Journal of Artificial Intelligence Research 56 (2016) 379-402 Submitted 09/15; published 06/16

Generating Models of a Matched Formula With a Polynomial Delay

Petr Savický SAVICKY@CS.CAS.CZ
Institute of Computer Science, The Czech Academy of Sciences
Pod Vodárenskou Věží 2, 182 07 Praha 8, Czech Republic

Petr Kučera KUCERAP@KTIML.MFF.CUNI.CZ
Department of Theoretical Computer Science and Mathematical Logic
Faculty of Mathematics and Physics, Charles University in Prague,
Malostranské nám. 25, 118 00 Praha 1, Czech Republic

Abstract

A matched formula is a CNF formula whose incidence graph admits a matching which matches a distinct variable to every clause. 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 linearly satisfiable formulas, which is a generalization of matched formulas containing the class considered above.

1. 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 the 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 & Johnson, 1979).

In this paper, we mostly deal with the problem of enumerating models of a formula. This problem is important in areas of research and applications, such as unbounded model checking (Kang & Park, 2005; McMillan, 2002) or data mining (Coquery, Jabbour, Sais, Salhi, et al., 2012). The success of modern SAT solvers inspired design of model counting and enumeration algorithms as well (see e.g. Jabbour, Lonlac, Sais, & Salhi, 2014; Morgado & Marques-Silva, 2005a, 2005b). In addition to the basic enumeration problem in which we do not require the models to be generated in any prescribed order, other versions have been considered, e.g. generating models by non-decreasing weight (Creignou, Olive, & Schmidt, 2011).

Another line of research concentrated on studying special classes of boolean formulas for which an enumeration algorithm with guaranteed complexity could be devised. One can easily find an example of a formula for which the set of models is exponentially larger than the size of the formula itself. In such a case it is reasonable to include the size of the output into the bound on the running time of an enumeration algorithm. More specifically we say that an algorithm which enumerates models of a formula runs in *output polynomial time* if its running time can be bounded by a polynomial in two variables, the size of the input (i.e. the input formula φ) and the size of the output (i.e. the number of models of φ). In this paper, we consider more restrictive setting as follows. The algorithm receives as input a formula and generates a sequence of all its models in such a way that 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 is called a polynomial delay. It should be clear that if we can enumerate models of a formula with a polynomial delay, then we can construct an output polynomial algorithm for this task as well. On the other hand, it can be much harder to get an enumeration algorithm with polynomial delay than an output polynomial algorithm. For an overview of various notions of enumeration complexity (see Johnson, Yannakakis, & Papadimitriou, 1988).

There are special classes of formulas for which polynomial delay enumeration algorithms have been described, this includes 2-CNF formulas, Horn formulas, generalized satisfiability problems and others (see e.g. Aceto, Monica, Ingólfsdóttir, Montanari, & Sciavicco, 2013; Creignou & Hébrard, 1997; Dechter & Itai, 1992; Kavvadias, Sideri, & Stavropoulos, 2000). In this paper, we describe another class of formulas for which a polynomial delay enumeration algorithm based on backtrack-free search can be described. On the contrary to such algorithms known for 2-CNF or Horn formulas, the splitting variable in each step cannot be chosen arbitrarily, however, the existence of a suitable variable is guaranteed and it can be efficiently identified.

In particular we consider the class of matched formulas introduced by 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 part consisting of clauses of φ and the other part 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 by Aharoni and Linial (1986) and 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 the formulas satisfying this condition were called *matched formulas* by Franco and Van Gelder. Since a matching of maximum size in a bipartite graph can be found in polynomial time (see e.g. Lovász & 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 have been considered as a base class in parameterized algorithms for satisfiability, see e.g. the book of Flum and Grohe (2006) for an overview of parameterized algorithms theory. In particular, Fleischner, Kullmann, and Szeider (2002) show that satisfiability of formulas with maximum deficiency bounded by a constant k can be decided in time $O(\|\varphi\| n^{O(k)})$ where $\|\varphi\|$ is the length of the input formula φ and n

denotes the number of its variables. This result was later improved by Szeider (2003) to an algorithm for satisfiability parameterized with maximum deficiency of a formula with complexity $O(2^k n^3)$. Parameterization based on backdoor sets with respect to matched formulas were considered by Szeider (2007).

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. 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 by 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 generalizations of matched formulas have also been considered in the literature. Kullmann (2000) generalized matched formulas into the class of linearly satisfiable formulas. Autarkies based on matchings were studied by Kullmann (2003). Szeider (2005) considered another generalization of matched formulas, the classes of biclique satisfiable and var-satisfiable formulas. Unfortunately, for both biclique satisfiable and var-satisfiable formulas it is hard to check if a formula falls into one of these classes (Szeider, 2005).

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.

The paper is organized as follows. After giving basic definitions in Section 2, we describe in Section 3 a specific simple splitting property of a class of formulas, which allows to generate the models of a formula from the class efficiently. In Section 4, we consider pure literal satisfiable formulas and prove that this class has the required splitting property. In Section 5, we consider the matched formulas and prove the required splitting property of a class of formulas, which generalizes both the matched and pure literal satisfiable formulas in a natural way. This implies an algorithm for generating all models of a matched formula or a formula from the more general class with a polynomial delay. In Section 6, we present complexity bounds for efficient versions of the algorithms from the previous sections. In Section 7, we show the negative result concerning linearly satisfiable formulas. Section 8 contains concluding remarks and some directions for further research.

2. Definitions

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

2.1 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 & 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 $a_1, \ldots, a_k \in \{0,1\}$ to the variables $x_1, \ldots, x_k$ is denoted as $\varphi[x_1 = a_1, x_2 = a_2, \ldots, x_k = a_k]$. 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 $x_1, \ldots, x_n$, 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.

2.2 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 *neighborhood* 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 2.1 (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 $X = \{x_1, \ldots, x_n\}$. 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, we shall call such matching as *clause saturated matching*. 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.

2.3 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 output polynomial. 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 $T(\varphi)$.

3. 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 3.1. 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.

We shall associate a splitting problem with a class of formulas U having splitting property.

