Generating models of a matched formula with a polynomial delay

Petr Savicky
Institute of Computer Science,
Academy of Sciences of Czech Republic,
Pod Vodárenskou Věží 2,
182 07 Praha 8, Czech Republic
savicky@cs.cas.cz

Petr Kučera
Department of Theoretical Computer Science
and Mathematical Logic,
Faculty of Mathematics and Physics,
Charles University,
Malostranské nám. 25,
118 00 Praha 1, Czech Republic
kucerap@ktiml.mff.cuni.cz

Abstract

A matched formula is a CNF formula, such that the system of the sets of the variables, which appear in individual clauses, has a system of distinct representatives. Such a formula is always satisfiable. Matched formulas are used, for example, in the area of parametrized complexity. We prove that the problem of counting the number of the models (satisfying assignments) of a matched formula is #P-complete. On the other hand, we define a class of formulas generalizing the matched formulas and prove that for a formula in this class, one can choose in polynomial time a variable suitable for splitting the tree for the search of the models of the formula. As a consequence, the models of a formula from this class, in particular of any matched formula, can be generated sequentially with a delay polynomial in the size of the input. On the other hand, we prove that this task cannot be performed efficiently for the linearly satisfiable formulas, which is a generalization of matched formulas containing the class considered above.

Introduction

In this paper we consider the problem of counting the models (satisfying assignments) and generating subsets of the models of a given formula in conjunctive normal form (CNF). It is well known that a problem of counting the models of a general CNF is #P-complete (Sipser 2006). The problem of generating the models of a general CNF formula is clearly also hard, because checking whether there is at least one satisfying assignment of the formula, the SAT problem, is NP-complete (Garey and Johnson 1979). That is why we consider subclasses of formulae for which the satisfiability problem can be solved in polynomial time.

In particular we consider the class of matched formulas introduced in (Franco and Van Gelder 2003). Given a CNF formula φ we consider its *incidence graph* $I(\varphi)$ defined as follows. $I(\varphi)$ is a bipartite graph with one partity consisting of clauses of φ and the other partity containing the variables of φ. An edge $\{x, C\}$ for a variable x and a clause C is in $I(\varphi)$ if x or $\neg x$ appears in C. It was observed in (Aharoni and Linial 1986; Tovey 1984) that if $I(\varphi)$ admits a matching (i.e. a set of pairwise disjoint edges) of size m (where m is the number of clauses in φ), then φ is satisfiable. Later in (Franco and Van Gelder 2003) the formulas satisfying this condition were called *matched formulas*. Since a matching of maximum size in a bipartite graph can be found in polynomial time (see e.g. (Lovász and Plummer 1986)), one can check efficiently whether a given formula is matched.

Given a general formula φ we can measure how far it is from being matched by considering its maximum deficiency $\delta^*(\varphi)$, the number of clauses which remain unmatched in a maximum matching of $I(\varphi)$. A formula φ is thus matched iff $\delta^*(\varphi) = 0$. A weaker notion of deficiency $\delta(\varphi) = m - n$, where m is the number of clauses and n the number of the variables in φ, is also often being considered.

Matched formulas play a significant role in the theory of satisfiability solving. Since their introduction, matched formulas were considered as a base class in parameterized algorithms for satisfiability, see e.g. (Flum and Grohe 2006) for an overview of parameterized algorithms theory. In particular, the authors of (Fleischner, Kullmann, and Szeider 2002) show that satisfiability of the formulas whose maximum deficiency is bounded by a constant can be decided in polynomial time. This result was later improved in (Szeider 2003) to an algorithm for satisfiability parameterized with maximum deficiency of a formula. Parameterization based on backdoor sets with respect to matched formulas were considered in (Szeider 2007).

Several generalizations of matched formulas were considered in the literature, too. In (Kullmann 2000), matched formulas were generalized into the class of linearly satisfiable formulas. Autarkies based on matchings were studied in (Kullmann 2003). Another generalization was considered in (Szeider 2005) as classes of bi-clique satisfiable and var-satisfiable formulas. Unfortunately, for both bi-clique and var-satisfiable formulas it is hard to check if a formula falls into one of these classes (Szeider 2005).

Since all matched formulas are trivially satisfiable, we ask a stronger question: How hard is it to count or enumerate the models of a matched formula? We prove that counting the models of a matched formula is a #P-complete problem, and turn our attention to generating models of a matched formula. Since the number of the models can be large, we consider the efficiency with respect to the input and output size.

