Computational Complexity of SAT, XSAT and NAE-SAT for linear and mixed Horn CNF formulas

Inaugural-Dissertation
zur
Erlangung des Doktorgrades
der Mathematisch-Naturwissenschaftlichen Fakultät
der Universität zu Köln

vorgelegt von
Tatjana Schmidt
aus Nowosibirsk

Köln, 2010

Berichterstatter:
Prof. Dr. Ewald Speckenmeyer
PD Dr. Stefan Porschen

Tag der mündlichen Prüfung: 28.06.2010

Contents

Chapter 1

Introduction

The classical propositional satisfiability problem (SAT) is a prominent problem, namely one of the first problems that have been proven to be NP-complete [15]. Important areas where CNF-SAT plays a vital role are formal verification [49], bounded model checking [14], and artificial intelligence. SAT also plays a fundamental role in computational complexity theory and in the theory of designing exact algorithms. Recently, the interest in designing exact algorithms providing better upper time bounds than the trivial ones for NP-complete problems and their NP-hard optimization counterparts has increased. In this context the investigation of exact algorithms for testing the satisfiability of propositional formulas in conjunctive normal form (CNF) is of particular significance. In addition, SAT is well known to be a fundamental NP-complete problem appearing naturally or via reduction as the abstract core of many application-relevant problems. As over the last years several powerful solvers for SAT have been developed, this is of specific interest (cf. e.g. [33, 11]). In industrial applications the modelling CNF formulas often are of a specific structure and therefore it would be desirable to have fast algorithms for such instances. In this context it turns out that after reducing many classical NP-complete problems to SAT, formulas of a restricted structure are generated, namely formulas $F = P \wedge H$ of a positive monotone 2-CNF part P and a Horn part H, called mixed Horn formulas (MHF) according to [43]. Thus designing good algorithms for solving formulas of this special structure is worthwhile and of great importance. As already shown in [43] graph colorability, problems for level graphs like level-planarity test or the NP-hard crossing-minimization problem [45], can be conveniently formulated in terms of MHF. One purpose of this thesis is to provide a more systematical insight into MHF as destination class and thus to illustrate that MHF has a central relevance in CNF. To that end we provide, in chapter 2, straightforward reductions to MHF-SAT for some prominent NP-complete problems, e.g. the Feedback Vertex Set, the Vertex Cover, the Dominating Set and the Hitting Set problem can easily be en-

coded as MHF.
Furthermore we consider restricted subclasses of MHF and show that they are also NP-complete w.r.t. SAT.
In chapter 2 we also provide algorithms for some NP-complete subclasses of MHF solving SAT significantly faster than in time $O((\sqrt[3]{3})^n) = O(1.443^n)$, the currently best bound for solving unrestricted members of the class MHF [43]. The first of these subclasses consists of formulas, where the Horn part is negative monotone and the variable graph corresponding to the positive 2-CNF part P consists of disjoint triangles only. For this class we provide an algorithm and give the running times for the cases that H is k-uniform, for $k \in \{3, 4, 5, 6, 7, 8\}$. Looking at this subclass of MHF is motivated by the fact that the analysis of an algorithm in [43] for solving unrestricted MHF has its worst-case behaviour just for this class of formulas.
The other NP-complete subclass of MHF actually consists of infinitely many subclasses with parameter $k \geq 3$. For fixed k a worst-case bound of $O(k^{\frac{n}{k}})$ is shown. For $k = 4, 5, 10$ the bases of the exponential growth are $k^{\frac{1}{k}} \approx 1.41, 1.38, 1.259$ resp., going to 1 with k tending to ∞. While the class looks artificial, the derivation of the running time deserves attention. In this case enumerating minimal satisfying assignments of the Horn part of the input formulas turns to be quite useful, whereas for unrestricted MHF and for the first subclass mentioned above enumerating minimal satisfying assignments of the 2-CNF part yields better bounds.
In addition, we consider mixed Horn formulas $F = P \wedge H \in MHF$ for which holds: H is negative monotone, $|c| \leq 3$, for all $c \in H$, and P consists of positive monotone 2-clauses. We solve SAT in running time $O(1.325^n)$ for this formula class by using the autarky principle. That means we can provide a better running time than the so far best running time of $O(p(n) \cdot 1.427^n)$ by S. Kottler, M. Kaufmann and C. Sinz [29] for this class of mixed Horn formulas. Afterwards we consider mixed Horn formulas $F = P \wedge H \in MHF$ for which holds: G_P consists of disjoint triangles, edges and isolated vertices and H consists of Horn clauses which have at most three literals but are not necessarily negative monotone and $V(P) = V(H)$. We can solve SAT in running time $O(1.41^n)$ for this formula class by applying the autarky principle. Furthermore, we present an algorithm which solves SAT for mixed Horn formulas with a linear, negative monotone and k-uniform Horn part and a P part which consists of positive monotone and disjoint 2-clauses only. Experimental results lead to the strong conjecture that its running time is better than $O((\sqrt[3]{3})^n)$, where n is the number of variables.
In chapter 2 we also treat some subclasses of MHF in P. As a matter of fact, there is an interesting connection between MHF-SAT and unrestricted SAT presented in [43]: If there is some $\alpha < \frac{1}{2}$ such that each MHF $M = P \wedge H$, where P has $k \leq 2n$ variables, can be solved in time $O(\|M\| 2^{\alpha k})$, then there is some $\beta \leq 2\alpha < 1$ such that SAT for an arbitrary CNF-formula F can be

decided in time $O(\|F\|2^{\beta n})$. Here $||F||$ denotes the length of F. Although some progress has been made recently in finding non-trivial bounds for SAT for arbitrary CNF formulas [17, 18], it would require a significant breakthrough in our understanding of SAT to obtain upper time bounds of the form $O(2^{(1-\epsilon)n})$, for some $\epsilon > 0$.

Recently, the propositional satisfiability problem (SAT) was shown to be NP-complete when restricted to the class of *linear* formulas in conjunctive normal form (CNF) in [42]. By definition, each pair of distinct clauses of a linear formula has at most one variable in common. Therefore, linear formulas yield a direct generalization of *linear hypergraphs* [4]. Linear formulas overlap only sparsely and there is some evidence that linear formulas form the algorithmically hard kernel for CNF-SAT, making this class specifically interesting especially regarding other variants of SAT.

In chapter 3 of this thesis, we investigate the computational complexity of some well-known variants of SAT, namely, not-all-equal SAT (NAE-SAT) and exact SAT (XSAT) restricted to linear CNF instances. Recall that deciding NAE-SAT, for a CNF formula, means to test for the existence of a truth assignment such that in each clause of the formula at least one literal evaluates to true and at least one to false. For solving XSAT, exactly one literal in each clause must evaluate to true and all others to false. Observe that for CNF formulas where all clauses have exactly two literals XSAT and NAE-SAT coincide. As shown in the seminal paper by Schaefer [47], both NAE-SAT and XSAT are NP-complete for the unrestricted CNF class. Whereas SAT gets trivial on monotone formulas, which by definition are free of negated variables, NAE-SAT and XSAT are well-known to remain NP-complete on that class. Note that monotone NAE-SAT coincides with the prominent NP-complete hypergraph bicolorability problem (also known as *set splitting* [21]); here the existence of a 2-coloring of the vertex set has to be checked such that no hyperedge gets colored monochrome. Moreover, monotone XSAT is closely related to the well-known NP-complete set partitioning problem (SPP) having many applications in combinatorial optimization. Recall that SPP takes as input a set M of elements and a collection $\mathcal{M}$ of subsets of M. It asks for a subfamily $\mathcal{T}$ of $\mathcal{M}$ such that each element of M occurs in exactly one member of $\mathcal{T}$. It is easy to see that the monotone XSAT variant coincides with SPP when the clauses overtake the roles of the elements in M and the variables are regarded as the members of $\mathcal{M}$ in such a way that a variable contains all clauses in which it occurs. Furthermore, monotone XSAT is also closely related to the well-known NP-complete problem Exact Hitting Set [21] which has many applications in combinatorial optimization.

The contributions of the present thesis are as the following: In chapter 3, we show that NAE-SAT and XSAT are NP-complete for monotone and linear formulas, where clauses have length greater or equal k, $k \geq 3$.

Recall that Schaefer's theorem in [47] classifies generalized satisfiability

problems (including XSAT) w.r.t. their complexity. However, this dichotomy theorem does not automatically apply if restrictions on the number of occurrences of variables in CNF formulas are given. E.g. in [32] it is shown that whereas unrestricted k-SAT is NP-complete, for $k \geq 3$, it behaves trivially (i.e. all formulas are satisfiable) if each clause has length exactly k and no variable occurs in more than $f(k)$ clauses; it gets NP-complete if variables are allowed to occur at most $f(k)+1$ times. Here $f(k)$ asymptotically grows as $\lfloor 2^k/(e \cdot k) \rfloor$; this bound has meanwhile been improved by other authors. In chapter 3, we also investigate the computational complexity of XSAT restricted to some subclasses of linear formulas defined through bounding the number of occurrences of variables. Here we prove the NP-completeness of XSAT for CNF formulas which are l-regular meaning that every variable occurs exactly l times, where $l \geq 3$ is a fixed integer. On that basis we can also prove the NP-completeness of XSAT for the subclass of linear and l-regular formulas. This result transfers to the monotone case. Moreover, we provide an algorithm solving XSAT for the subclass of monotone, linear and l-regular formulas faster than the so far best algorithm from J. M. Byskov et al. for CNF-XSAT with a running time of $O(2^{0.2325n})$ [12]. Our algorithm works by consequently reducing the average clause length of a formula whenever setting a variable to 1, until we either obtain a formula with an average clause length of 2 or an x-model for this formula. If an appropriate polynomial-time 2-SAT algorithm returns that the formula is unsatisfiable, then we backtrack until either no backtracking is possible or we have found an x-model. Using some connections to finite projective planes we can also show that XSAT remains NP-complete for linear and l-regular formulas which in addition are l-uniform (all clauses have the same length l) whenever $l = q + 1$, where q is a prime power. Thus XSAT most likely is NP-complete for the other values of $l \geq 3$ too. In addition, we are interested in exact linear formulas: here *each* pair of distinct clauses has exactly one variable in common. We show that NAE-SAT is polynomial-time decidable restricted to exact linear formulas. Reinterpreting this result enables us to give a partial answer to a long-standing open question as mentioned by T. Eiter in [20]: Classify the computational complexity of the symmetrical intersecting unsatisfiability problem (SIM-UNSAT).
The complexity of XSAT on exact linear formulas turns out to be more difficult. In chapter 3 we show the NP-completeness of XSAT for monotone and exact linear formulas which we can also establish for the subclass where clauses have length *at least* k, $k \geq 3$. This is somewhat surprising, since both SAT and not-all-equal SAT are polynomial-time solvable for exact linear formulas [42]. However, a difficulty arises when one tries to transfer the NP-completeness proof to the case when in addition all clauses are required to have length *exactly* k, for arbitrary $k \geq 3$. It might be possible that for these classes XSAT is polynomial-time solvable, which so far we can only show for $k \in \{3, 4, 5, 6\}$.

Finally, this thesis is devoted to studying the problems SAT and #SAT for a subclass of CNF-formulas whose *variable-clause graph* is k-outerplanar, $k \geq 1$. The variable-clause graph G_F of a CNF-formula F is defined as follows: The vertex set of G_F consists of the variables and clauses occurring in F. If a variable x occurs in a clause c, then the vertices x and c are joined by an edge. We consider formulas whose variable-clause graphs are k-outerplanar. A graph is planar if it can be embedded in the plane so that no two edges cross outside a vertex. An embedding of a graph $G = (V, E)$ is *1-outerplanar* if it is planar and all vertices lie on the outer face. For $k \geq 2$, an embedding of a graph $G = (V, E)$ is *k-outerplanar* if it is planar and if all vertices on the outer face are deleted, then a $(k-1)$-outerplanar embedding of the resulting graph is obtained. A graph is *k-outerplanar* if it has a k-outerplanar embedding. Formulas whose variable-clause graphs are k-outerplanar are denoted k-outerplanar. As planar 3-SAT is NP-complete according to D. Lichtenstein [34], the results described below are especially significant.

In chapter 4 of this thesis, we show that SAT can be solved in linear time for the 1-outerplanar formula class. Further we show that #SAT for k-outerplanar CNF formulas with n variables can be solved in time $O(n^{1.7(2k+1)})$ using the separator theorem by Lipton and Tarjan [35], which is in contrast to the $\#P$-hardness of the problem on general formulas. For unrestricted CNF formulas #SAT is known to be #P-complete [51]. In this thesis we first use the separator theorem by Lipton and Tarjan always allowing to partition an n-vertex set of a planar graph into exactly two sets A and B of at most $2n/3$ vertices each, plus a separator set C containing $O(n^{1/2})$ vertices, such that no vertex in A is adjacent to a vertex in B. Thus this separation seems to be more appropriate for our purposes. For 1-outerplanar formulas whose variable-clause graph is either free of cycles or consists of disjoint cycles without any chords we solve #SAT in linear time. Moreover, we prove that every CNF-formula F consisting of n variables whose variable-clause graph G_F is a path has at most ψ_{n+1} many different truth assignments, where ψ_{n+1} is the $(n+1)$st Fibonacci number. In addition we prove in chapter 4 that #SAT for k-circular-levelplanar formulas is solvable in time $O(k \cdot 16^k (2/3)^{5.13 \cdot \log_2 k} n^{5.13})$ by the separator theorem of Lipton and Tarjan [35]. So the class of k-circular-levelplanar formulas is fixed-parameter tractable with respect to the parameter k and thus belongs to the class FPT (see e.g. [19]). While we need polynomial-time to solve #SAT for a k-outerplanar formula F by the separator theorem of Lipton and Tarjan, we can actually solve #SAT in linear time using the technique of a nice tree decomposition of width at most $3k-1$ for the variable-clause graph of a k-outerplanar formula, as introduced by H.L. Bodlaender and T. Kloks in [7]. In [8] it is shown that the tree width of a k-outerplanar graph

is at most $3k-1$, and if the tree width of a graph G is at most $3k-1$, then a nice tree decomposition of width at most $3k-1$ can also be computed in linear time. In chapter 4 we further present an algorithm which uses dynamic programming bottom-up from the leaves to the root in a nice tree decomposition of width at most $3k-1$ of G_F to solve the #SAT problem in linear time for a k-outerplanar formula F. Finally, we consider the class of nested formulas defined by Knuth [27]. Using a graphical description of these formulas by Kratochvil and Krivanek [31], we prove, in chapter 4, that every nested formula belongs to the class of 2-outerplanar formulas. On that basis we show that #SAT can be solved in time $O(n^{8.5})$ by the separator theorem of Lipton & Tarjan and in linear time by the nice tree decomposition technique.

Chapter 2

Mixed Horn Formulas

2.1 Classical NP-complete Problems Encoded as MHF-SAT

In this section we provide reductions from some classical NP-complete problems to SAT and we will show that by complementing all literals of the corresponding SAT-instance F we obtain a mixed Horn instance $\overline{F}$. The Graph Colorability problem is already mentioned as an example of such a problem in [43]. Yet there are many more NP-complete problems which can easily be encoded into MHF-SAT. To demonstrate this, we explicitly transform some classical NP-complete problems, [25], to MHF-SAT, for example the Feedback Vertex Set, the Vertex Cover, the Hitting Set, the Dominating Set problem. Inspecting the reductions of several further NP-complete problems to SAT, as presented in [50], it turns out that most of them can easily be transferred to the MHF form.
The following abbreviations for propositional formulas are helpful for our reduction: The formula *at_most_one* (*at_least_one*) denotes that at most one (at least one) literal of the argument is true:

$$at\_most\_one\{l_1, \ldots, l_x\} := \bigwedge_{1 \le i < j \le x} (\overline{l_i} \vee \overline{l_j})$$
$$at\_least\_one\{l_1, \ldots, l_x\} := (l_1 \vee l_2 \vee \ldots \vee l_x)$$

With these fomulas it is easy to define *exactly_one*, which is true if, and only if, exactly one of its arguments is true:

$$exactly\_one\{l_1, \ldots, l_x\} := at\_most\_one\{l_1, \ldots, l_x\} \wedge at\_least\_one\{l_1, \ldots, l_x\}$$

We begin with the **Feedback Vertex Set** problem, which is defined as follows:
INSTANCE: $G = (V, A)$ a directed graph, $k \le |V|$ positive integer.
QUESTION: Is there a set $V' \subseteq V$, $|V'| \le k$, such that $G - \{V'\}$ has no

directed circles?

The Feedback Vertex Set problem can be reduced to SAT as follows [50]:

$$X = \{[v,i] | v \in V, 1 \leq i \leq |V|\}$$
$$F = \bigwedge_{1 \leq i \leq |V|} exactly\_one\{[v,i] | v \in V\} \wedge \bigwedge_{v \in V} exactly\_one\{[v,i] | 1 \leq i \leq |V|\}$$
$$\wedge \bigwedge_{(u,v) \in A} \bigwedge_{k < i \leq |V|} ([u,i] \Rightarrow at\_least\_one\{[v,j] | 1 \leq j \leq k, i < j \leq |V|\})$$

This reduction results from the fact that any directed acyclic graph (DAG) admits a linear ordering of its vertices such that all arcs are directed from left to right. The first two parts $\bigwedge$ describe a one-to-one labelling of the vertices, where the vertices with labels $\leq k$ are considered to build V'. In the last part $\bigwedge\bigwedge$ the DAG property is checked for the reduced graph. The existence of an edge between a vertex u with a label $i > k$ and a vertex v with a label j, where $k < j < i$, is not permissible. Let α be a satisfying truth assignment of F. Then all the vertices $v \in V$, for which holds $\alpha([v,i]) = 1$, where $1 \leq i \leq k$, belong to the Feedback Vertex Set V' and vice versa. Now we will consider F and show that we can accomplish a reduction to a mixed Horn instance. Let $V = \{v_1, \ldots, v_n\}$ then:

$$F = \bigwedge_{1 \leq i \leq n} \left(([v_1,i] \vee [v_2,i] \vee \ldots \vee [v_n,i]) \wedge \bigwedge_{1 \leq j < l \leq n} (\overline{[v_j,i]} \vee \overline{[v_l,i]}) \right)$$
$$\wedge \bigwedge_{v \in V} \left(([v,1] \vee [v,2] \vee \ldots \vee [v,n]) \wedge \bigwedge_{1 \leq j < l \leq n} (\overline{[v,j]} \vee \overline{[v,l]}) \right)$$
$$\wedge \bigwedge_{(u,v) \in A} \bigwedge_{k < i \leq n} (\overline{[u,i]} \vee ([v,1] \vee [v,2]$$
$$\vee \ldots \vee [v,k] \vee [v,i+1] \vee \ldots \vee [v,n]))$$

Complementing all literals in F we obtain $\overline{F} \in$ MHF:

$$\overline{F} = \bigwedge_{1 \leq i \leq n} \bigwedge_{1 \leq j < l \leq n} ([v_j,i] \vee [v_l,i]) \wedge \bigwedge_{v \in V} \bigwedge_{1 \leq j < l \leq n} ([v,j] \vee [v,l]) \wedge$$
$$\bigwedge_{1 \leq i \leq n} (\overline{[v_1,i]} \vee \overline{[v_2,i]} \vee \ldots \vee \overline{[v_n,i]}) \wedge \bigwedge_{v \in V} (\overline{[v,1]} \vee \overline{[v,2]} \vee \ldots \vee \overline{[v,n]}) \wedge$$
$$\bigwedge_{(u,v) \in A} \bigwedge_{k < i \leq n} ([u,i] \vee (\overline{[v,1]} \vee \overline{[v,2]}$$
$$\vee \ldots \vee \overline{[v,k]} \vee \overline{[v,i+1]} \vee \ldots \vee \overline{[v,n]}))$$

We have:

$$P := \bigwedge_{1\leq i\leq n}\bigwedge_{1\leq j<l\leq n}([v_j,i]\vee[v_l,i])\wedge\bigwedge_{v\in V}\bigwedge_{1\leq j<l\leq n}([v,j]\vee[v,l])$$

and

$$\begin{aligned}H := &\bigwedge_{1\leq i\leq n}(\overline{[v_1,i]}\vee\overline{[v_2,i]}\vee\ldots\vee\overline{[v_n,i]})\wedge\bigwedge_{v\in V}(\overline{[v,1]}\vee\overline{[v,2]}\vee\ldots\vee\overline{[v,n]})\wedge\\ &\bigwedge_{(u,v)\in A}\bigwedge_{k<i\leq n}([u,i]\vee(\overline{[v,1]}\vee\overline{[v,2]}\vee\ldots\vee\overline{[v,k]}\\ &\vee\overline{[v,i+1]}\vee\ldots\vee\overline{[v,n]}))\end{aligned}$$

Obviously $\overline{F} = P \wedge H$ is a mixed Horn formula which is satisfying the following: Let $\overline{\alpha}$ be a satisfying truth assignment of $\overline{F}$, then all the vertices $v \in V$ belong to the FVS V', for which holds: $\overline{\alpha}([v,i]) = 0$, where $1 \leq i \leq k$. Similarly, we can provide a reduction from the Feedback Arc Set problem to MHF-SAT. Recall that in the Feedback Arc Set problem instead of vertices we consider arcs.

Next we move on to the **Vertex Cover** problem being defined as:
INSTANCE: Graph $G = (V, E)$, positive integer $k \leq |V|$.
QUESTION: Is there a set $V' \subseteq V$, $|V'| \leq k$, such that for all $\{u, v\} \in E$ we have $\{u, v\} \cap V' \neq \emptyset$?
The Vertex Cover problem can be reduced to SAT as follows [50]:

$$\begin{aligned}X &= \{[v,i]|v\in V, 1\leq i\leq k\}\\ F &= \bigwedge_{1\leq i\leq k} at\_most\_one\{[v,i]|v\in V\}\\ &\quad\wedge\bigwedge_{(u,v)\in E} at\_least\_one\{[u,i],[v,i]|1\leq i\leq k\}\end{aligned}$$

The first part $\bigwedge$ implies that at most k vertices are chosen (literals $[v, i]$ set to true), and the second part $\bigwedge$ verifies that the chosen vertex set is indeed a vertex cover.
We now consider F and after some calculation we will assert that a reduction to a mixed Horn instance is possible.
Let $V = \{v_1, \ldots, v_n\}$ then we obtain:

$$\begin{aligned}F = &\bigwedge_{1\leq i\leq k}\bigwedge_{1\leq j<l\leq n}(\overline{[v_j,i]}\vee\overline{[v_l,i]})\\ &\wedge\bigwedge_{(u,v)\in E}([u,1]\vee[v,1]\vee[u,2]\vee[v,2]\vee\ldots\vee[u,k]\vee[v,k])\end{aligned}$$

Complementing all literals in F we obtain $\overline{F}$:

$$\overline{F} = \bigwedge_{1\leq i\leq k}\bigwedge_{1\leq j<l\leq n}([v_j,i]\vee[v_l,i]) \\ \wedge \bigwedge_{(u,v)\in E}(\overline{[u,1]}\vee\overline{[v,1]}\vee\overline{[u,2]}\vee\overline{[v,2]}\vee\ldots\vee\overline{[u,k]}\vee\overline{[v,k]})$$

For $\overline{F}$ we have: Let α be a satisfying truth assignment of $\overline{F}$. Then all the vertices belonging to the literals $[v,i]$ to which 0 is assigned form the vertex cover V'. We set

$$P = \bigwedge_{1\leq i\leq k}\bigwedge_{1\leq j<l\leq n}([v_j,i]\vee[v_l,i])$$

and

$$H = \bigwedge_{(u,v)\in E}(\overline{[u,1]}\vee\overline{[v,1]}\vee\overline{[u,2]}\vee\overline{[v,2]}\vee\ldots\vee\overline{[u,k]}\vee\overline{[v,k]})$$

Obviously P is a positive monotone 2-CNF formula and H is a Horn formula. Consequently the instance $\overline{F} = P \wedge H$ belongs to the class MHF.
The **Hitting Set** problem can also be encoded into MHF-SAT. It is defined as follows:
INSTANCE: Set C of subsets of a finite set S, a positive integer $k \leq |S|$.
QUESTION: Is there a set $S' \subseteq S$, $|S'| \leq k$, such that for all $c \in C$ we have $\overline{c \cap S' \neq \emptyset?}$

The Hitting Set problem can be transformed to SAT as follows [50]:

$$X = \{[s,i] | s \in S, 1 \leq i \leq k\}$$

$$F = \bigwedge_{1\leq i\leq k} at\_most\_one\{[s,i]|s\in S\} \wedge \bigwedge_{c\in C} at\_least\_one\{[s,i]|s\in c, 1\leq i\leq k\}$$

This is obviously a generalization of the Vertex Cover Problem, where $|c| = 2$ for all $c \in C$. The first part choses at most k elements from S which will be part of S'. The second part verifies that this selection indeed is a hitting set. From F we obtain:

$$F = \bigwedge_{1\leq i\leq k}\bigwedge_{1\leq j<l\leq |S|}(\overline{[s_j,i]}\vee\overline{[s_l,i]})\wedge \\ \bigwedge_{c\in C}([s_1,1]\vee\ldots\vee[s_1,k]\vee\ldots\vee[s_c,1]\vee\ldots\vee[s_c,k])$$

By $s_1, \ldots s_{|c|}$ we denote all elements of c, for an arbitrary $c \in C$. Complementing all literals of F we obtain a mixed Horn formula $\overline{F}$:

$$\overline{F} = \bigwedge_{1\leq i\leq k}\bigwedge_{1\leq j<l\leq |S|}([s_j,i]\vee[s_l,i])\wedge \\ \bigwedge_{c\in C}(\overline{[s_1,1]}\vee\ldots\vee\overline{[s_1,k]}\vee\ldots\vee\overline{[s_c,1]}\vee\ldots\vee\overline{[s_c,k]})$$

Obviously $\overline{F} = P \wedge H$ is a mixed Horn formula, where

$$P := \bigwedge_{1 \le i \le k} \bigwedge_{1 \le j < l \le |S|} ([s_j, i] \vee [s_l, i])$$

is a positive monotone 2-CNF formula and

$$H := \bigwedge_{c \in C} (\overline{[s_1, 1]} \vee \ldots \vee \overline{[s_1, k]} \vee \ldots \vee \overline{[s_c, 1]} \vee \ldots \vee \overline{[s_c, k]})$$

is a negative monotone Horn formula.
Now we consider the **Dominating Set** problem, which is defined as follows:
INSTANCE: Graph $G = (V, E)$, a positive integer $k \le |V|$.
QUESTION: Is there a set $V' \subseteq V$, $|V'| \le k$, such that for all $v \in V$ we have $(\{v\} \cup N(v)) \cap V' \neq \emptyset$?

Transforming the Dominating Set problem to SAT can be accomplished as follows [50]:

$$\begin{aligned} X &= \{[v, i] | v \in V, 1 \le i \le k\} \\ F &= \bigwedge_{1 \le i \le k} at\_most\_one\{[v, i] | v \in V\} \\ &\quad \wedge \bigwedge_{v \in V} at\_least\_one\{[w, i] | 1 \le i \le k, w \in (\{v\} \cup N(v))\} \end{aligned}$$

The first $\bigwedge$ assures that at most k vertices are selected, and the second $\bigwedge$ verifies that the chosen vertex-set is a dominating set. We denote by $N(v)$ the set of all vertices which are adjacent to the vertex v. If F is satisfiable and α is a model for F, then all the vertices belong to the dominating set which correspond to the variables to which 1 is assigned according to α. Let $V = \{v_1, \ldots, v_n\}$. Then we obtain from F:

$$\begin{aligned} F = &\bigwedge_{1 \le i \le k} \bigwedge_{1 \le j < l \le n} (\overline{[v_j, i]} \vee \overline{[v_l, i]}) \\ &\wedge \bigwedge_{v \in V} ([v, 1] \vee \ldots \vee [v, k] \vee [v_{N_1}, 1] \vee \ldots \vee [v_{N_1}, k] \vee \\ &[v_{N_2}, 1] \ldots \vee [v_{N_2}, k] \vee \ldots \vee [v_{N_v}, 1] \ldots \vee [v_{N_v}, k]) \end{aligned}$$

Let $v_{N_1}, \ldots, v_{N_v}$ be all the vertices adjacent to v in G. Complementing all literals of F we obtain $\overline{F}$:

$$\begin{aligned} \overline{F} = &\bigwedge_{1 \le i \le k} \bigwedge_{1 \le j < l \le n} ([v_j, i] \vee [v_l, i]) \\ &\wedge \bigwedge_{v \in V} (\overline{[v, 1]} \vee \ldots \vee \overline{[v, k]} \vee \overline{[v_{N_1}, 1]} \vee \ldots \vee \\ &\overline{[v_{N_1}, k]} \vee \overline{[v_{N_2}, 1]} \ldots \vee \overline{[v_{N_2}, k]} \vee \ldots \vee \overline{[v_{N_v}, 1]} \ldots \vee \overline{[v_{N_v}, k]}) \end{aligned}$$

We set

$$P := \bigwedge_{1\leq i\leq k} \bigwedge_{1\leq j<l\leq n} ([v_j, i] \vee [v_l, i])$$

and

$$H := \bigwedge_{v\in V} (\overline{[v,1]} \vee \ldots \vee \overline{[v,k]} \vee \overline{[v_{N_1},1]} \\ \vee \ldots \vee \overline{[v_{N_1},k]} \vee \overline{[v_{N_2},1]} \ldots \vee \overline{[v_{N_2},k]} \vee \ldots \vee \overline{[v_{N_v},1]} \ldots \vee \overline{[v_{N_v},k]})$$

Then P is a positive monotone 2-CNF formula and H is a negative monotone Horn formula. Therefore $\overline{F} = P \wedge H$ is a mixed Horn formula. Considering any satisfying truth assignment of $\overline{F}$ all the vertices corresponding to the variables to which 0 is assigned belong to the dominating set.

In [50] the **Edge Dominating Set** problem is reduced to the SAT-problem as follows::
INSTANCE: Graph $G = (V, E)$, a positive integer $k \leq |E|$.
QUESTION: Is there a set $E' \subseteq E$, $|E'| \leq k$, such that for each $e \in E$ an $f \in E'$ exists with $e \cap f \neq \emptyset$?

$$X = \{[e,i] | e \in E, 1 \leq i \leq k\} \cup \{[v] | v \in V\}$$
$$F = \bigwedge_{1\leq i\leq k} at\_most\_one\{[e,i] | e \in E\} \\ \wedge \bigwedge_{v\in V} (([v]) \Rightarrow at\_least\_one\{[e,i] | v \in e \in E, 1 \leq i \leq k\}) \\ \wedge \bigwedge_{\{u,v\}\in E} at\_least\_one\{[u],[v]\}$$

The first part chooses at most k edges, which are checked by the remaining two parts to be an Edge Dominating Set. We denote with $v \in e \in E$ all edges $e \in E$ which have v as one of their end points. We now consider F and after some calculation we assert that a reduction to a mixed Horn instance is possible.

$$F = \bigwedge_{1\leq i\leq k} \bigwedge_{1\leq j<l\leq |E|} (\overline{[e_j,i]} \vee \overline{[e_l,i]}) \\ \wedge \bigwedge_{v\in V} (\overline{[v]} \vee [e_1,1] \vee \ldots \vee [e_1,k] \vee \ldots \vee [e_v,1] \vee \ldots \vee [e_v,k]) \\ \wedge \bigwedge_{\{u,v\}\in E} ([u] \vee [v])$$

Let $e_1, \ldots, e_v$ be all the edges of E containing v, for an arbitrary $v \in V$.

Negating all literals we obtain the following mixed Horn formula:

$$\begin{aligned}\overline{F} = & \bigwedge_{1\le i\le k}\ \bigwedge_{1\le j<l\le |E|} ([e_j,i]\vee[e_l,i]) \\ & \wedge \bigwedge_{v\in V} ([v]\vee\overline{[e_1,1]}\vee\ldots\vee\overline{[e_1,k]}\vee\ldots\vee\overline{[e_v,1]}\vee\ldots\vee\overline{[e_v,k]}) \\ & \wedge \bigwedge_{\{u,v\}\in E} (\overline{[u]}\vee\overline{[v]})\end{aligned}$$

Hence $\overline{F} = P \wedge H$, where

$$P = \bigwedge_{1\le i\le k}\ \bigwedge_{1\le j<l\le |E|} ([e_j,i]\vee[e_l,i])$$

consists of positive 2-clauses only and

$$H = \bigwedge_{v\in V} ([v]\vee\overline{[e_1,1]}\vee\ldots\vee\overline{[e_1,k]}\vee\ldots\vee\overline{[e_v,1]}\vee\ldots\vee\overline{[e_v,k]}) \bigwedge_{\{u,v\}\in E} (\overline{[u]}\vee\overline{[v]})$$

is a Horn formula. Hence $\overline{F} \in MHF$ and therefore the **Edge Dominating Set** Problem can be reduced to MHF-SAT.

Now we consider the **Independent Set** Problem, which is defined as follows:
INSTANCE: Graph $G = (V, E)$, a positive integer $k > 0$.
QUESTION: Is there a set $V' \subseteq V$, $|V'| \geq k$, such that for all $\{u, v\} \in E$ holds:
$\{u, v\} \not\subseteq V'$?

We can reduce the **Independent Set** Problem to the SAT-problem. In [50], such a reduction is performed as follows:

$$\begin{aligned}X &= \{[v,i] | v \in V, 1 \le i \le k\} \\ F &= \bigwedge_{1\le i\le k} at\_least\_one\{[v,i] | v \in V\} \\ &\quad \wedge \bigwedge_{\{u,v\}\in E} at\_most\_one\{[u,i],[v,i] | 1 \le i \le k\}\end{aligned}$$

The first part $\bigwedge$ chooses at least k vertices and the second part $\bigwedge$ examines, whether no pair of those vertices is adjacent. In case F has a satisfying truth assignment α then all those vertices of V belong to the Independent

Set, which correspond to the variables in X being set to 1 according to α. Let $V = \{v_1, \ldots, v_n\}$. Then we obtain for F:

$$F = \bigwedge_{1 \le i \le k} ([v_1, i] \vee [v_2, i] \vee \ldots \vee [v_n, i]) \wedge \bigwedge_{\{u,v\} \in E} \bigwedge_{1 \le j < l \le k} ((\overline{[u, j]} \vee \overline{[v, l]}) \\ \wedge (\overline{[u, j]} \vee \overline{[u, l]}) \wedge (\overline{[v, j]} \vee \overline{[v, l]}) \wedge (\overline{[v, j]} \vee \overline{[u, l]}) \wedge (\overline{[u, j]} \vee \overline{[v, j]}))$$

Negating all literals in F we obtain $\overline{F}$:

$$\overline{F} = \bigwedge_{1 \le i \le k} (\overline{[v_1, i]} \vee \overline{[v_2, i]} \vee \ldots \vee \overline{[v_n, i]}) \wedge \bigwedge_{\{u,v\} \in E} \bigwedge_{1 \le j < l \le k} (([u, j] \vee [v, l]) \\ \wedge ([v, j] \vee [u, l]) \wedge ([u, j] \vee [u, l]) \wedge ([v, j] \vee [v, l]) \wedge ([u, j] \vee [v, j]))$$

Then

$$P = \bigwedge_{\{u,v\} \in E} \bigwedge_{1 \le j < l \le k} (([u, j] \vee [v, l]) \wedge ([v, j] \vee [u, l]) \\ \wedge ([u, j] \vee [u, l]) \wedge ([v, j] \vee [v, l]) \wedge ([u, j] \vee [v, j])$$

and

$$H = \bigwedge_{1 \le i \le k} (\overline{[v_1, i]} \vee \overline{[v_2, i]} \vee \ldots \vee \overline{[v_n, i]})$$

Evidently P is a positive monotone 2-CNF formula and H is a negative monotone Horn formula. Thus $\overline{F} = P \wedge H$ is a mixed Horn formula for which holds: All the vertices of V which correspond to the variables to which 0 is assigned in a model for $\overline{F}$ belong to the Independent Set.

Next we consider the **Minimum Maximal Matching** problem, which is defined as follows: INSTANCE: Graph $G = (V, E)$, a positive integer $k \le |E|$.
QUESTION: Is there a set $E' \subseteq E$, $|E'| \le k$, such that for all $\{u, v\} \in E$ holds: $\{u, v\} \cap V(E') \neq \emptyset$?
Let $V(E')$ denote all vertices occurring at the edges of E'. We can reduce the **Minimum Maximal Matching** Problem to the SAT-problem. In [50]

such a reduction is performed as follows:

$$
\begin{aligned}
X &= \{[e,i] | e \in E, 1 \leq i \leq k\} \\
F &= \bigwedge_{1 \leq i \leq k} at\_most\_one\{[e,i] | e \in E\} \\
&\quad \wedge \bigwedge_{e \cap f \neq \emptyset} at\_most\_one\{[e,i],[f,i] | 1 \leq i \leq k\} \\
&\quad \wedge \bigwedge_{e \in E} (none\{[e,i] | 1 \leq i \leq k\} \Rightarrow \\
&\quad at\_least\_one\{[f,i] | 1 \leq i \leq k, f \in E, e \cap f \neq \emptyset\})
\end{aligned}
$$

The first two parts of the formula make sure that a matching of size at most k is selected. The third part ensures that the chosen matching is indeed maximal. For $E = \{e_1, \ldots, e_n\}$ we obtain:

$$
\begin{aligned}
F = & \bigwedge_{1 \leq i \leq k} \bigwedge_{1 \leq j < l \leq n} (\overline{[e_j,i]} \vee \overline{[e_l,i]}) \\
& \wedge \bigwedge_{e \cap f \neq \emptyset} \bigwedge_{e,f \in E} \bigwedge_{1 \leq j < l \leq k} ((\overline{[e,j]} \vee \overline{[f,l]}) \wedge (\overline{[f,j]} \vee \overline{[e,l]}) \wedge (\overline{[e,j]} \vee \overline{[e,l]}) \\
& \wedge (\overline{[f,j]} \vee \overline{[f,l]})) \\
& \wedge \bigwedge_{e \cap f \neq \emptyset} \bigwedge_{e,f \in E} \bigwedge_{1 \leq j \leq k} (\overline{[e,j]} \vee \overline{[f,j]}) \\
& \wedge \bigwedge_{e \in E} \overline{((\overline{[e,1]} \wedge \ldots \wedge \overline{[e,k]})} \vee [f_1,1] \vee \ldots \vee \\
& [f_1,k] \vee [f_2,1] \vee \ldots \vee [f_2,k] \vee \ldots \vee [f_{|e|},1] \vee \ldots \vee [f_{|e|},k])
\end{aligned}
$$

We designate with $f_1, \ldots, f_{|e|}$, $|e| \in \mathbb{N}$, all edges $f_i \in E$, for which holds $e \cap f_i \neq \emptyset$.

Performing a further step yields:

$$
\begin{aligned}
F = & \bigwedge_{1\le i\le k}\ \bigwedge_{1\le j<l\le n} (\overline{[e_j,i]} \vee \overline{[e_l,i]}) \\
& \wedge \bigwedge_{e\cap f\neq\emptyset}\ \bigwedge_{e,f\in E}\ \bigwedge_{1\le j<l\le k} ((\overline{[e,j]} \vee \overline{[f,l]}) \wedge (\overline{[f,j]} \vee \overline{[e,l]}) \wedge (\overline{[e,j]} \vee \overline{[e,l]}) \\
& \wedge (\overline{[f,j]} \vee \overline{[f,l]})) \\
& \wedge \bigwedge_{e\cap f\neq\emptyset}\ \bigwedge_{e,f\in E}\ \bigwedge_{1\le j\le k} (\overline{[e,j]} \vee \overline{[f,j]}) \\
& \wedge \bigwedge_{e\in E} ([e,1] \vee \ldots \vee [e,k] \vee [f_1,1] \vee \ldots \vee \\
& [f_1,k] \vee [f_2,1] \vee \ldots \vee [f_2,k] \vee \ldots \vee [f_{|e|},1] \vee \ldots \vee [f_{|e|},k])
\end{aligned}
$$

Negating all literals of F we obtain $\overline{F}$:

$$
\begin{aligned}
\overline{F} = & \bigwedge_{1\le i\le k}\ \bigwedge_{1\le j<l\le n} ([e_j,i] \vee [e_l,i]) \\
& \wedge \bigwedge_{e\cap f\neq\emptyset}\ \bigwedge_{e,f\in E}\ \bigwedge_{1\le j<l\le k} (([e,j] \vee [f,l]) \wedge ([f,j] \vee [e,l]) \wedge ([e,j] \vee [e,l]) \\
& \wedge ([f,j] \vee [f,l])) \\
& \wedge \bigwedge_{e\cap f\neq\emptyset}\ \bigwedge_{e,f\in E}\ \bigwedge_{1\le j\le k} ([e,j] \vee [f,j]) \\
& \wedge \bigwedge_{e\in E} (\overline{[e,1]} \vee \ldots \vee \overline{[e,k]} \vee \overline{[f_1,1]} \vee \ldots \vee \overline{[f_1,k]} \vee \overline{[f_2,1]} \vee \ldots \vee \\
& \overline{[f_2,k]} \vee \ldots \vee \overline{[f_{|e|},1]} \vee \ldots \vee \overline{[f_{|e|},k]})
\end{aligned}
$$

We define

$$
\begin{aligned}
P = & \bigwedge_{1\le i\le k}\ \bigwedge_{1\le j<l\le n} ([e_j,i] \vee [e_l,i]) \\
& \wedge \bigwedge_{e\cap f\neq\emptyset}\ \bigwedge_{e,f\in E}\ \bigwedge_{1\le j<l\le k} (([e,j] \vee [f,l]) \wedge ([f,j] \vee [e,l]) \wedge ([e,j] \vee [e,l]) \\
& \wedge ([f,j] \vee [f,l])) \\
& \wedge \bigwedge_{e\cap f\neq\emptyset}\ \bigwedge_{e,f\in E}\ \bigwedge_{1\le j\le k} ([e,j] \vee [f,j])
\end{aligned}
$$

and

$$H = \bigwedge_{e \in E} (\overline{[e,1]} \vee \ldots \vee \overline{[e,k]} \vee (\overline{[f_1,1]} \vee \ldots \vee \overline{[f_1,k]}) \vee \overline{[f_2,1]} \vee \ldots \vee \overline{[f_2,k]} \\ \vee \ldots \vee \overline{[f_{|e|},1]} \vee \ldots \vee \overline{[f_{|e|},k]})$$

Then P is a positive monotone 2-CNF formula and H is a negative monotone Horn formula and thus $\overline{F} = P \wedge H$ is a mixed Horn formula.

The **Strong Connectivity** Problem also belongs to the class MHF. It is defined as follows:
INSTANCE: Directed graph $G = (V, A)$.
QUESTION: Is G strongly connected?
A directed graph G is *strongly connected* if, and only if, for each two vertices u, v of G there is a path from u to v and from v to u. Alternatively this can also be characterised as follows: A digraph G is *strongly connected* if, and only if, any fixed vertex v_0 has all other vertices as both descendants and ascendents. We can transform the **Strong Connectivity** Problem to the SAT-problem as follows [50]:

$$\begin{aligned} X &= \{[i,v] | -(|V|-1) \leq i \leq |V|-1, v \in V\} \\ F &= ([0,v_0]) \wedge none\{[0,v] | v \in (V - \{v_0\})\} \\ &\wedge \bigwedge_{v \in V} at\_least\_one\{[i,v] | 0 \leq i \leq |V|-1\} \\ &\wedge \bigwedge_{0 < i \leq |V|-1} \bigwedge_{v \in V} (([i,v]) \Rightarrow at\_least\_one\{[j,w] | 0 \leq j < i, (w,v) \in A\}) \\ &\wedge \bigwedge_{v \in V} at\_least\_one\{[i,v] | -(|V|-1) \leq i \leq 0\} \\ &\wedge \bigwedge_{-(|V|-1) \leq i < 0} \bigwedge_{v \in V} (([i,v]) \Rightarrow at\_least\_one\{[j,w] | i < j \leq 0, (v,w) \in A\}) \end{aligned}$$

The third and the fourth part of the above formula ensure all other vertices as descendants, and the last two parts verify all other vertices as ascendants,

both for the vertex $v_0 \in V$. Considering F we obtain further:

$$\begin{aligned}
F =&([0,v_0] \wedge \overline{[0,v_1]} \wedge \ldots \wedge \overline{[0,v_{n-1}]}) \wedge \bigwedge_{v\in V}([0,v] \vee \ldots \vee [|V|-1,v]) \\
&\wedge \bigwedge_{0<i\leq|V|-1} \bigwedge_{v\in V} (\overline{[i,v]} \vee [0,w_1] \vee \ldots \vee [i-1,w_1] \\
&\vee \ldots \vee [0,w_v] \vee \ldots \vee [i-1,w_v]) \\
&\wedge \bigwedge_{v\in V} ([-(|V|-1),v] \vee \ldots \vee [0,v]) \\
&\wedge \bigwedge_{-(|V|-1)\leq i<0} \bigwedge_{v\in V} (\overline{[i,v]} \vee [i+1,w_1'] \vee \ldots \vee [0,w_1'] \\
&\vee \ldots \vee [i+1,w_v'] \vee \ldots \vee [0,w_v'])
\end{aligned}$$

Let $v_1, \ldots, v_{n-1}$ be all vertices of V (without the vertex v_0),

$$\{w_1', \ldots, w_v'\} := \{w \in V | (v,w) \in A\}$$

and $\{w_1, \ldots, w_v\} := \{w \in V | (w,v) \in A\}$, for a $v \in V$. Negating all literals of F, we obtain $\overline{F}$:

$$\begin{aligned}
\overline{F} =&(\overline{[0,v_0]} \wedge [0,v_1] \wedge \ldots \wedge [0,v_{n-1}] \wedge \bigwedge_{v\in V}(\overline{[0,v]} \vee \ldots \vee \overline{[|V|-1,v]}) \\
&\wedge \bigwedge_{0<i\leq|V|-1} \bigwedge_{v\in V} ([i,v] \vee \overline{[0,w_1]} \vee \ldots \vee \overline{[i-1,w_1]} \\
&\vee \ldots \vee \overline{[0,w_v]} \vee \ldots \vee \overline{[i-1,w_v]}) \\
&\wedge \bigwedge_{v\in V} (\overline{[-(|V|-1),v]} \vee \ldots \vee \overline{[0,v]}) \\
&\wedge \bigwedge_{-(|V|-1)\leq i<0} \bigwedge_{v\in V} ([i,v] \vee \overline{[i+1,w_1']} \vee \ldots \vee \overline{[0,w_1']} \\
&\vee \ldots \vee \overline{[i+1,w_v']} \vee \ldots \vee \overline{[0,w_v']})
\end{aligned}$$

As no clause of $\overline{F}$ has more than one positive literal, $\overline{F}$ is a Horn-formula and thus belongs to the class MHF. Therefore we can encode each **Strong Connectivity** Problem into a MHF-SAT instance.
In the same way, we can similarly show that nearly all NP-complete problems introduced by Karp [25] have a natural and straightforward encoding as an MHF-SAT problem. Some of these problems are listed below:

- Monochromatic Triangle problem
- Partition Into Cliques problem
- Set-Splitting problem

- Maximum Leaf Spanning Tree problem
- Planar Subgraph problem

The number of Boolean variables in the resulting member of MHF is often larger than the relevant instance size of the original problem, e.g. in the case of the Feedback Vertex Set problem (FVS) n^2 vs. n. However, solving the MHF member by an up-to-date SAT solver is quite fast because of the large number of 2-clauses which allow iterated unit resolution phases, in the case of FVS of length n each time.

