic, J. Assoc. Comput. Mach. 38 (1991) 205–221.
[6] S.A. Cook, The complexity of theorem proving procedures, in: Proceedings of the third annual ACM symposium on the Theory of Computing, 1971, pp. 151–158.

[7] W.F. Dowling, J.H. Gallier, Linear-time algorithms for testing the satisfiability of propositional Horn formulae, J. Logic Program. 1 (3) (1984) 267–284.
[8] O. Dubois, Lecture held at DIMACS conference, Rutgers University, New Brunswick, NJ, March 1996.
[9] O. Dubois, P. Andre, Y. Boufkhad, J. Carlier, SAT versus UNSAT, in: Johnson and Trick (Eds.), Cliques, Coloring and Satisfiability: Second DIMACS implementation challenge, vol. 26 of DIMACS series in Discrete Mathematics and Computer Science. American Mathematical Society, Providence, RI, 1996, pp. 415–436.
[10] G. Gallo, M.G. Scutellá, Polynomially solvable satisfiability problems, Inform. Process. Lett. 29 (1988) 221–227.
[11] J. Gu, Global optimization for satisfiability (SAT) problem, IEEE Trans. Knowledge Data Eng. 6 (3) (1994) 361–381.
[12] 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, Providence, RI, 1996.
[13] H. Van Maaren, Elliptic approximations of propositional formulae, Technical Report 96-65, Faculty of Technical Mathematics and Informatics, Delft University of Technology, Delft, The Netherlands, 1996.
[14] H. Van Maaren, On the use of second order derivatives for the satisfiability problem, in: D. Du, J. Gu, P.M. Pardalos (Eds.), Satisfiability problem: Theory and applications, vol. 35 of DIMACS series in Discrete Mathematics and Computer Science, American Mathematical Society, Providence, RI, 1997, pp. 677–687.
[15] T.J. Schaefer, The complexity of satisfiability problems, in: Proceedings of the Tenth Symposium on the Theory of Computing, 1978, pp. 216–226.
[16] G.S. Tseitin, On the complexity of derivation in propositional calculus, Studies in Constructive Mathematics and Mathematical Logic Part 2 (1968) 115–125. Reprinted in J. Siekmann, G. Wrightson (Eds.), Automation of Reasoning, vol. 2, Springer, Berlin, 1983.
[17] A. Urquhart, Hard examples for resolution, J. ACM 34 (1987) 209–219.
[18] J.P. Warners, H. Van Maaren, Solving satisfiability problems using elliptic approximations — Effective branching rules, Technical Report 98-18, Department of Technical Mathematics and Informatics, Faculty of Information Technology and Systems, Delft University of Technology, Delft, The Netherlands, 1998, accepted for publication in Discr. Appl. Math.
[19] J.P. Warners, H. Van Maaren, A two-phase algorithm for solving a class of hard satisfiability problems, Oper. Res. Lett. 23 (3–5) (1999) 81–88.