More specifically, one can consider two variants of the algorithm. In one of them, the input of the algorithm is a formula and a natural number k. The output is a set of k different models of the formula, if such models exist, or all models of the formula, if their number is less than k. Such an algorithm is considered polynomially output-efficient, if its running time is at most $k\,\|\varphi\|^{O(1)}$, where $\|\varphi\|$ is the length of the input formula φ. For simplicity, we present our algorithms in another setting, when the algorithm receives as input a formula and generates a sequence of all its models. The time needed for generating the first model and the time between generating any two consecutive models in the sequence is polynomial in the length of the formula. This type of complexity bound will be called a polynomial delay in this paper, although the notion of a polynomial delay is sometimes understood in a weaker sense.

The main result of the paper is an algorithm, which generates models of a matched formula with a polynomial delay. The algorithm constructs a splitting tree, whose nodes correspond to either a matched or an unsatisfiable formula. However, in some cases, this strategy is not sufficient, since some nodes of the tree cannot be split in this way. We prove that such a node corresponds to a formula, which can be satisfied by iterated elimination of pure literals. Formulas with this property will be called pure literal satisfiable. These formulas were studied in (Kullmann 2000) as a subclass of linearly satisfiable formulas. If a node with a pure literal satisfiable formula is reached, the algorithm switches to a simpler strategy. We prove that the models of a pure literal satisfiable formula can be generated with a delay linear in the length of the formula. On the other hand, the #SAT problem for pure literal satisfiable formulas is #P-complete, because this problem is #P-complete for monotone 2CNFs (Valiant 1979a; 1979b), which are pure literal satisfiable.

Several classes generalizing matched formulas were considered in the literature such as biclique satisfiable and var-satisfiable formulas (Szeider 2005) and linearly satisfiable formulas (Kullmann 2000). We show in this paper that our result does not transfer to the class of linearly satisfiable formulas by demonstrating that it is not possible to generate models of a linearly satisfiable formula with a polynomial delay unless P=NP.

Note that this is an extended abstract and most of the proofs are therefore omitted.

Definitions

In this section we give the necessary definitions and recall the results we use in this paper.

Boolean functions

A *Boolean function of n variables* is a mapping $f : \{0,1\}^n \rightarrow \{0,1\}$. A *literal* is either a variable, called *positive literal*, or its negation, called *negative literal*. The negation of the variable x will be denoted $\neg x$ or $\overline{x}$. A *clause* is a disjunction of a set of literals, which contains at most one literal for each variable. Formula φ is in *conjunctive normal form (CNF)* or, equivalently, φ is a CNF formula, if it is a conjuction of clauses. We often treat a clause as a set of its literals and a CNF formula as a set of its clauses. It is a well known fact that every Boolean function can be represented by a CNF formula (see e.g. (Genesereth and Nilsson 1987)). The size of a formula φ is the number of the clauses in φ and will be denoted as $|\varphi|$. The length of a formula φ is the total number of occurrences of literals in φ, i.e. the sum of the sizes of the clauses in φ, and will be denoted as $\|\varphi\|$. Given a variable x and a value $a \in \{0,1\}$, $\varphi[x = a]$ denotes a formula originating from φ by substituting x with value a and the obvious simplifications consisting in removing falsified literals and satisfied clauses. We extend this notation to negative literals as well by setting $\varphi[\overline{x} = a] = \varphi[x = \overline{a}]$. The formula obtained from φ by assigning the values $a_1, \dots, a_k \in \{0,1\}$ to the variables $x_1, \dots, x_k$ is denoted as $\varphi[x_1 = a_1, x_2 = a_2, \dots, x_k = a_k]$. We say that a literal l is *pure* in a CNF formula, if it occurs in the formula and the negated literal $\neg l$ does not. A literal is *irrelevant* in a formula, if neither the literal nor its negation occurs in the formula. A variable is *pure*, if it appears only positively, or only negatively in φ, i.e. it appears in a literal, which is pure in φ.

Let φ be a formula defining a Boolean function f on n variables. An assignment of values $v \in \{0,1\}^n$ is a *model* of φ (also a *satisfying assignment*, or a *true point* of φ), if it satisfies f, i.e. if $f(v) = 1$. The set of models of φ is denoted as $T(\varphi)$. The models in $T(\varphi)$ are defined on the variables, which have an occurrence in φ. The set of the variables of the function defined by a formula can be larger, however, we do not introduce a special notation for this more general case. For algorithmic purposes, this is also not necessary, since adding an irrelevant variable to a formula changes the set of the models by adding this variable with both possible values to each element of the original set of models.