2.2 Some NP-complete Subclasses of MHF

This section is devoted to establishing NP-completeness of certain interesting subclasses of MHF. The first class to consider consists of formulas whose Horn clauses are k-uniform, which means they all have equal length k, where $k \geq 3$, and their 2-CNF part is positive monotone. We denote this class by $\mathrm{MH_kF^+}$. Secondly, we consider the class $\mathrm{MH_k^-F^+} \subset \mathrm{MH_kF^+}$ of mixed Horn formulas with a positive monotone 2-CNF part and negative monotone, k-uniform Horn clauses, $k \geq 3$. Afterwards, we consider a subclass of $\mathrm{MH_k^-F^+}$, for which additionally holds that the variable-graph G_P of the positive monotone 2-CNF part P consists of disjoint edges only. We denote such a class of formulas by $\mathrm{MH_k^-F^{d+}}$. Finally, we consider the class $\mathrm{LMH_k^-F^+} \subset \mathrm{MH_k^-F^+}$ consisting of mixed Horn formulas of $\mathrm{MH_k^-F^+}$, for which additionally holds that the Horn clauses are linear [42]. A CNF formula F is called *linear* if

(1) F contains no pair of complementary unit clauses and

(2) for all $c_1, c_2 \in F : c_1 \neq c_2$ we have $|V(c_1) \cap V(c_2)| \leq 1$.

Theorem 1. SAT *is* NP*-complete for the following* MHF *subclasses:* $\mathrm{MH_kF^+}$, $\mathrm{MH_k^-F^+}$, $\mathrm{MH_k^-F^{d+}}$ *and* $\mathrm{LMH_k^-F^+}$, *for* $k \geq 3$.

Proof. Note that the NP-completeness of all these MHF subclasses directly follows from Schaefer's theorem [47] because none of these subclasses is properly contained in any tractable CNF class due to Schaefer's theorem. However, direct reductions are of interest per se. To that end, it is easy to see that the encoding of k-colorability for graphs, $k \geq 3$, to SAT directly yields the NP-completeness of $\mathrm{LMH_k^-F^+} \subset \mathrm{MH_k^-F^+} \subset \mathrm{MH_kF^+}$. But this connection to graph colorability does not hold for the class $\mathrm{MH_k^-F^{d+}}$, for which we provide the following reduction: It is well known that the class k-CNF, $k \geq 3$, is NP-complete. Let $F \in k$-CNF be an arbitrary formula. Then we can reduce F to a SAT-equivalent formula $\widetilde{M_F}$ of the class $\mathrm{MH_k^-F^{d+}}$ in polynomial-time. The transformation is achieved in two main steps, the first of which is referred to as MHF-*reduction*:

1. Let $V^+(F) \subseteq V(F)$ be the set of all variables having a positive occurrence in F. For each variable $x \in V^+(F)$ we introduce a new variable $y_x \notin V(F)$ and perform the following steps: We replace each positive occurrence of $x \in V^+(F)$ in the k-clauses by $\overline{y_x}$, for each $x \in V^+(F)$. Let F' be the resulting formula. Next we add the constraints $\overline{y_x} \Leftrightarrow x$, for all $x \in V^+(F)$, to F', equivalent to

$$(\overline{y_x} \Leftrightarrow x) \Leftrightarrow ((y_x \vee x) \wedge (\overline{y_x} \vee \overline{x}))$$

yielding the new formula

$$M_F = F' \wedge \bigwedge_{x \in V^+(F)} (y_x \vee x) \wedge (\overline{y_x} \vee \overline{x})$$

Here F' only consists of Horn clauses of length k.
2. As formulas of each class $\mathrm{MH}_{\mathrm{k}}^{-}\mathrm{F}^{\mathrm{d}+}$ are allowed to contain positive 2-clauses only, but M_F also contains the negative monotone 2-clauses $(\overline{y_x} \vee \overline{x})$, for each $x \in V^+(F)$, we add to each such negative 2-clause exactly $(k-2)$ backbone variables $\overline{z_x^i}$ and the formulas $F_x^i \in \mathrm{MH}_{\mathrm{k}}^{-}\mathrm{F}^{\mathrm{d}+}$, for $i = 1, \ldots, k-2$, such that z_x^i is a backbone variable of F_x^i which has to be set to 1. All such formulas F_x^i, $i = 1, \ldots, k-2$, must be pairwise and also variable-disjoint with F'. Let $i = 1, \ldots k-2$, then an example of such a backbone-formula is:

$$\begin{aligned} F_x^i = (z_{x1}^i \vee z_{x2}^i) \wedge (z_{x3}^i \vee z_{x4}^i) \wedge (z_{x5}^i \vee z_{x6}^i) \wedge \ldots \wedge (z_{x(2k-1)}^i \vee z_{x2k}^i) \\ \wedge (\overline{z_{x1}^i} \vee \overline{z_{x3}^i} \vee \ldots \vee \overline{z_{x(2k-1)}^i}) \wedge \ldots \wedge (\overline{z_{x2}^i} \vee \overline{z_{x4}^i} \vee \ldots \vee \overline{z_{x(2k-1)}^i}) \end{aligned}$$

The Horn part consists of $2^k - 1$ negative monotone k-clauses, where each k-clause consists of the variables of a vertex cover of the positive monotone 2-CNF part $(z_1^i \vee z_2^i) \wedge (z_3^i \vee z_4^i) \wedge (z_5^i \vee z_6^i) \wedge \ldots \wedge (z_{2k-1}^i \vee z_{2k}^i)$. Since the positive monotone 2-CNF part has altogether 2^k negative monotone vertex covers, but there are only $2^k - 1$ negative monotone k-clauses in the Horn part, the formula F_x^i is satisfiable and has k backbone-variables, which have to be set to 1. We assume that the vertex cover which selects from each positive 2-clause the variable with the odd index does not appear as negative Horn clause in F_x^i. Therefore each variable of this clause is a backbone variable of F_x^i which has to be set to 1.

Finally we add the literals $\overline{z_x^1}, \ldots, \overline{z_x^{k-2}}$, for $x \in V^+(F)$ to each negative monotone clause $(\overline{y_x} \vee \overline{x})$ of M_F, and gain $(\overline{y_x} \vee \overline{x} \vee \overline{z_x^1} \vee \ldots \vee \overline{z_x^{k-2}})$. Altogether

we get

$$\widetilde{M_F} = F' \wedge \bigwedge_{x \in V^+(F)} \left((\overline{y_x} \vee \overline{x} \vee \overline{z_x^1} \vee \ldots \vee \overline{z_x^{k-2}}) \wedge F_x^1 \wedge \ldots \wedge F_x^{k-2} \right)$$
$$\wedge \bigwedge_{x \in V^+(F)} (y_x \vee x) \in \mathrm{MH_k^-F^{d+}}$$

It holds that $\widetilde{M_F}$ is satisfiable if, and only if, F is satisfiable: We assume that F is satisfiable. Let α be a model of F, then we set all the variables, which $\widetilde{M_F}$ and F have in common, also according to α. Each newly introduced variable y_x (such that $\overline{y_x}$ is equivalent to x) is set as follows: $y_x = 1 - \alpha(x)$. Since the added backbone formulas F_x^i are always satisfiable, $\widetilde{M_F}$ is also satisfiable. If F is unsatisfiable, we obviously cannot satisfy F' and $\widetilde{M_F}$ either. □

2.3 Algorithms for SAT of Further Mixed Horn Classes

In this section we consider some special classes of mixed Horn formulas, for which SAT can be solved in a better running time than $O(2^{0.5284n})$. Let H be k-**uniform** ($k \geq 3$), **negative** and **linear**; assume that all m clauses of H can be written as a sequence $c_1, \ldots, c_m$, such that

- all literals can be enumerated such that **either** the last literal of c_i and the first literal of c_{i+1} are equal ($i = 1, \ldots, m-1$)
  or the clauses c_i, c_{i+1} are variable-disjoint,
- no clause c_i ($i \in \{2, \ldots, m-1\}$) is allowed to share a variable with any other clause except with c_{i-1} and c_{i+1} as stated above.

Then we say H has **overlappings in boundary variables** only. Note that the first clause c_1 and the last clause c_m are also not allowed to share a variable.
Example:

$$H = (\overline{x_1} \vee \overline{x_2} \vee \overline{x_3} \vee \overline{x_4}) \wedge (\overline{x_4} \vee \overline{x_5} \vee \overline{x_6} \vee \overline{x_7}) \wedge (\overline{x_7} \vee \overline{x_8} \vee \overline{x_9} \vee \overline{x_{10}})$$

has overlappings in boundary variables only.
Each connected component of the incidence graph G_H for H looks like in Figure 1.

Definition 1. *We consider formulas $M = G \wedge H \in$ MHF with the following properties:*

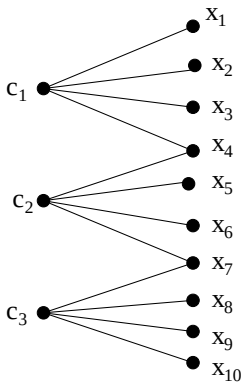


Figure 2.1: The incidence graph for
$H = (\overline{x_1} \vee \overline{x_2} \vee \overline{x_3} \vee \overline{x_4}) \wedge (\overline{x_4} \vee \overline{x_5} \vee \overline{x_6} \vee \overline{x_7}) \wedge (\overline{x_7} \vee \overline{x_8} \vee \overline{x_9} \vee \overline{x_{10}})$

*G consists of 2-clauses not necessarily positive monotone and H consists of linear Horn clauses, for which holds: All clauses are negative monotone, k-uniform, $k \geq 3$, and there is an ordering $c_1, c_2 \ldots, c_{|H|}$ of the clauses of H and an ordering of all literals in each clause, such that H has overlappings in boundary variables only. We denote this class by k-*BLMHF *(k-Boundary-Linear mixed Horn formulas).*

Example for a k-Boundary-Linear mixed Horn formula:

$$\begin{aligned} M =& (x_1 \vee \overline{x_2}) \wedge (x_3 \vee x_4) \wedge (x_5 \vee \overline{x_6}) \wedge (x_7 \vee x_{12}) \wedge (x_8 \vee x_{11}) \wedge (\overline{x_{10}} \vee \overline{x_{11}}) \\ & \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_3} \vee \overline{x_4}) \wedge (\overline{x_4} \vee \overline{x_5} \vee \overline{x_6} \vee \overline{x_7}) \\ & \wedge (\overline{x_8} \vee \overline{x_9} \vee \overline{x_{10}} \vee \overline{x_{11}}) \wedge (\overline{x_{11}} \vee \overline{x_{12}} \vee \overline{x_{13}} \vee \overline{x_{14}}) \end{aligned}$$

Obviously $M = G \wedge H$, where

$$G = (x_1 \vee \overline{x_2}) \wedge (x_3 \vee x_4) \wedge (x_5 \vee \overline{x_6}) \wedge (x_7 \vee x_{12}) \wedge (x_8 \vee x_{11}) \wedge (\overline{x_{10}} \vee \overline{x_{11}})$$

and

$$H = (\overline{x_1} \vee \overline{x_2} \vee \overline{x_3} \vee \overline{x_4}) \wedge (\overline{x_4} \vee \overline{x_5} \vee \overline{x_6} \vee \overline{x_7}) \wedge (\overline{x_8} \vee \overline{x_9} \vee \overline{x_{10}} \vee \overline{x_{11}}) \wedge (\overline{x_{11}} \vee \overline{x_{12}} \vee \overline{x_{13}} \vee \overline{x_{14}})$$

Note that H with this restriction belongs to a subclass of the class of nested formulas studied by Knuth [27].

Theorem 2. SAT *remains* NP*-complete for the class k-*BLMHF*, $k \geq 3$.*

Proof. We provide a polynomial-time reduction from k-CNF-SAT, which is NP-complete to k-BLMHF-SAT, proving the NP-completeness of the latter. Let $F \in k$-CNF. First we perform the MHF-reduction step as in the proof of Theorem 1 obtaining the corresponding formula M_F

$$M_F = F' \wedge \bigwedge_{x \in V^+(F)} (y_x \vee x) \wedge (\overline{y_x} \vee \overline{x})$$

Let the clauses of F' be labelled as follows: $c_1, c_2, \ldots, c_{|F'|}$. Then we proceed as follows:
At the beginning let $i = 1$, as long as $i < |F'|$ we consider c_i. As long as c_i has a common variable z with c_j, for some $j \geq i+1$, we replace $\overline{z}$ in c_j by $\overline{y_z}$, y_z not yet occurring in the set of variables of M_F, and add the 2-clauses $(\overline{z} \vee y_z) \wedge (\overline{y_z} \vee z)$ to M_F. Then we set $i := i + 1$. Let M'_F be the final resulting formula, then M'_F belongs to the class k-BLMHF (note the clauses of H are pairwise disjoint) and obviously M'_F is equivalent to F concerning SAT and the transformation can be performed in polynomial-time. □

Theorem 3. SAT *for* k-BLMHF*,* $k \geq 3$*, can be solved in time* $O((\sqrt[k]{k})^n)$*.*

Proof. The following algorithm solves SAT for the class k-BLMHF in time $O((\sqrt[k]{k})^n)$:

Algorithm for k-Boundary-linear MHF's

INPUT: $F = G \wedge H$ belonging to the class k-BLMHF.
OUTPUT: A model for F, if F is satisfiable; nil, otherwise.
begin
1.)If $V(G) \subset V(H)$ set all the variables in $V(H) - V(G)$ to 0 until $V(G) = V(H)$.
2.) Compute all (minimal) Hitting Sets $S_1, S_2, \ldots, S_r$ of H.
3.) Set $i = 1$.
4.) **while** $i \leq r$, **do**:

(a) Evaluate F by assigning 0 to all variables of S_i, then solve the 2-SAT search problem for the remaining part of G by a standard 2-SAT algorithm in polynomial-time.

(b) If the assignment t at hand is a model for F, then the algorithm stops with output t.

(c) Else undo the assignment of the last step and augment $i := i + 1$.

5.) Return nil.
end

Correctness: The algorithm k-BLMHF verifies for each minimal hitting set of H, whether the partial truth assignment resulting from setting all variables in the hitting set to 0 can be extended to a model of F by checking the remaining 2-CNF part in linear time [2]. Since we consider only the minimal hitting sets of H, we do not impose any restrictions concerning the satisfiability of H. This is due to the fact that for each model of F the set of all variables which are set to 0 either corresponds to a minimal hitting set of H or contains a minimal hitting set of H itself.

Analysis of the running time: Let $F = G \wedge H \in k$-BLMHF with n variables and let $V(G) = V(H)$. One can show that the number of minimal hitting sets is maximal if the Horn part of F consists of disjoint clauses only. So the running time of the algorithm k-BLMHF is dominated by this subclass of k-BLMHF, for which the number of minimal hitting sets of H is $k^{\lceil n/k \rceil}$, yielding the running time $O(p(n)(\sqrt[k]{k})^n)$, for a polynom p. As the sequence $(\sqrt[k]{k})_{k\in\mathbb{N}}$ decreases monotonously with increasing k, we obtain a running time better than $3^{n/3}$ for $k \geq 4$.
Note that the algorithm has a **fixed parameter tractable** running time of $O(p(n)k^{m/k})$, where $m = |V(H)|$ and $n = |V(G)|$ if $V(H) \subset V(G)$. □

Another NP-complete subclass of MHF we consider is the class MH_F$^{\triangle}$ consisting of mixed Horn formulas $M = P \wedge H$ with a negative monotone Horn part H and a positive monotone 2-CNF part P for which holds that the corresponding variable graph G_P consists of disjoint triangles only. We further demand that $V(P) = V(H)$. In case there is a variable $x \in V(P) - V(H)$ we set x to 1 and in case there is a variable $x \in V(H) - V(P)$ we set x to 0.

Theorem 4. SAT *remains* NP*-complete for the class* MH^{-}F$^{\triangle}$.

Proof. In Theorem 1 is shown that SAT remains NP-complete for the class MH$_{\text{k}}^{-}$F$^{\text{d}+}$. Now we provide a polynomial-time reduction from MH$_{\text{k}}^{-}$F$^{\text{d}+}$-SAT to MH^{-}F$^{\triangle}$-SAT. Let $F \in$ MH$_{\text{k}}^{-}$F$^{\text{d}+}$ with a Horn part consisting of negative monotone, k-uniform clauses and with a positive monotone 2-CNF part P whose corresponding variables graph G_P consists of disjoint edges only, that is P looks as follows:

$$(x_1 \vee x_2) \wedge (x_3 \vee x_4) \wedge (x_5 \vee x_6) \wedge \ldots \wedge (x_{n-1} \vee x_n)$$

Then for each such $(x_i \vee x_{i+1}) \in P$, $i \in \{1, \ldots, n-1\}$ we introduce a new variable $y_{i,i+1}$ and add the two clauses $(x_i \vee y_{i,i+1})$ and $(x_{i+1} \vee y_{i,i+1})$ to P. Let $\widetilde{P}$ be the resulting positive monotone 2-CNF formula. Then $G_{\widetilde{P}}$ obviously consists of disjoint triangles only. Further, for each $(x_i \vee x_{i+1}) \in P$, we add to H the following clauses each consisting of all variables of the same triangle of P, but all negated: $(\overline{x_i} \vee \overline{x_{i+1}} \vee \overline{y_{i,i+1}})$ and obtain $\widetilde{H}$. Now let F be satisfiable and let α be a model of F, then in $\widetilde{F} = \widetilde{P} \wedge \widetilde{H}$ we set all variables which F and $\widetilde{F}$ have in common according to α and set $y_{i,i+1} = 1$, for all $i \in \{1, \ldots, n-1\}$. This way we satisfy all the newly added 2-clauses of the $\widetilde{P}$ part. The newly added 3-clauses $(\overline{x_i} \vee \overline{x_{i+1}} \vee \overline{y_{i,i+1}})$ of $\widetilde{H}$ are also satisfied because it suffices to set exactly one variable in each clause $(x_i \vee x_{i+1})$ of P to 1 so that we can set the other variable to 0 and hence also satisfy all the newly added 3-clauses. If F is not satisfiable, $\widetilde{F}$ cannot be satisfied either because $F \subset \widetilde{F}$. □

Formulas $F = P \wedge H$ in MH^{-}F$^{\triangle}$ have an unrestricted Horn part H concerning the length of the clauses. So the natural question arises whether

NP-completeness also holds for the subclass of $\mathrm{MH}^{-}\mathrm{F}^{\triangle}$ where the Horn clauses are k-uniform, for a $k \geq 3$. Thus we next consider the class $MH_k^- F^{\triangle}$ of mixed Horn formulas, whose Horn part H is k-uniform, $k \geq 3$, and negative monotone. Further we demand that $V(P) = V(H)$. The following Theorem states the NP-completeness for this class.

Theorem 5. SAT *remains* NP*-complete for the class* $MH_k^- F^{\triangle}$, $k \geq 3$.

Proof. We provide a polynomial-time reduction from $MH_k^- F^{d+}$-SAT, which is NP-complete according to Theorem 1, to $MH_k^- F^{\triangle}$-SAT, proving the NP-completeness of the latter. To that end let $F \in MH_k^- F^{d+}$ be an arbitrary formula. For each clause $(x_i \vee x_j)$ of P, $x_i, x_j \in V(F)$, we introduce a new variable $y_{i,j}$ not yet occurring in $V(F)$ and add the two clauses $(x_i \vee y_{i,j})$, $(x_j \vee y_{i,j})$ to P. This way we obtain the positive monotone 2-CNF formula P' whose corresponding graph $G_{P'}$ consists of disjoint triangles only. For each newly introduced variable $y_{i,j}$ we add to H the following three negative monotone k-clauses (for $k \geq 4$):

$$(\overline{y_{i,j}} \vee \overline{z^1_{i,j,1}} \vee \ldots \vee \overline{z^{k-1}_{i,j,1}}), (\overline{y_{i,j}} \vee \overline{z^1_{i,j,2}} \vee \ldots \vee \overline{z^{k-1}_{i,j,2}}), (\overline{y_{i,j}} \vee \overline{z^1_{i,j,3}} \vee \ldots \vee \overline{z^{k-1}_{i,j,3}})$$

where the variables $z^p_{i,j,l}$, for $p \in \{1, \ldots, k-1\}, l \in \{1,2,3\}$ are newly introduced and pairwise different. Let H' be the resulting Horn part. To achieve the condition $V(P') = V(H')$ we add the clauses $(z^p_{i,j,1} \vee z^p_{i,j,2})$, $(z^p_{i,j,1} \vee z^p_{i,j,3})$, $(z^p_{i,j,2} \vee z^p_{i,j,3})$ to P', for each $p \in \{1, \ldots, k-1\}$ and obtain P'', whose corresponding graph $G_{P''}$ obviously still consists of disjoint triangles only.
In case $k = 3$ we add the following two clauses to H:

$$(\overline{y_{i,j}} \vee \overline{z_{i,j,1}} \vee \overline{z_{i,j,2}}) \wedge (\overline{y_{i,j}} \vee \overline{z_{i,j,1}} \vee \overline{z_{i,j,3}})$$

and we add the following clauses to P':

$$(z_{i,j,1} \vee z_{i,j,2}) \wedge (z_{i,j,2} \vee z_{i,j,3}) \wedge (z_{i,j,1} \vee z_{i,j,3})$$

Now we obviously have $V(P'') = V(H')$. Additionally $F' = P'' \wedge H'$ is satisfiable if, and only if, F is: Let F be satisfiable and let α be a model of F, then we set all the variables which F and F' have in common according to α and in each triangle $(x_i \vee x_j) \wedge (x_i, y_{i,j}) \wedge (x_j, y_{i,j})$ of P' we set $y_{i,j} = 1$ and hence satisfy P'. In case $k \geq 4$, we set $z^1_{i,j,1} = 0$, $z^2_{i,j,2} = 0$, $z^3_{i,j,3} = 0$ and hence satisfy the newly added clauses in H'. We assign 1 to all the other variables and this way also satisfy the newly introduced triangles in P''. In case $k = 3$ we set $z_{i,j,1} = 0$ and $z_{i,j,2} = z_{i,j,3} = 1$ and also satisfy F'. If F is not satisfiable, F' is also unsatisfiable, because $F \subset F'$. □

The next result is quite significant as we can provide a better running time than the so far best running time of $O(p(n) \cdot 1.427^n)$ by S. Kottler, M. Kaufmann and C. Sinz [29] for the class of mixed Horn formulas for which additionally holds: H is negative monotone, $|c| \leq 3$, for all $c \in H$, and P consists of positive monotone 2-clauses.

Theorem 6. *We consider mixed Horn formulas $F = P \wedge H \in MHF$ for which additionally holds: H is negative monotone, $|c| \leq 3$, for all $c \in H$, and P consists of positive monotone 2-clauses. Then we can solve* SAT *in running time $O(1.325^n)$ for this formula class.*

Proof. The following algorithm has a running time of $O(1.325^n)$ for this formula class.

Algorithm $\mathrm{MH}_{-\leq 3}\mathrm{F}^+$
begin
As long as F contains a unit clause or a pure literal apply the following steps 1.)-3.).
1.) Apply the unit clause rule and the pure literal rule.
2.) Suppose that G_P contains a disjoint triangle of 2-clauses $(x \vee y), (y \vee z)$, $(x \vee z)$ and H contains the clause $(\overline{x} \vee \overline{y})$. Then any satisfying assignment of F must set $z = 1$, hence recurse on $F[z = 1]$.
3.) If F contains the clause $(\overline{x} \vee \overline{y})$ and G_P contains an isolated edge $(x \vee y)$, i.e. neither x nor y appears in any other positive clauses, then recurse on $F - \{(\overline{x} \vee \overline{y})\}$.
* Remark that F now contains no unit clauses, no pure literals, and no two 2-clauses of the form $(x \vee y)$, $(\overline{x} \vee \overline{y})$.*
4.) If F contains a negative 2-clause, pick one, say $(\overline{x} \vee \overline{u})$. Since no variable is pure and H is negative monotone, there are positive 2-clauses $(x \vee y)$ and $(u \vee v)$. Next we will do the following recursive calls, each with a number of variables already assigned.

a) Clause $(x \vee y)$ is an isolated edge.
If $x = 1$, then $u = 0$ and $v = 1$. Furthermore, y is pure now and hence we can also set $y = 0$. If $x = 0$, then $y = 1$.

b) Clause $(x \vee y)$ is not an isolated edge, for example let $(y \vee w)$, $w \neq x$, be also a clause of P .
If $x = 1$, then $u = 0$ and $v = 1$. Note that the variable y is not pure now. If $x = 0$, then $y = 1$.

c) Clause $(x \vee y)$ belongs to a disjoint triangle $(x \vee y)$, $(x \vee z)$ and $(y \vee z)$.
If $x = 1$, then $u = 0$ and $v = 1$. If $x = 0$, then $y = 1 = z$.

d) Let $y = v$ and $(x \vee y) \wedge (y \vee u)$ be two clauses of P, which are not necessarily disjoint in P .
If $x = 0$, then $y = 1$. If $x = 1$, then $u = 0$ and $y = 1$.

e) Let $y = v$ and let the triangle $(x \vee y)$, $(u \vee y)$, $(x \vee u)$ be a part of G_P, not disjoint in G_P.
If $x = 0$, then $y = 1 = u$. If $x = 1$, then $u = 0$ and $y = 1$.

f) If P also contains the clause $(x \vee u)$, which is obviously not an isolated edge, then:
If $x = 0$, then $y = 1 = u$. If $x = 1$, then $u = 0$ and $v = 1$.

5.) Autarky principle: If F contains no negative 2-clause, we perform just any branching: Pick a variable x and recurse on $F[x = 0]$ and on $F[x = 1]$.
end

Autarky guarantees that we will have to do the branching from step 5.) no more than n times, where n is the number of variables of F, in the whole recursion tree. Thus autarky implies that step 5.) does not significantly influence the running time. The running time is clearly dominated by step 4.) and the number of recursive calls is bounded (up to a polynomial factor) by $T(n) \leq T(n-2)+T(n-3)$; this term results from the cases b), d) in 4.). So the running time is $O(p(n)a^n)$, where a is the biggest root of $a^3 - a - 1 = 0$ and this is roughly 1.325. □

The next result provides an improvement of the running time $O(poly(n)1.44^n)$ for the general mixed Horn formula class [43]. We present the algorithm $\mathrm{MH}_{\leq 3}\mathrm{F}^+$ whose running time is $O(poly(n)1.41^n)$ which is obviously better than the running time $O(poly(n)1.44^n)$ of algorithm MHFSAT for unrestricted mixed Horn formulas by Porschen and Speckenmeyer in [43].

Theorem 7. *Let us consider mixed Horn formulas $F = P \wedge H \in MHF$ for which holds: G_P consists of disjoint triangles, edges and isolated vertices and H consists of Horn clauses which have at most three literals, but are not necessarily negative monotone. Further we demand that $V(P) = V(H)$. Then we can solve* SAT *in running time $O(1.41^n)$ for this formula class.*

Proof. The following algorithm has a running time of $O(1.41^n)$ for this formula class.

Algorithm $\mathrm{MH}_{\leq 3}\mathrm{F}^+$
begin
Setting-1-rule: Let y be a variable which we have just set to 1. Then we have to differentiate the following two cases: If y is contained in a disjoint edge $(y \vee z)$ of G_P, we recurse on $F[z = 0]$ and on $F[z = 1]$. If y is contained in a triangle $(y \vee z) \wedge (y \vee w) \wedge (z \vee w)$ of G_P, we have to recurse on $F[z = 0, w = 1]$, $F[z = 1, w = 1]$ and on $F[z = 1, w = 0]$.
As long as possible apply the following steps 1.)-3.)
1.) Apply the unit clause rule and the pure literal rule.
2.) Suppose that G_P contains a disjoint triangle of 2-clauses

$$(x \vee y) \wedge (y \vee z) \wedge (x \vee z)$$

and H contains the clause $(\overline{x} \vee \overline{y})$. Any satisfying assignment of F must set $z = 1$, hence recurse on $F[z = 1]$ by applying the setting-1-rule.

3.) If F contains a negative 2-clause, pick one, say $(\overline{x} \vee \overline{u})$.
3.1) If there is only an isolated edge in G_P containing the variables x and u (that is the variables x and u do not occur in any other clauses of P except in $(x \vee u)$), then we do the following recursive calls: If $x = 0$, we set $u = 1$. If $x = 1$, we set $u = 0$.
3.2) Since $V(P) = V(H)$, there are positive 2-clauses $(x \vee y)$ and $(u \vee v)$. Next we do the following recursive calls, each with a number of variables already assigned.

a) The clauses $(x \vee y)$ and $(u \vee v)$ are isolated edges in G_P.
If $x = 1$, then we recurse on $F[v = 1, u = 0, y = 1]$ and on $F[v = 1, u = 0, y = 0]$. If $x = 0$, then $y = 1$.

b) The clause $(x \vee y)$ is contained in a disjoint triangle $(x \vee y)$, $(x \vee z)$, $(y \vee z)$ and the clause $(u \vee v)$ is an isolated edge in G_P.
If $x = 1$, then we recurse on $F[v = 1, u = 0, y = 1, z = 0]$, $F[v = 1, u = 0, y = 0, z = 1]$ and on $F[v = 1, u = 0, y = 1, z = 1]$. If $x = 0$, then $y = 1 = z$.

c) The clause $(x \vee y)$ belongs to a disjoint triangle $(x \vee y)$, $(x \vee z)$, $(y \vee z)$ and the clause $(u \vee v)$ also belongs to a disjoint triangle $(u \vee v)$, $(v \vee w)$, $(u \vee w)$ in G_P.
If $x = 1$, then we recurse on $F[v = 1, u = 0, w = 1, y = 1, z = 0]$, $F[v = 1, u = 0, w = 1, y = 0, z = 1]$ and on $F[v = 1, u = 0, w = 1, y = 1, z = 1]$. If $x = 0$, then $y = 1 = z$.

4.) If F contains a mixed 2-clause, pick one, say $(x \vee \overline{u})$. Since $V(P) = V(H)$, there are positive 2-clauses $(x \vee y)$ and $(u \vee v)$ and we can proceed analogously to step 3.).
5.) Autarky principle: If F contains no 2-clause, we perform just any branching: Pick a variable x and recurse on $F[x = 0]$ and on $F[x = 1]$ applying the setting-1-rule.
end

The running time is clearly dominated by steps 3.1) and 3.2) and the number of recursive calls is bounded (up to a polynomial factor) by

$$T(n) \leq \max\{T(n-2)+2T(n-4), T(n-3)+3T(n-5), T(n-3)+3T(n-6), 2T(n-2)\}$$

Obviously $T(n) \leq T(n-2) + 2T(n-4) = 2T(n-2)$ and this term results from 3.1). So the running time is $O(poly(n)a^n)$, where a is the biggest root of $a^2 - 2 = 0$ and this is $\sqrt{2} = 1.41$. □

2.3.1 Algorithm MH_F$^{\triangle}$

In this part of our thesis we present an algorithm solving SAT for formulas in MH_F$^{\triangle}$. The motivation for looking at this subclass of MHF has its source in the

analysis of the algorithm MHFSAT in [43]. Solving unrestricted MHF has its worst-case behaviour just for this class of formulas. The running time is stated below for k-uniform Horn parts, for $k \in \{3,4,5,6,7,8\}$. By backtracking variables of the Horn part are set to 0 without violating P. To this end, at least two variables in each triangle must be set to 1. When setting a variable x to 0 to satisfy a clause in the Horn part, the other two variables of the triangle $\triangle_x$ must be set to 1. Clauses of H that are satisfied are put on a stack. Variables set to 0 are stored in the set V and those set to 1 are stored in the set W.
So the main idea of Algorithm MH_F$^{\triangle}$ can be summarised as follows:

- Try to satisfy clauses of H by setting variables to 0.
- When a variable x is set to 0 then the other two variables of the triangle $\triangle_x$ in P, to which x belongs, must immediately be set to 1.
- Reduce the original formula by removing all clauses from H containing the variable which is set to 0 and put them on a stack. Further, remove all variables which are set to 1 from the remaining clauses in H and put them in the set W, the variables set to 0 put in V. Then, remove all clauses (triangles) from P which are satisfied.
- If obtaining a contradiction then **backtrack**, i.e. go a step back (to where the contradiction has not yet occurred), choose a different variable to set to 0 in H and now assign 1 to the variable which was set to 0 in the step before.
- If H and P are empty, then the formula is satisfiable.
- If one cannot backtrack anymore, meaning all possible combinations of setting variables in H to 0 have been checked and no satisfying assignment has been found, then the formula is not satisfiable.

Before presenting the algorithm we want to explain the way it works considering the following example:
Let $M = P \wedge H$, with $P = \triangle_a \wedge \triangle_b \wedge \triangle_c$, where $\triangle_j$ corresponds to $(j_1 \vee j_2) \wedge (j_2 \vee j_3) \wedge (j_1 \vee j_3)$, for $j \in \{a, b, c, \}$ and

$$\begin{aligned} H =& (\overline{a_1} \vee \overline{b_1} \vee \overline{c_1}) \wedge (\overline{a_2} \vee \overline{b_3} \vee \overline{c_2}) \wedge (\overline{a_3} \vee \overline{b_3} \vee \overline{c_1}) \wedge (\overline{a_2} \vee \overline{b_2} \vee \overline{c_3}) \\ & \wedge (\overline{a_1} \vee \overline{b_2} \vee \overline{c_1}) \wedge (\overline{a_3} \vee \overline{b_1} \vee \overline{c_2}) \wedge (\overline{a_3} \vee \overline{c_1} \vee \overline{c_2}) \end{aligned}$$

- Set $a_1 = 0$. This satisfies the clauses $(\overline{a_1} \vee \overline{b_1} \vee \overline{c_1})$, $(\overline{a_1} \vee \overline{b_2} \vee \overline{c_1})$ of H; remove them from H. As $a_1 = 0$, then $\triangle_a$ implies: $a_2 = a_3 = 1$. The remaining formula is: $P = \triangle_b \wedge \triangle_c$ and

$$H = (\overline{b_3} \vee \overline{c_2}) \wedge (\overline{b_3} \vee \overline{c_1}) \wedge (\overline{b_2} \vee \overline{c_3}) \wedge (\overline{b_1} \vee \overline{c_2}) \wedge (\overline{c_1} \vee \overline{c_2})$$

- Set $b_3 = 0$. This satisfies the clauses $(\overline{b_3} \vee \overline{c_2})$, $(\overline{b_3} \vee \overline{c_1})$, remove them from H. As $b_3 = 0$, then $\triangle_b$ implies: $b_1 = b_2 = 1$. The remaining formula is: $P = \triangle_c$ and $H = (\overline{c_3}) \wedge (\overline{c_2}) \wedge (\overline{c_1} \vee \overline{c_2})$.
- Unit clause rule: $c_3 = c_2 = 0$, but this violates $\triangle_c$. Hence **backtrack**:
- Consider $P = \triangle_b \wedge \triangle_c$ again and

$$H = (\overline{b_3} \vee \overline{c_2}) \wedge (\overline{b_3} \vee \overline{c_1}) \wedge (\overline{b_2} \vee \overline{c_3}) \wedge (\overline{b_1} \vee \overline{c_2}) \wedge (\overline{c_1} \vee \overline{c_2})$$

Now set $b_3 = 1$ and $c_2 = 0$ which implies $c_1 = c_3 = 1$ yielding: $P = \triangle_b$, $H = (\overline{b_3}) \wedge (\overline{b_2})$. Unit clause rule: $b_3 = b_2 = 0$ which is a contradiction to $b_3 = 1$. Hence **backtrack**:

- Consider the original formula and now set $a_1 = 1$, $b_1 = 0 \Longrightarrow b_2 = b_3 = 1$ obtaining: $P = \triangle_a \wedge \triangle_c$ and

$$H = (\overline{a_2} \vee \overline{c_2}) \wedge (\overline{a_3} \vee \overline{c_1}) \wedge (\overline{a_2} \vee \overline{c_3}) \wedge (\overline{a_1} \vee \overline{c_1}) \wedge (\overline{a_3} \vee \overline{c_1} \vee \overline{c_2})$$

- Set $a_2 = 0$, which implies $a_1 = a_3 = 1$, yielding: $P = \triangle_c$ and

$$H = (\overline{c_1}) \wedge (\overline{c_1}) \wedge (\overline{c_1} \vee \overline{c_2})$$

Unit clause rule: $c_1 = 0$. This satisfies all clauses of H and by setting $c_2 = c_3 = 1$, triangle $\triangle_c$ is also satisfied. Hence setting $b_1 = a_2 = c_1 = 0$ and the other variables to 1 satisfies $M = P \wedge H$.

ALGORITHM MH_F$^{\triangle}$

INPUT: Let a formula $M = (P \wedge H) \in$ MH_F$^{\triangle}$ be given and let k be the number of triangles in G_P.

OUTPUT: M **is satisfiable** if there is a satisfying truth assignment for M. Else M **is not satisfiable**.

begin

1.) Let $H = c_1 \wedge c_2 \wedge ... \wedge c_h$, where $|c_1| \leq \ldots \leq |c_h|$.

2.) Delete all clauses c_j, for which there is a clause c_i, $(i < j)$, such that $c_i \subset c_j$.

3.) Delete all clauses c_i of H, where $V(\triangle_j) \subset V(c_i)$, for a $\triangle_j \in G_P$.

4.) Set $V := \emptyset$, $W := \emptyset$, $i := 1$ and $s_H \leftarrow create$.

5.) For all $i = 1, \ldots, h$ set a Pointer $*$ in front of the first literal in the clause c_i:

$$(c_i, *) = (*\overline{x_{i_1}} \vee \ldots \vee \overline{x_{i_m}})$$

6.) Perform **Procedure Backtrack(c_1,$*$)**.

end

Procedure Backtrack(c_i,$*$)

begin

1.) Set all variables of the unit clauses and save all variables to which 0 is assigned in V and all variables assigned 1 in W. When assigning 0 to a variable x, search for the triangle $\triangle_x \in G_P$ also containing x and set the two other variables of $\triangle_x$ to 1. If there are complementary unit clauses or another contradiction, then the procedure stops and returns M **is not satisfiable**.

2.) If $(c_i, *) = (\overline{x_{i_1}} \vee \ldots \vee \overline{x_{i_l}} \vee \ldots \vee \overline{x_{i_m}}*)$ and $i = 1$, then the procedure stops end returns M **is unsatisfiable**.

3.) If $(c_i, *) = (\overline{x_{i_1}} \vee \ldots \vee \overline{x_{i_l}} \vee \ldots \vee \overline{x_{i_m}}*)$ and $i \neq 1$, then perform the following steps:

(a) Perform the operation pop_{s_H}: Delete all the clauses from the stack, which have been put there in the last procedure call, and insert them in the original order into H.

(b) Delete all variables x_i saved in V in the last procedure call from V and save them in W after deleting the two other variables from $\triangle_{x_i}$ from W.

(c) Verify, whether there are two variables of the same triangle saved in W. If yes, then set the third variable of this triangle to 0 and save it in $\widetilde{V}$.

(d) Set the Pointer at the beginning of the clause c_i:

$$c_i := (*\overline{x_{i_1}} \vee \ldots \vee \overline{x_{i_l}} \vee \ldots \vee \overline{x_{i_m}})$$

(e) Call the **Procedure Backtrack(c_{i-1},*)** .

4.) If $(c_i, *) = (\overline{x_{i_1}} \vee \ldots \vee *\overline{x_{i_l}} \vee \ldots \vee \overline{x_{i_m}})$, then set $j := i_l$.
5.) As long as $x_j \in W$ and $j \leq i_m$, augment j: $j = j + 1$.
6.) If $j = i_m + 1$, then perform the same steps as in 3. a)-e).
7.) If $x_j \notin W$ and $j \leq i_m$, then perform the following steps:

a) Set $x_j := 0; V := V \cup \{x_j\}; W := W \cup \triangle_{x_j}$;

b) Set the Pointer in front of $\overline{x_{j+1}}$:

$$(c_i, *) := (\overline{x_{i_1}} \vee \ldots \vee \overline{x_j} \vee *\overline{x_{j+1}} \vee \ldots \vee \overline{x_{i_m}})$$

c) Delete all clauses from H, which contain $\overline{x_j}$ and put them on stack s_H: $s_H := push_{s_H}(C_{x_j})$, where C_{x_j} is the set of all remaining Horn clauses containing the variable x_j.

d) If H is empty, then the procedure stops and returns M **is satisfiable**.

e) If H is not empty, then call the **Procedure Backtrack(c_{i+1},*)**.

end

Note that the algorithm works for arbitrary negative Horn parts. Specifically for l-uniform Horn parts, $l \in \mathbb{N}, l \geq 2$, a careful analysis of the running time of the algorithm yields the following results:

Theorem 8. *Algorithm* MH_F$^{\triangle}$ *decides satisfiability of l-uniform formulas $M \in$* MH_F$^{\triangle}$ *over n variables in time*

- $O(1.336^n)$, *for* $l = 3$.
- $O(1.384^n)$, *for* $l = 4$.
- $O(1.397^n)$, *for* $l = 5$.
- $O(1.408^n)$, *for* $l = 6$.
- $O(1.433^n)$, *for* $l = 7$.
- $O(1.436^n)$, *for* $l = 8$.

For $l = 3$ the running time is better than the so far best running time of $O(1.427^n)$ [29], for mixed Horn formulas with an arbitrary 3-uniform Horn part.

Remark 1. *For $l = 3$ a better running time of $O(1.273^n)$ can be achieved using a clever branching strategy. However, it is not trivial how to extend this branching algorithm to the cases $l \geq 4$.*

Proof. Correctness of Algorithm MH_F$^{\triangle}$:
Let $M = P \wedge H$ be an arbitrary formula in MH_F$^{\triangle}$ with $V(P) = V(H)$. As soon as a variable $x_j \in V(M)$ is set to 0, the two other variables of $\triangle x_j$ (the triangle to which x_j belongs) are immediately set to 1. This way we do not violate the

satisfiability of the P part because it is sufficient to set two variables of a triangle to 1 to satisfy the triangle. The Horn part is satisfied if all its clauses are on the stack s_H and thus the Horn part is empty. We put a clause $c_i \in H$ on a stack if it is satisfied by a variable which is set to 0. This is due to the fact that H is negative monotone. For each clause c_i we use a pointer $\cdot$ marking the one literal in c_i for which we have to check next whether assigning 0 (supposed it is not assigned to 1 yet) to its variable yields a model for F. As soon as a variable occurring in clause $c_i \in H$ is set to 0, the pointer moves over to its next literal and c_i itself is put on the stack s_H. If there is a clause $c_j \in H$ for which holds that the pointer is behind the last literal in c_j, then all variables of c_j have already been saved in W and thus set to 1. If $j = 1$, M is not satisfiable because in that case we have tested all possible assignments with one variable of c_j to 0 and none of them could yield a model for M. If $j \neq 1$, then we have to change the current partial assignment, therefore we backtrack: We take all the clauses already placed on stack s_H in the last $push_{s_H}()$-step, from the stack and include them in H again. From V we remove the variable x_p, belonging to the clause $c_{j-1} \in H$, which was saved in V in the last step and save x_p in W after having removed the two other variables from W which belong to the same triangle as x_p. Now the pointer is behind the literal $\overline{x_p}$ in front of $\overline{x_{p+1}}$, in the clause $c_{j-1} = (\ldots \vee \overline{x_p} \vee \cdot\, \overline{x_{p+1}} \vee \ldots)$. In the next step the algorithm accordingly checks whether $x_{p+1} = 0$ yields a model for M: We set the two other variables of the triangle containing x_{p+1} to 1 and save them in W. Then we put all clauses of H containing $\overline{x_{p+1}}$ on the stack s_H in a fixed order by the command $push_{s_H}(c_{x_{p+1}})$ and remove them from H. Via this procedure we test all possible assignments to 0 of variables to satisfy clauses in H without violating the satisfiability of P. All in all Algorithm MH_F$^{\triangle}$ works by using **depth-first search**: As soon as we note a contradiction implying that the current partial assignment does not yield a model for M, we backtrack, i.e. we undo the assignment of the last step and check whether another assignment can satisfy the formula. Therefore Algorithm $MH\_F^{\triangle}$ tests for all possible assignments whether they satisfy M and it outputs M **is not satisfiable**, if none of them can satisfy M. If Algorithm $MH\_F^{\triangle}$ has found a model for M then it outputs M **is satisfiable**.