Definition 3.2. Let U be a class of formulas with splitting property. The *splitting problem* relative to U is the following problem: Given a formula $\varphi \in U$, find a splitting variable for φ relative to U and the results of satisfiability tests for the formulas $\varphi[x = 0]$ and $\varphi[x = 1]$.

Note that the complexity of the splitting problem relative to U is also an upper bound on the time of a satisfiability test for formulas in U. This is because a formula φ is satisfiable, if and only if for any variable x we have that at least one of the formulas $\varphi[x = 0]$ and $\varphi[x = 1]$ is satisfiable. The result of these satisfiability checks for a splitting variable x is a required part of solution to the splitting problem.

Theorem 3.3. *If a class of formulas U has the splitting property and the splitting problem relative to U can be solved in time $c(\varphi)$, 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 $O(n \cdot c(\varphi))$.*

Proof. Construct a tree for φ in a DFS order using a splitting variable for every formula assigned to a non-leaf node. If a non-leaf node is labeled by φ and x is the splitting variable, the successors are labeled by $\varphi[x = 0]$ and $\varphi[x = 1]$. If some of these formulas is unsatisfiable, the corresponding successor becomes a 0 leaf. If some of these formulas is empty, the corresponding successor becomes a 1 leaf. The root of the tree is split even if φ is unsatisfiable, however, other nodes labeled by an unsatisfiable formula are not split.

Hence, except possibly of the root, there is no other node with two 0-leaves as successors. Since the length of every formula in the tree is at most $\|\varphi\|$, in each node, time $O(c(\varphi))$ is sufficient to choose a splitting variable, determine which of the successors is a leaf, and construct the formulas for the successors of the node.

Let us assume that u is a non-leaf node of the constructed tree different from the root. One of the successors of u can be labeled by an unsatisfiable formula. This is recognized by the splitting algorithm and this successor is a 0-leaf. Consequently, in time at most $O(c(\varphi))$, the construction of the tree continues at a satisfiable successor of u. Hence, in at most n splitting steps and time at most $O(n \cdot c(\varphi))$, a 1-leaf is reached. □

Remark 3.4. If φ contains a unit clause and U is closed under 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 3.5. If a class U satisfies that

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

then the splitting problem relative to U has polynomial 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. Class U with this property is sometimes also conservative. We can also say that this property is a particular form of self-reducibility (in a sense considered e.g. by Khuller & Vazirani, 1991). All classes of generalized satisfiability problem described by Creignou and Hébrard (1997) have this property in addition to other classes, consider, for instance, Horn formulas, SLUR formulas, 2CNFs, q-Horn formulas, etc. As an immediate corollary of Theorem 3.3, it is possible to generate the models of formulas in these classes with a polynomial delay.

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

4. 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 by 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 4.1. A *pure literal sequence* for a formula φ is a consistent sequence of literals $(l_1, \dots, l_k)$, such that for every $i = 1, \dots, k$, the literal l_i is either pure or irrelevant in the formula $\varphi[l_1, \dots, l_{i-1}]$. 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 $\varphi[l_1, \dots, l_{i-1}]$.

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 4.2. 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. Autarkies were studied e.g. by 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 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 3.5 and we have to put more effort into showing that pure literal satisfiable formulas have the splitting property and that the splitting problem relative to pure literal satisfiable formulas has polynomial 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 Algorithm 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 Corollary 4.4, the resulting reduced formula is uniquely determined by the input.

Lemma 4.3. *If a clause C of a CNF φ is removed by some run of* FINDPLS*, then it is removed by every run of* FINDPLS *with input φ.*

Proof. Let L and K be pure literal sequences produced by different runs of FINDPLS for φ. The formulas $\varphi[L]$ and $\varphi[K]$ are the corresponding reduced formulas and let C be a clause of φ not contained in $\varphi[L]$. Hence, L contains some of the literals of C. Since $\varphi[K]$ is a subset of φ, L is a pure literal sequence for $\varphi[K]$. If some literal of L is contained in $\varphi[K]$, then the first of such literals is pure in $\varphi[K]$. Since $\varphi[K]$ does not contain a pure literal, no literal of L is contained in $\varphi[K]$. In particular, C is not contained in $\varphi[K]$. $\square$

The following is an immediate corollary.

Algorithm 1 Constructing pure literal sequence

Require: A CNF formula φ.

Ensure: A maximal strict pure literal sequence L for φ and the corresponding reduced formula.

procedure FindPLS(φ)
    $\psi \leftarrow \varphi$
    Initialize a new empty list of literals L.
    Initialize Pure(ψ) as a set of pure literals in ψ.
    while Pure(ψ) $\neq \emptyset$ **do**
        Choose a literal l from Pure(ψ).
        Add l to L.
        $\psi \leftarrow \psi[l]$.
        Update Pure(ψ) to consist of pure literals in ψ.
    end while
end procedure

Corollary 4.4. *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 procedure FindPLS 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 6.1.

Lemma 4.5. *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 relative to pure literal satisfiable formulas 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.*

Proof. Let ψ be one of the formulas $\varphi[x_i = 0]$ and $\varphi[x_i = 1]$ and let $L' = (l_1, \ldots, l_{i-1})$. Clearly, L' is a pure literal sequence for ψ. Moreover, if ψ does not contain an empty clause, then L' assigns a value to some of the literals in every clause of ψ and hence, satisfies it. □

For now it is sufficient to show that the splitting problem relative to class of pure literal satisfiable formulas has polynomial complexity. Later in Theorem 6.2 we shall show that the splitting problem can in this case be solved in time $O(\|\varphi\|)$.

Lemma 4.6. *The splitting problem relative to class of pure literal satisfiable formulas has polynomial 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 4.5. 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 4.5 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 3.3, see Corollary 6.2.

Remark 4.7. 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.