A *partial assignment* assigns values only to a subset of the variables. For a formula of the variables $x_1, \dots, x_n$, it can be represented as a ternary vector $v \in \{0, 1, *\}^n$, where $v_i = *$ denotes the fact that x_i is not assigned a value by v.

Note that an empty clause does not admit a satisfying assignment and an empty CNF is satisfied by any assignment.

Matched formulas

In this paper we use standard graph terminology, see e.g. (Bollobás 1998). Given an undirected graph $G = (V, E)$, a subset of edges $M \subseteq E$ is a *matching* in G if the edges in M are pairwise disjoint. A *bipartite graph* $G = (A, B, E)$ is an undirected graph with disjoint sets of vertices A and B, and the set of edges E satisfying $E \subseteq A \times B$. For a set W of vertices of G, let $\Gamma(W)$ denote the *neighbourhood* of W in G, i.e. the set of all vertices adjacent to some element of W. We shall use the following well-known result on matchings in bipartite graphs:

Theorem 1 (Hall's Theorem)**.** *Let $G = (A, B, E)$ be a bipartite graph. A matching M of size $|M| = |A|$ exists if and only if for every subset S of A we have that $|S| \leq |\Gamma(S)|$.*

Let $\varphi = C_1 \wedge \ldots \wedge C_m$ be a CNF formula on n variables $X = \{x_1, \dots, x_n\}$. We associate a bipartite graph $I(\varphi) = (\varphi, X, E)$ with φ (also called the incidence graph of φ), where the vertices correspond to clauses in φ and the variables X. A clause C_i is connected to a variable x_j (i.e. $\{C_i, x_j\} \in E$) if C_i contains x_j or $\overline{x_j}$. A CNF formula φ is *matched* if $I(\varphi)$ has a matching of size m, i.e. if there is a matching which pairs each clause with a unique variable, this is called *perfect matching* of $I(\varphi)$. Note, that a matched CNF is trivially satisfiable, since each clause can be satisfied by the literal containing the variable matched to the given clause. A variable which is matched to some clause in a given matching M is called *matched* in M, it is *free* in M otherwise.

Generating models with a polynomial delay

The main goal of this paper is to describe an algorithm which given a matched formula φ generates the set $T(\varphi)$ of models of φ with a polynomial delay. Let us state more formally what we require of such an algorithm.

We say that an algorithm generates the models of a Boolean formula φ with a polynomial delay, if there is a polynomial p, such that the algorithm given a formula φ as an input satisfies the following properties.

1. It works in steps, each of which takes time $O(p(\|\varphi\|))$.
2. In each step, it either finds a model of φ different from the models obtained in the previous steps (in particular, any model in the first step) or determines that that there is no such model, so the previous steps already found all the models of φ.

If an algorithm with the properties above exists, it follows that we can construct the set $T(\varphi)$ of all models in time $O((|T(\varphi)| + 1) \cdot p(\|\varphi\|))$, which means that the algorithm is polynomial with respect to the size of the input and output. Note that since $T(\varphi)$ may be of exponential size with respect to $\|\varphi\|$, efficiency with respect to the size of the input and output is the best we can hope for when constructing $T(\varphi)$. Note also that often in the literature algorithms working with polynomial delay are allowed to spend time $O(p(|T'|, \|\varphi\|))$ to generate a new model, where T' is the set of models found in the previous steps. Our definition is more restrictive than this.

Efficient splitting tree algorithm

The idea of the algorithm is to construct a decision tree for the function represented by a given satisfiable CNF, such that every subtree larger than a single leaf contains a 1-leaf. The depth of the tree is at most the number of the variables. If this tree is searched in a DFS order, then the time needed in an arbitrary moment to reach a 1-leaf is at most n times the time needed to split a node. In the following, we show that for some classes of formulas including the matched formulas, it is possible to find a splitting procedure, which yields a tree as described above.

A *decision tree* for a Boolean function f is a labeled binary tree, where each inner node is labeled with a variable, while leaves and edges have labels 0 or 1. A decision tree computes $f(x)$ for a given assignment x by a process, which starts at the root and in each visited node follows the edge labeled by the value of the variable, which is the label of the node. The output is the label of the leaf reached by this process. If a computation path tests a variable, which was tested in the previous part of the path, then this test is redundant. We consider only trees without such redundant tests.