Analysis of the running time:

In the following let l denote the length of clauses in H and k the number of triangles in P, hence $k = n/3$, where n is the number of variables of a formula $M = P \wedge H \in MH\_F^{\triangle}$. For this running time analysis we assume that the clauses of H and the literals in each clause in H have a fixed order, for each formula $M = P \wedge H \in MH\_F^{\triangle}$.

Further, we denote with $WHN_k^l \subset MH\_F^{\triangle}$ a class of worst-case formulas for Algorithm $MH\_F^{\triangle}$ which are defined as follows.

The P part of the formulas $M = P \wedge H$ in WHN_l^k consists of k triangles which are designated with a, b, c, ..., where $k \geq \lceil \frac{l}{2} \rceil$. The Horn part H is negative monotone,

l-uniform, for $l \geq 2$, and has the following form:

$$
\begin{aligned}
&(\overline{a_1} \vee \overline{a_2} \vee \overline{b_1} \vee \overline{b_2} \vee \overline{c_1} \vee \overline{c_2} \vee \overline{d_1} \vee \overline{d_2} \vee \ldots)\\
&(\overline{b_1} \vee \overline{b_2} \vee \overline{c_1} \vee \overline{c_2} \vee \overline{d_1} \vee \overline{d_2} \vee \overline{e_1} \vee \overline{e_2} \vee \ldots)\\
&(\overline{c_1} \vee \overline{c_2} \vee \overline{d_1} \vee \overline{d_2} \vee \overline{e_1} \vee \overline{e_2} \vee \overline{f_1} \vee \overline{f_2} \vee \ldots)\\
&(\overline{d_1} \vee \overline{d_2} \vee \overline{e_1} \vee \overline{e_2} \vee \overline{f_1} \vee \overline{f_2} \vee \overline{g_1} \vee \overline{g_2} \vee \ldots)\\
&(\overline{e_1} \vee \overline{e_2} \vee \overline{f_1} \vee \overline{f_2} \vee \overline{g_1} \vee \overline{g_2} \vee \overline{h_1} \vee \overline{h_2} \vee \ldots)\\
&\vdots
\end{aligned}
$$

We denote with a_1, a_2, a_3 the variables of triangle a; by b_1, b_2, b_3 the variables of triangle b; by c_1, c_2, c_3 the variables of triangle c and so forth. We assume that there are altogether k triangles in P. If l is even, each clause of H has exactly two variables from each of the $l/2$ many triangles. If l is odd, then each clause of H contains exactly two variables from each of the $(l-1)/2$ many triangles and one variable from another triangle which we have not considered yet. Apart from the first two all variables of clause $c_i \in H$ are also contained in clause c_{i+1}. But clause c_{i+1} additionally contains two other variables, either (in case l is even) from a further triangle which we have not considered so far, or (in case l is odd) c_{i+1} contains one variable from the triangle of the last literal of clause c_i and a further variable from another triangle which we have not considered yet. We build the first k many clauses according to this scheme. Note that to make the last clauses l-uniform, for which there are no triangles left which were not already considered, we have to "fill up " these with arbitrary variables that have not been used so far. After building the first k many clauses of H this way further clauses are allowed to be of arbitrary feature as long as they are negative monotone and l-uniform. That means each formula belongs to the class WHN_l^k for which holds that by permutation of its Horn clauses we can achieve that its first k Horn clauses have the feature described above.

We show that among all input formulas with a 3-uniform Horn part formulas of the class WHN_3^k belong to the worst-case input formulas for Algorithm $MH\_F^{\triangle}$. Inductively we prove for each $l > 3$, that formulas of WHN_l^k belong to the worst-case input formulas for Algorithm $MH\_F^{\triangle}$ concerning the running time.
As already described above the Horn part of formulas in $WHN_3^k \subset MH\_F^{\triangle}$ has the following feature:

$$
\begin{aligned}
&(\overline{a_1} \vee \overline{a_2} \vee \overline{b_1})\\
&(\overline{b_1} \vee \overline{b_2} \vee \overline{c_1})\\
&(\overline{c_1} \vee \overline{c_2} \vee \overline{d_1})\\
&(\overline{d_1} \vee \overline{d_2} \vee \overline{e_1})\\
&(\overline{e_1} \vee \overline{e_2} \vee \overline{x_3})\\
&\vdots
\end{aligned}
$$

Each clause $c_i \in H$ contains two variables of the same triangle and the third variable belongs to another triangle not considered yet, so that the next clause $c_{i+1} \in H$ also

contains this third variable and additionally another variable of the same triangle. This way we can construct the first $k-1$ clauses. For constructing the kth clause c_k of H we choose the third variable in c_k from the set of all variables which do not appear in H yet and do not belong to the same triangle as the first two variables of c_k. The remaining clauses of the Horn part of a formula in class WHN_3^k may have arbitrary features as long as they are negative monotone and 3-uniform.
In the following we assume for each formula in $MH\_F^{\triangle}$ that all its Horn clauses and the literals in a Horn clause are enumerated and thus have a fixed order: $H = c_1 \wedge c_2 \wedge \ldots \wedge c_{|H|}$.
According to the function of Algorithm $MH\_F^{\triangle}$ a *search tree* t belonging to a formula $M = P \wedge H \in MH\_F^{\triangle}$ is constructed as follows: In the beginning we introduce a dummy vertex w, which is the root of the search tree and whose sons are the variables a_1, a_2, b_1, b_2,... of the first clause $c_1 = (a_1 \vee a_2 \vee b_1 \vee b_2 \vee \ldots)$ in H. All vertices of t, apart from the root, are the variables which we set to 0, according Algorithm $MH\_F^{\triangle}$, and we check whether this assignment yields a model for M. As soon as a variable x is set to 0, the two other variable of the triangle to which x belongs are set to 1 and saved in the set W. Clauses in H which contain a variable that is already set to 0 have to be removed from H and put on the stack s_H. Afterwards, we consider the current first clause of H. Variables which are already fixed to 1 are not allowed to appear as sons of a vertex in t. If the P part of M contains k many triangles, then the depth of the search tree t is at most k. We enumerate the sons of a vertex in t from left to right by $1, 2, 3$. Regarding the subtree whose root vertex is the second son, the variable which corresponds to the first son vertex is set to 1 because searching for a model with its assignment to 0 is unsuccessful. As soon as two variables of the same triangle are set to 1, we immediately assign 0 to the third variable. So, if the first two sons of a vertex x belong to the same triangle, then we set the third variable of this triangle to 0 as soon as we consider the subtree whose root vertex is the third son of x.

Definition 2. *We call two formulas F_1 and F_2 isomorphic, if F_1 and F_2 have the same clauses but in possibly different orders.*

Definition 3. *Let t be a search tree for Algorithm $MH\_F^{\triangle}$ and let F in $MH\_F^{\triangle}$ be a formula with a fixed order of its Horn clauses. We say F results from t, if the following holds: Let all sons of a vertex in t establish a clause. Writing the clauses occurring in t consecutively in a level from left to right and from upstairs to downstairs we obtain F.*

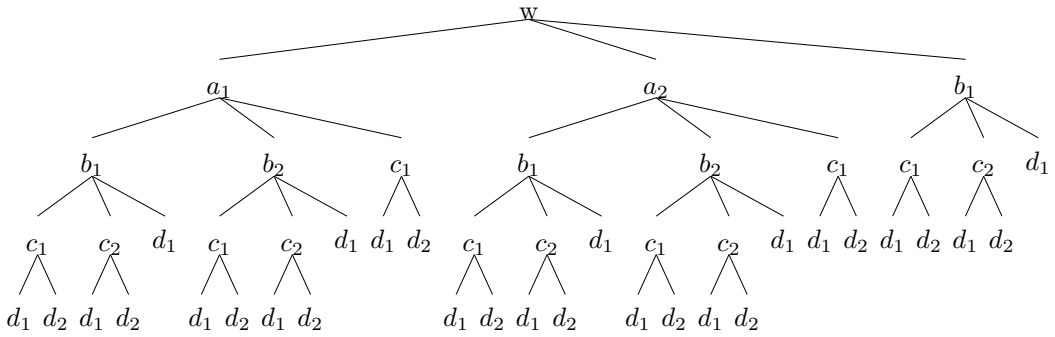
If there was a vertex with only two sons, then the corresponding clause would consist of two literals only, but as we have 3-uniform Horn formulas, we can fill up such clauses with an arbitrary literal. The only thing we have to take note of here is that the third literal does not appear in the tree above as a vertex. Put differently the tree structure should not be destroyed by the choice of the third literal.

Let k triangles be given. A *maximal search tree for k triangles* for Algorithm $MH\_F^{\triangle}$ is a tree with the maximal number of leaves among all trees which we obtain by setting exactly one variable to 0 in each triangle. The vertices of such a tree symbolise the variables we set to 0 in the Algorithm $MH\_F^{\triangle}$.

Here we also have to take into account that as soon as a variable is set to 0 the two other variables of the same triangle must be set to 1. If the search from a vertex in t was unsuccessful, we assign 1 to this vertex before considering the

subtree whose root vertex is its brother vertex. If the first two sons of a vertex in t correspond to variables of the same triangle, we set the third variable of this triangle to 0 when treating the subtree whose root is the third son.

Example:
Consider the 4 triangles a, b, c, d and let M be a formula in WHN_3^4. We obtain the following maximal search tree for Algorithm $MH\_F^{\triangle}$:



As soon as Algorithm $MH\_F^{\triangle}$ concluded that a model for M with the partial assignment $a_1 = 0$, $a_2 = 0$ does not exist, it assigns $a_1 = a_2 = 1, b_1 = 0$ and considers the subtree with root vertex b_1. We get $a_1 = a_2 = 1$ and thus we assign $a_3 = 0$. As $b_1 = a_3 = 0$ we only have $k - 2$ triangles left.

The following Theorem states that concerning the running time formulas in WHN_3^k establish the worst-case formulas for Algorithm $MH\_F^{\triangle}$.

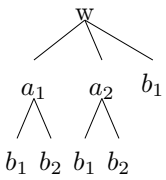
Theorem 9. *The Horn part of a formula F which results from a maximal search tree t for k triangles for Algorithm $MH\_F^{\triangle}$ is isomorph to the Horn part of a formula in WHN_3^k.*

We show the assertion of Theorem 9 by complete induction.

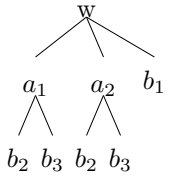
Proof. First we assume that in each clause the literals of the same triangle are written side by side.
Beginning of the induction: $k = 2$. Consider the two triangles $\triangle_a$ and $\triangle_b$. We denote by a_1, a_2, a_3 the variables of triangle a. The same also holds for triangle b.

In this case there are only two different (concerning their structure) maximal search trees for Algorithm $MH\_F^{\triangle}$, namely:

1.)



2.)



We obtain all other search trees for Algorithm $MH\_F^{\triangle}$ with five leaves by either exchanging a_i for b_i or permutating the indices of the variables in a triangle.

The formula resulting from the first search tree has the following form:

$$
\begin{aligned}
&(\overline{a_1} \vee \overline{a_2} \vee \overline{b_1})\\
&(\overline{b_1} \vee \overline{b_2} \vee \overline{a_3})\\
&\vdots
\end{aligned}
$$

Evidently this formula belongs to the class WHN_3^2.

The formula resulting from the second search tree has the following form:

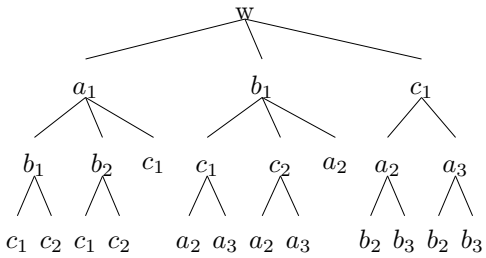
$$
\begin{aligned}
&(\overline{a_1} \vee \overline{a_2} \vee \overline{b_1})\\
&(\overline{b_2} \vee \overline{b_3} \vee \overline{a_1})\\
&(\overline{b_2} \vee \overline{b_3} \vee \overline{a_2})\\
&\vdots
\end{aligned}
$$

Obviously this formula is isomorphic to:

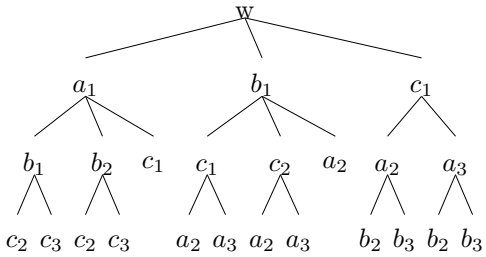
$$
\begin{aligned}
&(\overline{b_2} \vee \overline{b_3} \vee \overline{a_1})\\
&(\overline{a_1} \vee \overline{a_2} \vee \overline{b_1})\\
&(\overline{b_2} \vee \overline{b_3} \vee \overline{a_2})\\
&\vdots
\end{aligned}
$$

Because of its structure this formula also belongs to WHN_3^2.

To clarify the idea of the induction step, we additionally consider the case $k = 3$. Let the triangles $\triangle_a$, $\triangle_b$ and $\triangle_c$ be given. We denote the variables of triangle a with a_1, a_2, a_3. The same also holds for triangles b and c. In this case we obtain two (concerning their structure) different maximal search trees for Algorithm $MH\_F^{\triangle}$: 3.)



4.)



Note that we obtain all other maximal search trees for Algorithm $MH\_F^{\triangle}$ with 14 leaves by permutation of the indices of the variables in a triangle or by exchanging the variables a, b and c. The formula resulting from the third search tree is:

$$
\begin{aligned}
&(\overline{a_1} \vee \overline{b_1} \vee \overline{c_1})\\
&(\overline{b_1} \vee \overline{b_2} \vee \overline{c_1})\\
&(\overline{c_1} \vee \overline{c_2} \vee \overline{a_2})\\
&(\overline{a_2} \vee \overline{a_3} \vee \overline{c_2})\\
&(\overline{c_1} \vee \overline{c_2} \vee \overline{a_3})\\
&(\overline{a_2} \vee \overline{a_3} \vee \overline{c_3})\\
&(\overline{b_2} \vee \overline{b_3} \vee \overline{a_1})\\
&\vdots
\end{aligned}
$$

By permutation of its clauses we obtain:

$$
\begin{aligned}
&(\overline{b_1} \vee \overline{b_2} \vee \overline{c_1})\\
&(\overline{c_1} \vee \overline{c_2} \vee \overline{a_2})\\
&(\overline{a_2} \vee \overline{a_3} \vee \overline{c_3})\\
&\vdots
\end{aligned}
$$

Because of its structure this formula belongs to the class WHN_3^3. The following formula results from the fourth search tree:

$$
\begin{aligned}
&(\overline{a_1} \vee \overline{b_1} \vee \overline{c_1})\\
&(\overline{b_1} \vee \overline{b_2} \vee \overline{c_1})\\
&(\overline{c_1} \vee \overline{c_2} \vee \overline{a_2})\\
&(\overline{a_2} \vee \overline{a_3} \vee \overline{c_2})\\
&(\overline{c_2} \vee \overline{c_3} \vee \overline{a_3})\\
&(\overline{a_2} \vee \overline{a_3} \vee \overline{b_3})\\
&(\overline{b_2} \vee \overline{b_3} \vee \overline{a_1})\\
&\vdots
\end{aligned}
$$

By permutation of its clauses we obtain:

$$
\begin{aligned}
&(\overline{c_2} \vee \overline{c_3} \vee \overline{a_3})\\
&(\overline{a_3} \vee \overline{a_2} \vee \overline{b_3})\\
&(\overline{b_3} \vee \overline{b_2} \vee \overline{a_1})\\
&\vdots
\end{aligned}
$$

Obviously this formula belongs to the class WHN_3^3.
Induction step: $k \to k+1$.
The induction hypothesis is: For each maximal search tree for Algorithm $MH\_F^{\triangle}$ for k triangles the resulting formula F is isomorphic to a formula in WHN_3^k. We show that this also holds for $k+1$: Let $\triangle_1, \triangle_2, \ldots, \triangle_{k+1}$ be $k+1$ many triangles. We now construct a maximal search tree for $k+1$ many triangles. We choose an arbitrary $i \in \{1, \ldots, k+1\}$ and assign 0 to one variable of $\triangle_i$. Let us w.l.o.g. assign 0 to i_1 (and 1 to the two other variables of $\triangle_i$). So the variables of triangle i are fixed and we only have k triangles $\triangle_1, \triangle_2, \ldots, \triangle_{i-1}, \triangle_{i+1}, \ldots, \triangle_{k+1}$ left for the subtree whose root vertex is i_1. Hence we can apply the induction hypothesis on the k remaining triangles obtaining: For each maximal search tree for the k triangles $\triangle_1, \triangle_2, \ldots, \triangle_{i-1}, \triangle_{i+1}, \ldots, \triangle_{k+1}$ for Algorithm $MH\_F^{\triangle}$, the resulting formula $\widetilde{F}$ is isomorphic to a formula in WHN_3^k. W.l.o.g. let $\widetilde{F}$ be isomorphic to

the following formula in WHN_3^k:

$$
\begin{array}{l}
(\overline{(i+1)_1} \vee \overline{(i+1)_2} \vee \overline{(i+2)_1}) \\
\ldots \\
(\overline{j_1} \vee \overline{j_2} \vee \overline{(j+1)_1}) \\
(\overline{(j+1)_1} \vee \overline{(j+1)_2} \vee \overline{(j+2)_1}) \\
\ldots \\
(\overline{(k+1)_1} \vee \overline{(k+1)_2} \vee \overline{1_1}) \\
(\overline{1_1} \vee \overline{1_2} \vee \overline{2_1}) \\
\ldots \\
(\overline{(i-1)_1} \vee \overline{(i-1)_2} \vee \overline{i_3})
\end{array}
$$

We choose a $j \in \{1, \ldots, k+1\}$ and set one of the three variables of $\triangle_j$ to 0. W.l.o.g. we assign $j_1 = 0$ (and 1 to the two other variables of $\triangle_j$), so that all variables of $\triangle_j$ are fixed and we have k many triangles $\triangle_1, \triangle_2, \ldots, \triangle_{j-1}, \triangle_{j+1}, \ldots, \triangle_{k+1}$ left for the subtree with root vertex j_1. Note that now for triangle i holds: $i_1 = 1$. We assume that $j \neq i$. Now we can apply the induction hypothesis to the k remaining triangles obtaining the following: The resulting formula $\widehat{F}$, for each maximal search tree for the triangles $\triangle_1, \triangle_2, \ldots, \triangle_{j-1}, \triangle_{j+1}, \ldots, \triangle_{k+1}$, is isomorphic to a formula in WHN_3^k. W.l.o.g. let $\widehat{F}$ be isomorphic to the following formula in WHN_3^k:

$$
\begin{array}{l}
(\overline{(j+1)_1} \vee \overline{(j+1)_2} \vee \overline{(j+2)_1}) \\
\ldots \\
(\overline{(k+1)_1} \vee \overline{(k+1)_2} \vee \overline{1_1}) \\
(\overline{1_1} \vee \overline{1_2} \vee \overline{2_1}) \\
\ldots \\
(\overline{(i-2)_1} \vee \overline{(i-2)_2} \vee \overline{(i-1)_1}) \\
(\overline{(i-1)_1} \vee \overline{(i-1)_2} \vee \overline{(i+1)_2}) \\
\ldots \\
(\overline{(j-1)_1} \vee \overline{(j-1)_2} \vee \overline{i_2}) \\
(\overline{i_2} \vee \overline{i_3} \vee \overline{j_3})
\end{array}
$$

Composing the clauses of $\widetilde{F}$, $\widehat{F}$ we obtain:

$$
\begin{array}{l}
(\overline{j_1} \vee \overline{j_2} \vee \overline{(j+1)_1}) \\
(\overline{(j+1)_1} \vee \overline{(j+1)_2} \vee \overline{(j+2)_1}) \\
\ldots \\
(\overline{(k+1)_1} \vee \overline{(k+1)_2} \vee \overline{1_1}) \\
(\overline{1_1} \vee \overline{1_2} \vee \overline{2_1}) \\
\ldots \\
(\overline{(i-2)_1} \vee \overline{(i-2)_2} \vee \overline{(i-1)_1}) \\
(\overline{(i-1)_1} \vee \overline{(i-1)_2} \vee \overline{(i+1)_2}) \\
\ldots \\
(\overline{(j-1)_1} \vee \overline{(j-1)_2} \vee \overline{i_2}) \\
(\overline{i_2} \vee \overline{i_3} \vee \overline{j_3})
\end{array}
$$

This is obviously a formula of WHN_3^{k+1}.
In case $j = i$ we obtain a formula of class WHN_3^{k+1} by adding the clause

$$(\overline{i_1} \vee \overline{i_2} \vee \overline{(i+1)_1})$$

to $\widetilde{F}$ in front of its first clause:

$$
\begin{array}{l}
(\overline{i_1} \vee \overline{i_2} \vee \overline{(i+1)_1}) \\
(\overline{(i+1)_1} \vee \overline{(i+1)_2} \vee \overline{(i+2)_1}) \\
\ldots \\
(\overline{(k+1)_1} \vee \overline{(k+1)_2} \vee \overline{1_1}) \\
(\overline{1_1} \vee \overline{1_2} \vee \overline{2_1}) \\
\ldots \\
(\overline{(i-1)_1} \vee \overline{(i-1)_2} \vee \overline{i_3})
\end{array}
$$

□

So the search tree for Algorithm $MH\_F^{\triangle}$ belonging to formulas in WHN_3^k is maximal and hence formulas of class WHN_3^k are the worst-case formulas for Algorithm $MH\_F^{\triangle}$ concerning the running time. This results from the fact that as soon as we have found a maximal search tree (concerning the number of its leaves), the resulting formula F is isomorphic to a formula in WHN_3^k. Thus by a permutation of its clauses we can transfer F to a formula in WHN_3^k. To calculate the running time of Algorithm $MH\_F^{\triangle}$ for fomulas in $MH\_F^{\triangle}$ with l-uniform Horn parts and k many triangles in P it therefore suffices to consider the class WHN_l^k because its formulas are the most difficult formulas for Algorithm $MH\_F^{\triangle}$ concerning the running time.

Calculating the number of leaves in the search tree for Algorithm $MH\_F^{\triangle}$ which belongs to a formula in WHN_3^k yields the following results:

$\underline{k=2}$: $2^k + 2^{k-2}$

$\underline{k=3}$: $(2^k + 2^{k-2}) + 2^{k-2} = 2^k + 2\cdot 2^{k-2}$

$\underline{k=4}$: $(2^k + 2\cdot 2^{k-2}) + (2^{k-2} + 2^{k-4}) = 2^k + 3\cdot 2^{k-2} + 2^{k-4}$

$\underline{k=5}$: $(2^k + 3\cdot 2^{k-2} + 2^{k-4}) + (2^{k-2} + 2\cdot 2^{k-4} + 2^{k-4}) = 2^k + 4\cdot 2^{k-2} + 3\cdot 2^{k-4}$

$\underline{k=6}$: $(2^k + 4\cdot 2^{k-2} + 3\cdot 2^{k-4}) + (2^{k-2} + 3\cdot 2^{k-4} + 2^{k-6})$
$= (2^k + 5\cdot 2^{k-2} + 6\cdot 2^{k-4} + 2^{k-6})$

$\underline{k=7}$: $\underbrace{(2^k + 5\cdot 2^{k-2} + 6\cdot 2^{k-4} + 2^{k-6})}_{S_1} + \underbrace{(2^{k-2} + 4\cdot 2^{k-4} + 3\cdot 2^{k-6})}_{S_2}$
$= 2^k + 6\cdot 2^{k-2} + 10\cdot 2^{k-4} + 4\cdot 2^{k-6}$

The number of leaves in the search tree for a formula of the class WHN_3^k corresponds to the following polynomial:

$$p_k(k) = 2^k + (k-1)2^{k-2} + 2^{k-2} + \ldots + c_x\cdot 2^{k-x} + \ldots + .$$

For an arbitrary k we obtain $p_k(k)$ as follows: We consider $p_{k-1}(k-1)$, the polynomial for $k-1$ many triangles: Let $p_{k-1}(k-1) = S_1 + S_2$, where S_1, S_2 are the two summands consisting of 2-powers each. Then $p_{k-1}(k)$ is the 2-polynomial, for $k-1$ triangles where the exponent of each 2-power is increased at 1. Then we obtain: $p_k(k) = p_{k-1}(k) + S_1^{-1}$ with S_1^{-1} meaning that the exponents of each 2-powers occurring in S_1 are reduced at 1.

Altogether we obtain the following for the coefficients of the 2-powers:

- 2^k
- $(k-1)2^{k-2}$
- $(\sum_{j_1=1}^{k-3} j)2^{k-4} = \frac{(k-3)(k-2)}{2}2^{k-4} = O(k^2\cdot 2^k)$
- $$(\sum_{j_1=1}^{k-5}\sum_{j_2=1}^{j_1} j_2)2^{k-6} = \frac{1}{2}\sum_{j_1=1}^{k-5}(j_1^2 + j_1)2^{k-6}$$
  $$= (\frac{(k-5)(k-4)}{4} + \frac{(k-5)(k-4)(2(k-5)+1)}{12})2^{k-6}$$
  $$= O(k^3\cdot 2^{k-6})$$
- ...
- $(\sum_{j_1=1}^{k-(x-1)}\sum_{j_2=1}^{j_1}\cdots\sum_{j_{x/2-1}=1}^{j_{x/2-2}} m)2^{k-x}$, where $x = 2,4,6\ldots,\begin{cases}k, & k \text{ even}\\ k-1, & k \text{ odd}\end{cases}$
- ...
- $\begin{cases}(\frac{k^2}{8} + \frac{k}{4})2^2, & \text{if } k \text{ is even}\\ O(k^3)2^3, & \text{if } k \text{ is odd}\end{cases}$

- $\begin{cases} 2^0, & \text{if } k \text{ is even} \\ (\frac{k+1}{2})2^1, & \text{if } k \text{ is odd} \end{cases}$

Consequently we get:

$$
\begin{aligned}
& 2^k + (k-1)2^{k-2} + (\sum_{j_1=1}^{k-3} j_1)2^{k-4} + (\sum_{j_1=1}^{k-5}\sum_{j_2=1}^{j_1} j_2)2^{k-6} + (\sum_{j_1=1}^{k-7}\sum_{j_2=1}^{j_1}\sum_{j_3=1}^{j_2} j_3)2^{k-8} \\
& + ... + (\sum_{j_1=1}^{k-(x-1)}\sum_{j_2=1}^{j_1}\cdots\sum_{j_{x/2-1}=1}^{j_{x/2-2}} j_{x/2-1})2^{k-x} + \dots \\
& + \begin{cases} 2^0, & \text{if } k \text{ is even} \\ \sum_{j_1=1}^{2}\sum_{j_2=1}^{j_1}\cdots\sum_{j_{(k-3)/2}=1}^{j_{(k-5)/2}} j_{(k-3)/2})2^1, & \text{if } k \text{ is odd} \end{cases} \\
=& 2^k + (k-1)2^{k-2} + (\frac{(k-3)(k-2)}{2})2^{k-4} + (\sum_{j_1=1}^{k-5}\frac{(j_1+1)j_1}{2})2^{k-6} \\
& + (\sum_{j_1=1}^{k-7}\sum_{j_2=1}^{j_1}\sum_{j_3=1}^{j_2} j_3)2^{k-8} + \dots + (\sum_{j_1=1}^{k-(x-1)}\sum_{j_2=1}^{j_1}\cdots\sum_{j_{x/2-1}=1}^{j_{x/2-2}} j_{x/2-1})2^{k-x} \\
& + \dots + \begin{cases} 2^0, & \text{if } k \text{ is even} \\ (\frac{k+1}{2})2^1, & \text{if } k \text{ is odd} \end{cases}
\end{aligned}
$$

Proceeding we calculate the coefficient of 2^{k-x} of the 2-polynomial:

$$
\begin{aligned}
& (\sum_{j_1=1}^{k-(x-1)}\sum_{j_2=1}^{j_1}\cdots\sum_{j_{x/2-2}=1}^{j_{x/2-3}}\sum_{j_{x/2-1}=1}^{j_{x/2-2}} j_{x/2-1})2^{k-x} \\
& \approx (\int_{j_1=1}^{k-(x-1)}\int_{j_2=1}^{j_1}\cdots\int_{j_{x/2-2}=1}^{j_{x/2-3}}\int_{j_{x/2-1}=1}^{j_{x/2-2}} j_{x/2-1}\, dj_{x/2-1}\dots dj_1)2^{k-x} \\
& \approx (\int_{j_1=0}^{k-(x-1)}\int_{j_2=0}^{j_1}\cdots\int_{j_{x/2-2}=0}^{j_{x/2-3}}\int_{j_{x/2-1}=0}^{j_{x/2-2}} j_{x/2-1}\, dj_{x/2-1}\dots dj_1)2^{k-x} \\
& = (\int_{j_1=0}^{k-(x-1)}\int_{j_2=0}^{j_1}\cdots\int_{j_{x/2-2}=0}^{j_{x/2-3}}\frac{j_{x/2-2}^2}{2}\, dj_{x/2-2}\dots dj_1)2^{k-x} \\
& = (\frac{1}{2}\frac{1}{3}\cdots\frac{1}{x/2-1}\int_{j_1=0}^{k-(x-1)} j_1^{x/2-1}\, dj_1)2^{k-x} \\
& = (\frac{1}{2}\frac{1}{3}\cdots\frac{1}{x/2-1}\frac{1}{x/2}(k-x+1)^{x/2})2^{k-x} \\
& = \frac{(k-x+1)^{x/2}}{(x/2)!}2^{k-x}
\end{aligned}
$$

To calculate the faculty we use Stirling's formula:

$$(x/2)! \approx \sqrt{2\pi\frac{x}{2}}(\frac{x/2}{e})^{\frac{x}{2}} \geq (\frac{x/2}{e})^{\frac{x}{2}}$$

Hence:

$$\frac{(k-x+1)^{x/2}}{(x/2)!}2^{k-x} \leq \frac{2^{k-x}((k-x+1)2e)^{x/2}}{x^{\frac{x}{2}}} = O\left(2^{k-x}\left(\frac{(k-x)2e}{x}\right)^{\frac{x}{2}}\right)$$

As x depends on k we set $x = yk$ where $0 < y \leq 1$ resulting in:

$$O\left(2^{k-x}\left(\frac{(k-x)2e}{x}\right)^{\frac{x}{2}}\right) = O\left(\left(2^{(1-y)}\left(\frac{(1-y)2e}{y}\right)^{\frac{y}{2}}\right)^{k}\right)$$

Further we calculate the maximum of the function

$$f(y) = \left(2^{(1-y)}\left(\frac{(1-y)2e}{y}\right)^{\frac{y}{2}}\right)^{k}$$

for $0 < y \leq 1$. Obviously the maximum of f is at $y = 0,260209k$ so that $f(y) = 2,3845586^k$. As a result the Algorithm $MH\_F^{\triangle}$ has a running time of $O(2,3845586^k) = O(1.336^n)$, for a 3-uniform Horn part with n variables.

We now investigate the running time of Algorithm $MH\_F^{\triangle}$ for $l = 4, 5, 6, 7, 8$ and show that it has a better running time than $O(3^k)$ for $k \leq 4000$. We again exploit the fact that Algorithm $MH\_F^{\triangle}$ has a worst-case running time for formulas of class WHN_l^k.

$\underline{l = 4}$: Calculating the number of leaves in the search tree for Algorithm $MH\_F^{\triangle}$ belonging to a formula in WHN_4^k we get the following:

$\underline{k = 2}$: $2^k + 2^{k-1}$

$\underline{k = 3}$: $(2^k + 2^{k-1}) + 2^{k-1} = 2^k + 2\cdot 2^{k-1}$

$\underline{k = 4}$: $(2^k + 2\cdot 2^{k-1}) + (2^{k-1} + 2^{k-2}) = 2^k + 3\cdot 2^{k-1} + 2^{k-2}$

$\underline{k = 5}$: $(2^k + 3\cdot 2^{k-1} + 2^{k-2}) + (2^{k-1} + 2\cdot 2^{k-2}) = 2^k + 4\cdot 2^{k-1} + 3\cdot 2^{k-2}$

$\underline{k = 6}$: $(2^k + 4\cdot 2^{k-1} + 3\cdot 2^{k-2}) + (2^{k-1} + 3\cdot 2^{k-2} + 2^{k-3})$

$= (2^k + 5\cdot 2^{k-1} + 6\cdot 2^{k-2} + 2^{k-3})$

$\underline{k = 7}$: $\underbrace{(2^k + 5\cdot 2^{k-1} + 6\cdot 2^{k-2} + 2^{k-3})}_{S_1} + \underbrace{(2^{k-1} + 4\cdot 2^{k-2} + 3\cdot 2^{k-3})}_{S_2}$

$= 2^k + 6\cdot 2^{k-1} + 10\cdot 2^{k-2} + 4\cdot 2^{k-3}$

$\underline{k = 8}$: ...

For an arbitrary k we obtain $p_k(k)$ as follows: We consider $p_{k-1}(k-1)$, the 2-polynomial for $k-1$ many triangles: Let $p_{k-1}(k-1) = S_1 + S_2$, where S_1, S_2 are two summands whith each one consisting of 2-powers. Then $p_{k-1}(k)$ is the 2-polynomial, for $k-1$ triangles, where the exponent of each 2-power is increased at 1 and we obtain: $p_k(k) = p_{k-1}(k) + S_1$. So for the coefficients of the 2-powers we get the follwing:

- 2^k
- $(k-1)2^{k-1}$

- $(\sum_{j_1=1}^{k-3} j)2^{k-2}$
- $(\sum_{j_1=1}^{k-5} \sum_{j_2=1}^{j_1} j_2)2^{k-3}$
- ...
- $(\sum_{j_1=1}^{k-(2x-1)} \sum_{j_2=1}^{j_1} \cdots \sum_{j_{x-1}=1}^{j_{x-2}} j_{x-1})2^{k-x}$, $x = 1, 2, \ldots, \begin{cases} k/2, & k \text{ even} \\ (k+1)/2, & k \text{ odd} \end{cases}$

As already shown above we obtain:

$$(\sum_{j_1=1}^{k-(2x-1)} \sum_{j_2=1}^{j_1} \cdots \sum_{j_{x-1}=1}^{j_{x-2}} j_{x-1})2^{k-x} \approx \frac{(k-2x+1)^x}{x!}2^{k-x}$$

Subsequently we calculate the maximum of the function

$$f_k(x) = \frac{(k-2x+1)^x}{x!}2^{k-x}$$

where

$$x = 1, 2, 3, \ldots, \begin{cases} k/2, & \text{if } k \text{ is even} \\ (k+1)/2, & \text{if } k \text{ is odd} \end{cases}$$

By using a computer program we determine the maximal summand $c_x \cdot 2^{k-x}$ of $p_k(k)$ for $k = 1, \ldots, 4000$ and obtain: $c_x \cdot 2^{k-x} \leq 2.65^k$ for $k = 1, \ldots, 4000$. Thus Algorithm $MH\_F^{\triangle}$ has a running time of $O(2.65^k) = O(1.384^n)$ for formulas in $MH\_F^{\triangle}$ with n variables and whose Horn clauses are 4-uniform.

$\underline{l = 5}$:
Let $p(k)$ be the 2-polynomial, which corresponds to the number of leaves in the search tree for Algorithm $MH\_F^{\triangle}$ for formulas in WHN_5^k. Then we obtain:

$\underline{k = 2}$: $2^k + 2^{k-1}$

$\underline{k = 3}$: $(2^k + 2^{k-1}) + 2^{k-1} + 2^{k-3} = 2^k + 2 \cdot 2^{k-1} + 2^{k-3}$

$\underline{k = 4}$: $(2^k + 2 \cdot 2^{k-1} + 2^{k-3}) + (2^{k-1} + 2^{k-2}) + 2^{k-3}$
$= 2^k + 3 \cdot 2^{k-1} + 2^{k-2} + 2 \cdot 2^{k-3}$

$\underline{k = 5}$: $\underbrace{(2^k + 3 \cdot 2^{k-1} + 2^{k-2} + 2 \cdot 2^{k-3})}_{S_1} + \underbrace{(2^{k-1} + 2 \cdot 2^{k-2} + 2^{k-4})}_{S_2} + \underbrace{(2^{k-3} + 2^{k-4})}_{S_3}$
$= 2^k + 4 \cdot 2^{k-1} + 3 \cdot 2^{k-2} + 3 \cdot 2^{k-3+2} \cdot 2^{k-4}$

$\underline{k = 6}$: ...

For an arbitrary k we get $p_k(k)$ as follows: We consider $p_{k-1}(k-1)$, the 2-polynomial for $k-1$ many triangles: Let $p_{k-1}(k-1) = S_1 + S_2 + S_3$, where S_1, S_2, S_3 are three summands, each of them also consisting of 2-powers. Then $p_{k-1}(k)$ is the 2-polynomial, for $k-1$ many triangles, where we increase the exponent of each 2-power at 1. Then we obtain: $p_k(k) = p_{k-1}(k) + S_1 + S_2^{-1}$, where S_2^{-1} means that the exponents of all 2-powers occurring in S_2 are reduced at 1. We get the

following for the coefficients of the 2-powers:

$\rightarrow$ $\underline{2^k}$: 1

$\rightarrow$ $\underline{2^k}$: $(k-1)$

$\rightarrow$ $\underline{2^{k-2}}$: $\sum_{j=1}^{k-3} j$

$\rightarrow$ $\underline{2^{k-3}}$: $\sum_{j_1=1}^{k-5} \sum_{j_2=1}^{j_1} j_2 + \sum_{j_1=1}^{k-2} 1$

$\rightarrow$ $\underline{2^{k-4}}$: $\sum_{j_1=1}^{k-7} \sum_{j_2=1}^{j_1} \sum_{j_3}^{j_2} j_3 + 2 \sum_{j_1=1}^{k-4} j_1$

$\rightarrow$ $\underline{2^{k-5}}$: $\sum_{j_1=1}^{k-9} \sum_{j_2=1}^{j_1} \sum_{j_3=1}^{j_2} \sum_{j_4=1}^{j_3} j_4 + 3 \sum_{j_1=1}^{k-6} \sum_{j_2=1}^{j_1} j_2$

$\rightarrow$ $\underline{2^{k-6}}$: $\sum_{j_1=1}^{k-11} \sum_{j_2=1}^{j_1} \sum_{j_3=1}^{j_2} \sum_{j_4=1}^{j_3} \sum_{j_5=1}^{j_4} j_5 + 4 \sum_{j_1=1}^{k-8} \sum_{j_2=1}^{j_1} \sum_{j_3=1}^{j_2} j_3 + \sum_{j_1=1}^{k-5} j_1$

$\rightarrow$ $\underline{2^{k-7}}$: $\sum_{j_1=1}^{k-13} \sum_{j_2=1}^{j_1} \sum_{j_3=1}^{j_2} \sum_{j_4=1}^{j_3} \sum_{j_5=1}^{j_4} \sum_{j_6=1}^{j_5} j_6 + 5 \sum_{j_1=1}^{k-8} \sum_{j_2=1}^{j_1} \sum_{j_3=1}^{j_2} \sum_{j_4=1}^{j_3} j_4 + 3 \sum_{j_1=1}^{k-7} \sum_{j_2=1}^{j_1} j_2$

$\rightarrow \ldots$

Generally we obtain:

- 2^k
- $(k-1)2^{k-1}$
- $(\sum_{j_1=1}^{k-3} j)2^{k-2}$
- $(\sum_{j_1=1}^{k-5} \sum_{j_2=1}^{j_1} j_2 + (k-2))2^{k-3}$
- $\ldots$
- $(\sum_{j_1=1}^{k-(2x-1)} \cdots \sum_{j_{x-1}=1}^{j_{x-2}} j_{x-1} + (x-2)(\sum_{j_1=1}^{k-(2x-4)} \cdots \sum_{j_{x-3}=1}^{j_{x-4}} j_{x-3}) + \sum_{m=3}^{\lfloor (x+3)/3 \rfloor} \Big[(\sum_{j_1=1}^{x+4-3m} \cdots \sum_{j_{m-2}=1} j_{m-2}) \cdot (\sum_{j_1=1}^{k-2x+3m-2} \cdots \sum_{j_{x+1-2m}=1} j_{x+1-2m})\Big])2^{k-x}$,
  where $x = 1, 2, 3, \ldots, \begin{cases} k/2, & \text{if } k \text{ is even} \\ (k+1)/2, & \text{if } k \text{ is odd.} \end{cases}$

We have:

$$
\begin{aligned}
&(\sum_{j_1=1}^{k-(2x-1)} \cdots \sum_{j_{x-1}=1}^{j_{x-2}} j_{x-1} + (x-2) \sum_{j_1=1}^{k-(2x-4)} \cdots \sum_{j_{x-3}=1}^{j_{x-4}} j_{x-3} + \\
&\sum_{m=3}^{\lfloor (x+3)/3 \rfloor} \left[\sum_{j_1=1}^{x+4-3m} \cdots \sum_{j_{m-2}=1} j_{m-2} \cdot \sum_{j_1=1}^{k-2x+3m-2} \cdots \sum_{j_{x+1-2m}=1} j_{x+1-2m} \right]) 2^{k-x} \\
&\approx (\frac{(k-2x+1)^x}{x!} + \frac{(k-2x+4)^{x-2}}{(x-3)!} + \\
&\sum_{m=3}^{\lfloor (x+/3) \rfloor} \left[\frac{(x+4-3m)^{m-1}}{(m-1)!} \cdot \frac{(k-2x+3m-2)^{x+2-2m}}{(x+2-2m)!} \right]) 2^{k-x}
\end{aligned}
$$

Hence we obtain the following for the running time of Algorithm $MH\_F^{\triangle}$:

$$
\begin{aligned}
O((&\frac{(k-2x+1)^x}{x!} + \frac{(k-2x+4)^{x-2}}{(x-3)!} \\
&+ \sum_{m=3}^{\lfloor (x+/3) \rfloor} \left[\frac{(x+4-3m)^{m-1}}{(m-1)!} \cdot \frac{(k-2x+3m-2)^{x+2-2m}}{(x+2-2m)!} \right]) 2^{k-x})
\end{aligned}
$$

Let us consider the following function:

$$
\begin{aligned}
f_k(x) := (&\frac{(k-2x+1)^x}{x!} + \frac{(k-2x+4)^{x-2}}{(x-3)!} \\
&+ \sum_{m=3}^{\lfloor x/2 \rfloor - 1} \left[\frac{(x+4-3m)^{m-1}}{(m-1)!} \cdot \frac{(k-2x+3m-2)^{x+2-2m}}{(x+2-2m)!} \right]) 2^{k-x}
\end{aligned}
$$

For $k = 1, \ldots, 4000$ we have calculated the maximum of $f_k(x)$, for $x = 1, 2, 3, \ldots, \begin{cases} k/2, & \text{if } k \text{ is even} \\ (k+1)/2, & \text{if } k \text{ is odd} \end{cases}$ by using a computer program with an accuracy of 150 positions after decimal point. The result is that for a fixed k, the maximum of f_k is at $x = \lfloor 5k/21 \rfloor$. So for $k = 1, \ldots, 4000$ we get: $f_k(x) \leq 2.728^k$. Thus $O(2.728^k) = O(1.397^n)$ is the running time of Algorithm $MH\_F^{\triangle}$ for formulas in $MH\_F^{\triangle}$ with n variables whose clauses in the Horn part are 5-uniform and $k \leq 4000$.