5. 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 5.1. *The problem of determining $|T(\varphi)|$ given a matched formula φ is #P-complete.*

Proof. Let $\psi = C_1 \wedge C_2 \wedge \ldots \wedge C_m$ be an arbitrary CNF formula on n variables. Let $y_1, \ldots, y_m$ be new variables not appearing in ψ and let $D = (y_1 \vee y_2 \vee \ldots \vee y_m)$ be a clause. Let us define a CNF formula of $n + m$ variables φ equivalent to $\psi \vee D$ as

$$\varphi = (C_1 \vee D) \wedge (C_2 \vee D) \wedge \ldots \wedge (C_m \vee D) \ .$$

Clearly, φ is a matched formula and one can also observe that $|T(\psi)| = |T(\varphi)| - 2^n(2^m - 1)$. We have thus reduced the problem of counting the models of a general CNF formula ψ (i.e. the general #SAT problem) to the problem of counting the models of a matched CNF formula φ (i.e. the #SAT problem restricted to the matched formulas). □

Our goal is to show that we can generate the models of a matched formula with a polynomial delay. Theorem 3.3 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 5.2. A formula φ is called *pure literal matched*, if the reduced formula obtained by procedure FINDPLS 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. Later we shall show in Corollary 6.4 that the splitting problem relative to pure literal matched formulas can be solved in time $O(n^2 \cdot \|\varphi\|)$.

Theorem 5.3. *The class of pure literal matched formulas has the splitting property and the splitting problem relative to pure literal matched formulas has polynomial complexity.*

In order to prove Theorem 5.3, we have to show several statements concerning the structure of a matched formula. If V is a set of variables, we say that a clause is *limited* to V, if it contains only literals with variables from V.

Definition 5.4. Let V be a subset of the variables of a matched formula φ and let $\mathcal{C}$ denote the set of clauses in φ which are limited to V. The set V will be called a *critical block*, if $|\mathcal{C}| = |V|$. 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 limited to V, then by Hall's theorem (Theorem 2.1 above) we have $|\mathcal{C}| \leq \Gamma(\mathcal{C}) \leq |V|$. Critical blocks are those achieving the equality. These blocks have the following property.

Lemma 5.5. *Let V be a critical block of a matched formula φ. Then, in every clause saturated matching of $I(\varphi)$, the variables from V are matched to clauses limited to V.*

Proof. Let φ be a matched formula with a fixed clause saturated matching between the variables and the clauses of φ. If V is a critical block, then there are $|V|$ clauses limited to V and these clauses are matched to the variables in V. Since the variables matched to these clauses are different, each of the variables in V is matched to one of these clauses. □

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

Lemma 5.6. *The set of the critical blocks of a matched formula is closed under intersection.*

Proof. Let φ be a matched formula and let V_1, V_2 be critical blocks. If the intersection $V_1 \cap V_2$ is empty, the conclusion of the lemma is satisfied. If there is a variable $x \in V_1 \cap V_2$, then by Lemma 5.5, in every clause saturated matching of $I(\varphi)$, this variable is matched to a clause, which is limited to V_1 and also to V_2. Hence, the number of the clauses, which are limited to $V_1 \cap V_2$, is at least $|V_1 \cap V_2|$. Since φ is matched, the number of these clauses is equal to $|V_1 \cap V_2|$ by Hall's theorem. Hence, $V_1 \cap V_2$ is a critical block as required. □

If φ is a formula and x is a variable contained in at least one critical block, then Lemma 5.6 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 5.7. If φ is a matched formula with the same number of clauses and variables and x is one 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. Our aim is to show that if a formula φ is matched, then either we can find a splitting variable for φ relative to matched formulas, or φ is actually pure literal satisfiable. In order to show this property which is at the basis of our algorithm, we shall first investigate the structure of critical blocks with respect to matchings.

Lemma 5.8. *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 limited 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).*

Proof. By symmetry, we shall consider only the case $l = x$. Hence, by the assumptions, $\varphi[x = 0]$ is not a matched formula.

1. The critical block B_x is a subset of every critical block containing x. Hence, in order to prove the first part of the lemma, it is sufficient to show that there is at least one critical block B containing x, such that x does not occur negatively in the clauses limited to B. Let $\mathcal{C}$ be any set of the clauses for which the Hall's condition for the formula $\varphi[x = 0]$ is not satisfied so we have $|\Gamma(\mathcal{C})| < |\mathcal{C}|$. Let $V = \Gamma(\mathcal{C})$ be the set of the variables, which have an occurrence in some of the clauses of $\mathcal{C}$, and let $k = |V|$. There are at least $k + 1$ clauses in $\mathcal{C}$. Since every clause of $\mathcal{C}$ is limited to V, there are at least $k + 1$ clauses of $\varphi[x = 0]$ limited to V. Each of these clauses is either a clause of φ or is obtained from a clause of φ by removing the literal x. Consider the set of the clauses of φ limited to $V \cup \{x\}$. Since φ is matched, the Hall's condition is satisfied for this set. Hence, φ contains at most $k + 1$ clauses limited to $V \cup \{x\}$. Setting $x = 0$ leads to at least $k+1$ clauses limited to V. Hence, φ contains precisely $k + 1$ clauses limited to $V \cup \{x\}$ and none of them contains the literal $\neg x$. Hence, $V \cup \{x\}$ is a critical block with the required property and the proof of the first part of the lemma is finished.

2. Let us fix a clause saturated matching M of clauses to variables in $I(\varphi)$ and let D be a clause in φ which is matched to x. Since $\varphi[x = 0]$ is not matched, it follows that D contains the positive literal x, otherwise the same matching would work for $\varphi[x = 0]$ as well. Let C be a clause containing $\neg x$ and let y be a variable to which C is matched in M. Since C is different from D, we have $y \neq x$. By the assumptions, the set of all variables is a critical block for φ and, hence, the critical block B_y is well-defined. Since C is matched to y, we have that C is limited to B_y by Lemma 5.5. This implies $x \in B_y$, because $\neg x \in C$. Since B_y is a critical block containing x and B_x is the inclusion minimal critical block containing x, $B_x \subseteq B_y$. By the first part of the lemma, no clause limited to B_x contains $\neg x$ which implies that C is not limited to B_x and thus $B_x \neq B_y$. Together we get that $B_x \subset B_y$.