A decision tree representing the same function as a given CNF formula φ can be constructed top down as follows. The root of the tree is assigned to φ. For each non-leaf node of the tree assigned to a formula ψ, we choose an arbitrary split variable x, which has an occurrence in ψ, and assign the restricted formulas $\psi[x = 0]$ and $\psi[x = 1]$ to the successors. A node assigned to an empty formula becomes a 1-leaf and a node assigned to a formula, which contains an empty clause, becomes a 0-leaf. The resulting decision tree represents the function given by φ, although it can be too large for practical purposes. Each path from the root to an inner node u of the tree corresponds to a partial assignment which changes φ to a formula representing the function computed by the subtree, whose root is u. The depth of a tree for a function of n variables is at most n.

Each leaf node labeled with 1 represents a set of models of φ, more precisely, a leaf in depth d represents 2^{n-d} models of φ. Moreover, different leaves of the tree represent disjoint sets of models. Given a decision tree for the function represented by φ we can, by traversing it, generate all models of φ in time proportional to its size. This process leads to a large delay between generating successive models, if the tree contains large subtrees with only 0-leaves. The following condition on a class of formulas describes a situation, when this can be avoided.

Definition 2. Let U be a class of formulas, let $\varphi \in U$ and let x be a variable with an occurrence in φ. We say that x is a *splitting variable* for φ relative to U, if for every $a \in \{0, 1\}$, such that $\varphi[x = a]$ is satisfiable, we have $\varphi[x = a] \in U$.

A class of formulas U has the *splitting property*, if every formula in U containing a variable contains a splitting variable relative to U.

Definition 3. Let U be a class of formulas having the splitting property. The *splitting complexity* of a formula $\varphi \in U$, which contains a variable, is the time needed to find some of the splitting variables x for φ relative to U and the results of the satisfiability tests for the formulas $\varphi[x = 0]$ and $\varphi[x = 1]$. We say that the class U has a *splitting complexity* $c(\varphi)$ if every nontrivial formula $\varphi \in U$ and every nontrivial formula in U obtained from φ by a partial assignment has splitting complexity at most $c(\varphi)$. If moreover $c(\varphi) \leq p(\|\varphi\|)$ where p is a polynomial, then we say that U has a *polynomial splitting complexity*.

If $\varphi \in U$, then its splitting complexity is also an upper bound on the time of a satisfiability test for φ itself, since φ is satisfiable, if and only if at least one of the formulas $\varphi[x = 0]$ and $\varphi[x = 1]$, where x is a splitting variable, is satisfiable.

Theorem 4. *If a class of formulas U has the splitting property with the splitting complexity $c(\varphi)$, where $c(\varphi) \geq \|\varphi\|$ for each formula $\varphi \in U$, then the models of a formula $\varphi \in U$ with n variables can be generated with a delay $O(n \cdot c(\varphi))$.*

Remark 5. If φ contains a unit clause and U is closed under the unit propagation, then a variable x contained in a unit clause is a splitting variable, which can be identified efficiently. The reason is that if φ is known to be satisfiable, then one of the formulas $\varphi[x = a]$ contains an empty clause and, hence, the other is satisfiable.

Remark 6. Any class U satisfying that

1. the satisfiability of formulas in U can be tested in polynomial time, and
2. U is closed under partial assignments

has a polynomial splitting complexity. Indeed, in this case any variable in a formula φ from U is a splitting variable and the satisfiability tests for the corresponding restrictions can be obtained in polynomial time. Examples of such classes are Horn formulas, SLUR formulas, 2CNFs, q-Horn formulas etc. Hence, as an immediate corollary of Theorem 4, it is possible to generate the models of formulas in these classes with a polynomial delay.

The main result of this paper is that a slight generalization of matched formulas also has polynomial splitting complexity although the class of matched formulas is not closed under partial assignments.

Pure literal satisfiable formulas

Before considering matched formulas, let us make a small detour to the class of formulas which are satisfiable by iterated elimination of pure literals, which we call pure literal satisfiable. These formulas have already been considered in (Kullmann 2000) as a special case of linearly satisfiable formulas.

A set of literals is called *consistent*, if it does not contain contradictory literals. If l is a literal, let $\mathrm{assign}(l)$ be the assignment to the variable contained in l, which satisfies l. For a consistent set or sequence of literals L, let $\mathrm{assign}(L)$ be the partial assignment of the variables satisfying the literals in L. For a formula φ, $\varphi[L]$ is an abbreviation for $\varphi[\mathrm{assign}(L)]$.