$\underline{l = 6}$:
For an arbitrary k we obtain $p_k(k)$ as follows: We consider $p_{k-1}(k-1)$, the 2-polynomial for $k-1$ many triangles: Let $p_{k-1}(k-1) = S_1 + S_2 + S_3$, where S_1, S_2, S_3 are the three summands, each one also consisting of 2-powers. Then $p_{k-1}(k)$ is the 2-polynomial for $k-1$ many triangles, where we increase the exponent of each 2-power at 1. Then we get $p_k(k) = p_{k-1}(k) + S_1 + S_2$. Hence we obtain the following

for the coefficients of the 2-powers:

$$\to \underline{2^k}\colon 1$$
$$\to \underline{2^k}\colon (k-1)$$
$$\to \underline{2^{k-2}}\colon \sum_{j=1}^{k-2} j$$
$$\to \underline{2^{k-3}}\colon \sum_{j_1=1}^{k-4}\sum_{j_2=1}^{j_1} j_2 + \sum_{j_1=1}^{k-4} j_1$$
$$\to \underline{2^{k-4}}\colon \sum_{j_1=1}^{k-6}\sum_{j_2=1}^{j_1}\sum_{j_3=1}^{j_2} j_3 + \sum_{j_1=1}^{k-6}\sum_{j_2=1}^{j_1} j_2 + \sum_{j_1=1}^{k-5} j_1$$
$$\to \underline{2^{k-5}}\colon \sum_{j_1=1}^{k-8}\sum_{j_2=1}^{j_1}\sum_{j_3=1}^{j_2}\sum_{j_4=1}^{j_3} j_4 + + \sum_{j_1=1}^{k-8}\sum_{j_2=1}^{j_1}\sum_{j_3=1}^{j_2} j_3 + 2\sum_{j_1=1}^{k-7}\sum_{j_2=1}^{j_1}\sum_{j_3=1}^{j_2} j_3 + \sum_{j_1=1}^{k-7}\sum_{j_2=1}^{j_1} j_2$$
$$\to \underline{2^{k-6}}\colon \sum_{j_1=1}^{k-10}\sum_{j_2=1}^{j_1}\sum_{j_3=1}^{j_2}\sum_{j_4=1}^{j_3}\sum_{j_5=1}^{j_4} j_5 + \sum_{j_1=1}^{k-10}\sum_{j_2=1}^{j_1}\sum_{j_3=1}^{j_2}\sum_{j_4=1}^{j_3} j_4 + 3\sum_{j_1=1}^{k-9}\sum_{j_2=1}^{j_1}\sum_{j_3=1}^{j_2}\sum_{j_4=1}^{j_3} j_4 + 2\sum_{j_1=1}^{k-9}\sum_{j_2=1}^{j_1}\sum_{j_3=1}^{j_2} j_3 + \sum_{j_1=1}^{k-8}\sum_{j_2=1}^{j_1}\sum_{j_3=1}^{j_2} j_3$$
$$\to \dots$$

Generally we obtain the following for the coefficient of 2^{k-x}, for an arbitrary $x = 1, 2, 3, \dots,$ $\begin{cases} k/2, & \text{if } k \text{ is even} \\ (k+1)/2, & \text{if } k \text{ is odd} \end{cases}$

$$\sum_{j_1=1}^{k-(2x-2)} \cdots \sum_{j_{x-1}=1}^{j_{x-2}} j_{x-1} + \sum_{j_1=1}^{k-(2x-2)} \cdots \sum_{j_{x-2}=1}^{j_{x-3}} j_{x-2} + (x-3) \sum_{j_1=1}^{k-(2x-3)} \cdots \sum_{j_{x-2}=1}^{j_{x-3}} j_{x-2}$$
$$+ (x-4) \sum_{j_1=1}^{k-(2x-3)} \cdots \sum_{j_{x-3}=1}^{j_{x-4}} j_{x-3}$$
$$+ \sum_{m=5}^{x-1} \left[\sum_{j_1=1}^{x-m} \cdots \sum_{j_{\lceil (m-4)/2 \rceil}=1} j_{\lceil (m-4)/2 \rceil} \cdot \sum_{j_1=1}^{k-2x+m-\lfloor (m-2)/2 \rfloor} \cdots \sum_{j_{x-(1+\lceil (m-1)/2 \rceil)}=1} j_{x-(1+\lceil (m-1)/2 \rceil)} \right]$$
$$\approx \frac{(k-2x+2)^x}{x!} + \frac{(k-2x+2)^{x-1}}{(x-1)!} + (x-3)\frac{(k-2x+3)^{x-1}}{(x-1)!} + (x-4)\frac{(k-2x+3)^{x-2}}{(x-2)!}$$
$$+ \sum_{m=5}^{x-1} \left[\frac{(x-m)^{\lfloor (m+1)/2 \rfloor - 1}}{(\lfloor (m+1)/2 \rfloor - 1)!} \cdot \frac{(k-2x+m-\lfloor (m-2)/2 \rfloor)^{x-\lfloor m/2 \rfloor}}{(x-\lfloor m/2 \rfloor)!} \right]$$

Let us define the following function:

$$\begin{aligned} g_k(x) :&= \frac{(k-2x+2)^x}{x!} + \frac{(k-2x+2)^{x-1}}{(x-1)!} + (x-3)\frac{(k-2x+3)^{x-1}}{(x-1)!} \\ &+ (x-4)\frac{(k-2x+3)^{x-2}}{(x-2)!} \\ &+ \sum_{m=5}^{x-1}\left[\frac{(x-m)^{\lfloor (m+1)/2\rfloor -1}}{(\lfloor (m+1)/2\rfloor -1)!}\cdot\frac{(k-2x+m-\lfloor (m-2)/2\rfloor)^{x-\lfloor m/2\rfloor}}{(x-\lfloor m/2\rfloor)!}\right] \end{aligned}$$

For $k = 1, \ldots, 4000$ we have computed the maximum of $g_k(x)$, for $x = 1, 2, 3, \ldots, \begin{cases} k/2, & \text{if } k \text{ is even} \\ (k+1)/2, & \text{if } k \text{ is odd} \end{cases}$ by using a computer program which has an accuracy of 150 positions after decimal point: For $k = 1, \ldots, 4000$ we obtain: $g_k(x) \leq 2,79^k$. Hence $O(2,79^k) = O(1.408^n)$ is the running time of Algorithm $MH\_F^{\triangle}$ for formulas in $MH\_F^{\triangle}$ with n variables, where $k \leq 4000$ and whose Horn clauses are 6-uniform.

$\underline{l = 7}$:
The number of leaves in the search tree for Algorithm $MH\_F^{\triangle}$ for a formula in WHN_7^k corresponds to the following 2-polynomial:

$$p_k(k) = 2^k + (k-1)2^{k-1} + 2^{k-2} + \ldots + c_x \cdot 2^{k-x} + \ldots$$

Let $k \in \mathbb{N}$, then we obtain $p_k(k)$ as follows:

$$\begin{aligned} \underline{k=3}&: (2^k + 2^{k-1}) + 2^{k-1} + 2^{k-2} = (2^k + 2\cdot 2^{k-1} + 2^{k-2}) \\ \underline{k=4}&: (2^k + 2\cdot 2^{k-1} + 2^{k-2}) + (2^{k-1} + 2^{k-2}) + 2^{k-2} + 2^{k-4} \\ &= (2^k + 3\cdot 2^{k-1} + 3\cdot 2^{k-2} + 2^{k-4}) \\ \underline{k=5}&: (2^k + 3\cdot 2^{k-1} + 3\cdot 2^{k-2} + 2^{k-4}) + (2^{k-1} + 2\cdot 2^{k-2} + 2^{k-3}) + (2^{k-2} + 2^{k-3}) \\ &+ 2^{k-4} = (2^k + 4\cdot 2^{k-1} + 6\cdot 2^{k-2} + 2\cdot 2^{k-3} + 2\cdot 2^{k-4}) \\ \underline{k=6}&: \underbrace{(2^k + 4\cdot 2^{k-1} + 6\cdot 2^{k-2} + 2\cdot 2^{k-3} + 2\cdot 2^{k-4})}_{S_1} \\ &+ \underbrace{(2^{k-1} + 3\cdot 2^{k-2} + 3\cdot 2^{k-3} + 2^{k-5})}_{S_2} \\ &+ \underbrace{(2^{k-2} + 2\cdot 2^{k-3} + 2^{k-4})}_{S_3} + \underbrace{(2^{k-4} + 2^{k-5})}_{S_4} \\ &= (2^k + 5\cdot 2^{k-1} + 10\cdot 2^{k-2} + 7\cdot 2^{k-3} + 4\cdot 2^{k-4} + 2\cdot 2^{k-5}) \\ &\ldots \end{aligned}$$

For an arbitrary k we get $p_k(k)$ as follows: We consider $p_{k-1}(k-1)$, the 2-polynomial for $k-1$ many triangles: Let $p_{k-1}(k-1) = S_1 + S_2 + S_3 + S_4$, where $S_1, \ldots, S_4$ are the four summands consisting of 2-powers each. Then $p_{k-1}(k)$ is the 2-polynomial, for $k-1$ triangles, where we increase the exponent of each 2-power at 1. Then we gain $p_k(k) = p_{k-1}(k) + S_1 + S_2 + S_3^{-1}$. Here S_3^{-1} means that the exponents of all 2-powers occurring in S_3 are reduced at 1. So we obtain the following running time

for Algorithm $MH\_F^{\triangle}$: $O(c_x \cdot 2^{k-x})$, where $c_x \cdot 2^{k-x}$, for $x = 1, \ldots, \lfloor (k+1)/2 \rfloor$, is the biggest summand of the polynomial $p_k(k)$. By using a computer program with an accuracy of 150 positions after decimal point we calculate the maximal summand, for $k = 1, \ldots, 3000$, $c_x \cdot 2^{k-x}$ of $p_k(k)$ and obtain: $c_x \cdot 2^{k-x} \leq 2,94^k$, for $k = 1, \ldots, 3000$. Hence $O(2,94^k) = O(1.43^n)$ is the running time of Algorithm $MH\_F^{\triangle}$ for formulas in $MH\_F^{\triangle}$ with n variables, whose Horn clauses are 7-uniform, for $k \leq 4000$.
$\underline{l = 8}$:
The number of leaves in the search tree for Algorithm $MH\_F^{\triangle}$ for a formula in WHN_8^k corresponds to the following 2-polynomial:

$$p_k(k) = 2^k + (k-1)2^{k-1} + 2^{k-2} + \ldots + c_x \cdot 2^{k-x} + \ldots$$

Let $k \in \mathbb{N}$, then we obtain $p_k(k)$ as follows:

$\underline{k = 3}$: $(2^k + 2^{k-1}) + 2^{k-1} + 2^{k-2} = (2^k + 2 \cdot 2^{k-1} + 2^{k-2})$

$\underline{k = 4}$: $(2^k + 2 \cdot 2^{k-1} + 2^{k-2}) + (2^{k-1} + 2^{k-2}) + 2^{k-2} + 2^{k-3}$
$= (2^k + 3 \cdot 2^{k-1} + 3 \cdot 2^{k-2} + 2^{k-3})$

$\underline{k = 5}$: $(2^k + 3 \cdot 2^{k-1} + 3 \cdot 2^{k-2} + 2^{k-3}) + (2^{k-1} + 2 \cdot 2^{k-2} + 2^{k-3}) + (2^{k-2} + 2^{k-3})$
$+ 2^{k-3} = (2^k + 4 \cdot 2^{k-1} + 6 \cdot 2^{k-2} + 4 \cdot 2^{k-3}$

$\underline{k = 6}$: $\underbrace{(2^k + 4 \cdot 2^{k-1} + 6 \cdot 2^{k-2} + 4 \cdot 2^{k-3})}_{S_1} + \underbrace{(2^{k-1} + 3 \cdot 2^{k-2} + 3 \cdot 2^{k-3} + 2^{k-4})}_{S_2}$
$+ \underbrace{(2^{k-2} + 2 \cdot 2^{k-3} + 2^{k-4})}_{S_3} + \underbrace{(2^{k-3} + 2^{k-4})}_{S_4}$
$= (2^k + 5 \cdot 2^{k-1} + 10 \cdot 2^{k-2} + 10 \cdot 2^{k-3} + 3 \cdot 2^{k-4})$
$\ldots$

We consider $p_{k-1}(k-1)$, the 2-polynomial for $k-1$ many triangles: Let

$$p_{k-1}(k-1) = S_1 + S_2 + S_3 + S_4$$

where $S_1, \ldots, S_4$ are the four summands, consisting of 2-powers each. Then $p_{k-1}(k)$ is the 2-polynomial, for $k-1$ triangles, where the exponent of each 2-power is increased at 1. Next we get $p_k(k) = p_{k-1}(k) + S_1 + S_2 + S_3$. So we obtain the following running time for Algorithm $MH\_F^{\triangle}$: $O(c_x \cdot 2^{k-x})$, where $c_x \cdot 2^{k-x}$, for $x = 1, \ldots, \lfloor (k+1)/2 \rfloor$, is the biggest summand of polynomial $p_k(k)$. By using a computer program with an accuracy of 150 positions after decimal point we calculate the maximal summand $c_x \cdot 2^{k-x}$ of $p_k(k)$, for $k = 1, \ldots, 3000$, and obtain: $c_x \cdot 2^{k-x} \leq 2,9696^k$, for $k = 1, \ldots, 4000$. Hence $O(2,9696^k) = O(1.437^n)$ is the running time of Algorithm $MH\_F^{\triangle}$ for formulas in $MH\_F^{\triangle}$ with n variables, whose Horn clauses are 8-uniform.

□

2.4 Mixed Horn Formulas with Linear Horn Part

In the following we consider the class $LMH\underline{\ }F^{d+}$ consisting of mixed Horn formulas for which additionally holds: the Horn part consists of k-uniform, negative

monotone, linear clauses and the positive monotone part P consists of pairwise disjoint clauses in which each variable occurs only once.

Theorem 10. SAT *remains* NP*-complete for the class* $LMH^k_{-}F^{d+}$.

Proof. In [42] it is shown that SAT remains NP-complete for the class $LCNF(= k)$ of k-uniform linear formulas. Now we provide a polynomial-time reduction from $LCNF(= k)$-SAT to $LMH^k_{-}F^{d+}$-SAT proving the NP-completeness of the latter. To that end let F be an arbitrary formula of the class LCNF$(= k)$ and let $V(F) = \{x_1, x_2, \ldots, x_n\}$ be the set of all variables of F. For all variables $x_i \in V(F)$ occurring positively in F we replace each positive occurrence of x_i by $\overline{y_{x_i}}$ with $y_{x_i} \notin V(F)$ is a newly introduced variable, and add the positive monotone 2-clause $(x_i \vee y_{x_i})$ to F. Let F' be the resulting formula. Then $F' = H \wedge P$, where H consists of k-uniform, negative monotone, linear clauses and the P part consists of positive monotone pairwise disjoint 2-clauses. Now suppose F is satisfiable. Let α be a satisfying truth assignment for F. Then α' is a satisfying truth assignment for F' with α' defined as follows:
For all $i \in \{1, \ldots, n\}$: $\alpha'(x_i) = \alpha(x_i) \wedge \alpha'(y_{x_i}) = 1 - \alpha(x_i)$. Since $\alpha'(y_{x_i}) = 1 - \alpha(x_i)$ all clauses of P are obviously satisfied because exactly one variable of each clause of P is set to 1. The H-part is also satisfied, because α is a model for F.
Now assume F' is satisfiable and α' is a model for F'. Then α' satisfies all k-clauses in F, which are satisfied in F' by a $\overline{x_i}$, that is by $x_i = 0$, for $x_i \in V(F)$. Now let $c \in F$ be a k-clause in F whose corresponding clause $c' \in F'$ is satisfied by a $\overline{y_{x_j}}$ that is by setting $y_{x_j} = 0$, for $y_{x_j} \in V(F')$. Then the literal x_j is contained in c and hence $x_j = 1$ satisfies c. Thus α can be defined as α' restricted on the variables set $\{x_1, x_2, \ldots, x_n\}$. □

Next we analyse formulas of the class $LMH^k_{-}F^{d+}$. Let $F = P \wedge H \in LMH^k_{-}F^{d+}$. We assume that no clause of H contains both variables of one and the same clause of P, in other words each clause of H shares at most one variable with a clause $p \in P$, for all $p \in P$. This results from the fact that it suffices to set exactly one variable of each $p \in P$ to 1, hence we can set the other variable to 0 and this way satisfy all clauses in H which contain both variables of the same clause $p \in P$. The special structure of the formula class $LMH^k_{-}F^{d+}$ yields the following lemma.

Lemma 1. *Let* $F \in LMH^k_{-}F^{d+}$ *with* n *variables and* $k \geq n/2$. *Then* F *is satisfiable.*

Proof. Let $F = P \wedge H \in LMH^k_{-}F^{d+}$ and let n be the number of variables of F. As the Horn part H is linear and k-uniform, the following inequality holds for H according to [42]: $|H| \leq \frac{n(n-1)}{k(k-1)}$.
Since $k \geq n/2$, we obtain $|H| \leq \frac{4(n-1)}{n-2}$ and $\frac{4(n-1)}{n-2} \leq n/2$, for $n \geq 10$. So for $n \geq 10$ we obtain $|H| \leq n/2 = |P|$.

Next we consider two arbitrary clauses c_i, c_j of the Horn part. Let $|V(c_i) \cup V(c_j)|$ be the number of all different variables contained in the clauses c_i and c_j. Because of linearity of H we get:

$$|V(c_i) \cup V(c_j)| \geq k + (k-1) \geq \frac{n}{2} + (\frac{n}{2} - 1) = n - 1$$

So each pair of clauses in H contains at least $n - 1$ different variables. Now we consider the following bipartite incidence graph G_F for F: Let $V(G_F) = V_1 \cup V_2$ be

the vertex set partition of G_F, where V_1 consists of all clauses of H and V_2 consists of all clauses of P. There is an edge between a vertex in V_1 and a vertex in V_2 if the corresponding clause of H and the corresponding clause of P have a variable in common. Since $|H| \leq n/2 = |P|$ and for each $l \in \{1, \ldots, n/2\}$ each l vertices in V_1 have at least l neighbours in V_2 (because V_2 contains exactly $n/2$ many vertices and every two vertices of V_1 already have $n/2$ many neighbours), we can apply the classic Theorem of König-Hall [30, 23] for bipartite graphs stating that there exists a matching in G_F covering the component V_1 of the vertex set. In terms of the formula it means that we can uniquely assign a positive monotone 2-clause c_P of P to each Horn clause c_H of H so that the one variable which c_H and c_P have in common can be set to 0 to satisfy c_H. We set the other variable of c_P to 1 to satisfy c_P so that F is satisfied. □

Lemma 2. *Let $F \in LMH^k_- F^{d+}$, n the number of variables of F and $k = n/2 - j$, for $j \in \{1, \ldots, n/2 - 3\}$.*
If $n \geq \max\{2j + 5 + \sqrt{16j + 17}, 2j + 3 + \sqrt{2j^2 + 16j + 9}\}$, then F is satisfiable.

Proof. Let $F \in LMH^k_- F^{d+}$, let n be the number of variables of F and let $k = n/2 - j$, for $j \in \{1, \ldots, n/2 - 3\}$. As the Horn part H is linear and k-uniform, the following inequality holds according to [42]: $|H| \leq \frac{n(n-1)}{k(k-1)}$.
Since $k = n/2 - j$, we obtain

$$|H| \leq \frac{n(n-1)}{(n/2-j)(n/2-j-1)}$$

With the help of a computer program we calculate that for $n \geq 2j + 5 + \sqrt{16j + 17}$ we obtain $|H| \leq \frac{n}{2}$. We prove that every k many arbitrary clauses of H contain at least $n - 1$ many different variables:
Because of linearity of H the minimal number of variables contained in k different clauses of H is:

$$\sum_{l=0}^{k-1}(k-l) = \sum_{l=0}^{k-1}(n/2-j-l) = k(n/2-j) - \sum_{l=0}^{k-1} l = k(n/2-j) - k(k-1)/2$$

Since $k = n/2 - j$, we obtain:

$$k(n/2-j) - k(k-1)/2 \geq n \Leftrightarrow (n/2-j)^2 - (n/2-j)(n/4-j/2-1/2) \geq n$$

Again by using a computer program we calculate that for $n \geq 2j+3+\sqrt{2j^2 + 16j + 9}$ we obtain $(n/2-j)^2 - (n/2-j)(n/4-j/2-1/2) \geq n$.

Now define $x(j) := \max\{2j + 3 + \sqrt{2j^2 + 16j + 9}, 2j + 5 + \sqrt{16j + 17}\}$. For $n \geq x(j)$ each k many different clauses of H contain n different variables. So when considering the bipartite incidence graph G_F for F as defined in the proof of Lemma 1 (where the clauses of H are the vertices of G_F on the one side of the vertex set bipartition and the clauses of P are the vertices on the other side of the vertex set bipartition of G_F) each k different clauses of H have $n/2$ many neighbours with the clauses in P. It remains to prove that each l many clauses of H, for $l < k$, contain at least $2l$ many variables and thus have at least l many neighbours in G_F.

Because of linearity, each l many clauses of H, for $l < k$, obviously contain at least $\sum_{i=0}^{l-1}(n/2-j-i)$ many variables. We obtain

$$\sum_{i=0}^{l-1}(n/2-j-i) = l(n/2-j) - l(l-1)/2$$

and further

$$l(n/2-j) - l(l-1)/2 \geq 2l \Leftrightarrow n-2j-l \geq 3$$

As $l < k = n/2-j$ we get $n-2j-l > n-2j-(n/2-j) = n/2-j$. Thus $n-2j-l > n/2-j$ and $n/2-j \geq 3 \Leftrightarrow n \geq 2j+6$. Obviously $(2j+6) < x(j) \leq n$ and hence we obtain that each l many different clauses of H (for $l < k$) have at least l many neighbours in G_F, in case $n \geq x(j)$. So again we can apply the classic Theorem of König-Hall for bipartite graphs stating that there exists a matching in G_F covering the component H of the vertex set [30, 23]. In terms of the formula, this means we can uniquely assign a positive monotone 2-clause c_P of P to each Horn clause c_H of H such that the variable which c_H and c_P have in common can be set to 0 satisfying c_H and the other variable of c_P must be set to 1 to satisfy c_P. This way we can satisfy a formula F in $LMH^k_-F^{d+}$with $n \geq x(j)$ many variables and $k = n/2-j$. □

Now we consider the class $LMH^-_kF^{\triangle}$ of mixed Horn formulas. For its positive part P holds that the corresponding graph G_P consists of disjoint triangles only and its Horn part H is k-uniform, negative monotone and linear. Further we demand that $V(P) = V(H)$. The following Theorem states the NP-completeness of this class.

Theorem 11. SAT *remains* NP*-complete for the class* $LMH^-_kF^{\triangle}$

Proof. We provide a polynomial-time reduction from $LMH^-_kF^{d+}$-SAT which is NP-complete according to Theorem 10 to $LMH^-_kF^{\triangle}$-SAT, proving NP-completeness of the latter. To that end let $F = P \wedge H \in LMH^-_kF^{d+}$ be an arbitrary formula and let $k \geq 4$. For each clause $(x_i \vee x_j)$, $x_i, x_j \in V(F)$, of P we introduce a new variable $y_{i,j}$ not yet occurring in $V(F)$ and add the two clauses $(x_i \vee y_{i,j})$, $(x_j \vee y_{i,j})$ to P. This way we obtain the positive monotone 2-CNF formula P' whose corresponding graph $G_{P'}$ consists of disjoint triangles only. For each newly introduced variable $y_{i,j}$ we add the following three negative monotone, linear k-clauses (for $k \geq 4$) to H:

$$(\overline{y_{i,j}} \vee \overline{z^1_{i,j,1}} \vee \ldots \vee \overline{z^{k-1}_{i,j,1}}), (\overline{y_{i,j}} \vee \overline{z^1_{i,j,2}} \vee \ldots \vee \overline{z^{k-1}_{i,j,2}}), (\overline{y_{i,j}} \vee \overline{z^1_{i,j,3}} \vee \ldots \vee \overline{z^{k-1}_{i,j,3}})$$

where the variables $z^p_{i,j,l}$, for $p \in \{1,\ldots,k-1\}, l \in \{1,2,3\}$ are newly introduced and pairwise different. Let H' be the resulting Horn part. Then, to achieve the condition $V(P') = V(H')$, we add the clauses $(z^p_{i,j,1} \vee z^p_{i,j,2})$, $(z^p_{i,j,1} \vee z^p_{i,j,3})$, $(z^p_{i,j,2} \vee z^p_{i,j,3})$, for each $p \in \{1,\ldots,k-1\}$, to P' obtaining P'', whose corresponding graph $G_{P''}$ obviously still consists of disjoint triangles only.
In case $k = 3$ we perform the following steps: For each clause $(x_i \vee x_j)$, $x_i, x_j \in V(F)$, of the P-part we introduce a new variable $y_{i,j}$ not yet occurring in $V(F)$ and add the two clauses $(x_i, y_{i,j})$, $(x_j, y_{i,j})$ to P. This way we obtain the positive monotone 2-CNF formula P' whose corresponding graph $G_{P'}$ consists of disjoint

triangles only. For each newly introduced variable $y_{i,j}$ we add the following negative monotone, linear 3-clauses to H:

$$(\overline{y_{i,j}} \vee \overline{z_{i,j,1}} \vee \overline{z_{i,j,2}}) \wedge (\overline{y_{i,j}} \vee \overline{z_{i,j,3}} \vee \overline{z_{i,j,4}}) \wedge (\overline{z_{i,j,4}} \vee \overline{z_{i,j,5}} \vee \overline{z_{i,j,6}})$$

Here the variables $z_{i,j,l}$, for $l \in \{1,2,3,4,5,6\}$ are newly introduced and pairwise different. Let H' be the resulting Horn part, then H' is 3-uniform negative monotone and linear. Moreover we add to P' the following two triangles: $(z_{i,j,1} \vee z_{i,j,2})$,$(z_{i,j,1} \vee z_{i,j,3})$,$(z_{i,j,2} \vee z_{i,j,3})$ and $(z_{i,j,4} \vee z_{i,j,5})$,$(z_{i,j,5} \vee z_{i,j,6})$,$(z_{i,j,4} \vee z_{i,j,6})$ and obtain P''.
Now let F be satisfiable and α a model for F. We set all variables which F and $F' = P'' \wedge H'$ have in common according to α. In each new triangle $(x_i \vee x_j) \wedge (x_i, y_{i,j}) \wedge (x_j, y_{i,j})$ of P' we set $y_{i,j} = 1$ and hence satisfy P'. In case $k = 3$, we set $z_{i,j,1} = z_{i,j,4} = 0$ and $z_{i,j,2} = z_{i,j,3} = z_{i,j,5} = z_{i,j,6} = 1$ and satisfy all newly introduced Horn clauses and triangles. In case $k \geq 4$, we set $z^1_{i,j,1} = 0$, $z^2_{i,j,2} = 0$, $z^3_{i,j,3} = 0$ and hence satisfy the newly added clauses in H'. Then we assign 1 to all the other variables and in this way also satisfy the newly introduced triangles in P''. Let F' be satisfiable. As $F \subset F'$ and F' is satisfiable consequently F must be satisfiable, too. □

The computational complexity of mixed Horn formulas with a negative monotone and exact linear Horn part also turns out to be a compelling question. Let XLMH^-F denote the class of mixed Horn formulas $F = P \wedge H$ where the Horn part is negative monotone and exact linear and P consists of positive monotone 2-clauses only. We obtain the following result for the complexity of this class.

Theorem 12. SAT *remains* NP*-complete for the class* XLMH^-F.

Proof. We provide a polynomial-time reduction from $LMH^-_kF^{d+}$-SAT to XLMH^-F-SAT. Let $F = P \wedge H \in LMH^-_kF^{d+}$ be an arbitrary formula with variable set $V(F) = \{y_1, \ldots, y_n\}$. If the Horn part H is already exact linear then $F \in XLMH^-F$ and we are finished. Otherwise as long as there are two clauses c_i, c_j which do not have a variable in common introduce a new variable $x_{i,j}$ not yet occurring in $V(F)$ and enlarge c_i and c_j by $\overline{x_{i,j}}$. Let V' be the set of all these newly introduced variables. After having enlarged each pair of disjoint clauses by a newly introduced variable we obtain an exact linear and negative monotone Horn part H'. To provide SAT-equivalence to F we add to P, for each newly introduced variable $x_{i,j}$, the following 2-clauses: $(y_1 \vee x_{i,j}) \wedge (y_2 \vee x_{i,j}) \wedge \ldots \wedge (y_n \vee x_{i,j})$ and also $(x_{i,j} \vee x_{k,l})$, for each two newly introduced variables $x_{k,l}, x_{i,j} \in V'$. Let P' be the resulting positive monotone part and let $F' = P' \wedge H'$. F is satisfiable if, and only if, F' is:
Let F be satisfiable with model α and set all the variables which F and F' have in common according to α in F'. We set all the other variables of F' to 1 and in this way satisfy F'. On the other hand suppose F is not satisfiable. To satisfy H' in F' at least one of the newly introduced variables must be set to 0 in F'. But then P' implies that all the other variables of F' must be set to 1 which would not satisfy further clauses of H' (assuming that H' has more than one clause). Hence all the newly introduced variables in F' must be set to 1, but then F' reduces to F and cannot be satisfied either. □

The so far best algorithm solving SAT for an arbitrary mixed Horn formula $F = P \wedge H$ with n variables is the algorithm MHFSAT by Porschen and Speckenmeyer [43] with a running time of $O(2^{0.5284n})$. Algorithm MHFSAT first calculates all minimal vertex covers of P with P consisting of positive monotone 2-clauses only and then checks for each minimal vertex cover whether the partial truth assignment resulting by assigning 1 to all variables of a vertex cover can be extended to a model for H. In case G_P consists of disjoint edges only algorithm MHFSAT obviously needs $O(2^{n/2})$ running time. Below we present algorithm COUNT for formulas in $LMH^k_-F^{d+}$ which in contrast to algorithm MHFSAT does not work by calculating all vertex covers of P but by excluding minimal vertex covers of P for each $c \in H$ and counting the number of all these excluded vertex covers for all $c \in H$. If this number is $< 2^{n/2}$ F is satisfiable, else F is unsatisfiable. The idea here is the following: P has at most $2^{n/2}$ different minimal vertex covers. For each clause c of H we determine all minimal vertex covers of P which are excluded by c because setting all the variables of this minimal vertex cover to 1 would not satisfy c since then all literals in c would be set to 0. Note that each clause of H excludes exactly $2^{n/2-k}$ minimal vertex covers of P but it is possible that there are two clauses $c_i, c_j \in H$ such that c_i as well as c_j exclude one and the same minimal vertex cover m_c. So when adding the numbers of excluded minimal vertex covers by c_i and c_j we have to keep in mind that we add m_c only once instead of twice. It turns out to be quite difficult to analytically count the number M of all excluded minimal vertex covers by clauses of H. After having determined all excluded minimal vertex covers for P, we count their number M. If $M < 2^{n/2}$ then F has exactly $2^{n/2} - M$ models: We set all the variables of a minimal vertex cover of P, which is not excluded, to 1 and the remaining variables to 0 obtaining a model for F. In case $M = 2^{n/2}$ the formula is not satisfiable because we cannot satisfy P as no minimal vertex cover of P yields a model for F.

There are some experimental results which lead to the strong conjecture that algorithm COUNT works faster than algorithm MHFSAT for formulas in $LMH^k_-F^{d+}$, but unfortunately we are not able to prove this analytically. So the analysis of running time remains open for future work.

Algorithm **COUNT**

INPUT: $F = P \wedge H \in LMH^k_-F^{d+}$ with $V(P) = V(H)$ and there is no clause $c \in H$ for which there is a clause $p \in P$ such that $V(p) \subset V(c)$, that is no clause of H contains both variables of one and the same clause in P . Let n be the number of variables in F and let $n/2$ be the number of clauses of P .
OUTPUT: **satisfiable**, if F is satisfiable. Else **unsatisfiable**
begin

1. For each clause $c \in H$ create a set M_c containing all minimal vertex covers which are excluded by c:
   Let $c = (\overline{x_1} \vee \overline{x_2} \vee \ldots \vee \overline{x_k})$, then each minimal vertex cover m_c of M_c contains the variables $x_1, x_2, \ldots, x_k$. Additionally each m_c contains $\frac{n}{2} - k$ further variables from the remaining clauses of P which do not contain one of the variables $x_1, x_2, \ldots, x_k$. So each m_c contains from each clause of P exactly one variable. Hence $|M_c| = 2^{n/2-k}$ for each $c \in H$.
2. Let $H = c_1 \wedge c_2 \wedge \ldots \wedge c_{|H|}$ and let $M := \emptyset$.

3. **For** $i = 1$ to $|H|$ **do**
   (a) $M := M \cup M_{c_i}$;
   (b) Check whether M contains one and the same element more than once and remove all elements from M, which occur more than once in M until each element occurs exactly once in M.
   (c) If $|M| = 2^{n/2}$ then **return unsatisfiable**.
4. If $|M| < 2^{n/2}$ then **return satisfiable**.

end

2.5 Subclasses of MHF in P

This section is devoted to some interesting subclasses of MHF for which we can solve SAT in polynomial-time. The first class to be considered consists of mixed Horn formulas $F = P \wedge H$ with P being composed of disjoint, positive monotone 2-clauses only and H is negative monotone, k-uniform, for $k \geq 3$, linear and has overlappings in boundary variables only. Let $k\text{-}BLMHF^{d+}$ denote such a class of formulas. We show that we can solve SAT for formulas in $k\text{-}BLMHF^{d+}$ in polynomial-time as the following theorem states.

Theorem 13. *Formulas of the class $k\text{-}BLMHF^{d+}$, for $k \geq 3$, are always satisfiable.*

Proof. We assume that $V(P) = V(H)$, otherwise we set variables in $V(P) - V(H)$ to 1 and variables in $V(H) - V(P)$ to 0. The assertion of this Theorem directly follows from the classic Theorem of König-Hall for bipartite graphs [30, 23]: Let $F \in k\text{-}BLMHF^{d+}$ be an arbitrary formula. Then we consider the bipartite incidence graph G_F of F with vertex set partition $V_1 \cup V_2$ where the clauses of H are the vertices of V_1 and the disjoint 2-clauses of P are the vertices of V_2. There is an edge between a vertex in V_1 and a vertex in V_2 if the corresponding clauses $c_i \in H$ and $p_j \in P$ have at least one variable in common. Let n be the number of variables in F then $|V_2| = n/2$ and $|V_1| \leq \frac{n-1}{k-1} \leq \frac{n}{2}$. Moreover, we obtain that each i clauses of H, for all $i \in \{2, \ldots, |H|\}$, contain at least $i \cdot (k-1) + 1$ different variables. As $k \geq 3$, this yields $i \cdot (k-1) + 1 \geq 2 \cdot i$. Therefore in G_F all i vertices of V_1 have at least i neighbours in V_2, for all $i \in \{2, \ldots, |H|\}$, because all i clauses of H contain at least $2 \cdot i$ many different variables and all i 2-clauses in P contain exactly $2 \cdot i$ different variables. Now we can apply the classic Theorem of König-Hall for bipartite graphs stating that there exists a matching in G_F covering the component V_1 of the vertex set [30, 23]. In terms of the formula this means we can uniquely assign exactly one 2-clause $p_c \in P$ to each clause c of H. Setting the one variable which c and p_c share to 0 and the other variable of p_c to 1, for all $c \in H$, yields a model for F. □

In the last section we have shown with Theorem 12 that SAT remains NP-complete for formulas in $XLMH\_F$. But does NP-completeness still hold for its subclass, where the positive part P is supposed to consist of disjoint 2-clauses only? To answer this question we consider the class $XLMH\_F^{d+} \subseteq XLMH\_F$

of formulas of the class $XLMH\_F$ for which additionally holds that the 2-clauses of the positive monotone part P are pairwise disjoint. We obtain:

Theorem 14. *Formulas of the class $XLMH\_F^{d+}$ are always satisfiable.*

Proof. Let $F = P \wedge H$ be an arbitrary formula in $XLMH\_F^{d+}$. If a positive monotone 2-clause $p_i \in P$ exists for each clause $c_i \in H$ such that $V(p_i) \subset V(c_i)$, we set all variables of an arbitrary minimal vertex cover V_i of G_P, the variables graph for P, to 1 and all the other variables to 0 obtaining a model for F. This results from the fact that then in each clause $p_i \in P$ exactly one variable is set to 1 and the other one is set to 0. In consequence each clause of H contains at least one variable which is set to 0 because each clause of H contains both variables of a clause of P. Now suppose H contains a clause c and for c no clause exists in P whose variables both belong to c. Then we set all the variables of c to 0 and the remaining variables of F to 1. As H is exact linear, every other clause of H shares exactly one variable with c and is thus also satisfied because of negative monotonicity. As c does not contain both variables of any clause in P, in every clause of P at most one variable is now set to 0 and at least one variable is set to 1, hence P is also satisfied. □

After considering the class $XLMH\_F^{d+}$, it is now quite compelling to investigate whether the class of mixed Horn formulas $XLMH\_F^{\triangle}$ with a negative monotone, exact linear and k-uniform Horn part, whose P part consists of disjont triangles only is also polynomial-time solvable. So in the following we investigate this class of formulas.

Theorem 15. *Formulas in $XLMH\_F^{\triangle}$ are always satisfiable.*

Proof. Let $F = P \wedge H$ be an arbitrary formula in $XLMH\_F^{\triangle}$.

1. If there is a clause $c \in H$ which contains at most one variable of each triangle $\triangle_j \in P$, we set all the variables of c to 0 and all the remaining variables to 1 and this way satisfy F: As H is exact linear each clause of H has exactly one variable with c in common and is thus satisfied. As in each triangle of P at most one variable is set to 0 and the others (at least two) are set to 1, each triangle in P is also satisfied.

2. If there is no clause in H containing at most one variable of each triangle $\triangle_j \in P$, then for each clause c_i of H at least one triangle $\triangle_j$ must exist with c_i containing at least two variables of $\triangle_j$. Note that if there is a clause $c \in H$ which contains all three variables of the same triangle in P. Then this clause is always satisfiable as in each triangle exactly one variable has to be set to 0 and we can remove such a clause from H. In the following we assign all the clauses of H which contain exactly two variables of triangle $\triangle_j$ to triangle $\triangle_j$ of P. Note that because of exact satisfiability of H at most three clauses can be assigned to each triangle in P. Hence $|H| \leq n$.

   (a) If at most two clauses of H are assigned to each triangle of P then we proceed as follows: In case no clause of H is assigned to a triangle in P, we set one arbitrary variable of this triangle to 0 and the two other variables to 1. In case exactly one clause $c_x = (\overline{x_1} \vee \overline{x_2} \vee \ldots)$ is assigned to a triangle $\triangle_x = (x_1 \vee x_2) \wedge (x_2 \vee x_3) \wedge (x_1 \vee x_3)$ in P, we set w.l.o.g. $x_1 = 0$ and $x_2 = x_3 = 1$ and this way satisfy the clause c_x and $\triangle_x$. In

case exactly two clauses $c_x^1 = (\overline{x_1} \vee \overline{x_2} \vee \ldots)$ and $c_x^2 = (\overline{x_1} \vee \overline{x_3} \vee \ldots)$ are assigned to a triangle $\triangle_x = (x_1 \vee x_2) \wedge (x_2 \vee x_3) \wedge (x_1 \vee x_3)$ in P, we set the one variable which the two clauses of H and $\triangle_x$ have in common to 0 and the other two variables of $\triangle_x$ to 1: $x_1 = 0$, $x_2 = x_3 = 1$. This way we obviously satisfy c_x^1, c_x^2 and $\triangle_x$.

(b) The difficult case is if there is a triangle $\triangle_x = (x_1 \vee x_2) \wedge (x_2 \vee x_3) \wedge (x_1 \vee x_3)$ in P to which three clauses of H are assigned, namely $c_x^1 = (\overline{x_1} \vee \overline{x_2} \vee \ldots)$, $c_x^2 = (\overline{x_2} \vee \overline{x_3} \vee \ldots)$ and $c_x^3 = (\overline{x_1} \vee \overline{x_3} \vee \ldots)$. As long as there is such a triangle to which three clauses are assigned, we proceed as follows:

- If one of the clauses c_x^i, for $i \in \{1, 2, 3\}$, contains w.l.o.g. a variable a_1 belonging to the triangle $\triangle_a = (a_1 \vee a_2) \wedge (a_2 \vee a_3) \wedge (a_1 \vee a_3)$ for which holds that there is no clause of the form $(\overline{a_2} \vee \overline{a_3} \vee \ldots)$ in H, then (let w.l.o.g. $\overline{a_1}$ be contained in c_x^1) we can set $a_1 = 0$, $a_2 = a_3 = 1$ and $x_3 = 0$, $x_1 = x_2 = 1$ and this way satisfy the clauses c_x^i, for $i \in \{1, 2, 3\}$.
  In the following we call the clause $(\overline{a_2} \vee \overline{a_3} \vee \ldots)$ a *counter clause* for a_1, if a_1, a_2, a_3 are variables of the same triangle.
- If for each variable a_1 occurring in one of the clauses c_x^i, for $i \in \{1, 2, 3\}$, there is a counter clause $(\overline{a_2} \vee \overline{a_3} \vee \ldots)$ in H, then we check whether one of these counter clauses $(\overline{a_2} \vee \overline{a_3} \vee \ldots)$ contains a variable $b_1 \in V(H)$ for which there is no counter-clause $(\overline{b_2} \vee \overline{b_3} \vee \ldots)$ in H. In this case we set $b_1 = 0$ and hence satisfy $(\overline{a_2} \vee \overline{a_3} \vee \ldots)$ and thus we can now set $a_1 = 0$ and satisfy c_x^i, for a $i \in \{1, 2, 3\}$. But if there is also a counter clause $(\overline{b_2} \vee \overline{b_3} \vee \ldots)$ for each variable b_1 occurring in the counter clauses $(\overline{a_2} \vee \overline{a_3} \vee \ldots)$, then we check again for each variable c_1 occurring in the counter clauses $(\overline{b_2} \vee \overline{b_3} \vee \ldots)$ whether c_1 has a counter-clause $(\overline{c_2} \vee \overline{c_3} \vee \ldots)$ in H. If so, we repeat this procedure until we have found a variable without a counter clause. As $|H| \leq n$, the number of Horn clauses is bounded and hence we can always find a variable without a counter clause which we set to 0. Obviously this procedure needs polynomial-time.

So formulas of $XLMH\_F^{\triangle}$ are also always satisfiable. □

Next we consider the class $MH_{-d}F^{\triangle} \subset MHF$ of mixed Horn formulas $M = P \wedge H$ whose Horn part H is negative monotone and consists of disjoint clauses only and for whose positive 2-CNF part P holds that the corresponding graph G_P consists of disjoint triangles only. Further we demand that $V(P) = V(H)$. For a better study we dissect the class $MH_{-d}F^{\triangle} \subset MHF$ into the three following subclasses:

1. The Horn part H consists of negative monotone clauses of length ≥ 3, which are pairwise disjoint. Let $MH_{-d}F^{\triangle}_{\geq 3}$ denote this subclass of $MH_{-d}F^{\triangle}$.

   Theorem 16. *Formulas of the class $MH_{-d}F^{\triangle}_{\geq 3}$ are always satisfiable.*

   Proof. The assertion of this Theorem directly follows from the classic Theorem of König-Hall for bipartite graphs [30, 23]: Let $F \in MH_{-d}F^{\triangle}_{\geq 3}$ be an arbitrary formula. We consider the bipartite incidence graph G_F of F with

vertex set partition $V_1 \cup V_2$ where the clauses of H are the vertices of V_1 and the triangles of G_P are the vertices of V_2. There is an edge between a vertex in V_1 and a vertex in V_2, if the corresponding clause $c_i \in H$ and $\triangle_j \in P$ have a variable in common. Let n be the number of variables in F then $|V_2| = n/3$ and $|V_1| \leq n/3$. As the clauses of H are pairwise disjoint, we obtain that each i vertices of V_1 have at least i neighbours in V_2, for all $i \in \{1, \ldots, n/3\}$ because each i clauses of H contain at least $3 \cdot i$ different variables and every i triangles in P contain exactly $3 \cdot i$ different variables. Now we can apply the classic Theorem of König-Hall for bipartite graphs stating that there exists a matching in G_F covering the component V_1 of the vertex set [30, 23]. In terms of the formula that means we can uniquely assign exactly one triangle $\triangle_c$ to each clause c of H, setting the one variable which c and $\triangle_c$ share to 0 and the other two variables of $\triangle_c$ to 1. This way we obtain a model for F. □

2. The Horn part H consists of pairwise disjoint clauses of length < 3, let $MH_{-d}F^{\triangle}_{<3}$ denote the class of such formulas. We additionally demand that $V(P) = V(H)$.

   Theorem 17. *Formulas of the class $MH_{-d}F^{\triangle}_{<3}$ are not satisfiable.*

   Proof. Let $F = (P \wedge H) \in MH_{-d}F^{\triangle}_{<3}$ be an arbitrary formula, where $H = c_1 \wedge c_2 \wedge ... \wedge c_{|H|}$ and let m be the number of disjoint triangles in G_P. Let n be the number of variables in F. Then $3m = n \leq 2|H|$ leading to $m \leq \frac{2}{3}|H|$. To satisfy H we have to set at least one variable in each clause of H to 0, that means we have to assign 0 to $|H|$ many variables. But as we always have more clauses in H than triangles in P the pigeonhole principle implies that in order to satisfy H we have to set at least two variables to 0 in at least one triangle of P. This would violate the satisfiablility of P. Consequently F is not satisfiable. □

3. The Horn part H consists of monotone negative clauses and H has at least one clause of length < 3 and at least one of length ≥ 3. Further we demand that $V(P) = V(H)$. Let $MH_{-d}F^{\triangle}_{mxd}$ denote the class of such formulas.

   Theorem 18. SAT *can be solved in polynomial-time for formulas in $MH_{-d}F^{\triangle}_{mxd}$.*

   Proof. The following algorithm MXD solves SAT for the class $MH_{-d}F^{\triangle}_{mxd}$ in polynomial-time.
   Let $label(x_j)$ denote a function which assigns the same label to all three variables of a triangle of G_P. Variables of different triangles are distinctively labelled.

   Algorithm MXD

   INPUT: $F \in MH_{-d}F^{\triangle}_{mxd}$.
   OUTPUT: SATISFIABLE, if F is satisfiable, else UNSATISFIABLE.
   begin

(a) Assign to each variable $x \in V(F)$ a label, such that variables of the same triangle get the same label and variables of different triangles get different labels. Let $label(x)$ denote the label of variable $x \in V(F)$.

(b) We arrange the clauses of H in ascending order according to the number of their literals: $H = c_1 \wedge c_2 \wedge ... \wedge c_{|H|}$, where $|c_1| \leq \ldots \leq |c_{|H|}|$. Clauses of the same size and with the same labels are neighbours in this order. Let $c_i = (\overline{x_{i_1}} \vee \ldots \vee \overline{x_{i_m}})$ and let k be the number of triangles in G_P.

(c) We set $Z := \emptyset$.

(d) If $k < |H|$ then output UNSATISFIABLE.