□

The structure of the critical blocks will be used to show the following proposition needed to prove Theorem 5.3.

Theorem 5.9. *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. Let φ be a matched formula satisfying the assumptions and let us fix a clause saturated matching M of $I(\varphi)$. If there is a variable x in φ which is not matched to a clause, then assigning any value to x yields a matched formula. By assumption we can therefore suppose that such variable does not exist in φ and that each variable is matched to a clause. In this case, the numbers of clauses and variables of φ are equal and for each variable x in φ, B_x is well-defined.

Let n be the number of the variables and the clauses of φ. For each $i = 1, \ldots, n$, let l_i be the literal containing the variable x_i in the clause matched to this variable. For every $i = 1, \ldots, n$, the formula $\varphi[l_i]$ is matched and the formula $\varphi[\neg l_i]$ is not matched. Consider the strict partial order on the variables defined by

$$x < y \Leftrightarrow B_x \subset B_y \tag{1}$$

where $\subset$ means a strict inclusion. By Lemma 5.8, the variables which are maximal in this partial order are pure in φ. Let us consider a total ordering of the variables, which is consistent with the strict partial order (1). Using an appropriate renaming of the variables, we may assume that this ordering is $x_1, \ldots, x_n$, so for every i, j, if $x_i < x_j$, then $i < j$. Let us verify that using this ordering, the sequence $l_n, l_{n-1}, \ldots, l_1$ is a satisfying pure literal sequence for φ. Let us show by induction on $i = n, \ldots, 1$ that x_i is pure or irrelevant in the formula $\varphi[l_n, \ldots, l_{i+1}]$. It is true for $i = n$ by Lemma 5.8, because x_n is maximal in the order of variables induced by the inclusion of their critical blocks. Let us now fix some i and consider the partial assignment $\mathrm{assign}(l_n, \ldots, l_{i+1})$. By Lemma 5.8, each clause containing $\neg l_i$ is matched to a variable x_j satisfying $x_i < x_j$. Hence, these clauses are eliminated by the considered partial assignment and the variable x_i is pure or irrelevant in the formula $\varphi[l_n, \ldots, l_{i+1}]$. □

Proof of Theorem 5.3. Assume, φ is a pure literal matched formula. Let L be a pure literal sequence obtained by FindPLS 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 4.6. If ψ is not empty, then it is matched and not pure literal satisfiable. Hence, by Theorem 5.9, 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 $\text{assign}(L)$ 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 $\text{assign}(L)$ 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 4.6 is used. Hence, a splitting variable and the results of the required satisfiability tests can be obtained in polynomial time. □

Similarly as the class of matched formulas, also the class of pure literal matched formulas is closed under unit propagation. This implies that unit propagation can be used as part of the construction of the splitting tree, in particular by Remark 3.4 we can always select a variable in a unit clause as a splitting variable.

Proposition 5.10. *The class of pure literal matched formulas is closed under unit propagation.*

Proof. Assume, φ is a pure literal matched formula containing a unit clause $C = (l)$ where l is a literal. Let us prove that $\varphi[l]$ is a pure literal matched formula.

Let L be a pure literal sequence for φ. Observe that $\neg l$ cannot be contained in L, because $\varphi[\neg l]$ is unsatisfiable. In the rest of the proof, we distinguish, whether l is contained in L or not.

If l is contained in L, let L_1 denote the sequence of literals in L before l and let L_2 be the sequence of literals in L after l. For simplicity, this can be written as $L = (L_1, l, L_2)$. Some of the clauses of φ are missing in $\varphi[l]$ and some are changed by removing $\neg l$. Since l is not contained in L_1, the sequence L_1 is a pure literal sequence for $\varphi[l]$. Since any assignments to disjoint sets of variables commute, we have $\varphi[L_1, l] = \varphi[l, L_1]$ and, hence, the sequence L_2 is a pure literal sequence for both these formulas. Hence, the sequence $L' = (L_1, L_2)$ is a pure literal sequence for $\varphi[l]$. Since, moreover, $\varphi[L_1, l, L_2] = \varphi[l, L_1, L_2]$, the application of L' to $\varphi[l]$ leads to a matched formula. Consequently, $\varphi[l]$ is pure literal matched.

Let us now consider the case when l is not contained in L. In this case $\varphi[L]$ is a matched formula which contains a unit clause $C = (l)$, since this clause cannot be eliminated by satisfying any of the literals in L. In every maximum matching of $\varphi[L]$, clause C is matched to l. Thus satisfying l gives a matched formula $\varphi[L, l]$. Since $\varphi[L, l] = \varphi[l, L]$ and L is a pure literal sequence for $\varphi[l]$, this formula is pure literal matched. □

6. 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. The data structures used in the algorithms are similar to those described by Minoux (1988) or Murakami and Uno (2014). Let us first concentrate on the pure literal satisfiable formulas.

Lemma 6.1. *A maximal pure literal sequence L for a CNF formula φ can be constructed in time $O(\|\varphi\|)$.*

Proof. We use the approach presented in the linear time algorithm for unit propagation by Minoux (1988) to obtain an efficient version of procedure FindPLS in Algorithm 1. In addition to the initializations in Algorithm 1, we initialize some auxiliary data structures. These data structures are similar to those described by Murakami and Uno (2014). In particular, the occurences of the literals in the formula are represented as nodes arranged as a sparse matrix, whose rows correspond to literals and columns correspond to clauses. Each node contains an identification of the clause and the literal, whose occurence it represents. All the auxiliary data structures and their names are as follows:

- For each literal l we denote $cl(l)$ the row of the matrix, which is a doubly-linked list of nodes representing the occurences of l in ψ.
- For each clause $C \in \psi$ we denote $lit(C)$ the column of the matrix, which is a doubly-linked list of nodes corresponding to the occurences of literals in C.
- For each literal l we denote $cnt(l)$ a counter, which contains the size of list $lit(C)$ that is the number of clauses in which l appears.
- We initialize the set $\mathrm{Pure}(\psi)$ as a queue which always contains pure literals in ψ. These are the literals l for which $cnt(l) > 0$ and $cnt(\neg l) = 0$.