Definition 7. A *pure literal sequence* for a formula φ is a consistent sequence of literals $(l_1, \ldots, l_k)$, such that for every $i = 1, \ldots, k$, the literal l_i is either pure or irrelevant in the formula $\varphi[l_1, \ldots, l_{i-1}]$. In particular, l_1 is pure or irrelevant in φ. A pure literal sequence is called *strict*, if each of the literals l_i is pure in $\varphi[l_1, \ldots, l_{i-1}]$.

If L is a pure literal sequence for φ, the formula $\varphi[L]$ will be called the *reduced* formula corresponding to φ and L. If $\varphi[L]$ does not contain a pure literal, L will be called a *maximal* pure literal sequence for φ.

Definition 8. A formula φ is *pure literal satisfiable*, if there is a pure literal sequence L for φ, such that the reduced formula $\varphi[L]$ is empty or, equivalently, $\mathrm{assign}(L)$ is a satisfying assignment of φ.

An autarky for a formula φ is a partial assignment v of the variables, such that every clause is either satisfied or unchanged by v. For more on autarkies see e.g. (Kullmann 2000). Note that every initial segment of a pure literal sequence defines an assignment to the variables, which is an autarky. Moreover, one can easily verify that this property characterizes pure literal sequences.

Let us note that pure literal satisfiable formulas are not closed under partial assignments. Consider a formula ψ, which is either unsatisfiable or satisfiable and does not contain a pure literal. Let φ be the formula obtained from ψ by adding a new variable x as a positive literal to every clause. Formula φ is pure literal satisfiable, but $\varphi[x = 0] = \psi$ is not pure literal satisfiable. It follows that pure literal satisfiable formulas do not satisfy the second property required in Remark 6 and we have to put more effort into showing that pure literal satisfiable formulas have the splitting property and a polynomial splitting complexity.

For every CNF formula, it may be tested in polynomial time, whether it is pure literal satisfiable. In order to find a pure literal sequence witnessing this fact, the procedure FindPLS in Figure 1 uses a greedy approach, which at each step chooses and satisfies any pure literal in the current formula. This approach is meaningful, since if a literal is pure at some stage of the procedure, it either remains pure or becomes irrelevant in the following stages. The pure literal sequence obtained by the procedure depends on the nondeterministic choices made by the procedure, however, by Lemma 9, the resulting reduced formula is uniquely determined by the input.

Procedure FindPLS
Input: A CNF formula φ.
Output: A maximal strict pure literal sequence L for φ and the corresponding reduced formula.
Definition: For a formula φ, let $\mathrm{Pure}(\varphi)$ denote the set of the literals, which are pure in it.

```
initialize $\psi$ with the input formula $\varphi$
initialize $L$ with an empty list
while $\mathrm{Pure}(\psi)$ is not empty
    choose any literal $l$ from $\mathrm{Pure}(\psi)$
    add $l$ to $L$ and apply $\mathrm{assign}(l)$ to $\psi$
Output the list $L$ and the formula $\psi$
```

Figure 1: Constructing pure literal sequence

Lemma 9. *Let φ be a CNF formula and let L be a pure literal sequence obtained by FindPLS for φ.*

1. *The formula $\varphi[L]$ is uniquely determined by φ.*
2. *The formula φ is pure literal satisfiable, if and only if $\varphi[L]$ is empty.*

Since the running time of FindPLS procedure is polynomial in the length of the input formula, a maximal pure literal sequence for a formula can be constructed in polynomial time. The complexity of constructing a maximal pure literal sequence for a formula φ is, in fact, $O(\|\varphi\|)$ by Lemma 23.

Lemma 10. *Let $L = (l_1, \ldots, l_n)$ be a pure literal sequence for a formula φ, which contains a literal for each variable of φ. For $i = 1, \ldots, n$, denote by x_i the variable contained in l_i. If x_i is the variable with the largest index i among the variables, which have an occurence in φ, then x_i is a splitting variable for φ and each of the formulas $\varphi[x_i = 0]$ and $\varphi[x_i = 1]$ is satisfiable, if and only if it does not contain an empty clause.*

The following lemma is sufficient for this and the next section, however, the splitting complexity is, in fact, $O(\|\varphi\|)$ by Theorem 24.