(e) **For** $i := 1$ **to** $|H|$ **do** (*we consider the clause $c_i = (\overline{x_{i_1}} \vee \ldots \vee \overline{x_{i_m}})$*)

i. **For** $j := i_1$ **to** i_m **do**

- **If** $label(x_j) \notin Z$ **then** $Z := Z \cup \{label(x_j)\}$;

ii. **If** $|Z| < i$ **then** UNSATISFIABLE; **else** $i := i + 1$;

(f) F is SATISFIABLE

end

We need $O(||F||)$ running time to label all variables of F. The two **For**-loops have a running time of $O(|H||c_{|H|}|)$, where $c_{|H|}$ is the longest clause of H. So $O(||F||)$ is the running time of algorithm MXD.
To prove the correctness of algorithm MXD, the pigeonhole principle constitutes that in case F has more clauses in H than triangles in P, we cannot satisfy F because the clauses of H are pairwise disjoint. Next, if the first i clauses, for each $i \in \{1, \ldots, |H|\}$, contain less than i many different labels (which is equivalent to: the first i clauses of H have with less than i triangles in G_P variables in common), then we cannot satisfy F either because otherwise at least two variables of one and the same triangle in P would have to be set to 0. Else F is satisfiable which is a consequence from the König-Hall Theorem [30, 23]. □

Remark 2. *Note that algorithm MXD also works correctly when G_P consists of disjoint edges instead of triangles or of both disjoint edges and triangles.*

Corollary 1. SAT *can be solved in polynomial-time for the class $MH_{-d}F^{\triangle}$ of formulas, whose Horn part H consists of pairwise disjoint and negative monotone clauses and for whose positive monotone part P holds that the corresponding graph G_P consists of disjoint triangles only.*

Now we consider a generalisation of the class $MH_{-d}F^{\triangle}$, namely the class $MH_dF^{\triangle}$ of mixed Horn formulas, whose Horn part H consists of pairwise disjoint Horn clauses. These do not necessarily have to be negative monotone and for their positive monotone part P also holds that the corresponding graph G_P consists of disjoint triangles only. We examine whether SAT is also solvable in polynomial-time for this class. The next Theorem and its proof shed light on this question.

Theorem 19. SAT *can be solved in polynomial-time for formulas in $MH_dF^{\triangle}$.*

PROOF. Let $F = P \wedge H$ be an arbitrary formula in $MH_dF^{\triangle}$. Assuming H is negative monotone we can apply algorithm MXD to F solving the SAT search problem in polynomial-time for F. Else we set each variable x occurring positively in H to 1 and evaluate the formula. In this way we obtain a negative monotone Horn part H' consisting of pairwise disjoint clauses and a positive part P' such that $G_{P'}$ consists of disjoint triangles and edges only. Note that by setting the variables occurring positively in H to 1 we do not make any restrictions concerning the satisfiabilty of F.
Finally we apply the algorithm MXD on $F' := P' \wedge H'$ solving SAT for F' in polynomial-time according to Theorem 2. □

Next we treat the class $MH_{-d}F^{d+}$ of mixed Horn formulas $F = P \wedge H$, where G_P consists of disjoint edges and the negative monotone Horn part H of pairwise disjoint clauses only. Further we demand $V(P) = V(H)$, else we set all variables in $V(P)-V(H)$ to 1 an all variables in $V(H)-V(P)$ to 0 not affecting the satisfiability.

Theorem 20. SAT *can be solved in polynomial-time for the class* $MH_{-d}F^{d+}$.

PROOF. Let $F = (P \wedge H) \in MH_{-d}F^{d+}$ be an arbitrary formula. At first we assign all unit clause variables and evaluate the formula such that all clauses of F have at least length 2. If we obtain a contradiction such as an empty clause, then F is unsatisfiable. As for all $x \in V(F)$ holds x occurs exactly once in P and exactly once in H, we can otherwise conclude $w(x) = 2$ for all $x \in V(F)$. Thus we can apply Lemma 6 from [42] stating that a formula in which each clause has length at least k and each variable occurs at most j times is satisfiable if $k \geq j$. □

So far we have considered mixed Horn formulas with negative monotone and pairwise disjoint Horn clauses. Now we investigate whether SAT is also polynomial-time solvable for mixed Horn formulas whose Horn clauses are not pairwise disjoint.

Theorem 21. *Let* $F = P \wedge H \in MH\_F^{\triangle}$ *and let* k *be the number of triangles in* G_P*. If each clause in* H *has length exactly* l*,* $l \geq 2$*, and* H *contains more than* $\binom{3k}{l} - \binom{2k}{l}$ *many clauses,* $F = H \wedge P$ *is unsatisfiable.*

PROOF. Let $F = P \wedge H \in MH\_F^{\triangle}$ where H is l-uniform and let k be the number of triangles in G_P. Since there are k triangles in G_P, F has $3k$ variables. The Horn part H is negative monotone and only has clauses of length l, thus H contains at most $\binom{3k}{l}$ many different l-clauses.
In each triangle of G_P at most one variable can be set to 0 and the other variables - at least two - of this triangle must be set to 1. To satisfy F in each clause of H at least one variable must be set to 0. Setting at most one variable in each of the k triangles to 0 yields at least $2k$ many variables which have to be set to 1. Thus for each possible truth assignment which sets at most one variable of each triangle to 0 and at least two variables to 1 we obtain at least $\binom{2k}{l}$ many unsatisfiable l-clauses among the $\binom{3k}{l}$ many possible l-clauses. That means that each formula $F = P \wedge H \in MH\_F^{\triangle}$ with a l-uniform Horn part for which holds $|H| \geq \binom{3k}{l} - \binom{2k}{l}$ is unsatisfiable. □

Lemma 3. *Let* $F = P \wedge H \in MH\_F^{\triangle}$*,* k *the number of triangles in* G_P *and* $|c_i| \geq 2k + 1$*, for all* $c_i \in H$ *and* $i = 1, \ldots, |H|$*. Then* F *is satisfiable.*

PROOF. Let $F = P \wedge H \in MH\_F^{\triangle}$ with $|c_i| \geq 2k+1$, for all $c_i \in H$, $i = 1, \ldots, |H|$. For each $c_i \in H$ exists at least one triangle $\triangle_j \in G_P$, for one $j \in \{1, \ldots, k\}$, such that all variables of $\triangle_j$ are contained in c_i. Therefore setting exactly one arbitrary variable of each triangle to 0 and the other two variables to 1 yields a model for F. □

Lemma 4. *Let $F = P \wedge H \in MH\_F^{\triangle}$ and let k be the number of triangles in G_P. Further let $|c_i| = 2k$ for all clauses $c_i \in H$, for $i = 1, \ldots, |H|$. If $|H| < 3^k$ then $F = P \wedge H$ is satisfiable.*

PROOF. Let $F = P \wedge H \in MH\_F^{\triangle}$ and let k be the number of triangles in G_P. As $|c_i| = 2k$ for all clauses $c_i \in H$, at least one triangle $\triangle_j \in G_P$, for one $j \in \{1, \ldots, k\}$, exists for each clause $c_i \in H$, such that at least two variables of $\triangle_j$ occur in c_i. W.l.o.g. we do not consider clauses $c_i \in H$, for which there exists at least one triangle $\triangle_j \in G_P$ whose variables occur all in c_i, because these clauses are always satisfiable as shown in the proof of Lemma 3. Thus we assume that H only consists of clauses for which holds that each of them shares with each triangle of G_P exactly two variables. Note that this is the only possible case because of the condition $|c_i| = 2k$, for all clauses $c_i \in H$, for $i = 1, \ldots, |H|$. There are altogether exactly $\binom{3}{2}^k = 3^k$ many different clauses with this property. Suppose H contains all these possible 3^k many clauses, then H is not satisfiable. This results from the fact that in this case for each truth assignment which sets in each triangle exactly one variable to 0 and the other two variables to 1 we obtain exactly one unsatisfiable clause in H, namely the one consisting of all the variables set to 1. So, if $|H| < 3^k$ we can easily find a model for H. □

Theorem 22. *Let $F = P \wedge H \in MH\_F^{\triangle}$ and let $k \geq 3$ be the number of triangles in G_P. Further let $|c| = 2k-1$ for all clauses $c \in H$. If $|H| \leq 2k$, then F is satisfiable.*

PROOF. We consider the proof for the worst-case $|H| = 2k$. W.l.o.g. we do not consider clauses $c_i \in H$ with at least one triangle $\triangle_j \in G_P$ existing whose variables occur all in c_i because these clauses are always satisfiable by setting an arbitrary variable of $\triangle_j$ to 0.

Since $|c| = 2k-1$ for all $c \in H$, it follows that each clause of H contains at least one variable from each of the k different triangles. There are altogether $3k$ many different variables in F. The $2k$ many clauses of H contain altogether $2k(2k-1)$ many literals. It is $2k(2k-1) \geq 10k$, for $k \geq 3$. That means, that some of the $3k$ different variables occur more than once in H, more precisely: In average case each variable occurs $\frac{2k(2k-1)}{3k} = \frac{2}{3}(2k-1)$ many times in H.

- If $2k \equiv 1 \pmod 3$:
  Each of the $3k$ many different variables occurs either exactly $\frac{2}{3}(2k-1)$ times in H or there is a variable x occurring more than $\frac{2}{3}(2k-1)$ times in H. Obviously it is sufficient to have a variable x occurring at least $\frac{2}{3}(2k-1)$ times in H to satisfy F because setting x to 0 (and the two other variables of $\triangle_x$ to 1) satisfies at least $\frac{2}{3}(2k-1)$ many clauses of H and hence, after evaluating the formula, only $2k - \frac{2}{3}(2k-1) = \frac{2}{3}k + \frac{2}{3}$ many clauses remain in H. As $(\frac{2}{3}k + \frac{2}{3}) - (k-1) = -\frac{k}{3} + \frac{5}{3}$ and for $k \geq 5$ (note that for $k = 3, 4$

is $2k \not\equiv 1 \pmod 3$) we obtain: $-\frac{k}{3} + \frac{5}{3} \leq 0$, it follows that by setting in each of the $< k-1$ many remaining clauses one variable of the $k-1$ many remaining triangles to 0 we can satisfy H. Due to the fact that each clause of H contains at least one variable from each triangle of P we can uniquely assign one triangle $\triangle_j$ to each remaining clause c_h of H and set the one variable which c_h and $\triangle_j$ have in common to 0 and the two other variables of $\triangle_j$ to 1. This way F is satisfied.

- If $2k \not\equiv 1 \pmod 3$:
  In this case there is a variable x which occurs in at least $\lceil \frac{2}{3}(2k-1) \rceil$ many clauses and hence we can proceed as in the previous case.

□

The following corollary is a summary of the last results:

Corollary 2. *Let $M = P \wedge H \in MH\_F^{\triangle}$ and let $k \geq 1$ be the number of triangles in G_P, further let $|c_i| \geq 2k-1$ for all $i = 1, \ldots, |H|$. If $|H| \leq 2k$ then M is satisfiable.*

Chapter 3

Complexity of Linear Not-All-Equal SAT and Exact SAT Problems

Recently in [42] the propositional satisfiability problem (SAT) was shown to be NP-complete when restricted to the class of *linear* formulas in conjunctive normal form (CNF). Linear formulas yield a direct generalization of *linear hypergraphs*. Therefore linear formulas overlap only sparsely and there is some evidence that they form the algorithmically hard kernel for CNF-SAT, making this class specifically attractive with regard to other variants of SAT, too. In this chapter we investigate the computational complexity of some well-known variants of SAT, namely not-all-equal SAT (NAE-SAT) and exact SAT (XSAT) restricted to linear CNF instances. Recall that deciding NAE-SAT for a CNF formula means to test for the existence of a truth assignment such that in each clause of the formula at least one literal evaluates to true and at least one to false. For solving XSAT, exactly one literal in each clause must evaluate to true and all others to false. Observe that for CNF formulas where all clauses have exactly two literals XSAT and NAE-SAT coincide. As shown in the seminal paper by Schaefer [47], both NAE-SAT and XSAT are NP-complete for the unrestricted CNF class. Whereas SAT gets trivial on monotone formulas, which by definition are free of negated variables, NAE-SAT and XSAT are well known to remain NP-complete on that class. Note that monotone NAE-SAT coincides with the prominent NP-complete hypergraph bicolorability problem (also known as *set splitting* [21]). Moreover, monotone XSAT is closely related to the well-known NP-complete set partition problem (SPP) having many applications in combinatorial optimization. In addition, monotone XSAT is also closely related to the well-known NP-complete problem Exact Hitting Set [21] having many applications in combinatorial optimization.

3.1 NAE-SAT and XSAT-Complexity of Monotone Linear Formulas

Our first aim is to prove that XSAT and NAE-SAT behave NP-complete for arbitrary monotone linear CNF formulas. Recall that both XSAT and NAE-SAT remain NP-complete when restricted to monotone CNF formulas.

Theorem 23. *Both* XSAT *and* NAE-SAT *remain* NP*-complete when restricted to the class* LCNF_+ *of monotone linear formulas.*

Proof. As mentioned in the introduction, monotone NAE-SAT coincides with hypergraph bicolorability (which is the set splitting problem), and monotone XSAT corresponds to the set partition problem, both being well-known NP-complete problems (see e.g. [21]).

We first consider the monotone XSAT case and provide a polynomial-time reduction from CNF_+-XSAT to LCNF_+-XSAT. We then show that it can also be carried over to the second problem variant of interest, namely LCNF_+-NAE-SAT.

To that end, we take an arbitrary instance C from CNF_+. For each fixed variable $x_i \in V(C)$ having $r \geq 2$ occurrences in C, let say in the clauses $c_{j_1}, \ldots, c_{j_r}$ of C, we introduce the new variables $y_{x_i}^{j_1}, \ldots, y_{x_i}^{j_r} \notin V(C)$, and we replace the occurrence of x_i in c_{j_s} with $y_{x_i}^{j_s}$, for $1 \leq s \leq r$. Moreover, we introduce an auxiliary variable z_{x_i} also different from all variables. Finally, we add the following clauses to the formula, independently for each tuple $x_i, z_{x_i}, y_{x_i}^{j_1}, \ldots, y_{x_i}^{j_r}$, which is built for each fixed $x_i \in V(C)$ such that all new variables (i.e. variables not in $V(C)$) are pairwise distinct:

$$(*) \quad \{x_i, z_{x_i}\} \cup \bigcup_{1 \leq s \leq r} \{y_{x_i}^{j_s}, z_{x_i}\}$$

Observe that the resulting formula C' obtained from C after termination of the procedure just described is linear and positive monotone. The procedure runs in polynomial-time $O(n\|C\|)$ for n variables in $V(C)$. It remains to verify that $C \in \mathrm{XSAT}$ if and only if $C' \in \mathrm{XSAT}$. Let T denote the whole added 2-CNF formula generated according to $(*)$ which results after termination of the procedure described above.

First we show that any x-model of C which assigns a fixed truth value to variable x_i, assigns the same value to all new variables $y_{x_i}^{j_s}, 1 \leq s \leq r$, replacing x_i in C'. Indeed, this is ensured by subformula T since, for each tuple $x_i, y_{x_i}^{j_s}, 1 \leq s \leq r$, we have the implications according to XSAT:

$$\begin{aligned} x_i = 1 \quad &\Rightarrow \quad z_{x_i} = 0 \Rightarrow y_{x_i}^{j_s} = 1, \quad 1 \leq s \leq r \\ x_i = 0 \quad &\Rightarrow \quad z_{x_i} = 1 \Rightarrow y_{x_i}^{j_s} = 0, \quad 1 \leq s \leq r \end{aligned}$$

Conversely, according to T, any x-model of C' assigning a fixed truth value to one of the new variables $y_{x_i}^{j_s}$ replacing x_i must assign the same value to x_i and also to all other variables replacing x_i altogether demonstrating the XSAT-equivalence of $x_i \leftrightarrow y_{x_i}^{j_s}$, $1 \leq s \leq r$.

$$\begin{aligned} y_{x_i}^{j_s} = 1 \quad &\Rightarrow \quad z_{x_i} = 0 \Rightarrow x_i = 1,\ y_{x_i}^{j_{s'}} = 1, \quad 1 \leq s' \neq s \leq r \\ y_{x_i}^{j_s} = 0 \quad &\Rightarrow \quad z_{x_i} = 1 \Rightarrow x_i = 0,\ y_{x_i}^{j_{s'}} = 0, \quad 1 \leq s' \neq s \leq r \end{aligned}$$

The last observation directly implies that $C \in$ XSAT iff $C' \in$ XSAT.

Finally, we have no problem to cover the NAE-SAT case by the argumentation above because for the added 2-CNF part T NAE-SAT coincides with XSAT. Hence, we have $C \in$ NAE-SAT iff $C' \in$ NAE-SAT finishing the proof. □

Since for monotone CNF formulas NAE-SAT coincides with bicolorability of hypergraphs, we immediately obtain:

Corollary 3. *Bicolorability remains* NP*-complete when restricted to linear hypergraphs.*

The reduction given above adds 2-clauses to a non-linear input formula forcing all the newly-created variables to be assigned the same truth value in every model of C'. Therefore, if we consider the subclass $\text{LCNF}_+(\geq k)$ of LCNF_+, for fixed integer $k \geq 3$, where each formula contains only clauses of length at least k, then the reduction above does not work. Consequently, the question arises whether XSAT restricted to $\text{LCNF}_+(\geq k)$ remains NP-complete, too. Before giving the answer, we introduce some terminology and prove a useful Lemma.

Definition 4. *(1) Let C be a satisfiable formula. A variable $y \in V(C)$ is called a backbone variable of C, if y has the same value in each model of C.*

(2) Let C be an x-satisfiable formula. A variable $y \in V(C)$ is called an x-backbone variable of C, if y has the same value in each x-model of C.

It might be instructive to consider the following example: Let C be the x-satisfiable formula:

$$C = \{x_1x_2x_5, x_2x_3, x_1x_3x_4\}$$

where, for simplicity, clauses are represented as strings of literals. The only x-models of C obviously are: $x_1 = 0, x_2 = 1, x_3 = 0, x_4 = 1, x_5 = 0$ and $x_1 = 0, x_2 = 0, x_3 = 1, x_4 = 0, x_5 = 1$. For this reason x_1 is an x-backbone variable.

Lemma 5. *For each fixed $k \geq 3$, we can efficiently construct a monotone k-uniform linear formula C of $O(k^3)$ many variables and $O(k^2)$ many clauses such that C contains at least k x-backbone variables that all must be assigned 0.*

Proof. To construct the formula, we start with defining a first clause

$$c_0 = \{x, y_1, \ldots, y_{k-1}\}$$

Secondly we introduce a $(k-1) \times (k-1)$ variable matrix $A = (a_{ij})_{1 \leq i,j \leq k-1}$, and regard its rows as $(k-1)$-clauses. Enlarging each of these clauses with x yields $k-1$ additional k-clauses collected in set X having the property that $X \cup \{c_0\}$ is linear. Next we similarly build a set Y_1 of k-clauses as follows: first take the transpose A^T of A then enlarge each of its rows with y_1. Again $X \cup Y_1 \cup \{c_0\}$ is a linear clause set. Finally, let $\hat{A}^T$ be the matrix obtained from A^T by performing a cyclic shift of $i-1$ positions on the ith column of A^T. We form a clause set Y_2 enlarging the rows of $\hat{A}^T$ with y_2, each. Recall that by construction each clause of X contains variable x and each clause of Y_1, Y_2 contains y_1, y_2, respectively. Let C' denote the formula obtained from $\{c_0\} \cup X \cup Y_1 \cup Y_2$ by removing the last clause, referenced

to as c, from Y_2. For example, in case $k = 4$, C' (resp. c) looks as follows (omitting set embraces and representing clauses as literal strings):

$$C' = \begin{array}{cccc} x & y_1 & y_2 & y_3 \\ x & a_{11} & a_{12} & a_{13} \\ x & a_{21} & a_{22} & a_{23} \\ x & a_{31} & a_{32} & a_{33} \\ y_1 & a_{11} & a_{21} & a_{31} \\ y_1 & a_{12} & a_{22} & a_{32} \\ y_1 & a_{13} & a_{23} & a_{33} \\ y_2 & a_{11} & a_{23} & a_{32} \\ y_2 & a_{12} & a_{21} & a_{33} \end{array}, \quad c = y_2\ a_{13}\ a_{22}\ a_{31}$$

Clearly C' has $k + (k-1)^2 = O(k^2)$ variables and $3(k-1) = O(k)$ clauses. We claim that C' has at least two x-backbone variables which both have to be set to 0. From this claim the assertion is implied as follows: We construct $\lceil \frac{k}{2} \rceil$ many variable-disjoint copies of C' resulting in formula C. C clearly is k-uniform, linear, consists of $O(k^3)$ variables and $O(k^2)$ clauses. And C has at least $2 \cdot \lceil \frac{k}{2} \rceil \geq k$ x-backbone variables which must be assigned 0.

So the claim remains to be settled: We show that $C' \in$ XSAT and that each x-model of C' assignes 0 to x and y_1. First we show that neither x nor y_1 can be set to 1 by any x-model of C: Assume that x is set to 1 then all variables in A and therefore also all variables in A^T would be forced to 0 implying that y_1 must also be set to 1 x-contradicting the leading clause c_0. An analogous argumentation shows that y_1 is an x-backbone variable with truth value 0. Next we claim that a (canonical) x-model t for C' is provided by assigning 1 to all variables in the removed clause c, and 0 to all other variables. Since $C' \cup \{c\}$ is linear no two variables of c can occur in any clause of C'. So, t assigns to 1 at most one literal in each clause of C'. Clearly, c_0 and the remaining clauses of Y_2 all contain y_2 and therefore are x-satisfied. Finally, by construction all $k-1$ variables in $c - \{y_2\}$ are members of A and also of A^T. So as we have $k-1$ variables in $c - \{y_2\}$ and exactly $k-1$ rows in A resp. A^T, we have to distribute $k-1$ variables on $k-1$ rows in A resp. A^T. It follows that each row in A resp. A^T must contain exactly one variable in $c - \{y_2\}$. Because of linearity it is also clear that no variable in $c - \{y_2\}$ can occur twice in A resp. A^T. Therefore all clauses in X and Y_1 are x-satisfied finishing the proof. □

Now we are able to prove:

Theorem 24. *For each fixed $k \geq 3$,* XSAT *remains* NP*-complete restricted to* $\mathrm{LCNF}_+(\geq k)$.

Proof. For each fixed $k \geq 3$, the basic idea here essentially is to perform the reduction as shown in the proof of Theorem 23 from $\mathrm{CNF}_+(\geq k)$-XSAT thereby padding the added 2-clauses by x-backbone variables sets such that they get k-clauses and the XSAT status of the corresponding formulas is preserved.

More precisely, let Γ_k be the monotone k-uniform formula according to Lemma 5. For each added 2-clause form a copy of Γ_k such that all these copies are pairwise variable disjoint. Enlarge each 2-clause with $k-2$ of these x-backbone variables stemming from the corresponding Γ_k-copy. Observe that all these padding formulas Γ_k are already constructed as monotone formulas. Clearly, the resulting formula is linear and k-uniform and is in XSAT iff the original formula is because

all backbone variables always are assigned 0, and all Γ_k copies are x-satisfiable by construction. □

It is still an open question whether NAE-SAT restricted to $\mathrm{LCNF}_+(\geq k)$, for $k \geq 3$, remains NP-complete, too. Before giving the answer, we again introduce some terminology and then prove two useful Lemmas.

Definition 5. *A formula is called minimally nae-unsatisfiable if it is nae-unsatisfiable but removing an arbitrary clause from it yields a nae-satisfiable formula. We call a set $U \subseteq L(C)$ of literals in a nae-satisfiable formula C a nae-backbone set, if each nae-model of C sets the literals in U either all to 0 or all to 1.*

Lemma 6. *Given a formula C that is nae-unsatisfiable, then C contains a minimally nae-unsatisfiable subformula C'. Moreover, any clause c of C' forms a nae-backbone set in $C' - \{c\}$.*

Proof. First of all there must exist a nae-satisfiable subformula of C because any single clause of it has this property. Suppose there exists no minimally nae-unsatisfiable subformula of C, then each nae-unsatisfiable subformula of it has the property that it contains a clause which can be removed and the resulting formula remains nae-unsatisfiable. So, we inductively obtain a contradiction to the fact that there are nae-satisfiable subformulas of C.

To prove the second assertion, let C be a minimally nae-unsatisfiable formula, and $C' := C - \{c\}$, for arbitrary $c \in C$, then clearly C' is nae-satisfiable. We claim that c is a nae-backbone set in C'. First we observe that $V(c) \subseteq V(C')$ because otherwise there is a variable $u \in V(c)$ not occurring in C'. Evidently, we can always set u in such a way that c is nae-satisfied in C yielding a contradiction. Each nae-model of C' sets all literals in c either to 0 or all to 1, otherwise C would be nae-satisfiable again yielding a contradiction. So c forms a nae-backbone set in C'. □

Lemma 7. *If there is a linear, k-uniform and unsatisfiable formula C, for which additionally holds that each clause of C is either positive monotone or negative monotone, there is a linear, k-uniform and positive monotone formula C' which is not satisfiable concerning* NAE-SAT.

Proof. Given an arbitrary CNF formula C, then it is easy to see that $C \in$ NAE-SAT holds true if, and only if, $C \cup C^\gamma \in$ SAT, where C^γ denotes the formula obtained from C by complementing the literals of each clause in C. Now let C be a linear, k-uniform and unsatisfiable formula for which additionally holds that each clause of C is either positive monotone or negative monotone. As C is unsatisfiable $C \cup C^\gamma$ is unsatisfiable as well according to NAE-SAT. Further we can arrange $C \cup C^\gamma$ in $C' \cup (C')^\gamma$ such that C' only contains the positive monotone clauses, i.e. $C \cup C^\gamma = C' \cup (C')^\gamma$. Hence C' is a linear, k-uniform and positive monotone formula, which is obviously not satisfiable according to NAE-SAT. □

Now for the NAE-SAT case we obtain the NP-completeness for $\mathrm{LCNF}_+(\geq k)$ from Schaefer's Theorem [47] as follows:
For each $k \geq 3$ the class of linear, k-uniform formulas C, for which additionally holds that each clause of C is either positive monotone or negative monotone, is

NP-complete according to Schaefer's Theorem [47]. Let $\mathrm{LCNF}_{+,-}(= k)$ denote this class. Hence for each $k \geq 3$ there is a formula $F \in \mathrm{LCNF}_{+,-}(= k)$ which is not satisfiable. So according to Lemma 7 there is a linear, k-uniform and positive monotone formula C' which is not satisfiable concerning NAE-SAT for each $k \geq 3$. Hence this formula is specifically nae-unsatisfiable, and we can extract from it a nae-satisfiable monotone formula Γ_k having a nae-backbone set of k variables according to Lemma 6. Similarly performing the copy and padding steps as stated for the XSAT case above yields the same result. Applying Lemma 6 we therefore obtain:

Theorem 25. *For each $k \geq 3$,* NAE-SAT *remains* NP*-complete when restricted to* $\mathrm{LCNF}_{+}(\geq k)$. □

Remark 3. *Regarding the* NAE-SAT *case we are only able to find nae-unsatisfiable, linear and positive monotone formulas with clause length at most 3 and 4. For $k = 3$ the corresponding formula is given by the 3-block. For $k = 4$ the formula was constructed on base of the scheme described in [41] using the* SAT *solver designed in [11]. For each $k \geq 5$ the problem remains open.*

3.2 The linear, l-regular Formula Class

Theorem 26. NAE-SAT *remains* NP*-complete when restricted to the class* LCNF^{l}_{+} *of monotone, linear and l-regular formulas.*

Proof. According to Theorem 23 NAE-SAT is NP-complete for LCNF_{+}. Thus we provide a polynomial-time reduction from LCNF_{+}-NAE-SAT to LCNF^{l}_{+}-NAE-SAT proving the NP-completeness of the latter. So, let $C \in \mathrm{LCNF}_{+}$ be an arbitrary formula. For each variable $x \in V(C)$ with $w(x) > l$ we introduce $w(x) - (l-1)$ new, pairwise different variables $y^x_1, y^x_2, \ldots, y^x_{w(x)-(l-1)}$. Then the first $l-1$ occurrences of x in C remain, but each further occurrence of x we replace by a y^x_i, for $i = 1, \ldots, w(x) - (l-1)$. This way we obtain a formula C' where each variable occurs at most l times. To provide NAE-SAT-equivalence to C we now add the following 2-clauses to C' for each of these $x \in V(C)$ occurring more than l times in C:
If $w(x) - (l-1) \leq l$ we add the following 2-clauses to C' obtaining C'':

$$(x \vee z^x) \wedge (y^x_1 \vee z^x) \wedge \ldots \wedge (y^x_{w(x)-(l-1)} \vee z^x)$$

Let C be nae-satisfiable with $x = 1$ then $z^x = 0$ and hence $y^x_1 = y^x_2 = \ldots = y^x_{w(x)-(l-1)} = 1$. If C is nae-satisfiable with $x = 0$, the newly added 2-clauses imply $z^x = 1$ and hence $y^x_1 = y^x_2 = \ldots = y^x_{w(x)-(l-1)} = 0$. So C'' is nae-equivalent to C and obviously in this case all the variables occur at most l times in C''.
If $w(x) - (l-1) > l$, we add:

$$(x \vee z^x_1) \wedge (y^x_1 \vee z^x_1) \wedge \ldots \wedge (y^x_{l-1} \vee z^x_1) \wedge (v^x_1 \vee z^x_1) \wedge (v^x_1 \vee z^x_2) \wedge (y^x_l \vee z^x_2) \wedge$$
$$(y^x_{l+1} \vee z^x_2) \wedge \ldots \wedge (y^x_{2l-2} \vee z^x_2) \wedge (v^x_2 \vee z^x_2) \wedge (v^x_2 \vee z^x_3) \wedge (y^x_{2l-1} \vee z^x_3) \wedge \ldots$$

If C is nae-satisfiable with $x = 0$ the newly added 2-clauses imply $z^x_1 = 1$ and hence: $y^x_1 = y^x_2 = \ldots = y^x_{(l-1)} = 0$, $v^x_1 = 0$, hence $z^x_2 = 1$ and finally $y^x_i = 0$, for all

$i \in \{l, \ldots, w(x) - (l-1)\}$. If C is nae-satisfiable with $x = 1$ we also obtain $y_i^x = 1$, for all $i \in \{1, \ldots, w(x) - (l-1)\}$. So C'' is nae-equivalent to C and obviously in this case all the variables also occur at most l times in C''.
So we have transformed C to C'' such that C is equivalent to C'' according to NAE-SAT and all the variables in C'' occur at most l times. Thus it remains to transform C'' to a formula $C''' \in \text{LCNF}_+^l$ such that C is equivalent to C''' according to NAE-SAT. For this purpose we proceed as follows:
Let $x \in C''$ be a variable with $w(x) < l$. We add the following 2-clauses to C'': $(x \vee d_1^x) \wedge (x \vee d_2^x) \wedge \ldots \wedge (x \vee d_{l-w(x)}^x)$, where the variables d_i^x, for $i \in \{1, \ldots, l-w(x)\}$, are not already occurring in $V(C'')$ and are pairwise different. Further, to achieve l-regularity for the variables d_i^x, for $i = 1, \ldots, l - w(x)$, we add the following 2-clauses: $(d_i^x \vee a_1^x) \wedge (d_i^x \vee a_2^x) \wedge \ldots \wedge (d_i^x \vee a_{l-1}^x)$, for each $i \in \{1, \ldots, l-w(x)\}$, where the variables a_j^x, for $j \in \{1, \ldots, l-1\}$, are not already occurring in $V(C'')$ and are pairwise different. Now $w(d_i^x) = l$ and $w(a_j^x) = l - w(x)$, for $j = 1, \ldots, l-1$. Hence we have to add further 2-clauses: $(a_j^x \vee b_1^x) \wedge (a_j^x \vee b_2^x) \wedge \ldots \wedge (a_j^x \vee b_{w(x)}^x)$, where the variables b_i^x, for $i \in \{1, \ldots, w(x)\}$, are not already occurring in $V(C'')$ and are pairwise different. So now each variable occurs exactly l times apart from the b_i^x, for $i = 1, \ldots, w(x)$, which occur $l-1$ times in the current formula. Thus for each such b_i^x we add the following clauses to the current formula: $(b_i^x \vee e^{b_i^x}) \wedge (e^{b_i^x} \vee f_1^{b_i^x} \vee g_1^{b_i^x}) \wedge \ldots \wedge (e^{b_i^x} \vee f_{l-1}^{b_i^x} \vee g_{l-1}^{b_i^x}) \wedge (f_l^{b_i^x} \vee g_l^{b_i^x})$. Now $w(b_i^x) = w(e^{b_i^x}) = l$, so it only remains to establish l-regularity of the variables $g_j^{b_i^x}$ and $f_j^{b_i^x}$, for $j = 1, \ldots, l$. We can achieve this by adding in a final step the following 2-clauses: $(f_1^{b_i^x} \vee g_j^{b_i^x}) \wedge (f_2^{b_i^x} \vee g_{j+1}^{b_i^x}) \wedge \ldots \wedge (f_l^{b_i^x} \vee g_{j+l-1}^{b_i^x})$, for each $j \in \{2, \ldots, l\}$, where $l + i = i$, for $i > 1$. Let C''' be the formula we obtain after having added all these 2-clauses to C''. As the subformula consisting of all these 2-clauses added to C'' is obviously always nae-satisfiable, C'' and C''' are equivalent according to NAE-SAT. □

Theorem 27. SAT *remains* NP*-complete when restricted to the class* LCNF^l *of linear and* l*-regular formulas.*

Proof. It is a well known result that SAT is NP-complete when restricted to the linear formula class LCNF. Let $C \in \text{LCNF}$ be an arbitrary formula. If C is l-regular, $C \in \text{LCNF}^l$ and we are done. Otherwise for each variable $x \in V(C)$ occurring more than l times in C we proceed as follows: Let $c_1^x, c_2^x, \ldots, c_{w(x)}^x$ be all the clauses containing x, then we introduce new, pairwise different variables $y_1^x, \ldots, y_{w(x)}^x$ and replace the occurrence of x in c_i^x, for each $i \in \{1, \ldots, w(x)\}$, by y_i^x without affecting the polarity. Further, we add the following 2-clauses which provide SAT equivalence of the variables y_i^x with x:

$$(x \vee \overline{y_1^x}) \wedge (y_1^x \vee \overline{y_2^x}) \vee (y_2^x \vee \overline{y_3^x}) \wedge \ldots \wedge (y_{w(x)-1}^x \vee \overline{y_{w(x)}^x}) \wedge (y_{w(x)}^x \vee \overline{x})$$

Let C' be the resulting formula. Obviously $w(x) \leq l$, for all $x \in V(C')$. To provide l-regularity of the formula we now consider all the variables $x \in V(C')$ for which holds $w(x) < l$ and proceed as follows: For each such variable $x \in V(C')$ we add the following 2-clauses to C':

$$(x \vee z_1^x) \wedge (x \vee z_2^x) \wedge \ldots \wedge (x \vee z_{l-(w(x))}^x)$$

where z_i^x, for $i \in \{1, \ldots, l - w(x)\}$ are new, pairwise different variables not yet occurring in $V(C')$. Now $w(x) = l$, but to achieve l-regularity for z_i^x we have to add further clauses: For each $i \in \{1, \ldots, l - w(x)\}$ we add:

$$(z_i^x \vee a_1^x) \wedge (z_i^x \vee a_2^x) \wedge \ldots \wedge (z_i^x \vee a_{l-1}^x)$$

where a_j^x, for $j \in \{1, \ldots, l-1\}$ are new and pairwise different variables. We have now achieved $w(z_i^x) = l$, for all $i \in \{1, \ldots, l - w(x)\}$ but it still remains to establish l-regularity for the a_i^x, for $i \in \{1, \ldots, l-1\}$ because now $w(a_i^x) = l - w(x)$. Thus we finally add $(a_i^x \vee b_1^x) \wedge (a_i^x \vee b_2^x) \wedge \ldots \wedge (a_i^x \vee b_{w(x)}^x)$ with b_j^x, for $j \in \{1, \ldots, w(x)\}$, new and pairwise different variables. The last clause we add to C' to establish l-regularity is $(b_1^x \vee b_2^x \vee \ldots \vee b_{w(x)}^x)$. Let C'' be the formula that we get when adding all these clauses to C'. Obviously the resulting formula which consists of all these newly added clauses to C' is always satisfiable no matter how x is set. Thus C and C'' are SAT-equivalent. □

3.3 XSAT on Linear Formulas with Regularity Conditions

Recall that LCNF denotes the linear formula class, XLCNF the exact linear formula class. Moreover, $k - A$, $(\geq k) - A$ or $(\leq k) - A$ denote formulas of the class $A \in \{\text{CNF}, \text{LCNF}, \text{XLCNF}\}$ for which additionally holds that their clauses have length exactly k, at least k or at most k. Then the classes A^l, $A^{\geq l}$ or $A^{\leq l}$ contain formulas of the class $A \in \{\text{CNF}, \text{LCNF}, \text{XLCNF}\}$, for which additionally holds that all variables in these formulas occur exactly l times, at least l times or at most l times. A_+ denotes the positive monotone formulas of the class $A \in \{\text{CNF}, \text{LCNF}, \text{XLCNF}\}$.

In this section we focus on the following classes:

$$\text{CNF}_+ \supset k\text{-CNF}_+ \supset k\text{-CNF}_+^{\leq l} \supset k\text{-CNF}_+^{l} \supset (\leq l)\text{-LCNF}_+^{\geq l} \supset (\leq l)\text{-LCNF}_+^{l}$$
$$\text{CNF}_+ \supset \text{CNF}_+^{l} \supset \text{LCNF}_+^{l} \supset (\leq l)\text{-LCNF}_+^{l}$$

Clearly, the same inclusion relations are also valid for the non-monotone counterparts. For $k \leq 2$ or $l \leq 2$, all above classes behave polynomial-time solvable regarding XSAT and also NAE-SAT, which is even the case for variable-weighted optimisation versions of these problems [42].

For CNF_+ and k-CNF_+ the NP-completeness of XSAT is well known. For LCNF_+, k-LCNF_+ and $(\geq k)$-LCNF_+ the NP-completeness was shown in the last section. For the remaining classes we prove the NP-completeness of XSAT in this section. The next two results treat the non-linear case, which will be referred to later.

Theorem 28. XSAT *remains* NP*-complete for* k-$\text{CNF}_+^{\leq l}$ *and* k-$\text{CNF}^{\leq l}$, $k, l \geq 3$.

Proof. We provide a polynomial-time reduction from k-CNF_+-XSAT (which is NP-complete [21]) to k-$\text{CNF}_+^{\leq l}$ establishing NP-completeness of the latter and thus of k-$\text{CNF}^{\leq l}$. To that end, let C be an arbitrary formula in k-CNF_+. For each $x \in V(C)$ with $w_C(x) > l$, we introduce $p := w_C(x) - (l-1)$ new variables

$x_1, x_2, \ldots, x_p$. Let the first $l-1$ occurrences of x remain unchanged and replace the p remaining occurrences of x by the variables $x_1, x_2, \ldots, x_p$. Let C' be the resulting formula. Next we introduce new, pairwise different variables a_{ij}, for $i = 1, \ldots, p$, $j = 1, \ldots, k-1$, and add the following clauses to C' which ensure XSAT-equivalence of the newly introduced variables $x_1, x_2, \ldots, x_p$ with x.

$$\begin{aligned}
&(x \vee a_{11} \vee a_{12} \vee \ldots \vee a_{1,k-1}) \wedge (x_1 \vee a_{11} \vee a_{12} \vee \ldots \vee a_{1,k-1}) \\
&\wedge (x_1 \vee a_{21} \vee a_{22} \vee \ldots \vee a_{2,k-1}) \wedge (x_2 \vee a_{21} \vee a_{22} \vee \ldots \vee a_{2,k-1}) \\
&\wedge \ldots \\
&\wedge (x_{p-1} \vee a_{p1} \vee a_{p2} \vee \ldots \vee a_{p,k-1}) \wedge (x_p \vee a_{p1} \vee a_{p2} \vee \ldots \vee a_{p,k-1})
\end{aligned}$$

Hence C and C' are XSAT-equivalent, and obviously no variable occurs more than l times in C'. □

Theorem 29. XSAT *remains* NP*-complete for* k-CNF$^l_+$ *and* k-CNF^l, $k, l \geq 3$.

Proof. We provide a polynomial-time reduction from k-CNF$^{\leq l}_+$-XSAT (which is NP-complete) to k-CNF$^l_+$-XSAT similar to the technique in [32]. Let C be an arbitrary formula in k-CNF$^{\leq l}_+$ with variable set $V(C) = \{x_1, \ldots, x_n\}$. We introduce l pairwise variable-disjoint copies $C_1, \ldots, C_l$ of C, such that the variables in C_i are $\{x^i_1, \ldots, x^i_n\}$, for $i = 1, \ldots, l$. For each $x_j \in V(C)$ with $w_C(x_j) < l$ we construct the formulas $D_{x_j,1}, \ldots, D_{x_j,l-w_C(x_j)}$ with

$$D_{x_j,i} = \bigwedge_{r=1}^{l} \left(x^r_j \vee a_{i,1} \vee \ldots \vee a_{i,k-1}\right)$$

for $1 \leq i \leq l - w_C(x_j)$. Note that $a_{i,j}$ occurs exactly l times in $D_{x_j,i}$ and nowhere else, for $i = 1, \ldots, l - w_C(x_j), j = 1, \ldots, k-1$. Defining

$$C' = \bigwedge_{i=1}^{l} C_i \wedge \bigwedge_{x_j \in V(C)} \bigwedge_{i=1}^{l-w_C(x_j)} D_{x_j,i}$$

we observe that x^i_j occurs $w_C(x_j)$ times in C_i and once in each $D_{x_j,i}$, for $i = 1, \ldots, l - w_C(x_j)$. Thus each x^i_j occurs l times in C'. So C' belongs to k-CNF$^l_+$.

We show that $C \in$ XSAT if and only if $C' \in$ XSAT. Let C be x-satisfiable, then we can use a fixed x-model t of C to x-satisfy the copies $C_1, \ldots, C_l$ of C. If $t(x_j) = 1$ we also set $x^r_j = 1$, for $r = 1, \ldots, l$, and $a_{i,1} = \ldots = a_{i,k-1} = 0$ yielding a x-model for $D_{x_j,i}$, for all $x_j \in V(C)$ and $1 \leq i \leq w_C(x_j)$. If $t(x_j) = 0$ we assign $x^i_j = 0$, for $i = 1, \ldots, l$ and we set $a_{i,1} = 1$ as well as $a_{i,2} = \ldots = a_{i,k-1} = 0$ yielding a x-model for $D_{x_j,i}$, for all $x_j \in V(C)$ and $1 \leq i \leq w_C(x_j)$. The reverse direction is obvious. □

The situation is different from the SAT case where there are k and l, for $k, l \geq 3$, such that k-CNF^l-SAT is polynomial-time solvable as mentioned in the introduction [32]. Now we are able to pay attention to the linear and l-regular classes.

Theorem 30. XSAT *remains* NP*-complete for* LCNF$^l_+$, *and* LCNF^l, $l \geq 3$.

PROOF. We provide a polynomial-time reduction from CNF^l_+-XSAT (which is NP-complete) to LCNF^l_+-XSAT. Let C be an arbitrary formula in CNF^l_+. Is C is not linear, we proceed as follows for each variable $x_i \in V(C)$: Since each fixed variable $x_i \in V(C)$ has exactly l occurrences in C, namely in the clauses $c_{j_1}, \ldots, c_{j_l}$, we introduce a new variable $y^{j_s}_{x_i} \notin V(C)$, for each such occurrence $2 \leq s \leq l$, except for the first occurrence of x_i in c_{j_1}. Then we replace each occurrence of x_i in c_{j_s} (except in c_{j_1}) with $y^{j_s}_{x_i}$, for $2 \leq s \leq l$. Let C' be the resulting formula. Then C' is obviously linear, monotone and each variable occurs exactly once in C'. For each $x_i \in V(C)$, we introduce new, pairwise different variables $z^{x_i}_1, \ldots, z^{x_i}_{l-1} \notin V(C')$. Next we add the following 2-clauses to C' providing XSAT-equivalence of the variables $x_i, y^{j_2}_{x_i}, \ldots, y^{j_l}_{x_i}$:

$$\begin{aligned} P_{x_i} =& (x_i \vee z^{x_i}_1) \wedge (x_i \vee z^{x_i}_2) \wedge \ldots \wedge (x_i \vee z^{x_i}_{l-1}) \\ & \wedge (y^{j_2}_{x_i} \vee z^{x_i}_1) \wedge (y^{j_2}_{x_i} \vee z^{x_i}_2) \wedge \ldots \wedge (y^{j_2}_{x_i} \vee z^{x_i}_{l-1}) \\ & \wedge (y^{j_3}_{x_i} \vee z^{x_i}_1) \wedge (y^{j_3}_{x_i} \vee z^{x_i}_2) \wedge \ldots \wedge (y^{j_3}_{x_i} \vee z^{x_i}_{l-1}) \\ & \wedge \ldots \\ & \wedge (y^{j_l}_{x_i} \vee z^{x_i}_1) \wedge (y^{j_l}_{x_i} \vee z^{x_i}_2) \wedge \ldots \wedge (y^{j_l}_{x_i} \vee z^{x_i}_{l-1}) \end{aligned}$$

Observe that $C'' := \bigwedge_{x_i \in V(C)} P_{x_i} \wedge C'$ is l-regular: $w_{C''}(z^{x_i}_r) = l$, for $r = 1, \ldots, l-1$, $w_{C''}(y^{j_s}_{x_i}) = l$, for $s = 2, \ldots, l$, and $w_{C''}(x_i) = l$, for each $x_i \in V(C)$. Since every variable appears only once in C', C' is linear. Obviously P is linear; and the variables $z^{x_i}_1, \ldots, z^{x_i}_{l-1}$ do not occur in C', so $P \wedge C' = C''$ is linear, too. C is XSAT-equivalent with C'', because of the XSAT-equivalence of the variables $x_i, y^{j_2}_{x_i}, \ldots, y^{j_l}_{x_i}$ and P_{x_i} are always x-satisfiable, for all i. □

Finding a concrete reduction for the NP-completeness proof of XSAT for k-LCNF^l_+ is a very complicated problem. We are only able to show NP-completeness of XSAT for l-LCNF^l, where $l = q + 1$ and q is a prime power. This is due to the fact that we can exploit block formula patterns providing backbone-variables. A k-block formula directly corresponds to a finite projective plane of order $k-1$. Unfortunately it is a hard open question to decide whether a projective plane exists for a given $k \in \mathbb{N}$ [46]. However, it is a well known fact in combinatorics that for prime power orders the corresponding projective planes can easily be computed.