All these data structures can be initialized by traversing ψ in linear time. It is important to note that each node represents an occurence of a literal l in a clause C. As such the structure representing the node contains four pointers, two for doubly-linked list $lit(C)$ and two for double-linked list $cl(l)$. Thus removing this node from any of these lists can be performed in constant time.

In procedure FindPLS we repeat the following steps – find a pure literal l in ψ, add l to L and apply $\mathrm{assign}(l)$ to ψ. Finding a pure literal amounts to dequeueing it from $\mathrm{Pure}(\psi)$. When applying $\mathrm{assign}(l)$ we remove all clauses containing l (these are now satisfied) and remove $\neg l$ from the remaining clauses. Let ψ_1 consist of clauses in ψ which contain l and let ψ_0 consist of clauses in ψ which contain $\neg l$. We claim that $\mathrm{assign}(l)$ can be applied to ψ in time $O(\|\psi_1\| + |\psi_0|)$.

1. Removing clauses in ψ_1 means going through the list $cl(l)$ and for each clause C in this list and each literal l' in $lit(C)$ (including l), remove the corresponding node from $cl(l')$ and make the list $lit(C)$ inaccessible. This requires time $O(1)$ for each literal l'. During this operation we also decrement the counters $cnt(l')$ of literals in $lit(C)$ and if any of their negated counterparts becomes pure, we add it to queue $\mathrm{Pure}(\psi)$.
2. Removing all occurrences of $\neg l$ means going through the list $cl(\neg l)$ and for each clause C in this list, remove the corresponding node from $cl(\neg l)$ and from $lit(C)$. This can be done in time $O(1)$ for each occurrence of $\neg l$.

Repeating these steps for all literals which are included into L requires a constant number of operations on each occurrence of a literal in the input formula φ which implies the total time $O(\|\varphi\|)$. $\square$

Theorem 6.2. *The splitting problem relative to pure literal satisfiable formulas can be solved in time $O(\|\varphi\|)$ where φ is the input pure literal satisfiable formula. Moreover, the set $T(\varphi)$ of the models of a pure literal satisfiable formula φ can be generated with a delay of $O(\|\varphi\|)$.*

Proof. Using the efficient version of FindPLS guaranteed by Lemma 6.1, all the operations used in the proof of Lemma 4.6 can be done in time $O(\|\varphi\|)$. This implies the first statement of the theorem. The same procedure will be used as a preprocessing step for the algorithm proving the second statement. In time $O(\|\varphi\|)$, the preprocessing produces a pure literal sequence $L = (l_1, \ldots, l_n)$, which contains a literal for each variable of φ. The auxiliary data structures $cl(l)$, $lit(C)$ and $cnt(l)$ used in the preprocessing will be used also later, so they can be stored or reconstructed when needed.

By construction of L, the assumption of Lemma 4.5 is satisfied for φ and L. If the method from Lemma 4.5 is used to find a splitting variable for such a formula, then each of the corresponding restrictions either contains an empty clause or also satisfies the assumption of Lemma 4.5 with L. Hence, the sequence L can be used for selecting a splitting variable in all nodes of a splitting tree for φ.

The DFS search is controlled by a stack of postponed nodes, which is initialized with the root before the search starts. The search is split into a sequence of descending branches. Each of the descending branches starts by removing a node from the stack and resuming the search from this node. If a visited node has two satisfiable successors, DFS continues to one of them and the other is put onto the stack. If a node has a single satisfiable successor, then the stack is not modified. Each descending branch ends when a 1-leaf is found. For an estimate of the delay, we estimate the total time needed to construct nodes in one of these descending branches as follows.

The indices in L of the splitting variables chosen in a descending branch are monotonically decreasing. Hence, the total time needed to search for all the splitting variables in one descending branch is $O(n)$ and, hence, $O(\|\varphi\|)$.

The time needed for manipulations with the formula in one descending branch is as follows. When a node is removed from the stack, the auxiliary data structures $cl(l)$, $lit(C)$ and $cnt(l)$ are computed for the original formula φ and then modified according to the sequence of settings of the variables along the path from the root to the current node. This can be done in time $O(\|\varphi\|)$. Then, in each node of the descending branch, both assignments of the chosen variable are computed and a satisfiable successor is selected. In one node, this can be done in time $O(k)$, where k is the number of the occurrences of the chosen variable in φ. When a satisfiable successor is selected, the auxiliary structures are updated according to it. The total time needed for these operations in one descending branch is $O(\|\varphi\|)$ using a similar argument as in the proof of Corollary 6.1.

By combining the above estimates, the total time for constructing a descending branch and, hence, the delay between generating two consecutive models, is $O(\|\varphi\|)$. $\square$

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

Lemma 6.3. *The splitting problem relative to pure literal matched formulas can be solved in time $O(n \cdot \|\varphi\|)$ where φ is the input formula on n variables.*

Proof. Following the proof of Theorem 5.3, we first find a pure literal sequence L for φ which can be done in time $O(\|\varphi\|)$ by Lemma 6.1. If $\psi = \varphi[L]$ is an empty formula, the last variable in L is a splitting variable. Otherwise ψ is matched and we find a maximum matching M for ψ. This step can be performed in time $O(\|\varphi\| \cdot \sqrt{n})$ (see Hopcroft & Karp, 1973). Then, we search for a variable x in ψ, such that both $\psi[x = 0]$ and $\psi[x = 1]$ are matched. Such a variable exists by Theorem 5.9. If the number of the clauses of φ is less than n, any variable not used in the matching has this property. Otherwise, we check for every variable, whether $\psi[x = a]$ is matched for $a \in \{0, 1\}$. If the assignment $x = a$ satisfies the matched literal containing x, then $\psi[x = a]$ is matched. In the rest of the proof, we estimate the complexity of each of the at most n checks for the assignments falsifying a matched literal.