Lemma 11. *The class of pure literal satisfiable formulas has a polynomial splitting complexity.*

Proof. If φ is pure literal satisfiable, then a pure literal sequence, which satisfies it, can be obtained by FindPLS in polynomial time. If the sequence does not contain literals for all variables, it is extended in polynomial time by appending arbitrary literals for the missing variables to obtain a pure literal sequence satisfying the assumption of Lemma 10. Then, this lemma implies a method to select a splitting variable and obtain the results of the satisfiability test for the corresponding restrictions in polynomial time. □

If a pure literal sequence satisfies the assumption of Lemma 10 for a formula φ, then the same sequence can be used to find a splitting variable for all formulas in a splitting tree for φ. Using this, the models of a pure literal satisfiable formula can be generated with a delay smaller than the general bound from Theorem 4, see Corollary 24.

Remark 12. The sign of a literal for a given variable, which occurs in a strict pure literal sequence, is not uniquely determined. Each of the variables y_1 and y_2 can occur both positively and negatively in a strict pure literal sequence for the formula

$$(x_1 \vee y_1) \wedge (x_2 \vee \overline{y_1}) \wedge (x_3 \vee y_2) \wedge (x_4 \vee \overline{y_2}) \wedge (y_1 \vee y_2)\ .$$

For example, $(x_2, y_1, x_3, \overline{y_2})$ and $(x_4, y_2, x_1, \overline{y_1})$ are strict pure literal sequences for this formula.

Matched formulas

In this section we concentrate on matched formulas. Let us start with showing that the problem of determining the number of models of a matched formula φ, i.e. the size $|T(\varphi)|$, is as hard as a general #SAT problem.

Theorem 13. *The problem of determining $|T(\varphi)|$ given a matched formula φ is #P-complete.*

Our goal is to show that we can generate the models of a matched formula with a polynomial delay. Theorem 4 cannot be used for this directly, since the class of the matched formulas does not have the splitting property as can be seen from the following example. Consider the formula

$$(x_1 \vee x_2) \wedge (x_1 \vee x_3) \wedge (x_2 \vee x_3)\ .$$

This formula is matched, but it has no splitting variable. Indeed, setting x_1 to 0 leads to a satisfiable, yet not matched formula $(x_2)(x_3)(x_2 \vee x_3)$ and by symmetry this is true for variables x_2 and x_3 as well. In order to achieve our objective, we have to consider a richer class of formulas. The class we consider generalizes matched and pure literal satisfiable formulas as follows. Note that an empty formula is matched, since it corresponds to an empty graph and we can formally assume that an empty graph possesses the required matching.

Definition 14. A formula φ is called *pure literal matched*, if the reduced formula obtained by FindPLS procedure for φ is matched.

Elimination of a pure literal preserves the property of being matched, since a pure literal is an autarky. Hence, a matched formula is pure literal matched. Clearly, every pure literal satisfiable formula is pure literal matched, since its reduced formula is empty and, hence, matched.

The basic idea of an efficient splitting algorithm for the matched formulas is presented in the following theorem. The splitting complexity of the pure literal matched formulas can be bounded as $O(n^2 \cdot \|\varphi\|)$ by Theorem 26.

Theorem 15. *The class of pure literal matched formulas has the splitting property with a polynomial splitting complexity.*

In order to prove Theorem 15, we have to show several statements concernig the structure of a matched formula. If V is a set of variables, we say that a clause is *restricted* to V, if it contains only literals with variables from V.

Definition 16. Let V be a subset of the variables of a matched formula φ. The set V will be called a *critical block*, if the number of clauses of φ, which are restricted to V, is equal to $|V|$. Formally, if V is empty, it is also a critical block.

Note that if φ is a matched formula, V is a subset of its variables, and $\mathcal{C}$ is the set of the clauses in φ restricted to V, then by Hall's theorem (Theorem 1 above) we have $|\mathcal{C}| \leq \Gamma(\mathcal{C}) \leq |V|$. Critical blocks are those achieving the equality. Basic property of critical blocks with respect to the possible perfect matchings of $I(\varphi)$ is as follows.

Lemma 17. *Let V be a critical block of a matched formula φ. Then, in every perfect matching of $I(\varphi)$, the variables from V are matched to clauses restricted to V.*

Another useful property of the set of the critical blocks is as follows.

Lemma 18. *The set of the critical blocks of a matched formula is closed under intersection.*

If φ is a formula and x is a variable contained in at least one critical block, then Lemma 18 implies that there is a unique inclusion minimal critical block of φ containing x, which is equal to the intersection of all critical blocks of φ containing x. If a matched formula has the same number of clauses and variables, then every variable is contained in a critical block, since the set of all the variables of the formula is a critical block.