Theorem 31. XSAT *remains* NP*-complete for* l-LCNF^l*, for* $l = q + 1$*, where* q *is a prime power.*

PROOF. We provide a polynomial-time reduction from l-CNF^l_+ to l-LCNF^l, for $l = q + 1$, where q is a prime power. Let $C \in l$-CNF^l_+ be an arbitrary formula and $V(C) = \{x_1, x_2, \ldots, x_n\}$ the set of its variables. For each variable $x_i \in V(C)$ we proceed like in the beginning of the proof of Theorem 30 obtaining the corresponding formulas C' and $P = \bigwedge_{x_i \in V(C)} P_{x_i}$. Next we enlarge each 2-clause of P by exactly $l - 2$ many x-backbone variables all of which must be assigned to 0 and obtain l-clauses this way. The x-backbone variables are provided via l-block formulas as follows: Each such l-block formula B_l is l-regular, l-uniform and exists whenever $l = q + 1$, for q prime power [42]. B_l is not x-satisfiable but removing an arbitrary clause of B_l yields a x-satisfiable formula, where the variables of the removed clauses are x-backbone variables which have to be set to 1 [39]. Therefore

we provide $n(l-1)(l-2)$ many l-block formulas which are pairwise variable-disjoint. Removing a clause of each of them in total yields $n(l-1)(l-2)l$ distinct x-backbone variables. Since in P we have $n(l-1)l$ many 2-clauses, each of which needs $l-2$ variables to become an l-clause, this fits perfectly. Let P' be the formula obtained from P this way. Then $C'' := C' \wedge P'$ is l-regular and l-uniform by construction. Moreover, C'' is x-satisfiable if, and only if, C is x-satisfiable because P provides the XSAT-equivalence of the original variables with the replaced ones which is preserved by P' through the x-backbone 0 variables. Note that C'' is non-monotone as we have to negate the x-backbone variables when adding them to the clauses of P . □

This result provides evidence that NP-completeness also holds for all values of $l \geq 3$. Unfortunately, the proof does not easily transfer to the monotone case due to the fact that we have not been able to find suitable formulas providing backbone variables which must be set to 0. However considering the monotone case we are able to treat the following larger classes.

Theorem 32. XSAT *is* NP*-complete for* $(\leq l)$-LCNF$^l_+$, $(\leq l)$-LCNF^l, $l \geq 3$.

Proof. We provide a polynomial time reduction from l-CNF$^l_+$-XSAT (which is NP-complete according to Theorem 29) to $(\leq l)$-LCNF$^l_+$-XSAT. Let C be an arbitrary formula in l-CNF$^l_+$. If C is not linear, we proceed as in the beginning of the proof of Theorem 30 obtaining the corresponding formulas C' and $P = \bigwedge_{x_i \in V(C)} P_{x_i}$. Again $C'' := \bigwedge_{x_i \in V(C)} P_{x_i} \wedge C'$ is l-regular and linear by construction. Moreover, each clause of C'' has a clause length of at most l, because in C' each clause has a length of exactly l and P consists of 2-clauses only. C'' is XSAT-equivalent with C which is ensured by P. □

Remark 4. *Note that* XSAT *also remains* NP*-complete for the class* $(\leq l)$-LCNF$^{\geq l}_+$ *and for* $(\leq l)$-LCNF$^{\geq l}$, $l \geq 3$.

3.4 An Algorithm Solving XSAT For LCNF$^l_+$

In this section we present an algorithm solving XSAT for the class LCNF$^l_+$ running faster than the so far best algorithm for XSAT by J. M. Byskov et al. [12]. Our Algorithm AVRG uses backtracking and exploits the l-regularity of the formulas to apply an *average clause length* argument. Let $C \in$ LCNF$^l_+$ be a formula with average clause length α. Then Algorithm AVRG works as follows: It searches a variable $x \in V(C)$ such that the formula $C(x)$ consisting of all clauses containing x has a clause legth $\geq \alpha$. Setting $x = 1$ and the other variables in the x-clauses to 0 we obtain a formula C' with an average clause length $\alpha_1 \leq \alpha$. As long as the formula is not empty and the average clause length of the remaining formula is still > 2 we search for a variable x_i such that $C(x_i)$ has at least the current average clause length. In case the formula is empty, we have found an x-model for C. If the average clause length of the current formula is ≤ 2, we can solve XSAT for this formula in polynomial-time by applying a polynomial-time algorithm to this formula. If it is not x-satisfiable, we backtrack: We undo the setting made in the last recursion step such that the variable x which we have set to 1 before in the last recursion step we now set to 0 and the other variables in the x-clauses we do not assign for now.

If no backtracking is possible and no x-model has been found yet, C is unsatisfiable.

Algorithm AVRG

INPUT: $C^0 \in \text{LCNF}^l_+$ with the average clause length $\alpha_0 = \frac{n \cdot l}{m}$, where n is the number of variables and m the number of clauses of C^0.
OUTPUT: an x-model, if C^0 is x-satisfiable, C^0 *is not x-satisfiable*, else.
begin

1. As long as the current formula is not empty do:
   (a) Apply the unit clause rule.
   (b) Calculate the average clause length α_i of the current formula C^i.
   (c) If the average clause length α_i of the current formula C^i is ≤ 2:
   - Solve XSAT for C^i by applying a polynomial-time 2-XSAT algorithm.
   - If the algorithm returns an x-model for C^i, then an x-model for C^0 is also found.
   - Else backtrack: Let x_{i-1} be the variable which is now set to 1 and for which holds that it has been set to 1 as last (of all the variables of C^0 which are also set to 1 actually). Add all x_{i-1}-clauses which we have deleted before to the current formula. Further undo the assignment of variables which we have performed in the steps after x_{i-1} was set to 1 and set $x_{i-1} = 0$. Note that now especially all the variables in the x_{i-1}-clauses (apart from x_{i-1}) are not assigned.
   - If no backtracking is possible and the formula is not empty yet then return C^0 *is not x-satisfiable*

   (d)
   - Search for a variable $x_i \in V(C^i)$ of the current formula C^i, such that the subformula $C^i(x_i)$, consisting of all the x_i-clauses, has an average clause length of at least α_i.
   - Set $x_i = 1$ and all the other variables of $C(x_i)$ to 0.
   - Remove $C^i(x_i)$ from C^i and also remove all the variables which are set to 0 from the remaining clauses in C^i, obtaining the formula C^{i+1}.
   - If the formula C^{i+1} is empty, then an x-model for C is found.
   - If there is a contradiction now (e.g. an empty clause), backtrack to the formula C^i and undo the setting of the variables in $C^i(x_i)$. Instead, assign $x_i = 0$ now. If no backtracking is possible, return C^0 *is not x-satisfiable*

end

In the following we analyse the running time of **Algorithm AVRG** for the class LCNF^l_+, which is NP-complete according to Theorem 30. Let $C^0 \in \text{LCNF}^l_+$ be an arbitrary formula with n variables and m clauses.
If $l \geq n$, we obtain for all $x_i \in V(C^0)$ that C^0 has at least one unit clause (x_i), this is due to the pigeonhole principle. Hence we must set $x_i = 1$, for all $x_i \in V(C^0)$, and thus cannot x-satisfy C^0.
So in the following we assume $3 \leq l < n$. Let $\alpha_0 := \frac{||C^0||}{|C^0|} = \frac{n \cdot l}{m}$ be the average

clause length of C^0. Let C^i be a subformula of C^0 with the average clause length α_i. Then we can always find a variable x such that the subformula $C^i(x) \subset C^i$ consisting only of all x-clauses has the average clause length $\geq \alpha_i$.
This is due to the following Theorem:

Theorem 33. *Let C be a l-regular* CNF *formula and let $\alpha = \frac{||C||}{|C|}$ be the average clause length of C. Then there is a variable $x \in V(C)$ such that $\alpha(x) \geq \alpha$, where $\alpha(x) = \frac{||C(x)||}{|C(x)|}$ and $C(x)$ is the subformula of C consisting of all clauses containing x.*

Before starting with the proof of Theorem 33 we first prove the following useful Lemma:

Lemma 8. *Let C be a* CNF *formula. Then*

$$|C| \cdot \sum_{c \in C} |c|^2 \geq ||C||^2 = \alpha^2 |C|^2$$

Here $\alpha = \frac{||C||}{|C|}$ is the average clause length of C and $||C|| = \sum_{c \in C} |c|$.

Proof. If $C = \emptyset$ then obviously the assertion of Lemma 8 is true. Now let $|C| \geq 1$, then for all $c \in C$ there is a $\delta_c \in \mathbb{Z}$ such that $|c| = \alpha + \delta_c$ and $\sum_{c \in C} \delta_c = 0$. Since $\delta_c^2 \geq 0$ and $2\alpha\delta_c \geq 0$ we obtain

$$\begin{aligned} |C| \cdot \sum_{c \in C} |c|^2 &= |C| \sum_{c \in C} (\alpha^2 + \delta_c^2 + 2\alpha\delta_c) \\ &\geq |C| \sum_{c \in C} \alpha^2 \\ &= |C|^2 \alpha^2 = ||C||^2 \end{aligned}$$

□

Applying Lemma 8 we are now able to prove Theorem 33.

Proof. Since C is l-regular, we get $|C(x)| = l$ and hence $\alpha(x) = \frac{||C(x)||}{l}$ for all $x \in V(C)$. Suppose there is a l-regular formula C such that for all $x \in V(C)$: $\alpha(x) < \alpha$. It follows that $\sum_{x \in V(C)} \alpha(x) < \alpha |V(C)|$ (*). Now it is

$$\alpha(x) = \frac{||C(x)||}{l} = \frac{\sum_{c \in C(x)} |c|}{l}$$

So we get

$$\begin{aligned}\sum_{x\in V(C)} \alpha(x) &= \sum_{x\in V(C)} \frac{\sum_{c\in C(x)} |c|}{l} \\ &= \frac{\sum_{x\in V(C)} \sum_{c\in C(x)} |c|}{l} \\ &= \frac{\sum_{c\in C} |c|^2}{l} \\ &\overset{L.8}{\geq} \frac{||C||^2}{l|C|} \\ &= \frac{||C||}{|C|}\frac{||C||}{l} \\ &= \alpha\cdot |V(C)|\end{aligned}$$

Hence we obtain $\sum_{x\in V(C)} \alpha(x) \geq \alpha\cdot |V(C)|$ which is a contradiction to (*). Therefore at least one variable $x \in V(C)$ with $\alpha(x) \geq \alpha$ must exist.

□

We consider a tree t whose number of leaves symbolises the number of backtracks and hence determines the running time of Algorithm AVRG. Let b_0 be the maximal number of branches in the 0th level of the tree and hence the maximal number of variables set to 0 in the formula C^0 such that the average clause length of C^0 is still > 2. Then we obtain for b_0:

$\alpha_0 - \frac{b_0\cdot l}{m} > 2 \Leftrightarrow b_0 < (\alpha_0 - 2)\frac{m}{l}$. That means if $b_0 \geq (\alpha_0 - 2)\frac{m}{l}$, then the average clause length of C^0 is ≤ 2. So the 0th level of t has at most $b_0 := (\alpha_0 - 2)\frac{m}{l}$ many branches. In the jth branch of level 0 exactly j variables are set to 0, for $j = 0, \ldots, b_0$. To calculate the number of branches of the 1st level which are adjacent to the ith node d_i^0 of the 0th level, for $i = 1, \ldots, b_0$, we need to know how many variables were set to 0 before in the 0th level along the path from the root down to d_i^0 . Let $x_0 \in \{1, \ldots, b_0\}$ denote the number of variables set to 0 in the 0th level. Then we define

$$\alpha_0'(x_0) = \alpha_0 - \frac{x_0\cdot l}{m}$$

and

$$\alpha_1(x_0) = \alpha_0'(x_0) - \frac{l(l-1)(\alpha'(x_0) - 1)}{m - l}$$

Note that $\alpha_0'(x_0)$ denotes the average clause length of C^0 after having set exactly x_0 many variables to 0, for $x_0 \in \{1, \ldots, b_0\}$. $\alpha_1(x_0)$ denotes the average clause length after having set x_0 many variables to 0, one variable y to 1 (which is altogether occurring in l many clauses of average clause length $\alpha_0'(x_0)$) and all other variables occurring in the y-clauses to 0. Now we are able to calculate the maximal number of branches which are adjacent to a node in the 0th level:

$$b_1(x_0) = (\alpha_1(x_0) - 2)\frac{m - l}{l}$$

So $b_1(x_0)$ is the maximal number of branches going out from a node in the 0th level where at first x_0 many variables are set to 0 in C^0, for $x_0 \in \{1, \ldots, b_0\}$. Next

we calculate the number of branches going out from an arbitrary node d^{i-1} of the $(i-1)$th level. Note that in order to do this we need to consider the path Pd^{i-1} from the root of the tree down to d^{i-1} and also need to know x_k, i.e. the number of variables which are set to 0 in the kth level of Pd^{i-1}, for all $k = 0, 1, \ldots, i-1$. We obtain inductively:

$$b_i(x_0, x_1, \ldots, x_{i-1}) = (\alpha_i(x_0, x_1, \ldots, x_{i-1}) - 2)\frac{m - i \cdot l}{l}$$

where

$$\alpha_i(x_0, x_1, \ldots, x_{i-1}) = \alpha'_{i-1}(x_0, x_1, \ldots x_{i-1}) - \frac{l(l-1)(\alpha'_{i-1}(x_0, x_1, \ldots x_{i-1}) - 1)}{m - i \cdot l}$$

$\alpha_i(x_0, x_1, \ldots, x_{i-1})$ is the average clause length of the formula, where in the kth level exactly x_k many variables are set to 0 before having set a variable to 1, for $k = 0, 1, \ldots, i-1$.

$$\alpha'_{i-1}(x_0, x_1, \ldots x_{i-1}) = \alpha_{i-1}(x_0, x_1, \ldots x_{i-2}) - \frac{x_{i-1} \cdot l}{m - (i-1) \cdot l}$$

$\alpha'_{i-1}(x_0, x_1, \ldots x_{i-1})$ is the average clause length of the formula, where in the kth level exactly x_k many variables are set to 0 before setting a variable to 1, for $k = 0, 1, \ldots, i-2$, and in the $(i-1)$th level x_{i-1} many variables are set to 0 (and no variable has been set to 1 here yet).

Let $N(t)$ be the number of leaves in t then $O(N(t))$ is the running time of Algorithm AVRG. Using a computer implementation which calculates the number of leaves in t for different inputs of α_0, l and n we compare $N(t)$ with $2^{0.2325n}$ (see Appendix, Experimental Results) and obtain that Algorithm AVRG asymptotically performs much better for the class LCNF^l_+ than the so far best algorithm from J. M. Byskov et al. for CNF-XSAT with a running time of $O(2^{0.2325n})$ [12]. One can also easily verify that the algorithm from J. M. Byskov et al. does not perform better for the class LCNF^l_+ than in $O(2^{0.2325n})$. So Algorithm AVRG has the so far best running time for LCNF^l_+ concerning XSAT.

Correctness of the algorithm AVRG: When setting a variable $y \in V(F)$ of a $LCNF^l_+$ formula to 1, the algorithm immediately sets all other variables in the y-clauses to 0 and hence x-satisfies all y-clauses. After evaluating the current formula we obtain a formula with l clauses less and the average clause length has diminished. Repeating this process until we obtain a formula with an average length ≤ 2 we achieve that we can solve XSAT in polynomial-time for the remaining formula. When obtaining either a contradiction or *not x-satisfiable*, we go a step back and reverse the assignment of the last step. If we cannot go back anymore and no x-model has been found yet,the formula obviously is not x-satisfiable.

The following theorem summarises our results:

Theorem 34. *Algorithm AVRG has the so far best running time for* LCNF^l_+ *concerning* XSAT.

3.5 The Exact Linear Case

This section is devoted to a certain subclass of linear formulas, namely the exact linear ones. Such formulas, as defined earlier, have the property to be linear, and in addition the variable sets of each two distinct clauses have *exactly* one variable in common. Exact linear formulas are quite small instances since the number of clauses never exceeds the number of variables [38, 42]. In [42] it is shown that SAT restricted to XLCNF is polynomial time solvable. In this section we show that this also holds for NAE-SAT. To that end, we will use a result for the satisfiability problem restricted to a certain CNF subclass published recently:

Theorem 35. *[Theorem 3 in [44]] For* $C \in$ CNF*, such that the variable sets of each pair of clauses have exactly one or all members in common,* SAT *and moreover* #SAT *can be decided, respectively solved, in polynomial-time.*

Here as usual #Π denotes the counting version of a decision problem Π, searching for the number of solutions that a given instance of Π has. On the basis of the last theorem, we obtain:

Corollary 4. NAE-SAT *restricted to exact linear formulas is polynomial-time solvable and moreover* #NAE-SAT *is polynomial-time solvable.*

Proof. Given an arbitrary CNF formula C, it is easy to see that $C \in$ NAE-SAT holds true if, and only if, $C \cup C^\gamma \in$ SAT, where C^γ denotes the formula obtained from C by complementing the literals of each clause in C. Now let $C \in$ XLCNF be arbitrarily chosen, then $C \cup C^\gamma$ is a formula such that *each pair* of clauses has exactly one or all members in common. So, according to Theorem 35, we derive that NAE-SAT is polynomial-time decidable (and solvable) for exact linear formulas. Moreover, it is easy to see that each fixed nae-model of C gives rise to a unique SAT model of $C \cup C^\gamma$ and vice versa. Hence the corresponding model spaces are in 1-to-1-correspondence implying that #NAE-SAT can be solved in polynomial-time using of Theorem 35. □

In 1996, T. Eiter [20] mentioned a problem called *symmetrical intersecting monotone UNSAT (SIM-UNSAT)*, which is computationally equivalent to a problem called IM-UNSAT that in turn forms the hard core of several interesting combinatorial problems arising in different areas. In [20] Eiter asks the question concerning the computational complexity of SIM-UNSAT (resp. IM-UNSAT) which has been open for 15 years, already in 1996. As far as we know this question has not been answered so far. Instances of SIM-UNSAT have the form $C \cup C^\gamma$, where C is a set of pairwise intersecting *monotone* clauses. The task is to decide whether such an instance is unsatisfiable. Observe that according the proof above, solving NAE-SAT for exact linear formulas in polynomial-time means to solve SIM-UNSAT in polynomial time restricted to the special case where the monotone clauses intersect pairwise in exactly one variable. In that way, we have given a partial answer to Eiter's problem, however, the general case clearly remains open.

3.5.1 XSAT for Exact Linear Formula Classes

This section is mainly devoted to considering XSAT for exact linear formulas. Besides proving the NP-completeness of XLCNF-XSAT we also provide polynomial-

time subclasses. The next theorem describes a quite interesting result und its proof is really complex.

Theorem 36. XSAT *remains* NP*-complete for* XLCNF_+ *and* XLCNF*.*

Proof. We give a polynomial-time reduction from LCNF_+-XSAT to XLCNF_+-XSAT. Let $C = c_1 \wedge c_2 \wedge \ldots \wedge c_m \in \text{LCNF}_+$ be an arbitrary formula that is not exact linear, otherwise we are done. As long as there is a pair of clauses $c_i, c_j \in C$, $i, j \in \{1, \ldots, m\}$ which do not share a variable, introduce a new variable z that does not occur in the current formula and enlarge both c_i and c_j with the variable z. The resulting formula C' obviously is exact linear. Let Z denote the collection of all newly introduced variables this way. Next, we add at least $m + 1$ further clauses collected in D whereas C' is modified to $\tilde{C}'$ so that the resulting formula $C'' := \tilde{C}' \wedge D$ stays exact linear and becomes XSAT-equivalent with C.
The construction of D and the modification of C' proceeds hand in hand: Initially, D is empty. As long as there is a variable $z_i \in Z$ not occurring in any clause of D, add a new clause d to D containing z_i and a new distinguished variable u (which is required to be contained in each clause of D). For each clause c_i of the current formula C' such that $V(d) \cap V(c_i) = \varnothing$ introduce a new variable $w_{d,i}$ and add it to d and c_i. Let W denote the collection of all these newly introduced variables..
When all variables in Z occur in D, but D still contains less than $m + 1$ clauses, add sufficiently many new clauses to D each containing u. Each such new clause e is filled-up by m new variables $y_{e,1}, \ldots, y_{e,m}$ such that $y_{e,r}$ is added to W and to the rth clause of the current formula C', $1 \leq r \leq m$. Finally, all newly introduced variables in $Z \cup W$ occur in D and in the final version $\tilde{C}'$ of C' and the formula is exact linear.
Let C be x-satisfiable with x-model t. Obviously, t can be extended to all variables of C'' by setting all newly introduced variables of $W \cup Z$ to 0 and $u = 1$. This yields a x-model for C''. Conversely, let C be x-unsatisfiable, and assume that C'' is x-satisfiable. Then $\tilde{C}'$ can only become x-satisfiable by setting at least one variable $x \in Z \cup W$ to 1. As each variable of $Z \cup W$ also occurs in D, there must be a clause $d_i \in D$ with $x \in d_i$. Hence $u = 0$ in d_i, and thus, to x-satisfy D, there must be exactly one variable from $Z \cup W$ set to 1 in each of its clauses. As D has at least $m + 1$ clauses, at least $m + 1$ distinct variables from $Z \cup W$ must be set to 1 in D. Since all these variables occur in $\tilde{C}'$, but $\tilde{C}'$ has exactly m clauses, the pigeonhole principle implies that there is a clause in $\tilde{C}'$ containing at least two variables set to 1. This yields a contradiction, hence C'' is x-unsatisfiable, too.
To illustrate this reduction, consider the input formula

$$C = (x_1 \vee x_2 \vee x_3) \wedge (x_4 \vee x_5 \vee x_6) \wedge (x_1 \vee x_7 \vee x_8) \in \text{LCNF}_+$$

At first we obtain C' by making the clauses of C exact linear introducing the new variables $Z = \{z_1, z_2\}$:

$$\begin{aligned} C' = &(x_1 \vee x_2 \vee x_3 \vee z_1) \\ &\wedge (x_4 \vee x_5 \vee x_6 \vee z_1 \vee z_2) \\ &\wedge (x_1 \vee x_7 \vee x_8 \vee z_2) \end{aligned}$$

Next we add clauses $D = \{d_1, d_2\}$ each containing a fixed variable u so that all variables in Z occur in the new clauses. To preserve the exact linearity we need to

introduce new variables $W = \{w_1, w_2\}$:

$$\begin{aligned}
&(x_1 \vee x_2 \vee x_3 \vee z_1 \vee w_2)\\
\wedge&(x_4 \vee x_5 \vee x_6 \vee z_1 \vee z_2)\\
\wedge&(x_1 \vee x_7 \vee x_8 \vee z_2 \vee w_1)\\
\wedge&\underbrace{(u \vee z_1 \vee w_1)}_{=:d_1}\\
\wedge&\underbrace{(u \vee z_2 \vee w_2)}_{=:d_2}
\end{aligned}$$

In this example D has only two clauses, so we have to add two more clauses d_3, d_4 to ensure XSAT-equivalence and preserve exact linearity, finally yielding $W = \{w_1, w_2, y_1, \ldots, y_6\}$, and:

$$\begin{aligned}
C'' =&(x_1 \vee x_2 \vee x_3 \vee z_1 \vee w_2 \vee y_1 \vee y_4)\\
&\wedge(x_4 \vee x_5 \vee x_6 \vee z_1 \vee z_2 \vee y_2 \vee y_5)\\
&\wedge(x_1 \vee x_7 \vee x_8 \vee z_2 \vee w_1 \vee y_3 \vee y_6)\\
&\wedge(u \vee z_1 \vee w_1)\\
&\wedge(u \vee z_2 \vee w_2)\\
&\wedge\underbrace{(u \vee y_1 \vee y_2 \vee y_3)}_{=:d_3}\\
&\wedge\underbrace{(u \vee y_4 \vee y_5 \vee y_6)}_{=:d_4} \quad \in \text{XLCNF}_+
\end{aligned}$$

□

It is not hard to see that the result above sharpens the long-standing NP-hardness result for clique packing of a graph maximizing the number of covered edges of Hell and Kirkpatrick [24]. Recently Chataigner et al. have provided significant approximation (hardness) results regarding the clique packing problem [13]. It could be interesting to investigate in the future whether similar approximation results can be gained for XSAT on (X)LCNF.

Next we are interested in XSAT for $(\geq k)$-XLCNF$_+$, with $k \geq 3$. To prove its NP-completeness, we need to consider the class $(\geq |C|)$-LCNF$_+$ consisting of all monotone and linear formulas C such that each clause has at least length $|C|$.

Lemma 9. *Every formula C in $(\geq |C|)$-LCNF$_+$ is x-satisfiable.*

Proof. Let C be a formula in $(\geq |C|)$-LCNF$_+$ with $m := |C|$ clauses and assume there is a clause $c_0 \in C$ containing at least the variables $x_1, \ldots, x_m$ such that $w_C(x_i) \geq 2$, for $1 \leq i \leq m$. Due to linearity this implies that there are clauses c_i, $1 \leq i \leq m$, such that $x_i \in V(c_i)$, thus $|C| \geq |\{c_0, c_1, \ldots, c_m\}| \geq m+1$ yielding a contradiction. It follows that each clause c of C contains at least one literal which occurs only once in C. Hence, setting exactly these variables to 1 x-satisfies C. □

Let $(k, |C|-1)$-LCNF$_+$, $k \geq 3$, denote the class of all monotone and linear formulas C such that each clause has at least length k and at most length $|C|-1$.

According to Theorem 4 in [39] XSAT is NP-complete for $(\geq k)$-LCNF$_+$, for each fixed $k \geq 3$. According to Lemma 9 $(\geq |C|)$-LCNF$_+$ behaves trivially for XSAT. Since XSAT is NP-complete for k-LCNF$_+$ and k-LCNF$_+ \subseteq (k, |C|-1)$-LCNF$_+$, it follows that XSAT is NP-complete for $(k, |C|-1)$-LCNF$_+$, too.

We were not able to establish the NP-completeness for k-XLCNF$_+$, i.e. uniform formulas, regarding XSAT. However, on behalf of the NP-completeness of $(k, |C|-1)$-LCNF$_+$-XSAT just shown, we can provide the next result by using the same technique as in the proof of Theorem 36 starting with a formula in $(k, |C|-1)$-LCNF$_+$.

Theorem 37. XSAT *remains* NP*-complete for* $(\geq k)$-XLCNF$_+$*, for each* $k \geq 3$.

Proof. We provide a polynomial-time reduction from $(k, |C|-1)$-LCNF$_+$-XSAT to $(\geq k)-$XLCNF$_+$-XSAT establishing NP-completeness of the latter. For this purpose let $C \in (k, |C|-1)$-LCNF$_+$ be an arbitrary input formula: $C = c_1 \wedge c_2 \wedge \ldots \wedge c_{|C|}$ and $k \leq |c_i| \leq |C|-1$ for all $i \in \{1, \ldots, |C|\}$. Hence $k \leq |C|-1$. We procced as follows: If C is already exact linear, we are done. Otherwise, as long as there is a pair of clauses $c_i, c_j \in C$, $i, j \in \{1, \ldots, |C|\}$ which do not share a variable, we introduce a new variable $z_i \notin V(C)$ not occurring in the formula yet and enlarge the clauses c_i and c_j with the variable z_i. Let $C^{'}$ be the resulting formula. Obviously C' is exact linear and each clause in C' has length $\geq k$. Let $Z = \{z_1, \ldots, z_p\} = V(C') - V(C)$ be the set of all newly introduced variables. We obtain: Each $z_i \in Z$ occurs exactly twice in C'. To ensure XSAT-equivalent of C and C' we add further clauses to C', which provide the XSAT-equivalence of C': We add at least $|C|+1$ clauses to C' and denote this new set of clauses with $D = \{d_1, \ldots, d_q\}$, where $q \geq |C|+1$. A detailed construction of such a clause set D is presented in the proof of Theorem 36. As d_i shares exactly one variable with each clause of C' we obtain $|d_i| = |C|+1$, for each clause d_i, $i \in \{1, \ldots, q\}$. Thus we have $|d_i| = |C|+1 > |C|-1 \geq k$. Furthermore, $u \in V(d_i)$, for all $i \in \{1, \ldots, q\}$, where u is a variable not yet occurring in C', so the newly introduced clauses are exact linear. Hence $C'' := C' \wedge d_1 \wedge d_2 \wedge \ldots \wedge d_q$ is positive monotone, exact linear and XSAT equivalent to C as shown in the proof of Theorem 36. Further, as each clause of C has at least length k and $|d_i| \geq k$, C'' belongs to the class $(\geq k)-$XLCNF$_+$. □

3.5.2 XSAT for k-XLCNF$_+$

In this section we consider the class k-XLCNF$_+$ of k-uniform, exact linear and positive monotone CNF formulas, for $k \in \mathbb{N}, k \geq 2$. Notice that for $l > k$ the class k-XLCNF$_+^l$ is empty because of exact linearity and k-uniformity:
Suppose there is a k-uniform, l-regular, exact linear and positive monotone CNF formula C which is not an empty formula for $l > k$. Then there is a variable $x \in V(C)$ which occurs in exactly l clauses $c_1, \ldots, c_l$. The formula consisting only of the clauses $c_1, \ldots, c_l$ is not l-regular; to make it l-regular we have to add further clauses. But we cannot add a further clause c_{l+1} such that $c_1, \ldots, c_l, c_{l+1}$ are exact linear and k-uniform, because c_{l+1} must share a variable with each of the clauses $c_1, \ldots, c_l$ and hence c_{l+1} would have a length of at least l, where $l > k$ which is a contradiction to the k-uniformity of C. Therefore the class k-XLCNF$_+^l$, for $l > k$, is empty.

As mentioned above, there are some difficulties in establishing the complexity of XSAT for arbitrary k-uniform, exact linear formulas. We were not able to establish the complexity of XSAT for these formulas. Instead we present the polynomial-time solvability of XSAT for the k-uniform subclasses k-XLCNF$_+$, where $k \leq 6$. For that purpose we introduce several lemmas.

Lemma 10. *Let $C \in k$-XLCNF$_+$. If there is a variable $x \in V(C)$ with $w(x) > k$, then $x \in V(c)$, for all $c \in C$.*

Proof. Suppose there is a clause $c_i \in C$ that does not contain x. Then c_i must share exactly one variable with every clause containing x. There are more than k many clauses containing x, but c_i is only k-uniform. Hence C has only clauses which contain x. □

Lemma 11. *The class k-XLCNF$_+^k$ is not x-satisfiable.*

Proof. Let $C \in k$-XLCNF$_+^k$. Then C is a k-block formula according to Lemma 20 in [39] and thus not x-satisfiable according to Lemma 3 in [39]. □

Lemma 12. *Let $C \in k$-XLCNF$_+$ and let $x \in V(C)$ be a variable with $w(x) = k-1$ in C. Then C is x-satisfiable.*

Proof. Let $C \in k$-XLCNF$_+$ and $x \in V(C)$ with $w(x) = k-1$. Let $c_1, \ldots, c_{k-1}$ be the clauses containing x. We set $x = 1$ in $c_1, \ldots, c_{k-1}$ and assign 0 to the other variables in these clauses. This way we x-satisfy the clauses $c_1, \ldots, c_{k-1}$ and remove them from the formula C. Now we consider the remaining clauses $c_j \in C - \{c_1, \ldots, c_{k-1}\}$, which satisfy $V(c_j) \cap (V(c_i) - \{x\}) \neq \emptyset$, for all $i = 1, \ldots, k-1$. Hence each of the remaining clauses contains $k-1$ distinct variables already assigned to 0. When we remove these from all of the remaining clauses $c_j \in C - \{c_1, \ldots, c_{k-1}\}$ the remaining formula consists of unit clauses only, and thus C is x-satisfiable. □

Lemma 13. *Let $C \in k$-XLCNF$_+$ be a k-uniform, exact linear and positive monotone* CNF *formula and let $c \in C$ be a clause of C with $w(x) = k$ for all $x \in V(c)$. Then C is not x-satisfiable.*

Proof. Let $c = (x_1 \vee x_2 \vee x_3 \vee \ldots \vee x_k) \in C$ with $w(x_i) = k$, for all $i = 1, \ldots, k$. We claim C has exactly $1 + k(k-1)$ many clauses. Indeed, because of the exact linearity of C every other clause of C must have exactly one variable with c in common. Since each of the variables $x_1, \ldots, x_k$ occurs exactly k times, the formula C must have the following structure: C only consists of the x_i-clauses, for a $i \in \{1, \ldots, k\}$, i.e. there is no clause in C which does not contain the variable x_i, for a $i \in \{1, \ldots, k\}$. Each x_i-clause occurs exactly k times. Let n be the number of variables occurring in C. Altogether there are k many x_1-clauses containing $1+k(k-1)$ different variables as a reason of linearity. Hence we get $n \geq 1+k(k-1)$. Suppose $n > 1 + k(k-1)$. Then there exists a variable y in C not occurring in any of the x_1-clauses. As each clause of C must share exactly one variable with c, an x_j-clause c' must exist, for a $j \in \{2, \ldots, k\}$, containing y. Now c' has two literals, namely x_j, y and thus only $k-2$ free positions left. But there are $k-1$ further x_1-clauses with which c' must share exactly one variable. This is a contradiction.

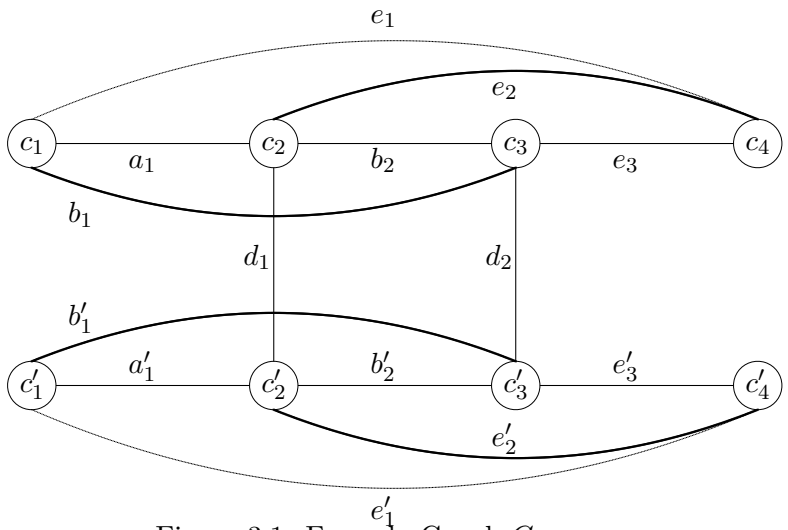


Figure 3.1: Formula Graph G_C

Hence $n = 1 + k(k-1) = |C|$ and according to Lemma 20 in [41] C is a k-block formula und thus not x-satisfiable. □

Lemma 14. *Let $C \in$ CNF with $w(x) \leq 2$ for all $x \in V(C)$. Then we can decide* XSAT *for C in polynomial-time.*

Proof. We construct a formula graph G_C, such that C is x-satisfiable if, and only if, G_C has a perfect matching. Concerning this construction, we distinguish two cases:

1. There is no clause in C containing a unique variable, then G_C coincides with the intersection graph of C. Usually the intersection graph is defined as follows: The vertices of G_C are the clauses of C: $c_i \in V(G_C) \Leftrightarrow c_i \in C$. We connect two clause-vertices $c_i,\ c_j$ by an edge if they share a variable. Each edge is labelled with the corresponding intersection variable.
2. C contains unique variables. We make two copies of the intersection graph and for each unique variable x we join the vertices corresponding to the clauses containing x in either copy.

It is easy to see that C is x-satisfiable if, and only if, G_C has a perfect matching. It is obvious that this formula graph can be constructed in polynomial-time and it is well known that a perfect matching can be found in polynomial-time. This construction is illustrated in the following example:
Let $C = \{c_1, c_2, c_3, c_4\}$ be a positive linear matching formula with clauses

$$c_1 = \{a_1, b_1, e_1\},\ c_2 = \{a_1, b_2, e_2, d_1\},\ c_3 = \{b_1, b_2, e_3, d_2\},\ c_4 = \{e_1, e_2, e_3\}$$

The graph in Figure 3.1 corresponds to the formula C where the vertices of the copy of the intersection graph are marked by $'$. A perfect matching of G_C as well as the x-model of C is $\{b_1, e_2\}$. □

Theorem 38. *Let $C \in k$-XLCNF$_+$ with $w(x) = 2$ for all $x \in V(C)$. If k is even, C is not x-satisfiable. If k is odd, C is x-satisfiable.*

PROOF. We consider the intersection graph G_C whose vertices are the clauses of C. There is an edge between two clause vertices c_i, c_j if they have a variable x_l in common. We write e_{x_l} for the edge between c_i and c_j. Since C is exact linear, i.e. each two clauses share a variable, our clause-graph is a complete graph. Now the idea is that C is x-satisfiable if, and only if, G_C has a perfect matching. Let m be the number of clauses of C then G_C has m vertices and resulting from exact satisfiability $m = 2 + (k-1)$.

- If k is even, m is odd. Clearly a graph with an odd number of vertices cannot have a perfect matching, thus we cannot x-satisfy C.
- Let k be odd, then m is even. Evidently, a complete graph with an even number of vertices admits a perfect matching. Let M be the perfect matching of G_C. For each $e_{x_i} \in M$ we set $x_i = 1$ and assign 0 to all the other variables not belonging to the edges in M.

□

Lemma 15. *Let $C \in k$-XLCNF$_+$ containing a clause $c = (x_1 \vee x_2 \vee \ldots \vee x_k) \in C$ such that $w(x_1) = w(x_2) = \ldots = w(x_k) = k-2$. Then we can decide* XSAT *for C in polynomial-time.*

PROOF. Let $C \in k$-XLCNF$_+$ and $c_1 = (x_1 \vee x_2 \vee \ldots \vee x_k) \in C$ with

$$w(x_1) = w(x_2) = \ldots = w(x_k) = k-2$$

XSAT-evaluating C according to the setting $x_i = 1$, for any fixed $i \in \{1, \ldots, k\}$, yields a formula $C[x_i]$ in 2-LCNF$_+$ because of exact linearity and k-uniformity. Therefore XSAT for $C[x_i]$ can be decided in linear-time [42]. Hence, in the worst-case we have to check every such formula $C[x_i]$, $1 \leq i \leq k$, yielding a polynomial-time worst-case running time of $O(k \cdot ||C||)$. □

Lemma 16. *Let $C \in k$-XLCNF$_+$ with at least $k-3$, $k \geq 4$, variables occurring exactly k times in C and*

$$k + (k-1)(k-4) < |C| < k + (k-1)(k-1)$$

Then we can decide XSAT *for C in polynomial-time.*

PROOF. Let $x_1, x_2 \ldots, x_{k-3}$ be $(k-3)$ distinct variables each occurring k times in C and let w.l.o.g. $c_1 = (x_1 \vee x_2 \vee \ldots \vee x_{k-3} \vee x_{k-2} \vee x_{k-1} \vee x_k)$. It is left to the reader to verify that because of exact linearity such a clause must exist. As C has more than $k + (k-1)(k-4)$ many clauses we obtain $w(x) \geq k-3$ for all $x \in V(C) - \{x_{k-2}, x_{k-1}, x_k\}$. Especially as C has more than k many clauses, we cannot set $x_i = 1$ for a $i = 1, \ldots, k-3$ because then in one of the clauses of C all the variables would be set to 0. Hence we set $x_i = 0$ for all $i = 1, \ldots, k-3$. As C has more than $k+(k-1)(k-4)$ many clauses at least one clause $c_p = (x_j \vee y_2 \vee \ldots \vee y_k)$, $j \in \{k-2, k-1, k\}$, exists in C which is not a x_i-clause, for a $i = 1, \ldots, k-3$. We

obtain for c_p that each variable $y_i \in V(c_p) - \{x_j\}$ occurs at least $k-2$ times in C. Setting y_i to 1, for a $i \in \{2, \ldots, k\}$, and all other variables in the y_i-clauses to 0 yields a formula with a clause length ≤ 2 which we can solve in polynomial-time. If $y_i = 1$, for all $i \in \{2, \ldots, k\}$, does not yield an x-model for C, we consider, if existent, another x_l-clause $c_q = (x_l \vee z_2 \vee \ldots \vee z_k)$ for a $l \in \{k-2, k-1, k\}, l \neq j$. As every variable $z_i \in V(c_q) - \{x_l\}$ occurs at least $k-2$ times in C, setting $z_i = 1$, for a $i \in \{2, \ldots, k\}$, and all the other variables in the z_i-clauses to 0 yields a formula with a clause length ≤ 2 for which we can solve XSAT in polynomial-time. If again for all $i \in \{2, \ldots, k\}$ $z_i = 1$ does not yield an x-model for C, then C is not x-satisfiable. This is because if we assign $x_j = x_l = 1$, then c_1 will contain two variables which have to be set to 1 which is a contradiction. If there is no such x_l-clause for a $l \in \{k-2, k-1, k\}, l \neq j$, then we set $x_j = 1$ and all the other variables in the x_j-clauses, especially in c_1, to 0. Other variables from the x_j-clauses we set to 0 and delete these clauses. So we have $(k-1)(k-3)$ clauses left; let C' be the remaining formula. Setting a variable $a \in V(C')$ to 1 (which is not marked), yields $(k-2)(k-3)(k-4)$ literals which are set to 0 in C' as a consequence because each variable occurs exactly $k-3$ times in C' and the clause length is $k-1$. We have to distribute these $(k-2)(k-3)(k-4)$ literals onto $(k-3)(k-1) - (k-3) = (k-3)(k-2)$ clauses which yields exactly $k-4$ literals in average case that are set to 0 in each clause of C'. That means in the remaining formula C'' each clause has an average length of $(k-1) - (k-4) = 3$ literals. Now setting an arbitrary variable which is not marked in C'' to 1, yields $2 \cdot (k-3)(k-4)$ literals which have to be set to 0 as a consequence and C'' has only $(k-1)(k-3) - 2(k-3) = (k-3)^2$ clauses left. Hence the average clause length is then $3 - \frac{2 \cdot (k-3)(k-4)}{(k-3)^2} = 3 - \frac{2 \cdot (k-4)}{(k-3)} \leq 2$ and this formula can be solved in polynomial-time. □

Let $k \in \mathbb{N}$ such that a k-block formula B_k exists and let $C \in k\text{-XLCNF}_+$ be a formula which is not a k-block formula. Then C can be *embedded* into a k-block formula, if we can expand C by adding further clauses such that C becomes a k-block formula, that is $|C| = k + (k-1)^2$. This is equivalent to the so called *Church-Rosser property.*
A k-block formula B_k is not x-satisfiable but removing an arbitrary clause from B_k yields a x-satisfiable formula $F \subset B_k$. By applying the following theorem we can conclude that not every $C \in k\text{-XLCNF}_+$, which is not a k-block formula (for $k \in \mathbb{N}$ for which a k-block formula B_k exists) is x-satisfiable because it cannot always be embedded into a k-block formula.

Theorem 39. *Let $k \in \mathbb{N}$ be such that a k-block formula exists and let $C \in k\text{-XLCNF}_+$ such that $w(x) \leq k$ for all $x \in V(C)$. Then C in general cannot be embedded into a k-block formula. In other words generating k-block formulas does not have the Church-Rosser property.*

Proof. Consider the following counterexample for $k = 5$:

$$
\begin{aligned}
C = &(x_1 \vee x_2 \vee x_3 \vee x_4 \vee x_5)\\
&(x_1 \vee x_6 \vee x_7 \vee x_8 \vee x_9)\\
&(x_1 \vee x_{10} \vee x_{11} \vee x_{12} \vee x_{13})\\
&(x_1 \vee x_{14} \vee x_{15} \vee x_{16} \vee x_{17})\\
&(x_1 \vee x_{18} \vee x_{19} \vee x_{20} \vee x_{21})\\
&\\
&(x_2 \vee x_6 \vee x_{10} \vee x_{14} \vee x_{18})\\
&(x_2 \vee x_7 \vee x_{11} \vee x_{15} \vee x_{19})\\
&(x_2 \vee x_8 \vee x_{12} \vee x_{16} \vee x_{20})\\
&(x_2 \vee x_9 \vee x_{13} \vee x_{17} \vee x_{21})\\
&\\
&(x_3 \vee x_6 \vee x_{11} \vee x_{16} \vee x_{21})\\
&(x_3 \vee x_7 \vee x_{10} \vee x_{17} \vee x_{20})\\
&(x_3 \vee x_8 \vee x_{13} \vee x_{15} \vee x_{18})\\
&(x_3 \vee x_9 \vee x_{12} \vee x_{14} \vee x_{19})
\end{aligned}
$$

Obviously C is not a 5-block formula and satisfies $w(x) \leq 5$, for all $x \in V(C)$. But we cannot add another 5-uniform clause c to C such that $C \cup \{c\}$ remains exact linear. On the other hand recall that a 5-block formula exists [43]. □

Consequently we are ready to establish:

Theorem 40. *The classes k-XLCNF$_+$, for $k \in \{1, 2, 3, 4, 5, 6\}$, can be x-solved in polynomial-time.*

Proof. Let $C \in k$-XLCNF$_+$ be arbitrarily chosen. We set $w_C(x) := w(x)$ since C is fixed, and provide a case analysis guided by k:

1. $k = 3$:

   - If there is a variable $x \in V(C)$ with $w(x) \geq 4$, then $x \in c$, for all $c \in C$, according to Lemma 10. In this case we set $x = 1$ and assign 0 to all other variables in $V(C)$ obtaining an x-model for C.
   - If $w(x) = 3$, for all $x \in V(C)$, then C is not x-satisfiable according to Lemma 11.
   - If there is a variable $x \in V(C)$ with $w(x) = 3$ (but there is at least one variable $y \in V(C)$ which does not satisfy $w(y) = 3$) then we set all variables occurring 3 times in C to 0, because setting such a variable to 1 would yield a clause where all literals are set to 0, in case C has more than 3 clauses. Let C' be the resulting formula, then each variable occurs ≤ 2 times in C' and we can solve such a formula in polynomial-time according to Lemma 14.