Partial assignment can be performed in time $O(\|\psi\|) = O(\|\varphi\|)$. During partial assignment the satisfied clauses are removed and the occurences of variable x are removed from the remaining clauses. We modify matching M into a matching N for $\psi[x = a]$ accordingly, that is we remove pairs containing a satisfied clause and the pair containing x. If in $|N| = m$ (where m is the number of clauses in $\psi[x = a]$), we are done. Otherwise we know that $|N| = m - 1$, since at most one pair containing a clause of $\psi[x = a]$, specifically, the pair containing a literal on x, was removed from M when forming N. It remains to check whether N is already a maximum matching or whether there is a better matching. This can be tested by looking for a single augmentating path in $I(\psi[x = a])$ for matching N. An augmentating path can be found using a breadth first search in time linear in the size of the graph $I(\psi[x = a])$ (see e.g. Hopcroft & Karp, 1973; Lovász & Plummer, 1986). Hence, the test, whether $\psi[x = a]$ is matched can be done in time $O(\|\psi\|) = O(\|\varphi\|)$. □

As a corollary of Lemma 6.3 and the general bound from Theorem 3.3, we get the following.

Corollary 6.4. *Models of a pure literal matched formula φ on n variables can be generated with a delay $O(n^2 \cdot \|\varphi\|)$.*

Proof. By Lemma 6.3 we can find a splitting variable for a pure literal matched formula φ in time $O(n \cdot \|\varphi\|)$, in the same time we can determine the satisfiability of formulas $\varphi[x = 0]$ and $\varphi[x = 1]$ as well. By Theorem 3.3 we thus get that the delay is $O(n^2 \cdot \|\varphi\|)$. □

7. 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, the splitting problem relative to linearly satisfiable formulas does not have polynomial complexity unless P=NP. This consequence follows also unconditionally from Example 7.10, which presents a linearly satisfiable formula of 4 variables, which does not have a splitting variable with respect to the class of linearly satisfiable formulas.

Let us recall the notation introduced by Kullmann, which is used below to present the definition and basic facts concerning the linearly satisfiable formulas. If l is a literal, then $\mathrm{var}(l)$ is the variable in this literal. If v is a partial assignment, then $v(l)$ is the value of this assignment on literal l.

Definition 7.1 (Kullmann, 2000)**.** Let φ be a CNF formula and let v be a non-empty partial assignment of the variables of φ. We say that v is a *simple linear autarky*, if there is an *associated weight function* w which assigns each variable x evaluated by v a positive real number $w(x)$ such that for all clauses C of φ we have

$$\sum_{l \in C, v(l)=1} w(\mathrm{var}(l)) \geq \sum_{l \in C, v(l)=0} w(\mathrm{var}(l)) \,. \tag{2}$$

Clearly, if any literal in C is falsified by v, then there must be a literal satisfied by v as well. Therefore a simple linear autarky is an autarky. Kullmann showed that we can check whether there is a simple linear autarky v for a CNF formula φ and find one, if it exists, by solving several linear programs.

If a literal l is pure in a formula, then the partial assignment $v(l) = 1$ with the weight $w(\mathrm{var}(l)) = 1$ is a simple linear autarky for the formula. As another example, consider any satisfying assignment of a satisfiable 2-CNFs. Such an assignment with the same weight for all variables forms a simple linear autarky. Similarly, pure Horn CNFs without unit clauses are satisfiable by a simple linear autarky which assigns value 0 and equal weight to all variables. On the other hand, if a pure Horn CNF formula contains a unit clause, it can be satisfiable and have no simple linear autarky. An example is the formula

$$(x_1) \wedge (\overline{x_1} \vee x_2) \wedge (\overline{x_1} \vee x_3) \wedge (x_1 \vee \overline{x_2} \vee \overline{x_3}) \ ,$$

which has no simple linear autarky by Theorem 7.8 and Lemma 7.9 below.

By considering iterative application of simple linear autarkies to a formula we can get the class of linearly satisfiable formulas defined as follows.

Definition 7.2 (Kullmann, 2000)**.** The class of *linearly satisfiable formulas* is defined as the smallest class satisfying the following two properties:

1. An empty CNF is linearly satisfiable.

2. Let φ be a CNF, such that there is a simple linear autarky v for φ and $\varphi[v]$ is linearly satisfiable. Then so is φ.

In other words, a CNF formula φ is linearly satisfiable if by subsequent applications of linear autarkies we obtain an empty formula. A composition of simple linear autarkies is called *linear autarky* by Kullmann (2000) and the class of linearly satisfiable formulas therefore consists of formulas which are satisfiable by a linear autarky. Kullmann showed that all matched formulas are linearly satisfiable. Since a pure literal is a simple linear autarky, any pure literal satisfiable formula is linearly satisfiable. Similarly, any pure literal matched formula defined in Section 5 is linearly satisfiable by simple linear autarkies for the pure literals concatenated with the linear autarky for the resulting matched formula.

While for matched and pure literal satisfiable formulas we have presented algorithms which generate models of these formulas with polynomial delay, it is not possible to extend this result to linearly satisfiable formulas unless P=NP. Let us first present a construction, which is used in a reduction argument.

Let φ be an arbitrary 3-CNF formula with variables $x_1, \ldots, x_n$ and clauses $c_1, \ldots, c_m$. Consider new variables y_1, y_2, y_3 and let α be the formula consisting of the clauses

$$\begin{array}{ll} (y_1 \vee \overline{y_2}),\ (y_2 \vee \overline{y_3}),\ (y_3 \vee \overline{y_1}), & \\ (c_j \vee y_1 \vee y_2 \vee y_3), & j = 1, \ldots, m \\ (x_i \vee \overline{y_1}) & i = 1, \ldots, n\ . \end{array}$$

Recall that the number of the models of a formula is the number of the satisfying assignments of the variables, which have an occurrence in it. Hence, in the next lemma, $T(\alpha)$ and $T(\varphi)$ are defined on different sets of the variables.

Lemma 7.3. *Formula α is linearly satisfiable and the number of its models is $|T(\alpha)| = |T(\varphi)| + 1$.*

Proof. Each clause c_i of φ contains three literals. Hence, in each clause of α, the number of the positive literals is at least the number of the negative literals. It follows that the assignment of all variables to 1 with equal weight for all variables defines a simple linear autarky, which satisfies α. Hence, this formula is linearly satisfiable.