Definition 19. If φ is a matched formula with the same number of clauses and variables and x is some of its variables, then let B_x denote the inclusion minimal critical block containing x.

The notation B_x does not specify the formula, since it will always be clear from the context.

Lemma 20. *Let φ be a matched formula with the same number of clauses and variables. Let l be a literal containing a variable x and let us assume that the formula $\varphi[\neg l]$ is not matched. Then*

1. *the literal l is pure or irrelevant in the clauses of φ restricted to B_x,*
2. *if a clause C of φ contains $\neg l$, then in every matching for φ, C is matched to a variable y, such that $B_x \subset B_y$ (where $\subset$ denotes strict inclusion).*

The structure of the critical blocks will be used to show the following proposition needed to prove Theorem 15.

Theorem 21. *Let φ be a matched formula. If for every variable x, which has an occurence in φ, there is $a \in \{0, 1\}$, such that $\varphi[x = a]$ is not matched, then φ is pure literal satisfiable.*

Proof of Theorem 15. Assume, φ is a pure literal matched formula. Let L be a pure literal sequence obtained by Find-PLS procedure for φ and let $\psi = \varphi[L]$, which is, by the assumption, a matched formula. Since L is maximal, ψ does not contain a pure literal. If ψ is empty, then φ is itself a pure literal satisfiable formula and we can find a splitting variable for φ by the method from Lemma 11. If ψ is not empty, then it is matched and not pure literal satisfiable. Hence, by Theorem 21, there is a variable x of ψ, such that $\psi[x = 0]$ and $\psi[x = 1]$ are both matched. Since L does not contain a literal with the variable x, the application of $\mathrm{assign}(L)$ and $x = a$ commute for each $a \in \{0, 1\}$. Hence, L is a pure literal sequence for the formula $\varphi[x = a]$ and the application of $\mathrm{assign}(L)$ to $\varphi[x = a]$ leads to $\psi[x = a]$, which is matched. Hence, for each $a \in \{0, 1\}$, the formula $\varphi[x = a]$ is pure literal matched and the variable x is a splitting variable for the formula φ.

A time polynomial in the length of the formula is sufficient to select a splitting variable x as in the proof above. If ψ is nonempty, the satisfiability of $\varphi[x = 0]$ and $\varphi[x = 1]$ is guaranteed by the choice of x. If ψ is empty, φ is pure literal satisfiable and the method from Lemma 11 is used. Hence, a splitting variable and the results of the required satisfiability tests can be obtained in a polynomial time. $\square$

Similarly as the class of matched formulas, also the class of pure literal matched formulas is closed under the unit propagation. This implies that the unit propagation can be used as part of the construction of the splitting tree by Remark 5.

Proposition 22. *The class of pure literal matched formulas is closed under the unit propagation.*

Algorithms and complexity

In this section, we prove specific complexity bounds for the algorithms presented in the previous sections. The complexity bounds are derived for the RAM model with the unit cost measure and the word size $O(\log \|\varphi\|)$, where φ is the input formula. Let us first concentrate on the pure literal satisfiable formulas.

Lemma 23. *A maximal pure literal sequence L for a CNF formula φ can be constructed in time $O(\|\varphi\|)$.*

Theorem 24. *A pure literal satisfiable formula φ has splitting complexity $O(\|\varphi\|)$. Moreover, the set $T(\varphi)$ of the models of φ can be generated with a delay $O(\|\varphi\|)$.*

Now let us concentrate on the time complexity of selecting a splitting variable of a pure literal matched formula.

Lemma 25. *The splitting complexity of a pure literal matched formula φ of n variables is $O(n \cdot \|\varphi\|)$.*

As a corollary of Lemma 25 and the general bound from Theorem 4, we get the following.

Corollary 26. *Models of a pure literal matched formula φ on n variables can be generated with a delay $O(n^2 \cdot \|\varphi\|)$.*

Linearly satisfiable formulas

In this section we consider the class of linearly satisfiable formulas. By results of (Kullmann 2000), this class generalizes both the matched formulas and the pure literal satisfiable formulas and, by combining the proofs, also the class of pure literal matched formulas. In this section, we show that it is not possible to generate models of linearly satisfiable formulas with a polynomial delay unless P=NP.