- If $w(x) \leq 2$, for all $x \in V(C)$, then we can decide XSAT for C in polynomial-time according to Lemma 14.

2. $k = 4$:

   - If there is a variable $x \in V(C)$ with $w(x) \geq 5$, then $x \in c$, for all $c \in C$, according to Lemma 10. In this case we set $x = 1$ and assign 0 to all other variables in $V(C)$ obtaining a x-model for C.
   - If $w(x) = 4$, for all $x \in V(C)$, C is not x-satisfiable according to Lemma 11.
   - If there is at least one variable $x \in V(C)$ with $w(x) = 4$ but there is also at least one variable $y \in V(C)$ which does not satisfy $w(y) = 4$), we proceed as follows:
     If C has no other clauses except the ones containing x, we set $x = 1$ and all other variables of $V(C)$ to 0 and obtain an x-model for C. Else we consider a clause $c \in C$ that is not an x-clause.
     Let $c = (x_{i_1} \vee x_{i_2} \vee x_{i_3} \vee x_{i_4})$ be such a clause. Then each variable of c occurs at least twice in C because c must share a variable with each of the four x-clauses and c is 4-uniform. Setting $x_{i_j} = 1$, for one $j \in \{1, 2, 3, 4\}$, yields a remaining formula where each clause has length ≤ 2 and for such a formula XSAT can be solved in polynomial-time.
   - If there is a variable $x \in V(C)$ with $w(x) = 3$, C is x-satisfiable according to Theorem 12.
   - If $w(x) \leq 2$, for all $x \in V(C)$, then we can decide XSAT for C in polynomial-time according to Lemma 14.

3. $k = 5$:

   - If there is a variable $x \in V(C)$ with $w(x) \geq 6$, then $x \in c$, for all $c \in C$, according to Lemma 10. In this case we set $x = 1$ and assign 0 to all other variables in $V(C)$ obtaining a x-model for C.
   - If $w(x) = 5$, for all $x \in V(C)$, then C is not x-satisfiable according to Lemma 11.
   - If there is a variable $x \in V(C)$ with $w(x) = 4$, then C is x-satisfiable according to Theorem 12.
   - As long as a variable x exists in $V(C)$ occurring ≥ 3 times, we set $x = 1$ and the other variables in all x-clauses to 0. Because of the exact linearity the clause length of the resulting formula C' reduces to ≤ 2 and we can solve such a formula in polynomial-time. If C' is not x-satisfiable, we undo the last assignment so that all variables of C are unassigned again and consider another variable occurring ≥ 3 times in C which we set to 1 (and the other variables in the same clauses to 0) and check whether this setting x-satisfies C. After we have considered all the variables occurring ≥ 3 times in C and have found out that assigning 1 to any of them does not yield an x-model for C, we set all the variables occurring ≥ 3 times in C to 0 and hence obtain a formula C' where each variable occurs ≤ 2 times. Then we can solve XSAT for C' in polynomial-time according to Lemma 14.

- If $w(x) \leq 2$, for all $x \in V(C)$, we can decide XSAT for C in polynomial-time according to Lemma 14.

4. $k = 6$: Let $C \in$ 6-XLCNF$_+$ be an arbitrary formula with variable set $V(C)$. We provide a case analysis guided by the number of occurrences of variables in $V(C)$.

   - If there is a variable $x \in V(C)$ with $w(x) \geq 7$, then $x \in c$, for all $c \in C$, according to Lemma 10. In this case we set $x = 1$ and assign 0 to all the other variables in $V(C)$. This way we get an x-model for C.
   - If $w(x) = 6$, for all $x \in V(C)$, then C is x-unsatisfiable according to Lemma 11.
   - If there is a variable $x \in V(C)$ with $w(x) = 5$, then C is x-satisfiable according to Lemma 12.
   - If $w(x) = 4$, for all $x \in V(C)$, then we can decide XSAT for C in polynomial-time according to Lemma 15.
   - If there is a variable $x \in V(C)$ with $w(x) = 6$ as well as a variable $y \in V(C)$ with $w(y) \neq 6$ and C only consists of clauses containing x, then we set $x = 1$ and all other variables in $V(C)$ to 0 obtaining an x-model for C.
   - If there is a variable $x \in V(C)$ with $w(x) \geq 4$, then setting x to 1, and all other variables in the clauses containing x to 0 yields a formula only containing clauses of length ≤ 2. Such a formula can be checked for XSAT in polynomial-time. This way we treat each variable $z \in V(C)$ with $w(z) \geq 4$ until an x-model is found. In case we have not found an x-model yet, we proceed as follows:
     (a) If there is no variable $x \in V(C)$ with $w(x) = 3$, we set all the variables x with $w(x) \geq 4$ to 0. Hence the resulting formula C' contains only variables occurring ≤ 2 times in C' and by using Lemma 14 we can solve C' in polynomial time.
     (b) If there is a variable $x \in V(C)$ with $w(x) = 3$, then after having set x to 1 and all other variables to 0 in the clauses containing x, we obtain a formula C' which is in 3-LCNF$_+$. If there is no further variable occurring three times in C', we set all variables occurring ≥ 4 times to 0 and can decide x-satisfiability of C' in polynomial-time according to Lemma 14. Otherwise, there is another variable y with $w(y) = 3$ in C'. Then we set $y = 1$ and to 0 all other variables in the clauses containing y. Now all y-clauses are x-satisfied and there are at most six clauses in the remaining formula that do not share any variable with any of the clauses containing y. Hence all clauses, except for at most six, do share at least one variable with one clauses containing y. Since these variables are all set to 0, the remaining formula only contains clauses of length two at most (except for at most six clauses) which can be decided for XSAT in polynomial-time.
   - If $w(x) \leq 3$, for all $x \in V(C)$, and there is a variable $x \in V(C)$ with $w(x) = 3$ then we proceed as follow. After having set x to 1 and all other variables in the clauses containing x to 0, the remaining formula

C' is in 3-LCNF$_+$. If all variables occur at most twice in C', then we can decide x-satisfiability of C' in polynomial-time according to Lemma 14. Otherwise there is still a variable y with $w(y) = 3$ in C'. In that case we set $y = 1$ and all variables in the clauses containing y to 0. Now all y-clauses are x-satisfied and there are at most six clauses which do not share any variable with at least one of the y-clauses. Hence we can decide XSAT in polynomial-time as above.

- If $w(x) \leq 2$, for all $x \in V(C)$, we can decide XSAT for C in polynomial-time according to Lemma 14.

□

3.6 Connection to Combinatorial Optimization Problems

Our results imply the NP-completeness for some subversions of the well-known combinatorial optimization problems: Exact Hitting Set on linear hypergraphs, Set Partitioning on linear hypergraphs and Exact Hitting Set on exact linear hypergraphs. A *hypergraph* is a pair $H = (V, E)$ where $V = V(H)$ is a finite set, the *vertex set* and $E = E(H)$ is a family of subsets of V the *hyperedge set* such that for each $x \in V$ there is an edge containing it. We call a hypergraph *linear* if each two hyperedges have at most one variable in common and *exact linear* if each two hyperedges have exactly one variable in common. Recall that Set Partitioning takes as input a finite hypergraph with a vertex set M and a set of hyperedges $\mathcal{M}$ (i.e. subsets of M). It asks for a subfamily $\mathcal{T}$ of $\mathcal{M}$ such that each element of M occurs in exactly one member of $\mathcal{T}$. It is easy to see that monotone XSAT coincides with Set Partitioning when the clauses take on the roles of vertices in M and the variables are regarded as the hyperedges in $\mathcal{M}$ in such a way that a variable contains all clauses in which it occurs.
Exact Hitting Set, however, is nothing but monotone XSAT translated to the hypergraph (or set system) language. Exact Hitting Set is the variant of the Hitting Set problem in which each set must be hit by exactly one element of the Hitting Set. The Hitting Set problem is defined as follows: Let $H = (V, E)$ be a hypergraph with vertex set V and hyperedge set E. Then a set $S \subset V$ is called *Hitting Set* of H if for all edges $e \in E$: $S \cap V(e) \neq \emptyset$. Therefore, we directly obtain that Exact Hitting Set remains NP-complete for linear, l-regular hypergraphs. And a simple dualization argument implies that the same is true for Set Partitioning on that specific class of hypergraphs.
Moreover, it easily follows that Exact Hitting Set is NP-complete for exact linear hypergraphs. However, XLCNF$_+$-XSAT cannot be reduced to Set Partitioning for exact linear hypergraphs; here the dualization argument fails.

Corollary 5.

- *Exact Hitting Set on linear hypergraphs is* NP*-complete.*
- *Set Partitioning on linear hypergraphs is* NP*-complete.*
- *Exact Hitting Set on exact linear hypergraphs is* NP*-complete.*

Proof.

- We can reduce LCNF_+-XSAT to Exact Hitting Set on linear hypergraphs in polynomial-time.
- We can reduce LCNF_+-XSAT to Set Partitioning on linear hypergraphs in polynomial-time.
- We can reduce XLCNF_+-XSAT to Exact Hitting Set on exact linear hypergraphs in polynomial-time.

□

Observe that Set Partitioning for exact linear hypergraphs is trivial in the sense that it has no solution unless the input hypergraph consists of one hyperedge only.

Chapter 4

k-Outerplanar Formulas

4.1 An Algorithm Solving SAT for Outerplanar Formulas in Linear Time

As already mentioned in the introduction SAT is NP-complete for the planar formula class. The outerplanar formulas form a subclass of the planar formula class. In this section we illustrate how SAT can be solved in linear time for the outerplanar formula class. The algorithm presented for this purpose works by exploiting the structure of outerplanar formulas. In the last section of this chapter we discuss an algorithm which uses dynamic programming in a nice tree decomposition of the corresponding variable-clause graph G_F of an outerplanar formula F. Since the treewidth of an outerplanar formula is 2, this algorithm also needs linear time to solve SAT for an outerplanar formula. Nevertheless, as we are especially interested in the structure of the variable-clause graphs of outerplanar formulas, we devote this section to a longer algorithm which exploits a preceding detailled analysis of the outerplanar formula class and thus illustrates the outerplanar formulas.

Definition 6. *Let K be an outerplanar circle, then we call an edge inside K a chord.*

If K is an outerplanar circle with q variable vertices and q clause vertices, K has obiously at most $q-2$ many chords.

Let $F = c_1 \wedge \ldots \wedge c_m$ be a CNF formula. In the following we consider F as a set of its clauses, i.e. $F = \{c_1, c_2, \ldots, c_m\}$ and call F a *matched formula* if $|F'| \leq |V(F')|$, for every subset $F' \subset F$ where $|F|$ denotes the number of clauses in F. It is not difficult to see that matched formulas are always satisfiable with respect to SAT. To show this we consider the bipartite incidence graph $G_F = (V, E)$ of a mathcing formula F with the vertex set partition $V = V(F) \cup F$, i.e. we partition V into a set of clauses and a set of variables. The edge set E consists of edges between variables and clauses: If a variable x is contained in a clause c, then there is an edge between x and c. It is easy to see that every subset $F' \subseteq F$ has the neighbourhood $N(F') = V(F') \subseteq V(F)$ in G_F. Because $|F'| \subseteq |V(F')| = |N(F')|$ for every subset $F' \subseteq F$, we can apply the classical Theorem of Koenig-Hall [23, 30] for bipartite graphs stating the existence of a matching in G_F covering the component F of the vertex set. In terms of the formula, this means that there is a set of variables,

corresponding to the vertices of the matching edges so that each of them is assigned uniquely to a clause of F sothat no clause is left out. Since these variables are all distinct, the corresponding literals can independently be set to 1 yielding a model of F. In the following, we denote with MAP the class of outerplanar formulas which are matching formulas. As one can easily see, outerplanar formulas whose variable-clause graphs consist of paths (each two of which are allowed to share a vertex), disjoint circles and single vertices belong to the class MAP and thus are solvable.

As shown in [43], each matched formula F with n variables is satisfiable and a model can be found in $O(\sqrt{n}||F||)$ where $||F||$ denotes the sum of the length of all clauses. So a model can be found in $O(\sqrt{n}||F||)$ for each formula $F \in MAP$ with n variables. But can a model for formulas of the class MAP also be found in linear time? To answer this question we analyse the special structure of outerplanar formulas.
To begin with, we fix some notation. Recall that a *backbone variable* x of a CNF formula F has the same assignment in all models of F, that means either $x = 0$ or $x = 1$ in all models of F.
Let K be an outerplanar circle with chords which are not allowed to cross inside the circle. Omitting the chords we obtain the *annulus* of K which we denote by K_R. Let K^i be a circle then we denote the corresponding formula by C^i, the annulus of K^i by K^i_R and its corresponding 2-CNF formula by C^i_R. When considering the satisfiability of a circle K^i we mean the satisfiability of C^i. The next lemma states that each backbone variable of a circle is also a backbone variable of the corresponding annulus.

Lemma 17. *Let K^i be a circle of G_F with possible chords. Then we obtain: If x is a backbone variable of K^i, then x is also a backbone variable of K^i_R.*

Proof. Let C^i_R be the 2-CNF formula corresponding to K^i_R. Then for each clause c^i_R of C^i_R there is exactly one clause c^i of C^i such that $V(c^i_R) \subset V(c^i)$ because C^i_R results from C^i by shortening clauses of C^i. Thus each model of C^i_R is obviously also a model of C^i. So let us assume that there is a backbone variable x of C^i which is not a backbone variable of C^i_R, then there is a model of C^i_R in which x is set to 1 and another in which x is set to 0. But each such model is also a model of C^i and hence x is not a backbone variable of C^i which is a contradiction to the assumption. □

The following theorem gives a direct characterisation of a backbone variable of an annulus K_R and is thus very useful for our algorithm solving outerplanar formulas.

Theorem 41. *Let K be a circle of G_F consisting of the variable vertices $x_1, \ldots, x_k$ and the clause vertices $c_1, \ldots, c_k$. Further let K_R be the annulus of K. Then x_1 is a backbone variable of the 2-CNF formula C_R corresponding to K_R if, and only if, we can obtain an implicational chain of the form*

$$x_1 \Rightarrow l(x_2) \Rightarrow l(x_3) \Rightarrow \ldots \Rightarrow l(x_k) \Rightarrow \overline{x_1}$$

or

$$\overline{x_1} \Rightarrow l(x_2) \Rightarrow l(x_3) \Rightarrow \ldots \Rightarrow l(x_k) \Rightarrow x_1$$

from C_R.
Moreover K_R has at most one backbone variable.

PROOF. Let K_R be the annulus resulting from K by omitting possible chords. Let $x_1, c_1, x_2, c_2 \ldots, x_k, c_k$ be the range of the vertices in clockwise direction. Omitting the polarities of the variables we can write for each clause:

$$c_1 = \{x_1, x_2\}, c_2 = \{x_2, x_3\}, c_3 = \{x_3, x_4\}, \ldots, c_k = \{x_k, x_1\}$$

Each 2-clause $\{x_i, x_{i+1}\}$ corresponds to an implication $\overline{x_i} \Rightarrow x_{i+1}$ or $\overline{x_{i+1}} \Rightarrow x_i$. An implication has the value true if the premise is false or in case $1 \Rightarrow 1$. Hence the only case when the implication is always false is $1 \Rightarrow 0$. Let us consider the corresponding implication for each 2-clause: If we can build a single implicational chain, e.g. $x_1 \Rightarrow \ldots \Rightarrow \overline{x_1}$, with all these k implications, then x_1 is a backbone variable. By setting $x_1 = 0$ we would obtain the implication $0 \Rightarrow \ldots \Rightarrow 1$ which is always true because the premise is false. By assigning $x_1 = 1$ we obtain the implication $1 \Rightarrow \ldots \Rightarrow 0$ which is obviously false. Hence $x_1 = 0$ is the only solution for the implicational chain and thus x_1 is a backbone variable for the 2-CNF formula belonging to K_R.

Next we show that if x_1 is a backbone variable of K_R, then the implicational chain for x_1 is either $\overline{x_1} \Rightarrow \ldots \Rightarrow x_1$ or $x_1 \Rightarrow \ldots \Rightarrow \overline{x_1}$ for x_1.

Suppose such an implicational chain does not exist, i.e. we have at least two implicational chains $x_1 \Rightarrow \ldots \Rightarrow x_i$ and $\overline{x_i} \Rightarrow \ldots \Rightarrow \overline{x_1}$ which we cannot compose to a single implicational chain. In this case we can assign 0 as well as 1 to x_1 to satisfy the formula belonging to K_R: When setting $x_1 = 1$ we have to assign $x_i = 1$. When assigning $x_1 = 0$ we can assign 0 as well as 1 to x_1 to satisfy the two implicational chains.

Now let x_1 be a backbone variable of K_R and let $x_1 \Rightarrow \ldots \Rightarrow \overline{x_1}$ be the implicational chain. In consequence w have to assign $x_1 = 0$ to satisfy K_R. Suppose there is another variable $x_i, 1 < i \leq k$ on K_R which is also a backbone variable of K_R, then, according to our previous considerations, we have two implicational chains $x_1 \Rightarrow \ldots \Rightarrow x_i$ and $x_i \Rightarrow \ldots \Rightarrow \overline{x_1}$. But these chains can neither be composed to the single implicational chain $x_i \Rightarrow \ldots \Rightarrow \overline{x_i}$ nor to $\overline{x_i} \Rightarrow \ldots \Rightarrow x_i$. Hence x_i cannot be a backbone variable which is a contradiction to our assumption. □

In the following we write $x_i \overset{+(-)}{\longrightarrow} c_j$, if there is a chord in K between the vertices x_i and c_i and x_i occurs positively (negatively) in c_j.

Theorem 42. *Let K be a circle (with possible chords) of the outerplanar variable-clause graph G_F belonging to the outerplanar formula F such that C_R, the 2-CNF formula belonging to K_R, can be written as the following implicational chain*

$$\overline{x_1} \Rightarrow x_2 \Rightarrow x_3 \Rightarrow \ldots \Rightarrow x_n \Rightarrow x_1$$

i.e. x_1 is a backbone variable of K_R. Further we assume that no variables of K_R are assigned yet. If one of the following three cases occurs, then x_1 is not a backbone variable of the formula C corresponding to K.

1. *K has a chord $x_i \overset{+}{\longrightarrow} c_j$, for $1 \leq i < n$, $1 < j \leq n$ and $i < j$.*
2. *K has a chord $x_i \overset{-}{\longrightarrow} c_j$, for $1 < i \leq n$, $1 \leq j < n$ and $i > j$.*
3. *K has a chord $x_1 \overset{-}{\longrightarrow} c_j$, for $1 < j \leq n$.*

We call these three cases the chords criterion. If none of these three cases occurs, x_1 is also a backbone variable of C.

Proof.

As x_1 is a backbone variable of C_R in each model of C_R the variable x_1 has the value 1 and as each model of C_R is also a model of C we obtain a model of C where $x_1 = 1$. If K has no chords, we obtain $C_R = C$ and C corresponds to the following implicational chain $\overline{x_1} \Rightarrow x_2 \Rightarrow x_3 \Rightarrow \ldots \Rightarrow x_n \Rightarrow x_1$. We can only satisfy this implicational chain for $x_1 = 1$, hence x_1 is also a backbone variable of C.

1. If K has a chord $x_i \xrightarrow{+} c_j$, for $1 \leq i < n$, $1 < j \leq n$ and $i < j$, then the following assignment is also a model for C:
   - $x_1 = 0$
   - $x_k = 1$, for $1 < k \leq j$
   - $x_k = 0$, for $j < k \leq n$.
2. If K has a chord $x_i \xrightarrow{-} c_j$, for $1 < i \leq n$, $1 \leq j < n$ and $i > j$, then the following assignment is also a model for C:
   - $x_1 = 0$
   - $x_k = 1$, for $1 < k \leq j$
   - $x_k = 0$, for $j < k \leq n$.
3. If K has a chord $x_1 \xrightarrow{-} c_j$, for $1 < j \leq n$, then the following assignment is also a model for C:
   - $x_1 = 0$
   - $x_k = 1$, for $1 < k \leq j$.
   - $x_k = 0$, for $j < k \leq n$.

□

For the reason of completeness we also present the counterpart of the last theorem:

Let K be a circle (with possible chords) of the outerplanar variable-clause graph G_F of the outerplanar formula F such that the 2-CNF formula C_R corresponding to K_R can be written as the following implicational chain

$$x_1 \Rightarrow x_2 \Rightarrow x_3 \Rightarrow \ldots \Rightarrow x_n \Rightarrow \overline{x_1}$$

meaning that x_1 is a backbone variable of K_R. Further, no variables of K are set yet. If one of the following three cases occurs, then x_1 is not a backbone variable of the formula C belonging to K.

1. K has a chord $x_i \xrightarrow{+} c_j$, for $1 \leq i < n$, $1 < j \leq n$ and $i < j$.
2. K has a chord $x_i \xrightarrow{-} c_j$, for $1 < i \leq n$, $1 \leq j < n$ and $i > j$.
3. K has a chord $x_1 \xrightarrow{+} c_j$, for $1 < j \leq n$.

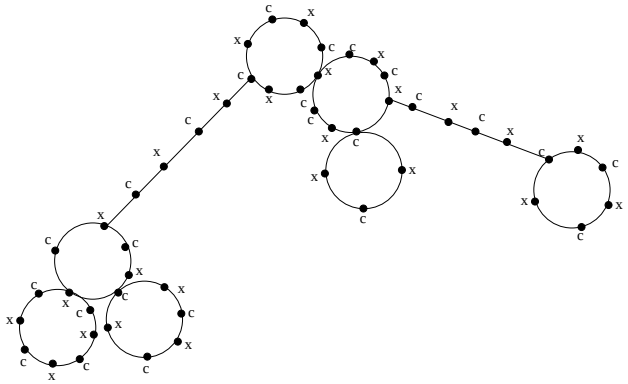


Figure 4.1: Example for a circle component

If none of these three cases occurs, then x_1 is also a backbone variable of C.

Let F be an outerplanar formula with the variable-clause graph G_F and let K^1 and K^2 be two circles of G_F which share the variable vertex x only. Let x be a backbone 1 variable of K^1_R and a backbone 0 variable of K^2_R. Then G_F is unsatisfiable if x remains a backbone variable of K^1 as well as of K^2. This follows immediately from the definition of a backbone variable. We call such a situation a *backbone conflict*. We call a connected component KK of an outerplanar graph G_F a *circle-component* if KK consists of circles for which holds that each two circles are either disjoint or share a single vertex only or they are joined by a path.

For each circle-component KK we can create a *superstructure graph* B_{KK} as follows: The vertices of B_{KK} are the circles of KK. If there is a circle K^1 which is joined with another circle K^2 by a path in KK then there is an edge in B_{KK} between the vertices K^1 and K^2 corresponding to the path. If two circles K^1 and K^2 share a single variable (clause) vertex x (c) then there is an edge in B_{KK} between the vertices K^1 and K^2 labelled with x (c).
Note that because of the outerplanarity of G_F the superstructure graph B_{KK} must be a tree. Now we present an algorithm solving SAT for outerplanar formulas in linear time.

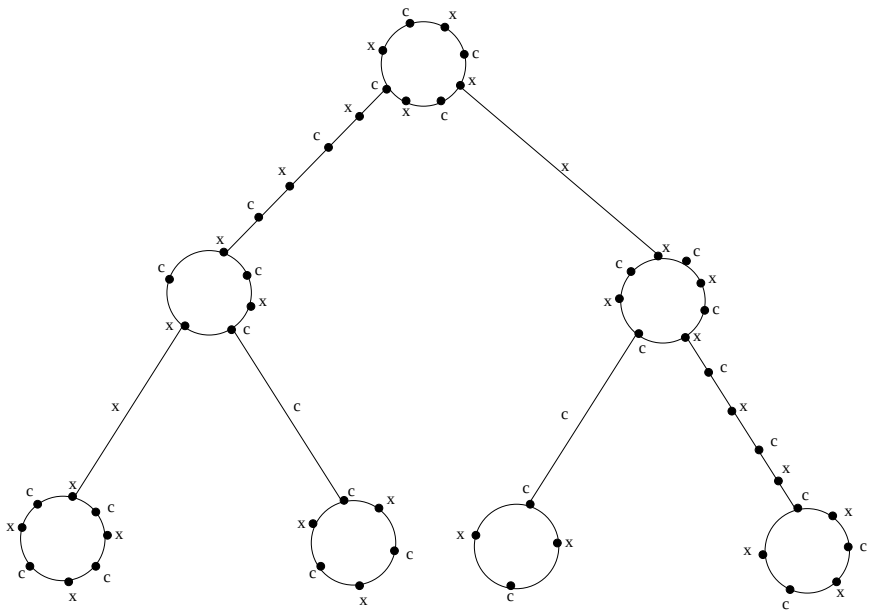


Figure 4.2: Example for a superstructure graph

Algorithm ASP

INPUT: A formula F in conjunctive normal form with an outerplanar variable-clause graph G_F.
OUTPUT: A model if F is satisfiable and UNSATISFIABLE, else.

1. If F is a 2-CNF formula, we solve F by an appropriate linear-time 2-SAT algorithm.
2. By an outerplanar embedding we determine the outerplanar variable-clause graph G_F belonging to F.
3. As long as there is a variable vertex x_i in G_F which is joined to only one clause-vertex c_j, we set x_i such that it satisfies the clause c_j: $l(x_i) = 1$ in c_j.
4. As long as there is a clause vertex c_j in G_F which is adjacent to only one variable vertex x_i, we set x_i so that it satisfies c_j: $l(x_i) = 1$ in c_j.
5. As long as G_F has disjoint circles (with possible chords) we perform the following for each of these circles: Let K^1 be such a circle and let

   $$x_1 - c_1 - x_2 - c_2 - x_3 - c_3 - \ldots - x_n - c_n - x_1$$

   be all the vertices of K^1 in clockwise direction. Then we assign x_i such that it satisfies c_i, that is $l(x_i) = 1$ in c_i, for all $i \in \{1, \ldots, n\}$.
6. If G_F consists of m circles $K^1, K^2, \ldots K^m$ only (there are no paths or other outerplanar constructions in G_F) and each two circles are allowed to share at most one vertex, then F is satisfiable: Let

   $$x_1^i - c_1^i - x_2^i - c_2^i - x_3^i - c_3^i - \ldots - x_n^i - c_n^i - x_1^i$$

be the vertices of K^i in clockwise direction, for $i = 1 \ldots, m$. Then we set x_k^i such that $l(x_k^i) = 1$ in c_k^i, that means by the assignment of x_k^i the clause c_k^i is satisfied, for $i = 1, \ldots, m$, $k = 1, \ldots, n$. This way we obtain a model for F.

7. If now the remaining graph G_F consists of several circle-components, we solve each circle-component KK by the **procedure circle-component**(KK).
8. If the **procedure circle-component**(KK) returns a model for each circle-component, then the algorithm ASP also returns this model for F. If the **procedure circle-component**(KK) cannot find a model for at least one circle-component KK, then F is unsatisfiable and the algorithm ASP returns UNSATISFIABLE.

Procedure circle-component(KK)
INPUT: A circle-component KK of the outerplanar variable-clause graph G_F which belongs to the outerplanar formula F.
OUTPUT: A model, if the CNF formula C_{KK} corresponding to the graph KK is satisfiable, UNSATISFIABLE, else.

1. We determine the superstructure tree B_{KK} belonging to KK.
2. We perform the following for each circle K in KK:
   Let K_R be the annulus belonging to the circle K, then we write each 2-clause $(l(x_i) \vee l(x_{i+1}))$ which lies on K_R as the corresponding implication $\overline{l(x_i)} \Rightarrow l(x_{i+1})$ or $\overline{l(x_{i+1})} \Rightarrow l(x_i)$and check whether we can build a single implicational chain of all these implications. In case we can put all these implications together to a single implicational chain w.l.o.g. $x_1 \Rightarrow \ldots \Rightarrow \overline{x_1}$ or $\overline{x_1} \Rightarrow \ldots \Rightarrow x_1$, i.e. x_1 is a backbone variable of K_R according to Theorem 41. Note that it is the only backbone variable of K_R in this case.
3. For all circles K whose annulus K_R has a backbone variable x_1 we check with help of Theorem 42 whether x_1 also remains a backbone variable of K:
   Let K_R be the annulus with the backbone variable x_1 and let

   $$\overline{x_1} \Rightarrow x_2 \Rightarrow x_3 \Rightarrow \ldots \Rightarrow x_n \Rightarrow x_1$$

   be the respective implicational chain according to Theorem 41. Further let $x_1 - c_1 - x_2 - c_2 - x_3 - c_3 - \ldots - x_n - c_n - x_1$ be the range of the vertices on K_R in clockwise direction. Then we obtain for each variable x_i, $i > 1$: x_i occurs positively in c_{i-1}, negatively in c_i and x_1 occurs positively in c_1 and c_n on K_R.
   Next we check whether there is a chord $x_i \xrightarrow{g} c_j$ on K^1 where $g \in \{+, -\}$ which satisfies one of the following three cases of the chords criterion:

   (a) K^1 has a chord $x_i \xrightarrow{+} c_j$, where $1 \leq i < n$, $1 < j \leq n$ and $i < j$.

   (b) K^1 has a chord $x_i \xrightarrow{-} c_j$, where $1 < i \leq n$, $1 \leq j < n$ and $i > j$.

   (c) K^1 has a chord $x_1 \xrightarrow{-} c_j$, where $1 < j \leq n$.

   If none of these three cases occurs for any chord of the current circle K^1, then x_1 is a backbone-variable for K^1, too. Therefore we set $x_1 = 1$.
   We perform these steps until we have considered all the circles of KK and found all the backbone variables.

4. We assign all the backbone variables of KK such that they make it possible to satisfy the formula.

5. If there are two circles K^1 and K^2 sharing a variable vertex x where x is a backbone 1 variable of K^1 and a backbone 0 variable of K^2, then there is a backbone conflict and hence KK is unsatisfiable. The procedure stops and returns UNSATISFIABLE.

6. We start with considering the leaves of the tree B_{KK}. As soon as we have solved all the leaf-circles BK in B_{KK} by **procedure circle (BK)**, we consider the father vertices VK for all BK and solve these again by **procedure circle (VK)**. In general, we do not solve a circle until we have solved all its descendants in B_{KK} by **procedure circle**. So beginning with the leaves in B_{KK} and as long as we have not reached the root of the tree yet, we check whether we can apply the following steps:

   - Let K^i be the current circle in B_{KK}. If K^i is joined to its father VK^i in B_{KK} by an edge labelled with x, we proceed as follows:
     - If x is already assigned (e.g as a possible backbone variable of VK^i), then we check by the **procedure circle** (K^i), whether the circle K^i can be satisfied with this assignment of x.
       (a) If **procedure circle** (K^i) returns that K^i is unsatisfiable, then the corresponding formula C_{KK} of KK is unsatisfiable and the procedure stops with the output UNSATISFIABLE.
       (b) If the **procedure circle** (K^i) returns a model α for K^i, under consideration of the assignment of x, then we set all variables of K^i according to α. After having solved all the sibling-circles GK^i of K^i we solve VK^i, the father circle of K^i.
     - If x is not assigned yet, we check by the **procedure circle** $(K^i_{x=1})$ and **procedure**$(K^i_{x=0})$, whether K^i can be satisfied for $x = 1$ as well as for $x = 0$.
       (a) If K^i can be satisfied for $x = 1$ as well as for $x = 0$, we do not fix x directly but first consider all the sibling-circles GK^i of K^i before considering the father VK^i of K^i.
       (b) If w.l.o.g. K^i can only be satisfied for $x = 1$, we assign $x = 1$, solve all the sibling-circles of K^i and afterwards solve the father VK^i of K^i. Note that x is already fixed to 1 when we consider VK^i.
       (c) If K^i cannot be satisfied for neither $x = 1$ nor $x = 0$, the fromula C_{KK} corresponding to KK is unsatisfiable and the procedure stops with the output UNSATISFIABLE.
   - Let K^i be the current circle of B_{KK}. If K^i is joined to its father circle VK^i in B_{KK} by an edge labelled with the clause vertex c, then we proceed as follows:

     We check by **procedure circle** (K^i) whether K^i is satisfiable.
     - If yes, particularly c is satisfied, hence we label c as satisfied in VK^i.

  - If not, we consider VK^i after having applied **procedure circle** on all its sibling-circles before.

- Let K^i be the current circle of B_{KK}. If K^i is joined to its father circle VK^i in B_{KK} by an edge which corresponds to the path P between the circles K^i and VK^i in KK, we proceed as follows:
  - If K^i and its father-circle VK^i are joined by the path $P = x_1 - c_1 - x_2 - c_2 - \ldots - x_n - c_n$ which shares the variable vertex x_1 with K^i and the clause vertex c_n with VK^i, then we proceed as follows:
    * We check whether K^i can be satisfied for $x_1 = 1$ as well as for $x_1 = 0$ by **procedure circle** $(K^i_{x_1=1})$ and by **procedure circle** $(K^i_{x_1=0})$. If K^i can be satisfied for $x_1 = 1$ as well as for $x_1 = 0$, we set x_1 so that its assignment satisfies c_1. Further we set x_2 so that it satisfies c_2. In general we set x_i so that its assignment satisfies c_i, for $i = 1, \ldots, n$. It follows that c_n is now satisfied in VK^i and thus we label c_n as satisfied in VK^i.
    * If w.l.o.g. K^i is only satisfiable for $x_1 = 1$, we assign $x_1 = 1$ and check whether $x_1 = 1$ also satisfies c_1. If so, we assign x_i so that its assignment satisfies the clause vertex c_i of the path P, for $i = 2, \ldots, n$. Therefore, c_n is now satisfied in VK^i and thus we label c_n as satisfied in VK^i. But if c_1 cannot be satisfied by $x_1 = 1$, we assign x_{i+1} such that it satisfies the clause c_i for $i = 1, \ldots, n-1$. If x_n occurs with the same polarity in c_n as in c_{n-1}, we label c_n as satisfied in VK^i.
    * But if K^i cannot be satisfied for neither $x_1 = 0$ nor $x_1 = 1$, then the procedure stops with the output UNSATISFIABLE.
  - If the two circles K^i and its father circle VK^i are joined together by the path

    $$P = x_1 - c_1 - x_2 - c_2 - \ldots - x_{n-1} - c_{n-1} - x_n$$

    such that P shares x_1 with K^i and x_n with VK^i, then we proceed as follows:
    * We check whether we can satisfy K^i for $x_1 = 1$ as well as for $x_1 = 0$ by **procedure circle** $(K^i_{x_1=1})$ and **procedure circle** $(K^i_{x_1=0})$. If K^i can be satisfied for $x_1 = 1$ as well as for $x_1 = 0$, we set x_1 so that its assignment satisfies c_1. Furthermore, we set x_i such that its assignment satisfies c_i, for $i = 1, \ldots, n-1$. Note that x_n remains unassigned.
    * If w.l.o.g. K^i is only satisfiable for $x_1 = 1$, we set $x_1 = 1$ and check whether $x_1 = 1$ also satisfies c_1. If so, we set x_i such that it satisfies c_i, for $i = 2, \ldots, n-1$. But if the assignment $x_1 = 1$ does not satisfy the clause c_1, we set x_{i+1} such that its assignment satisfies the clause c_i, for all $i = 1, \ldots, n-1$. As a result the variable x_n has a fixed value in VK^i.
    * If K^i cannot be satisfied for neither $x_1 = 1$ nor $x_1 = 0$, the formula C_{KK} corresponding to KK is unsatisfiable and the procedure stops with the output UNSATISFIABLE.

– If the two circles K^i and VK^i are joined together by the path

$$P = c_1 - x_1 - c_2 - x_2 - \ldots - c_{n-1} - x_{n-1} - c_n$$

which shares the clause vertex c_1 with K^i and the clause vertex c_n with VK^i, we proceed as follows:

* We check by **procedure circle** (K^i) whether we can find a model for K^i. If K^i is satisfiable, we set all the variables of K^i according to the model. Further we set x_1 such that it satisfies c_2. In general, we set x_i such that its assignment satisfies c_{i+1}, for all $i = 1, \ldots, n-1$. As a consequence the clause c_n in VK^i is now satisfied and we label c_n as satisfied in VK^i.
* If **procedure circle** (K^i) returns UNSATISFIABLE, we set x_1 such that it satisfies the clause vertex c_1. Further we check whether this assignment of x_1 satisfies the clause c_2, that is whether x_1 occurs with the same polarity in c_2 as in c_1. If so, we further set x_i such that it satisfies c_{i+1}, for all $i = 2, \ldots, n-1$. But if the assignment of x_1 does not satisfy c_2 because x_1 occurs with different polarities in c_1 and in c_2, we set x_2 so that it satisfies the clause c_2. Afterwards we check whether this assignment of x_2 also satisfies c_3, i.e. whether x_2 occurs with the same polarity in c_2 as in c_3. If so, then we further set x_i such that it satisfies c_{i+1}, for all $i = 3, \ldots, n-1$. Otherwise, we set x_3 such that its assignment satisfies the clause c_3 and so forth. If x_{n-1} occurs with the same polarity in the clause c_n as in c_{n-1} then c_n is also satisfied by the assignment of x_{n-1} and we label c_n as satisfied in VK^i. Otherwise VK^i remains unmodified by the path.

– If the two circles K^i and VK^i are joined together by the path

$$P = c_1 - x_1 - c_2 - x_2 - \ldots - c_{n-1} - x_{n-1} - c_n - x_n$$

which shares the clause vertex c_1 with K^i and the variable vertex x_n with VK^i then we proceed as follows:

* We check by **procedure circle**(K^i) whether K^i is satisfiable. If so, we determine a model for K^i and set all the variables of K^i according to this model. Further we set x_1 such that it satisfies the clause c_2. Afterwards we set x_2 such that it satisfies c_3 and generally we set x_i such that it satisfies the clause c_{i+1}, for all $i = 1, \ldots, n-1$. Note that this does not influence the variable x_n which P shares with VK^i and so x_n remains unassigned.
* If **procedure circle** (K^i) returns UNSATISFIABLE, we set x_1 so that it satisfies the clause c_1. Next we check whether c_2 is also satisfied by this assignment of x_1. If so, we further set x_i such that it satisfies c_{i+1}, for $i = 2, \ldots, n-1$. But if this assignment of x_1 does not satisfy c_2, we set x_2 such that it satisfies c_2 and check whether this assignment also satisfies c_3. If so, we further set x_i so that it satisfies c_{i+1}, for $i = 3, \ldots, n-1$. Else we set x_3 so that it satisfies c_3 and so forth until c_n is satisfied.

7. As soon as we have reached the root circle WK of B_{KK} we check by **procedure circle (WK)** whether WK is satisfiable.

   - If WK is not satisfiable, the formula C_{KK} corresponding to KK is not satisfiable, either, and the procedure stops and the output UNSATISFIABLE.
   - If WK is satisfiable, we set all the variables of WK according to an arbitrary model of WK. If all the other circles of B_{KK} are satisfiable and we have already found a model for all them, the procedure stops and the output is a model for C_{KK}.
   - If WK is satisfiable and there is a circle K^j in B_{KK} for which we have not found a model yet, we proceed as follows for all these circles: (Note that we only have to solve those circles K^j for which we have not fixed a model yet in the first passage through the tree from the leaves up to the root of the tree):
     Beginning with the root circle we move down the tree B_{KK} until we have reached the leaves and have also found a model for all the leaves (and hence a model for C_{KK}) or until we have found a circle in B_{KK} which cannot be satisfied. In the latter case the procedure stops with the output UNSATISFIABLE. We do not solve a circle K^j until we have found a model for all its ancestors. For each such circle K^j with father circle VK^j we proceed as follows:

   - If K^j is joined to VK^j by an edge labelled with a variable vertex x, then x is now fixed in K^j because of VK^j, w.l.o.g. let $x = 0$. As K^j can be satisfied for $x = 0$ as well as for $x = 1$, we set all variables of K^j according to the model which fixes $x = 0$.
   - If K^j is joined to VK^j by an edge labelled with a clause vertex c, we check by **procedure circle** $(K^j - \{c\})$, whether $K^j - \{c\}$ is satisfiable. If not, then C_{KK} is unsatisfiable and the procedure stops with the output UNSATISFIABLE. Else the **procedure circle** $(K^j - \{c\})$ returns a model for $K^j - \{c\}$ and we set all variables of K^j according to this model and consider the next circle (in the range downstairs) in B_{KK} for which we have not determined a model yet.

procedure circle(K^i)
INPUT: A circle K^i with possible chords and possibly fixed variables.
OUTPUT: A model if C_{K^i}, the formula corresponding to K^i, is satisfiable and UNSATISFIABLE, else.

1. All clauses which are labelled as satisfiable are invisible for the procedure and not considered here.
2. As long as there is a variable vertex x_i in K^i which is adjacent to only one clause vertex c_j, we set x_i such that it satisfies the clause c_j.
3. As long as there is a clause vertex c_j in K^i which is adjacent to only one variable vertex x_i, we set x_i such that it satisfies the clause c_j.
4. If the 2-CNF formula corresponding to the annulus K^i_R of K^i is satisfiable, then K^i is satisfiable, too, and each model for K^i_R is also a model for K^i.

5. First we fix an order for the vertices of K_R^i and determine a marker S_M for each chord $S = x_i \longrightarrow c_j$ for which x_i is not assigned yet. This marker contains the following information:
   Let $x_m \Rightarrow \ldots \Rightarrow l(x_i)$ resp. $l(x_i) \Rightarrow \ldots \Rightarrow x_p$ be the two possible implicational chains for x_i, where $l(x_i) \in \{x_i, \overline{x_i}\}$, then the marker remembers whether there is a variable fixed to 1 in $x_m \Rightarrow \ldots \Rightarrow l(x_i)$ respectively to 0 in $l(x_i) \Rightarrow \ldots \Rightarrow x_p$.

6. If the 2-CNF formula corresponding to K_R^i is not satisfiable, K_R^i satisfies one of the two following cases:

   (a) The 2-CNF formula contains a smallest false implicational chain e.g. $x_k \Rightarrow \ldots \Rightarrow x_{k+l}$ where $x_k = 1$, $x_{k+l} = 0$ is already fixed and $x_{k+l} \neq \overline{x_k}$. In this case K_R^i is not satisfiable. We assume that this is the only false implicational chain on K_R^i. In order to test whether we can satisfy K^i when taking the chords into consideration, we check if one of the following cases occurs:

      i. There is a no chord of the form $x_i \xrightarrow{+(-)} c_j$, with $k \leq j \leq k+l$. In this case K^i is unsatisfiable and the procedure stops with the output UNSATISFIABLE.

      ii. There is a chord of the form $x_i \xrightarrow{+} c_j$, with $k \leq i < k+l$, $k < j \leq k+l$ and $i < j$.
      In this case the following assignment enables a model for K^i:
      $x_k = \ldots = x_j = 1$ and $x_{j+1} = \ldots = x_{k+l} = 0$.

      iii. There is a chord of the form $x_i \xrightarrow{-} c_j$, with $k < i \leq k+l$, $k \leq j < k+l$ and $i > j$.
      In this case the following assignment enables a model for K^i:
      $x_k = \ldots = x_j = 1$ and $x_{j+1} = \ldots = x_{k+l} = 0$.

      iv. There is a chord of the form $x_i \xrightarrow{+(-)} c_j$, with $i < k$ or $i > k+l$ and $k \leq j \leq k+l$ and x_i is already set to 1 (resp. to 0). Then the following assignment enables a model for K^i:
      $x_k = \ldots = x_j = 1$ and $x_{j+1} = \ldots = x_{k+l} = 0$.

      v. There is a chord of the form $S = x_i \xrightarrow{+(-)} c_j$, with $i < k$ or $i > k+l$ and $k \leq j \leq k+l$ and x_i is not assigned yet. Then we proceed as follows:

         - If there is only one such chord, we set x_i such that c_j is satisfied by the assignment of x_i, then

           $$x_k = \ldots = x_j = 1, x_{j+1} = \ldots = x_{k+l} = 0$$

           satisfies the clauses $c_k, \ldots c_{k+l}$. Next we check with help of the marker S_M whether the current assignment of x_i yields a false implicational chain $1 \Rightarrow 0$:
           - The implicational chain $x_m \Rightarrow \ldots \Rightarrow l(x_i)$ already contains a variable set to 1, $l(x_i) = x_i$ and $x_i = 0$.
           - The implicational chain $x_m \Rightarrow \ldots \Rightarrow l(x_i)$ already contains a variable set to 1, $l(x_i) = \overline{x_i}$ and $x_i = 1$.

– The implicational chain $l(x_i) \Rightarrow \ldots \Rightarrow x_p$ already contains a variable set to 0, $l(x_i) = x_i$ and $x_i = 1$.
– The implicational chain $l(x_i) \Rightarrow \ldots \Rightarrow x_p$ already contains a variable set to 0, $l(x_i) = \overline{x_i}$ and $x_i = 0$.

If one of these cases occurs, the remaining 2-CNF formula cannot be satisfied apart from $x_k \Rightarrow \ldots \Rightarrow x_{k+l}$ because with the current assignment of x_i we obtain an additional false implicational chain. If we can satisfy this chain applying (6a), then K^i is satisfiable, else unsatisfiable.