Any model of α satisfies $y_1 = y_2 = y_3$. An assignment containing $y_1 = y_2 = y_3 = 1$ is a model of α if and only if $x_i = 1$ for $i = 1, \ldots, n$. An assignment containing $y_1 = y_2 = y_3 = 0$ is a model of α if and only if the assignment of the variables x_i is a model of φ. This implies the second part of the statement of the lemma. □

Since the formula α can be constructed for every 3-CNF formula φ, the lemma implies the following immediate corollary.

Corollary 7.4. *It is an NP-complete problem to determine, whether a general linearly satisfiable formula has at least* 2 *models.*

Note that this implies NP-hardness of #SAT problem restricted to the linearly satisfiable formulas. This problem is, in fact, also #P-complete, since it is #P-complete to count models of monotone formulas, which are pure literal satisfiable and, hence, linearly satisfiable.

In Example 7.10 below, we present a linearly satisfiable formula, which has no splitting variable relative to the class of linearly satisfiable formulas. For analysis of this example, we use a characterization of simple linear autarkies obtained using the clause-variable matrix.

Definition 7.5. Let φ be a CNF formula with clauses $c_1, \ldots, c_m$ and variables $x_1, \ldots, x_n$. The clause-variable matrix of this formula is the matrix $A = \{a_{j,i}\}$ of the dimension $m \times n$ defined as

$$a_{j,i} = \begin{cases} 1 & x_i \in c_j \\ -1 & \neg x_i \in c_j \\ 0 & \text{otherwise}\ . \end{cases}$$

If $u \in \mathrm{R}^m$, then $u \geq 0$ means $u_j \geq 0$ for all $j = 1, \ldots, m$. Kullmann showed the following proposition.

Lemma 7.6 (Kullmann, 2000)**.** *A formula φ with the clause-variable matrix A has a simple linear autarky, if and only if there is a nonzero $z \in \mathrm{R}^n$, such that $Az \geq 0$. Moreover, a linear autarky can be obtained from such a vector z using the assignment*

$$v(x_i) = \begin{cases} 1 & \text{if } z_i > 0 \\ 0 & \text{if } z_i < 0 \\ * & \text{if } z_i = 0 \end{cases}$$

and the weight function $w(x_i) = |z_i|$.

Let us present the well-known Farkas lemma in the form used in the proof of Theorem 7.8.

Theorem 7.7 (Farkas lemma)**.** *Let A be an $m \times n$ real matrix and $b \in \mathrm{R}^n$. Then, exactly one of the following statements is true.*

1. *There is a vector $y \in \mathrm{R}^m$, such that $y \geq 0$ and $y^t A = b^t$.*
2. *There is a vector $z \in \mathrm{R}^n$, such that $Az \geq 0$ and $b^t z < 0$.*

A linear combination of real vectors with non-negative coefficients will be called, for simplicity, a non-negative combination.

Theorem 7.8. *Assume, φ is a formula of n variables and m clauses and A is its clause-variable matrix of the dimension $m \times n$. Then, exactly one of the following statements is true:*

(a) φ has a linear autarky,

(b) every vector in R^n is a non-negative combination of the rows of A.

Proof. First, assume, both (a) and (b) are satisfied. Lemma 7.6 implies that there is a non-zero $z \in \mathrm{R}^n$, such that $Az \geq 0$. By (b), there is a non-negative vector $y \in \mathrm{R}^m$, such that $y^t A = -z^t$. Multiplying this by z from the right, we get

$$y^t Az = -z^t z < 0 \ .$$

This is a contradiction, since both y and Az are non-negative.

Assume, (b) is not satisfied. Hence, there is a vector $b \in \mathrm{R}^n$, which is not a non-negative combination of the rows of A. By Farkas lemma, there is a vector $z \in \mathrm{R}^n$, such that $Az \geq 0$ and $b^t z < 0$. Since the latter condition implies that z is non-zero, there is a simple linear autarky for φ by Lemma 7.6 which means that (a) is satisfied. □

Lemma 7.9. *Assume, A is a matrix of dimension $m \times n$, such that* $\mathrm{rank}(A) = n$ *and there is a vector $u \in \mathrm{R}^m$ with all components positive, such that $u^t A = 0$. Then, every vector in R^n is a non-negative combination of the rows of A.*

Proof. By the assumption, the linear space generated by the rows of A is R^n. Hence, for every $z \in \mathrm{R}^n$, there is $v \in \mathrm{R}^m$, such that $v^t A = z$. For a sufficiently large real number s, the vector $v + su$ has all components non-negative and $(v + su)^t A = z$. □

Note that every linearly satisfiable CNF formula of at most 3 variables has a splitting variable relative to the class of linearly satisfiable CNF formulas, since setting any variable to a constant leads to a formula of at most 2 variables, which is at most quadratic and, hence, is satisfiable if and only if it is linearly satisfiable.

Example 7.10. *Denote* $E = \{a \in \{0,1\}^4 \mid 2 \le a_1+a_2+a_3+a_4 \le 3\}$ *and for every Boolean variable* x*, let* $x^1 = x$ *and* $x^0 = \overline{x}$*. The formula*

$$\beta(x_1, x_2, x_3, x_4) = \bigwedge_{a \in E} \bigvee_{i=1}^{4} x_i^{a_i}$$

is linearly satisfiable, but has no splitting variable relative to the class of linearly satisfiable formulas.

Proof. In every clause, the number of positive literals is at least the number of negative literals. Hence, the formula β is linearly satisfiable by Lemma 7.6 with $z = (1,1,1,1)$.