As a consequence, linearly satisfiable formulas do not have a polynomial splitting complexity unless P=NP. This consequence follows also unconditionally from a fact that there is a linearly satisfiable formula which does not have a splitting variable with respect to the class of linearly satisfiable formulas.

We omit the details due to space limitations.

Conclusion and directions for further research

In this paper we have shown that it is possible to generate the models of a matched formula φ of n variables with delay $O(n^2 \cdot \|\varphi\|)$. As a byproduct we have shown that the models of a pure literal satisfiable formula φ (i.e. a formula satisfiable by iterated pure literal elimination) can be generated with delay $O(\|\varphi\|)$. We have also shown that this result cannot be generalized for the class of linearly satisfiable formulas, since it is not possible to generate models of linearly satisfiable formulas with a polynomial delay unless P=NP.

Let us mention that the procedure for generating the models with a bounded delay can be extended to formulas, for which a small strong backdoor set with respect to the class of matched formulas with empty clause detection can be found. Assume, B is such a backdoor set for a formula φ, i.e. B is a set of variables satisfying that any partial assignment to variables in B leads to a matched formula, or to a formula containing an empty clause. Then we can generate the decision tree for φ (and thus generate its models) in time $O(2^{|B|}\,\|\varphi\| + T(f)\,n^2\,\|\varphi\|)$. Unfortunately, searching for strong backdoor sets with respect to the class of matched formulas is hard (Szeider 2007).

An interesting question is whether our approach could be used with the parameterized satisfiability algorithm based on maximum deficiency (see (Szeider 2003)) in order to get a parameterized algorithm for generating the models of a general formula.

Acknowledgement

Petr Savický was supported by CE-ITI and GAČR under the grant number GBP202/12/G061 and by the institutional research plan RVO:67985807. Petr Kučera was supported by the Czech Science Foundation (grant GA15-15511S).

References

Aharoni, R., and Linial, N. 1986. Minimal non-two-colorable hypergraphs and minimal unsatisfiable formulas. *Journal of Combinatorial Theory, Series A* 43(2):196 – 204.

Bollobás, B. 1998. *Modern Graph Theory*, volume 184 of *Graduate Texts in Mathematics*. Springer.

Fleischner, H.; Kullmann, O.; and Szeider, S. 2002. Polynomial-time recognition of minimal unsatisfiable formulas with fixed clause-variable difference. *Theoretical Computer Science* 289(1):503 – 516.

Flum, J., and Grohe, M. 2006. *Parameterized complexity theory*, volume 3. Springer.

Franco, J., and Van Gelder, A. 2003. A perspective on certain polynomial-time solvable classes of satisfiability. *Discrete Appl. Math.* 125(2-3):177–214.

Garey, M., and Johnson, D. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. San Francisco: W.H. Freeman and Company.

Genesereth, M., and Nilsson, N. 1987. *Logical Foundations of Artificial Intelligence*. Los Altos, CA: Morgan Kaufmann.

Kullmann, O. 2000. Investigations on autark assignments. *Discrete Applied Mathematics* 107(1–3):99 – 137.

Kullmann, O. 2003. Lean clause-sets: generalizations of minimally unsatisfiable clause-sets. *Discrete Applied Mathematics* 130(2):209 – 249. The Renesse Issue on Satisfiability.

Lovász, L., and Plummer, M. D. 1986. *Matching Theory*. North-Holland.

Sipser, M. 2006. *Introduction to the Theory of Computation*, volume 2. Thomson Course Technology Boston.

Szeider, S. 2003. Minimal unsatisfiable formulas with bounded clause-variable difference are fixed-parameter tractable. In Warnow, T., and Zhu, B., eds., *Computing and Combinatorics*, volume 2697 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg. 548–558.

Szeider, S. 2005. Generalizations of matched CNF formulas. *Annals of Mathematics and Artificial Intelligence* 43(1-4):223–238.

Szeider, S. 2007. Matched formulas and backdoor sets. In Marques-Silva, J., and Sakallah, K., eds., *Theory and Applications of Satisfiability Testing – SAT 2007*, volume 4501 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg. 94–99.

Tovey, C. A. 1984. A simplified NP-complete satisfiability problem. *Discrete Applied Mathematics* 8(1):85 – 89.

Valiant, L. 1979a. The complexity of computing the permanent. *Theoretical Computer Science* 8(2):189 – 201.

Valiant, L. 1979b. The complexity of enumeration and reliability problems. *SIAM Journal on Computing* 8(3):410–421.