Since β is invariant under any permutation of the variables, it is sufficient to prove that x_4 is not a splitting variable. Since every clause of β contains a negative literal, we have $\beta(0,0,0,0) = 1$. It follows that the formula $\beta[x_4 = 0]$ is satisfiable. One can verify that

$$\beta[x_4 = 0] = \bigwedge_{a \in E'} \bigvee_{i=1}^{3} x_i^{a_i} \ ,$$

where $E' = \{a \in \{0,1\}^3 \mid 1 \le a_1 + a_2 + a_3 \le 2\}$. In order to prove that $\beta[x_4 = 0]$ is not linearly satisfiable, consider its clause-variable matrix with the columns corresponding to x_1, x_2, x_3, which is

$$\begin{pmatrix} 1 & 1 & -1 \\ 1 & -1 & 1 \\ -1 & 1 & 1 \\ 1 & -1 & -1 \\ -1 & 1 & -1 \\ -1 & -1 & 1 \end{pmatrix} .$$

This matrix has rank 3, since each of the vectors $(2,0,0)$, $(0,2,0)$, $(0,0,2)$ is a sum of two rows out of the first three. Moreover, the sum of all the rows of this matrix is the zero vector. Hence, the formula $\beta[x_4 = 0]$ does not have a linear autarky by Lemma 7.9 and Theorem 7.8. □

8. 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. Let us assume that 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).

The algorithms described in this paper for the cases of pure literal satisfiable and pure literal matched formulas can be used in a general algorithm for model enumeration which is based on splitting tree. This, in turn, is any DPLL based enumeration algorithm. To this end, a similar approach to the one described by Stefan Szeider (2003) can be used. Together with a formula φ we would keep a maximum matching M of $I(\varphi)$. This maximum matching can then be maintained through the reduction and assignment steps performed in the enumeration algorithm. Once the algorithm arrives at a matched formula, it can select splitting variables in the way we have described in this paper which has guaranteed polynomial delay.

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.

Acknowledgments

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

Aceto, L., Monica, D., Ingólfsdóttir, A., Montanari, A., & Sciavicco, G. (2013). *Logic for Programming, Artificial Intelligence, and Reasoning: 19th International Conference, LPAR-19, Stellenbosch, South Africa, December 14-19, 2013. Proceedings*, chap. An Algorithm for Enumerating Maximal Models of Horn Theories with an Application to Modal Logics, pp. 1–17. Springer Berlin Heidelberg, Berlin, Heidelberg.

Aharoni, R., & 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*, Vol. 184 of *Graduate Texts in Mathematics*. Springer.

Coquery, E., Jabbour, S., Sais, L., Salhi, Y., et al. (2012). A SAT-based approach for discovering frequent, closed and maximal patterns in a sequence. In *Proceedings of ECAI*.

Creignou, N., & Hébrard, J.-J. (1997). On generating all solutions of generalized satisfiability problems. *Informatique théorique et applications*, *31*(6), 499–511.

Creignou, N., Olive, F., & Schmidt, J. (2011). *Theory and Applications of Satisfiability Testing - SAT 2011: 14th International Conference, SAT 2011, Ann Arbor, MI, USA, June 19-22, 2011. Proceedings*, chap. Enumerating All Solutions of a Boolean CSP by Non-decreasing Weight, pp. 120–133. Springer Berlin Heidelberg, Berlin, Heidelberg.

Dechter, R., & Itai, A. (1992). Finding all solutions if you can find one. In *AAAI-92 Workshop on Tractable Reasoning*, pp. 35–39.

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

Flum, J., & Grohe, M. (2006). *Parameterized complexity theory* (1st edition)., Vol. 3 of *Texts in Theoretical Computer Science. An EATCS Series*. Springer-Verlag Berlin Heidelberg.

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

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

Genesereth, M., & Nilsson, N. (1987). *Logical Foundations of Artificial Intelligence*. Morgan Kaufmann, Los Altos, CA.

Hopcroft, J. E., & Karp, R. M. (1973). An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal on computing*, *2*(4), 225–231.

Jabbour, S., Lonlac, J., Sais, L., & Salhi, Y. (2014). Extending modern sat solvers for models enumeration. In *IEEE 15th International Conference on Information Reuse and Integration (IRI), 2014*, pp. 803–810. IEEE.

Johnson, D. S., Yannakakis, M., & Papadimitriou, C. H. (1988). On generating all maximal independent sets. *Information Processing Letters*, *27*(3), 119 – 123.

Kang, H.-J., & Park, I.-C. (2005). Sat-based unbounded symbolic model checking. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, *24*(2), 129–140.

Kavvadias, D. J., Sideri, M., & Stavropoulos, E. C. (2000). Generating all maximal models of a Boolean expression. *Information Processing Letters*, *74*(3–4), 157–162.

Khuller, S., & Vazirani, V. V. (1991). Planar graph coloring is not self-reducible, assuming $P \neq NP$. *Theoretical Computer Science*, *88*(1), 183 – 189.

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., & Plummer, M. D. (1986). *Matching Theory*. North-Holland.

McMillan, K. L. (2002). *Computer Aided Verification: 14th International Conference, CAV 2002 Copenhagen, Denmark, July 27–31, 2002 Proceedings*, chap. Applying SAT Methods in Unbounded Symbolic Model Checking, pp. 250–264. Springer Berlin Heidelberg, Berlin, Heidelberg.

Minoux, M. (1988). LTUR: A simplified linear time unit resolution algorithm for Horn formulae and computer implementation. *Information Processing Letters*, *29*, 1 – 12.

Morgado, A., & Marques-Silva, J. (2005a). Algorithms for propositional model enumeration and counting. Tech. rep., Instituto de Engenharia de Sistemas e Computadores, Investigação e Desenvolvimento, Lisboa.

Morgado, A., & Marques-Silva, J. (2005b). Good learning and implicit model enumeration. In *Tools with Artificial Intelligence, 2005. ICTAI 05. 17th IEEE International Conference on*, pp. 6 pp.–136.

Murakami, K., & Uno, T. (2014). Efficient algorithms for dualizing large-scale hypergraphs. *Discrete Applied Mathematics*, *170*, 83–94.

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

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

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., & Sakallah, K. A. (Eds.), *Theory and Applications of Satisfiability Testing – SAT 2007*, Vol. 4501 of *Lecture Notes in Computer Science*, pp. 94–99. Springer Berlin Heidelberg.

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.