- If there are several chords $S = x_i \xrightarrow{+(-)} c_j$ for which holds $i < k$ resp. $i > k + l$, $k \leq j \leq k + l$ and x_i is not fixed yet, we perform the following for each such chord S as long as we have not found a model for K^i yet:
  We set $x_i = 1$ (resp. $x_i = 0$) and

$$x_k = \ldots = x_j = 1, x_{j+1} = \ldots = x_{k+l} = 0$$

  Then we check by means of the marker S_M whether we obtain a false implicational chain $1 \Rightarrow 0$ by the assignment of x_i:
  – The implicational chain $x_m \Rightarrow \ldots \Rightarrow l(x_i)$ already contains a variable set to 1, $l(x_i) = x_i$ and $x_i = 0$.
  – The implicational chain $x_m \Rightarrow \ldots \Rightarrow l(x_i)$ already contains a variable set to 1, $l(x_i) = \overline{x_i}$ and $x_i = 1$.
  – The implicational chain $l(x_i) \Rightarrow \ldots \Rightarrow x_p$ already contains a variable set to 0, $l(x_i) = x_i$ and $x_i = 1$.
  – The implicational chain $l(x_i) \Rightarrow \ldots \Rightarrow x_p$ already contains a variable set to 0, $l(x_i) = \overline{x_i}$ and $x_i = 0$.

  If one of these cases occurs, the current assignment of x_i yields a false implicational chain as a consequence. If we can satisfy this one by applying (6a), then K^i is satisfiable. If not, we undo (release) the assignment of the variables $x_k, \ldots x_{k+l}$ and test whether there is another chord $x_i \xrightarrow{+(-)} c_j$, with $i < k$ resp. $i > k + l$ and $k \leq j \leq k + l$ where x_i is not fixed yet, yielding a model for K^i.
- If we have considered all the chords $x_i \xrightarrow{+(-)} c_j$ with $i < k$ resp. $i > k + l$ and $k \leq j \leq k + l$, where x_i is not fixed yet, and have not found a model for K^i then K^i is unsatisfiable and the procedure stops with the output UNSATISFIABLE.

(b) If K^i_R corresponds to $\overline{x_1} \Rightarrow \ldots \Rightarrow x_1$ (resp. $x_1 \Rightarrow \ldots \Rightarrow \overline{x_1}$), $x_1 = 0$ (resp. $x_1 = 1$) is already fixed and there are no other smaller false implicational chains on K^i_R, then we solve K^i as follows (w.l.o.g. let $\overline{x_1} \Rightarrow \ldots \Rightarrow x_1$ and $x_1 = 0$ be fixed): If K^i has no chords, it is unsatisfiable and the procedure stops with the output UNSATISFIABLE. Else we check whether there is a chord of the form $x_i \xrightarrow{+(-)} c_j$ in K^i:

i. Let $x_i \xrightarrow{+} c_j$ be a chord on K^i, with $i < j$, where $x_i = 1$ is already fixed. Then the clause c_j is satisfied by x_i and as there are no false implicational chains on $K^i_R - \{c_j\}$ the remaining 2-CNF formula

on $K_R^i - \{c_j\}$ is satisfiable. We solve the formula corresponding to $K_R^i - \{c_j\}$ by a linear 2-SAT algorithm.

ii. Let $x_i \xrightarrow{-} c_j$ be a chord, with $i > j$ and $x_i = 0$ is already fixed. Then the clause c_j is satisfied due to the assignment of x_i and as there are no false implicational chains on $K_R^i - \{c_j\}$ the remaining 2-CNF formula on $K_R^i - \{c_j\}$ is satisfiable. We solve the formula corresponding to $K_R^i - \{c_j\}$ by a linear 2-SAT algorithm.

iii. Let $x_1 \xrightarrow{-} c_j$ be the chord, with $j > 1$. Then the clause c_j is satisfied by the assignment of x_1 and as there are no false implicational chains on $K_R^i - \{c_j\}$ the remaining 2-CNF formula on $K_R^i - \{c_j\}$ is satisfiable. We solve the formula corresponding to $K_R^i - \{c_j\}$ by a linear 2-SAT algorithm.

iv. If there is only one chord $x_i \xrightarrow{+(-)} c_j$ with $i < j$ $(i > j)$ where x_i is not fixed, we set x_i so that it satisfies c_j.
If there are several chords of this form, we perform the following for each such chords:
We assign x_i so that it satisfies c_j and check whether this assignment satisfies the remaining 2-CNF formula on $K_R^i - \{c_j\}$. If so, we have found a model for K^i. Otherwise, there is now a false implicational chain, e.g. $x_i \Rightarrow \ldots \Rightarrow x_k$, with $x_i = 1, x_k = 0$ or $x_k \Rightarrow \ldots \Rightarrow \overline{x_i}$, with $x_i = 1, x_k = 1$. We check whether we can solve it by applying (6a). If so, we have found a model. Else we consider the next chord which satisfies (iv). If we have considered all such chords and still have not found a model, then K^i is unsatisfiable and the procedure stops with the output UNSATISFIABLE.

Corollary 6. SAT *is solvable in linear time for the outerplanar formula class.*

PROOF. **Analysis of the running time:**

- If G_F consists of disjoint circles, circles which share a clause vertex only and paths which may have a vertex in common (we assume that F has no unit clauses), then F is a matched formula and thus satisfiable. Steps 1.) - 6.) can obviously be performed in linear time.
- **procedure circle(K^i)**: Let K^i be a circle with q clause vertices and q variable vertices. Then K^i has at most $q - 2$ many chords. At first we solve the 2-CNF formula corresponding to the annulus K_R^i. Obviously we need $O(q)$ time for this. If the 2-CNF formula corresponding to the annulus K_R^i is satisfiable, the model for this formula is also a model for the formula corresponding to K^i.
  If the 2-CNF formula is not satisfiable because there is a false implicational chain, we take the chords into consideration and test whether we can find a chord which has the property of one of the cases (ii)-(iv) of **5(a)** in **procedure circle(K^i)**. If the 2-CNF formula is not satisfiable as a result of a false implicational chain $x_k \Rightarrow \ldots \Rightarrow x_{k+l}$ on the annulus K_R^i with $x_k = 1, x_{k+l} = 0$ already fixed and $x_{k+l} \neq \overline{x_k}$, then we check whether we can find a chord which satisfies one of the cases (ii)-(iv) of **5(a)** in **procedure circle(K^i)**. If not, we check for all chords $x_i \xrightarrow{+(-)} c_j$ with the property of

case (v) in **5(a)** whether by setting $x_i = 1$ (resp. $x_i = 0$) they provide a satisfying assignment for K^i. As soon as we have found such a chord providing a model for K^i the procedure outputs this model. If K^i is unsatisfiable, we perform this process recursively until we have considered all the chords with the property of case (v) in **5(a)** and have ascertained that none of them provides a model for K^i. Then the procedure stops with the output UNSATISFIABLE. Hence we consider each chord at most once. Using markers we need constant time to check for each chord whether by the assignment of the chord variable we obtain a false implicational chain on the annulus. Since K^i has at most $q-2$ many chords we need $O(q)$ running time to solve a circle with q many variable and q many clause vertices. Case **5(b)** of **procedure circle** (K^i) also yields a running time of $O(q)$ because in worst-case we have to check all chords.

- **procedure circle-component(KK)**:
  For each circle-component KK we run through the superstructure tree B_{KK} at most twice, that is we consider each vertex of B_{KK} at most twice: At first we run from the leaves up to the root circle of the tree. In case the root circle is satisfiable and we have not determined a model for each circle in B_{KK} yet, we run from the root down towards the leaves until we have either found a model for all circles of KK or until we have encountered an unsatisfiable circle in B_{KK}. It follows that **procedure circle()** is called at most twice for each circle of B_{KK}.

  As we can solve a circle K of q many variable and q many clause vertices in $O(q)$ running time, the running time for an outerplanar formula F of n variables is $O(n)$.

Proof of the correctness: We show that algorithm ASP outputs a satisfying truth assignment for an outerplanar formula F in case F is satisfiable, and UNSATISFIABLE else.
Considering **procedure circle component (KK)**:

First, we determine the backbone variables for all circles in KK. According to Lemma (17) the backbone variable x_1 of a circle K is a backbone variable of K_R and according to Theorem (41) x_1 is a backbone variable of K_R if, and only if, we can obtain an implicational chain $x_1 \Rightarrow \ldots \Rightarrow x_n \Rightarrow \overline{x_1}$ or $\overline{x_1} \Rightarrow \ldots \Rightarrow x_n \Rightarrow x_1$ from the 2-CNF formula corresponding to K_R. Therefore it is sufficient to consider the 2-CNF formula C_R corresponding to the annulus K_R in order to check whether a circle K has a backbone variable or not because By Theorem (42) we obtain if one of the following three cases occurs:

1. K has a chord $x_i \overset{+}{\longrightarrow} c_j$, where $1 \leq i < n$, $1 < j \leq n$ and $i < j$.
2. K has a chord $x_i \overset{-}{\longrightarrow} c_j$, where $1 < i \leq n$, $1 \leq j < n$ and $i > j$.
3. K has a chord $x_1 \overset{-}{\longrightarrow} c_j$, where $1 < j \leq n$.

x_1 is also a backbone variable for K.

Having found all the backbone variables we fix them because there is no other possibility to set them to obtain a model for F. Obviously the assignment of the backbone variables is no restriction for the satisfiability of the formula. If we have discovered that two circles K^1 and K^2 share a variable x which is a backbone variable of both circles but w.l.o.g. a backbone 0 variable for K^1 and a backbone 1

variable for K^2, then we have a backbone conflict and thus F is not satisfiable, hence algorithm ASP outputs UNSATISFIABLE.
If there is no backbone conflict in KK, we conctruct the superstructure tree B_{KK} for KK and starting with the leaves of B_{KK} we call **procedure (BK)** for each leaf BK. On the way up to the root circle WK of B_{KK} we call for each circle K **procedure (K)**: If K is joined to its father circle VK by an edge labelled with a variable vertex x, we call **procedure circle** $(K_{x=0})$ as well as **procedure** $(K_{x=1})$. If w.l.o.g. only **procedure** $(K_{x=1})$ outputs a satisfying assignment, we fix $x = 1$ because $x = 0$ does not yield a model for F. If both procedure calls output a satisfying truth assignment, we do not fix x yet because this would pose a restriction on the father VK of K as we do not know yet whether VK can also be satisfied for both values of x. Suppose VK is only satisfiable for $x = 0$ but we have already fixed $x = 1$, **procedure circle** $(VK_{x=1})$ would output UNSATISFIABLE and this would have consequences for all ancestors of VK in B_{KK}.
If on our way from the leaves up to the root of the tree we have called **procedure circle** (K) for a circle K which is joined to its father by an edge labelled with a clause vertex c and the output is UNSATISFIABLE, then we do not consider the father VK of K before having solved all its sibling circles and continue this way until having reached the root circle WK. If the **procedure circle (WK)** outputs a satisfying assignment for WK, we fix all the variables of WK according to this satisfying assignment and move downstairs towards the leaves of the tree B_{KK}. When arriving at a circle K in B_{KK}, its father VK must be satisfiable and all variables of VK already fixed according to a model. Hence the clause c in K is now satisfied and we can omit c from K and thus call **procedure circle** $(K - \{c\})$. If **procedure circle** $(K - \{c\})$ outputs UNSATISFIABLE too, then K cannot be satisfied and thus KK is unsatisfiable. In this case **procedure circle-component** (KK) outputs UNSATISFIABLE.
In summary, we firstly fix for each circle the backbone variables. On the way from the leaves up to the root we fix a variable only if its assignment does not pose a restriction on the satisfiability of the remaining circles of the tree. That means we only fix those variables for which only one assignment can possibly satisfy the circle (or path) or whose assignment does not influence the variables of other circles. As soon as we have reached the root circle WK and found a model for WK, we fix all the variables of WK according to this model and consider all those circles on our way downstairs for which we have not determined a model yet because either both values have been possible for the variable x joining K with its father circle VK or because **procedure circle** (K) returned UNSATISFIABLE. In the latter case K was joined to its father VK by an edge labelled with the clause vertex c.
If for a circle K which is joined to its father VK by an edge labelled with a variable vertex x we have not determined a model yet because there is a satisfying truth assignment for $x = 0$ as well as for $x = 1$, then when moving downstairs in the tree from the root to the leaves and arriving at K the variable x is fixed. Therefore we decide for the one satisfying truth assignment where x has this fixed value. Thus we do not need to call the **procedure circle** (K) once again. Hence, algorithm ASP works correctly. □

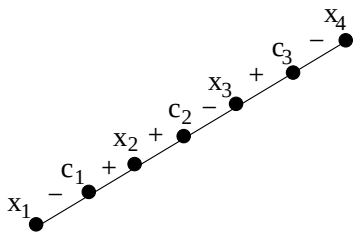


Figure 4.3: A path with four variable-vertices

4.2 Solving the Counting Problem #SAT for Outerplanar Formulas by the Separator Theorem of Lipton & Tarjan

In this section we provide a polynomial-time algorithm solving #SAT for outerplanar formulas. The basic idea is a divide-and-conquer approach based on the seminal separator theorem for planar graphs due to Lipton and Tarjan [35], which for convenience is stated next:

Theorem 43 ($\sqrt{n}$-separator theorem)**.** *Let G be a planar graph of n vertices. The vertex set of G can be partitioned into three sets A, B, C such that no edge joins a vertex in A with a vertex in B; neither A nor B contains more than $2n/3$ vertices, and C contains no more than $2\sqrt{2}\sqrt{n}$ vertices.*

We call C a *separator set* and the vertices contained in C the *separator vertices*. It is well known that every outerplanar graph has a separator set of size two and that such a separator set can be computed in linear time [8]. A useful variant of the theorem above for the special case of outerplanar graphs is due to Maheshwari et al. [36] stating that the separator set then has at most two vertices.

Let us emphasize some easy notions concerning circle-free graph patterns which can be treated as special cases in our algorithm which is described below.

Definition 7. *(1) A connected outerplanar graph G without circles is called a tree. (2) Given a tree consisting of intersecting paths which satisfy the following condition: Every two paths are only allowed to intersect in vertices corresponding to variables. We then define the mainpath as an arbitrary fixed path of the tree.*

Let P be a path consisting of n vertices and let α be a fixed satisfying truth assignment over the variables $x_1, \ldots, x_n$ of P. We write $M(x_i = \alpha(x_i))$ for the number of all different satisfying truth assignments of P in which the variables $x_i, \ldots, x_n$ are set according to α. The variables $x_1, \ldots, x_{i-1}$ can be set arbitrarily as long as P is satisfied. Similarly, we write $M(x_i = \overline{\alpha(x_i)})$ for the number of all different satisfying truth assignments of P, where the variables $x_{i+1}, \ldots, x_n$ are set according to α, $x_i = \overline{\alpha(x_i)}$ and the variables $x_1, \ldots, x_{i-1}$ can be assigned arbitrarily as long as P is satisfied.

Next we consider an example to demonstrate the Procedure NumberPath(P) which is used in our algorithm for the special case that a current formula corresponds to a path. So, let P be the path illustrated in Figure 4.3. An edge between

a variable vertex x_i and a clause vertex c_j labelled with $+(-)$ means that x_i occurs as a positive (negative) literal in c_j. Let α be the following fixed truth assignment for the variables x_2, x_3, x_4: $\alpha(x_2) = 1, \alpha(x_3) = 0, \alpha(x_4) = 0$. Then we have:

- $M(x_2 = \alpha(x_2)) = 2$ because the assignments $x_1 = 0, x_2 = 1, x_3 = 0, x_4 = 0$ as well as $x_1 = 1, x_2 = 1, x_3 = 0, x_4 = 0$ satisfy P .
- $M(x_2 = \overline{\alpha(x_2)}) = 1$ because for $x_2 = 0$ we have only one single choice to set x_1 to satisfy P, namely: $x_1 = 0$. Therefore the assignment

  $$x_1 = 0, x_2 = 0, x_3 = 0, x_4 = 0$$

  is the only possible assignment, so that the variables x_3, x_4 are set according to α and $x_2 = \overline{\alpha(x_2)}$.
- $M(x_3 = \alpha(x_3)) = M(x_2 = \alpha(x_2)) + M(x_2 = \overline{\alpha(x_2)}) = 2+1 = 3$. In this case there are three different satisfying truth assignments that force the variables x_3, x_4 to be set according to α.
- $M(x_3 = \overline{\alpha(x_3)}) = 2$ because if we set $x_3 = 1$ and $x_4 = 0$, the assignment of x_3 does not satisfy the clause c_2. Thus we have to set variable x_2 appropriately to satisfy clause c_2, namely: $x_2 = 1 = \alpha(x_2)$. There are altogether $M(x_2 = \alpha(x_2)) = 2$ satisfying truth assignments, which enforce the setting $x_2 = 1$.
- $M(x_4 = \alpha(x_4)) = M(x_3 = \alpha(x_3)) + M(x_3 = \overline{\alpha(x_3)}) = 3+2 = 5$. Accordingly, there are altogether 5 different satisfying truth assignments which enforce the setting of variable x_4 according to α.
- $M(x_4 = \overline{\alpha(x_4)}) = 2$ because given $x_4 = 1$, x_4 does not satisfy the clause c_3. So we have to set $x_3 = 1 = \overline{\alpha(x_3)}$. There are altogether $M(x_3 = \overline{\alpha(x_3)}) = 2$ satisfying assignments which enforce the setting $x_3 = 1$.
- Altogether there are $M(x_4 = \alpha(x_4)) + M(x_4 = \overline{\alpha(x_4)}) = 5 + 2 = 7$ truth assignments which satisfy P.

Next we introduce an algorithm counting all models of outerplanar formulas using a divide-and-conquer strategy based on the separator theorem mentioned above. This algorithm works recursively using subprocedures when the graph of the input formula is a tree:

Algorithm Number ASP (G_F)

INPUT: An outerplanar formula F with graph G_F.
OUTPUT: $N(G_F)$ = Number of all satisfying truth assignments for F
BEGIN

1. As long as there is a clause-vertex c_j in G_F which is only adjacent to a single variable-vertex x_i we fix the value of x_i such that c_j is satisfied by the assignment of x_i and do the following:We eliminate all the clause-vertices which are satisfied by the assignment of x_i, then we eliminate the vertex x_i and all edges incident with the vertex x_i from G_F.
2. If there is a clause-vertex not adjacent to any variable-vertex (i.e. there is an empty clause), the formula F is unsatisfiable and the procedure returns with the output $N(G_F) = 0$. Else the algorithm proceeds with the next step.

3. If G_F is a path, F is satisfiable and we determine the number of all satisfying assignments of G_F by the **Procedure NumberPath (G_F)**.

4. In case G_F is a tree, we determine the number of all satisfying assignments of G_F by the **Procedure NumberTree (G_F)**.

5. (a) Using the separator theorem we determine two vertices x_{i_1} and c_{j_1} so that G_F is partitioned into two subgraphs G_F^1 and G_F^2. These two have just the vertices x_{i_1} and c_{j_1} in common and neither of them contains more than $\frac{2n}{3}$ variable-vertices. Furthermore there is no edge joining a vertex from $G_F^1 - \{x_{i_1}, c_{j_1}\}$ with a vertex from $G_F^2 - \{x_{i_1}, c_{j_1}\}$.

   (b) We derive $G_F^1(x_{i_1} = 1)$ resp. $G_F^2(x_{i_1} = 1)$ by eliminating all those clause-vertices which are satisfied by the assignment $x_{i_1} = 1$, the variable-vertex x_{i_1} and all edges incident with x_{i_1}.

   Equally we derive $G_F^1(x_{i_1} = 0)$ resp. $G_F^2(x_{i_1} = 0)$ by eliminating (from G_F^1 resp. G_F^2) all the clause-vertices which are satisfied by the assignment $x_{i_1} = 0$, the vertex x_{i_1} and all the edges incident with x_{i_1}. Further we obtain $G_F^i(x_{i_1} = 1) - \{c_{j_1}\}$ resp. $G_F^i(x_{i_1} = 0) - \{c_{j_1}\}$, $i = 1, 2$, by eliminating the clause-vertex c_{j_1} and all edges incident with it from $G_F^i(x_{i_1} = 1)$ resp. $G_F^i(x_{i_1} = 0)$.

   (c) Recursively we compute the number of all satisfying truth assignments of
   $G_F^i(x_{i_1} = 1)$, $G_F^i(x_{i_1} = 0)$, $G_F^i(x_{i_1} = 1) - \{c_{j_1}\}$ and $G_F^i(x_{i_1} = 0) - \{c_{j_1}\}$, for $i = 1, 2$, by:
   Algorithm Number ASP($G_F^1(x_{i_1} = 1)$),
   Algorithm Number ASP($G_F^1(x_{i_1} = 0)$),
   Algorithm Number ASP($G_F^2(x_{i_1} = 1)$),
   Algorithm Number ASP($G_F^2(x_{i_1} = 0)$),
   Algorithm Number ASP($G_F^1(x_{i_1} = 1) - \{c_{j_1}\}$),
   Algorithm Number ASP($G_F^1(x_{i_1} = 0) - \{c_{j_1}\}$),
   Algorithm Number ASP($G_F^2(x_{i_1} = 1) - \{c_{j_1}\}$) and
   Algorithm Number ASP($G_F^2(x_{i_1} = 0) - \{c_{j_1}\}$).

   (d) Let $N(G_F)$ be the number of all satisfying truth assignments of G_F and let $N(G_F^i(x_{i_1} = 1))$ be the number of all satisfying truth assignments of $G_F^i(x_{i_1} = 1)$, for $i = 1, 2$. Further let $N(G_F^i(x_{i_1} = 0))$ be the number of all satisfying truth assignments of $G_F^i(x_{i_1} = 0)$, for $i = 1, 2$. Moreover let $N(G_F^i(x_{i_1} = 1) - \{c_{j_1}\})$ resp. $N(G_F^i(x_{i_1} = 0) - \{c_{j_1}\})$ be the number of all satisfying truth assignments of $G_F^i(x_{i_1} = 1) - \{c_{j_1}\}$ resp. $G_F^i(x_{i_1} = 0) - \{c_{j_1}\}$, for $i = 1, 2$. Then we have:

$$
\begin{aligned}
N(G_F) = \max\{&N(G_F^1(x_{i_1} = 1) - \{c_{j_1}\}) \cdot N(G_F^2(x_{i_1} = 1)),\\
&N(G_F^2(x_{i_1} = 1) - \{c_{j_1}\}) \cdot N(G_F^1(x_{i_1} = 1))\}\\
+ \max\{&N(G_F^1(x_{i_1} = 0) - \{c_{j_1}\}) \cdot N(G_F^2(x_{i_1} = 0)),\\
&N(G_F^2(x_{i_1} = 0) - \{c_{j_1}\}) \cdot N(G_F^1(x_{i_1} = 0))\}
\end{aligned}
$$

Procedure NumberPath (P)
INPUT: A path P admitting n variable-vertices such that P begins and ends with

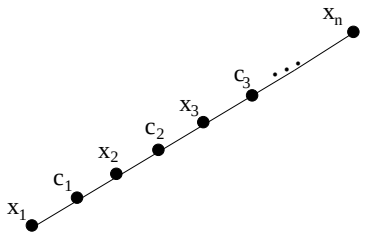


Figure 4.4: A path with n variable-vertices

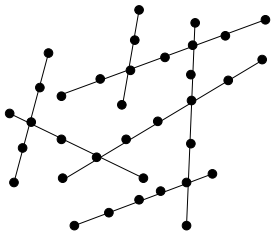


Figure 4.5: A tree.

a variable-vertex (cf. Figure 4.4).
OUTPUT: $N(P)$ = The number of all satisfying truth assignments of P .

1. Let $\alpha : \{x_2, \ldots, x_n\} \longrightarrow \{0, 1\}$ be the following truth assignment for P : For all $i = 2, \ldots, n$
$$\alpha(x_i) = \begin{cases} 1, & \text{if } x_i \text{ occurs in } c_{i-1} \\ 0, & \text{if } \overline{x_i} \text{ occurs in } c_{i-1} \end{cases}$$
α satisfies all clauses of P by definition. For satisfying P the value of the variable x_1 is not relevant, so we are allowed to set either $x_1 = 1$, or $x_1 = 0$. Hence we have:

2. Initially holds: $M(x_2 = \alpha(x_2)) = 2$ and $M(x_2 = \overline{\alpha(x_2)}) = 1$.

3. For $i = 3$ to n do
$M(x_i = \alpha(x_i)) = M(x_{i-1} = \alpha(x_{i-1})) + M(x_{i-1} = \overline{\alpha(x_{i-1})})$ and
$$M(x_i = \overline{\alpha(x_i)}) = \begin{cases} M(x_{i-1} = \alpha(x_{i-1})), & \text{if } x_{i-1} \text{ has the same} \\ & \text{polarity in } c_{i-1} \text{ as in } c_{i-2} \\ M(x_{i-1} = \overline{\alpha(x_{i-1})}), & \text{else.} \end{cases}$$

4. $N(P) = M(x_n = \alpha(x_n)) + M(x_n = \overline{\alpha(x_n)})$.

Procedure NumberTree (B)
INPUT: A formula whose graph is a tree B (and not only a path) of n variable-vertices (cf. Figure 4.5).
OUTPUT: $N(B)$:= Number of all satisfying truth assignments of B.

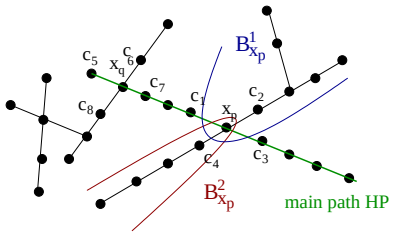


Figure 4.6: Every two paths may have only a variable-vertex in common

1. If B consists of paths which satisfy the following condition: Each pair of distinct paths is only allowed to have a variable-vertex in common and there is a variable-vertex x_p in B which is adjacent to at least three clause-vertices. An example is illustrated in Figure 4.6

   On this premise we proceed in the following way: Let HP be the main-path of B and let x_p be a variable-vertex of the main path which is adjacent to $l \geq 1$ clause-vertices $c_1, \ldots, c_l$ that do not lie on HP and let α be a satisfying truth assignment for HP. For every such variable-vertex x_p of the main path we proceed as follows: Let $B^1_{x_p}, \ldots, B^l_{x_p}$ be all the subtrees of B which only intersect HP in the variable-vertex x_p. We compute them recursively by **Algorithm Number ASP** $\left(B^i_{x_p=\alpha(x_p)}\right)$, and **Algorithm Number ASP** $\left(B^i_{x_p=\overline{\alpha(x_p)}}\right)$, for $i = 1, \ldots, l$. Let $N\left(B^i_{x_p=\alpha(x_p)}\right)$ resp. $N\left(B^i_{x_p=\overline{\alpha(x_p)}}\right)$ be the number of all satisfying truth assignments of $B^i_{x_p=\alpha(x_p)}$ resp. $B^i_{x_p=\overline{\alpha(x_p)}}$. By **Algorithm Number ASP** (HP) which calls **Procedure NumberPath**(HP) we deal with the main path. As soon as we reach x_p, we get one of the following equations:

$$M\left(x_p = \alpha(x_p)\right) = \left[M\left(x_{p-1} = \alpha(x_{p-1})\right) + M\left(x_{p-1} = \overline{\alpha(x_{p-1})}\right)\right] \cdot N\left(B^1_{x_p=\alpha(x_p)}\right) \cdot \ldots \cdot N\left(B^l_{x_p=\alpha(x_p)}\right)$$

   or

$$M(x_p = \overline{\alpha(x_p)}) = \begin{cases} M(x_{p-1} = \alpha(x_{p-1})) \cdot N(B^1_{x_p=\overline{\alpha(x_p)}}) \cdot \ldots \cdot N(B^l_{x_p=\overline{\alpha(x_p)}}); \text{if } x_{p-1} \text{ or } \bar{x}_{p-1} \text{ in } c_{p-1} \cap c_{p-2} \\ M(x_{p-1} = \overline{\alpha(x_{p-1})}) \cdot N(B^1_{x_p=\overline{\alpha(x_p)}}) \cdot \ldots \cdot N(B^l_{x_p=\overline{\alpha(x_p)}}); \text{else.} \end{cases}$$

2. If B contains a clause-vertex c_i which is adjacent to $k \geq 3$ variable-vertices $x_{i_1}, \ldots, x_{i_k}$, as illustrated in Figure 4.7, then we partition B into k subtrees $B_{x_{i_1}} \ldots, B_{x_{i_k}}$ such that every subtree is connected with the other $k - i$ subtrees only by the clause-vertex c_i. We write $B_{x_{i_j}}$ for the subtree including the vertex x_{i_j}, for $j = 1, \ldots, k$, without the clause-vertex c_i. For every $B_{x_{i_j}}$, for $j = 1, \ldots, k$, we compute the number of all satisfying truth assignments of $B_{x_{i_j}}$ as follows:

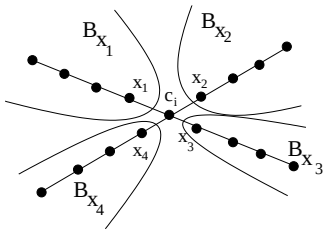


Figure 4.7: A clause-vertex adjacent to 4 variable-vertices

(a) If $B_{x_{i_j}}$ is a path, we compute the number of all satisfying truth assignments of $B_{x_{i_j}}$ by **Procedure NumberPath($B_{x_{i_j}}$)**.

(b) If $B_{x_{i_j}}$ is a tree and does not have a clause-vertex which is adjacent to at least three variable-vertices, $B_{x_{i_j}}$ has a variable-vertex x_p, which is adjacent to at least three clause-vertices. In this case we compute the number of all models of $B_{x_{i_j}}$ by applying (**1.**).

(c) If $B_{x_{i_j}}$ has a clause-vertex which is adjacent to at least three variable-vertices, then we compute the number of all models of $B_{x_{i_j}}$ by applying (**2.**).

(d) Let $N(B_{x_{i_j}})$ be the number of all models of $B_{x_{i_j}}$, for $j = 1, \ldots, k$. Then it holds:

$$N(B) = N(B_{x_{i_1}}) \cdot \ldots \cdot N(B_{x_{i_k}}) - N(B_{x_{i_1}\neg c_i}) \cdot \ldots \cdot N(B_{x_{i_k}\neg c_i})$$

Here $N(B_{x_{i_j}\neg c_i})$, for $j = 1, \ldots, k$, denotes the number of all models of $B_{x_{i_j}}$ where the variable x_{i_j} is set such that its assignment does not satisfy the clause c_i.

END

Remark 5. *1. The separator set in (5) may also consist of two variable-vertices or two clause-vertices.*

- *If the separator set consists of two variable-vertices x_{i_1}, x_{i_2}, we have to treat the following subgraphs:*

Algorithm Number ASP$(G_F^1(x_{i_1} = 1 \wedge x_{i_2} = 1))$,
Algorithm Number ASP$(G_F^1(x_{i_1} = 0 \wedge x_{i_2} = 1))$,
Algorithm Number ASP$(G_F^1(x_{i_1} = 1 \wedge x_{i_2} = 0))$,
Algorithm Number ASP$(G_F^1(x_{i_1} = 0 \wedge x_{i_2} = 0))$,
Algorithm Number ASP$(G_F^2(x_{i_1} = 1 \wedge x_{i_2} = 1))$,
Algorithm Number ASP$(G_F^2(x_{i_1} = 0 \wedge x_{i_2} = 1))$,
Algorithm Number ASP$(G_F^2(x_{i_1} = 1 \wedge x_{i_2} = 0))$ *and*
Algorithm Number ASP$(G_F^2(x_{i_1} = 0 \wedge x_{i_2} = 0))$.

Let $N(G_F^i(x_{i_1} = \delta_1 \wedge x_{i_2} = \delta_2))$, for $i = 1, 2$ and $\delta_1, \delta_2 \in \{0, 1\}$, be the number of all models of $G_F^i(x_{i_1} = \delta_1 \wedge x_{i_2} = \delta_2)$. For $N(G_F)$, the

number of all models of G_F, holds:

$$\begin{aligned} N(G_F) =& N(G_F^1(x_{i_1} = 1 \wedge x_{i_2} = 1))\cdot N(G_F^2(x_{i_1} = 1 \wedge x_{i_2} = 1)) \\ &+ N(G_F^1(x_{i_1} = 0 \wedge x_{i_2} = 1))\cdot N(G_F^2(x_{i_1} = 0 \wedge x_{i_2} = 1)) \\ &+ N(G_F^1(x_{i_1} = 1 \wedge x_{i_2} = 0))\cdot N(G_F^2(x_{i_1} = 1 \wedge x_{i_2} = 0)) \\ &+ N(G_F^1(x_{i_1} = 0 \wedge x_{i_2} = 0))\cdot N(G_F^2(x_{i_1} = 0 \wedge x_{i_2} = 0)) \end{aligned}$$

- *If the separator set consists of two clause-vertices c_{i_1}, c_{i_2}, we have to treat the following subgraphs:*

  Algorithm Number ASP$(G_F^1 - \{c_{i_1}\})$,
  Algorithm Number ASP$(G_F^1 - \{c_{i_1}, c_{i_2}\})$,
  Algorithm Number ASP$(G_F^1 - \{c_{i_2}\})$,
  Algorithm Number ASP(G_F^1),
  Algorithm Number ASP$(G_F^2 - \{c_{i_1}\})$,
  Algorithm Number ASP$(G_F^2 - \{c_{i_1}, c_{i_2}\})$,
  Algorithm Number ASP$(G_F^2 - \{c_{i_2}\})$ *and*
  Algorithm Number ASP(G_F^2).

  We get :

$$\begin{aligned} N(G_F) = \max\{&N(G_F^1 - \{c_{i_1}\})\cdot N(G_F^2 - \{c_{i_2}\}), N(G_F^1 - \{c_{i_1}, c_{i_2}\})\cdot N(G_F^2), \\ &N(G_F^1 - \{c_{i_2}\})\cdot N(G_F^2 - \{c_{i_1}\}), N(G_F^1)\cdot N(G_F^2 - \{c_{i_1}, c_{i_2}\})\}. \end{aligned}$$

2. *If we have a* CNF *formula F whose outerplanar graph G_F has $l > 1$ connected components $Z_1, \ldots, Z_l$, we do the following:*

   We apply the ***Algorithm Number ASP*** *on each single connected component. Let $N(Z_i)$ be the number of all models of the component Z_i, for $i = 1, \ldots, l$. Then $N(F) = N(Z_1)\cdot N(Z_2)\cdot \ldots \cdot N(Z_l)$ is the number of all models of F.*

The **Algorithm Number ASP()** previously described leads to:

Theorem 44. *The counting problem* #SAT *for outerplanar formulas with n variables is solvable in time $O(n^{5.13})$. For outerplanar formulas whose graph is either free of circles or consists of disjoint circles without chords we can solve* #SAT *in linear time.*

Proof. We establish the theorem by proving the correctness and stated time complexity of the Algorithm Number ASP() starting with the analysis of the running time: Let F be a CNF formula with n variables and a connected outerplanar graph G_F. If G_F is a tree, thus free of circles, we can calculate the number of all models of F in linear time. As we visit each vertex of G_F only once in the **Procedure NumberTree** as well as in the **Procedure NumberPath**, each of the two procedures takes linear running time. The same argument holds when G_F consists of pairwise disjoint circles without any chords because by setting an arbitrary variable of a circle without any chords, we obtain a path.

If G_F has circles, we treat G_F recursively by the separator-theorem: We determine two vertices x_{i_1} and c_{j_1} such that G_F is partitioned in two subgraphs G_F^1

and G_F^2 which only have the two vertices x_{i_1} and c_{j_1} in common and it holds that neither G_F^1 nor G_F^2 contain more than $\frac{2n}{3}$ variable-vertices. Further there is no edge connecting a vertex from $G_F^1 - \{x_{i_1}, c_{j_1}\}$ with a vertex from $G_F^2 - \{x_{i_1}, c_{j_1}\}$. From G_F^1 we get $G_F^1(x_{i_1} = 1)$ (resp. $G_F^1(x_{i_1} = 0)$), by setting the variable $x_{i_1} = 1$ (resp. $x_{i_1} = 0$)in G_F^1, eliminating all clauses, which are satisfied by the assignment $x_{i_1} = 1$ (resp. $x_{i_1} = 0$) and further eliminating x_{i_1} and all edges incident with x_{i_1}.

Likewise we obtain the subgraphs $G_F^2(x_1 = 1)$ and $G_F^2(x_1 = 0)$ from G_F^2. Next we build $G_F^1(x_1 = 1) - \{c_{j_1}\}$, $G_F^2(x_1 = 1) - \{c_{j_1}\}$, $G_F^1(x_1 = 0) - \{c_{j_1}\}$ and $G_F^2(x_1 = 0) - \{c_{j_1}\}$, by eliminating the clause vertex c_{j_1} from $G_F^1(x_{i_1} = 1)$, $G_F^2(x_{i_1} = 1)$, $G_F^1(x_{i_1} = 0)$ and $G_F^2(x_{i_1} = 0)$, in case this one is not satisfied yet through the setting of x_{i_1}. Next we apply the **Algorithm Number ASP** to the eight subgraphs:

Algorithm Number ASP($G_F^1(x_{i_1} = 1)$),
Algorithm Number ASP($G_F^1(x_{i_1} = 0)$),
Algorithm Number ASP($G_F^2(x_{i_1} = 1)$),
Algorithm Number ASP($G_F^2(x_{i_1} = 0)$),
Algorithm Number ASP($G_F^1(x_{i_1} = 1) - \{c_{j_1}\}$),
Algorithm Number ASP($G_F^1(x_{i_1} = 0) - \{c_{j_1}\}$),
Algorithm Number ASP($G_F^2(x_{i_1} = 1) - \{c_{j_1}\}$) and
Algorithm Number ASP($G_F^2(x_{i_1} = 0) - \{c_{j_1}\}$).

As soon as a subgraph is free of circles, we can compute the number of all its models in linear time by the **Procedure NumberTree (B)** or by the **Procedure NumberPath (P)**.

Let $T(n)$ be the running time to compute the number of all models of an outerplanar formula with n variables. Then we obtain the following recurrence for the running time:

$$T(1) = O(1)$$
$$T(n) = 8 \cdot T\left(\frac{2}{3}n\right) + O(n) + O(1), n > 1$$

Since we can determine the separator set for an outerplanar graph in linear time, we get $O(n)$ for the running time to determine a separator.

At each step of the recursion we obtain eight new subgraphs, to which we apply the **Algorithm Number ASP()**. As with every separation step the variable set has diminishes to at most 2/3 of the variable set of the previous graph the recursion tree has maximal depth l satisfying $(2/3)^l n = 1$. Therefore we have:

$$l = \log_{3/2}(n) = \frac{\log_2(n)}{\log_2(3/2)}$$

We need $O(1)$ running time to combine the solutions of the different subgraphs. Thus the solution of our recurrence is: $T(n) = n^{\log_{3/2} 8} = n^{5.13}$. Therefore $T(n) = O(n^{5.13})$ is the running time for #SAT for outerplanar formulas.

Now we consider the correctness of the **Algorithm Number ASP()**. Concerning **Procedure NumberPath(P)** we have the following invariant: $M(x_i = \alpha(x_i))$,

for $i = 2, \ldots, n$, denotes the number of all models assigning the variables $x_i, \ldots, x_n$ according to α and $M(x_i = \overline{\alpha(x_i)})$ denotes the number of all models assigning the variables $x_{i-1}, \ldots, x_n$ according to α and $x_i = \overline{\alpha(x_i)}$. Hence

$$N(P) = M(x_n = \alpha(x_n)) + M(x_n = \overline{\alpha(x_n)}$$

denotes the number of all models for the path P.

Concerning **Procedure NumberTree(B)** we similarly have the following invariant: If B is a tree for which holds that every two paths are only allowed to have a variable-vertex in common, we consider the main path (HP) and apply the **Algorithm Number ASP(HP)** to HP which then calls the **Procedure NumberPath(HP)** to apply to HP. For every variable x_p of the main path which is adjacent to $l > 2$ clause-vertices $c_1, \ldots, c_l$ we have the following invariant:

$$M(x_p = \alpha(x_p)) = \Big[M(x_{p-1} = \alpha(x_{p-1})) + M(x_{p-1} = \overline{\alpha(x_{p-1})})\Big] \cdot N(B^1_{x_p=\alpha(x_p)}) \cdot \ldots \cdot N(B^l_{x_p=\alpha(x_p)})$$

and

$$M(x_p = \overline{\alpha(x_p)}) = \begin{cases} M(x_{p-1} = \alpha(x_{p-1})) \cdot N(B^1_{x_p=\overline{\alpha(x_p)}}) \cdot \ldots \cdot N(B^l_{x_p=\overline{\alpha(x_p)}}); \text{if } x_{p-1} \text{ or } \bar{x}_{p-1} \text{ is in } c_{p-1} \cap c_{p-2} \\ M(x_{p-1} = \overline{\alpha(x_{p-1})}) \cdot N(B^1_{x_p=\overline{\alpha(x_p)}}) \cdot \ldots \cdot N(B^l_{x_p=\overline{\alpha(x_p)}}); \text{else.} \end{cases}$$

In other words we partition B into paths so that we only need to recursively treat each single path by the **Algorithm Number ASP** and finally to combine the number of the solutions of the common variables.

If B is a tree with a clause-vertex c which is adjacent to at least three variable-vertices, then we partition B into k subtrees, where k is the number of the variable-vertices which are adjacent to c, such that every two of the k subtrees have only the vertex c in common. Then we remove c from every subtree and compute for every subtree the number of all its models separately. Next we multiply all these numbers and subtract the number of all truth assignments not satisfying c.

If G_F is neither a path nor a tree, G_F must have a circle. Then we treat G_F by the divide and conquer strategy using the separator theorem. □

The next Theorem implicates that an upperbound for path formulas is provided by the Fibonaccci numbers.

Theorem 45. *Let F be an outerplanar* CNF*-formula consisting of n variables whose graph G_F is a path. Then F has at most*

$$\psi_{n+1} = \frac{1}{\sqrt{5}}\left(\phi^{n+1} - (1-\phi)^{n+1}\right)$$

many different satisfying truth assignments, where ψ_{n+1} is the $(n+1)$. th Fibonacci number and $\phi = \frac{1+\sqrt{5}}{2}$.

Proof. Let F be a CNF Formula with n variables whose graph G_F is a path P beginning and ending with a variable-vertex. Then F is satisfiable and F has the

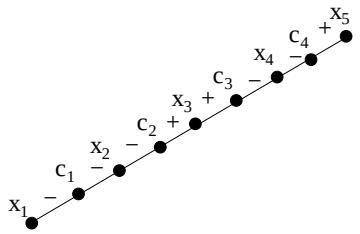


Figure 4.8: Pure literals path

largest possible number of truth assignments if it only consists of pure literals. An example is illustrated in Figure 4.8.

We compute the number of all models of G_F by the **Procedure NumberPath** (G_F). Since every variable x_{i-1} occurs with the same polarity in c_{i-1} as in c_{i-2}, it holds:

$$M(x_i = \overline{\alpha(x_i)}) = M(x_{i-1} = \alpha(x_{i-1}))$$

We set $\psi_0 := 1$. It holds $M(x_2 = \overline{\alpha(x_2)}) = 1 =: \psi_1$, $M(x_2 = \alpha(x_2)) = 2 =: \psi_2$. Further, we define: $\psi_i := M(x_i = \alpha(x_i))$, for $i = 1, \ldots, n$. We recursively obtain for $i = 3, \ldots, n+1$:

$$\begin{aligned}\psi_i = M(x_i = \alpha(x_i)) &= M(x_{i-1} = \alpha(x_{i-1})) + M(x_{i-1} = \overline{\alpha(x_{i-1})}) \\ &= M(x_{i-1} = \alpha(x_{i-1})) + M(x_{i-2} = \alpha(x_{i-2})) \\ &= \psi_{i-1} + \psi_{i-2}\end{aligned}$$

Thus we obtain:

$$\begin{aligned}N(P) &= M(x_n = \alpha(x_n)) + M(x_n = \overline{\alpha(x_n)}) \\ &= M(x_n = \alpha(x_n)) + M(x_{n-1} = \alpha(x_{n-1})) \\ &= \psi_n + \psi_{n-1} = \psi_{n+1}\end{aligned}$$

So F has

$$N(P) = \psi_{n+1}$$

many satisfying truth assignments. The ψ_i, for $i = 1, \ldots, n+1$, are the famous Fibonacci-numbers.
It is well known that $\psi_n = \frac{\phi^n - (1-\phi)^n}{\sqrt{5}}$, where $\phi = \frac{1+\sqrt{5}}{2}$. Thus we get

$$N(P) = \frac{1}{\sqrt{5}} \left(\phi^{n+1} - (1-\phi)^{n+1}\right)$$

□

Theorem 46. *Let F be an outerplanar formula consisting of n variables whose graph G_F is a circle without any secants. Then F has at most*

$$\frac{1}{\sqrt{5}} \left(\phi^n - (1-\phi)^n\right) + \frac{1}{\sqrt{5}} \left(\phi^{n-2} - (1-\phi)^{n-2}\right)$$

many different satisfying truth assignments.

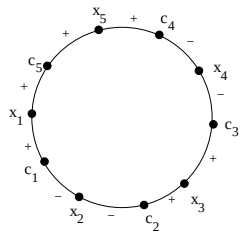


Figure 4.9: Pure literals circle

Proof. Let F be a CNF formula with n variables whose graph G_F is a circle without any chords. Then F is satisfiable and has the largest possible number of truth assignments if it only consists of pure literals. An example is illustrated in 4.9. We set $x_1 = 1$ resp. $x_1 = 0$ and compute the number of all satisfying truth assignments of $G_{F_{x_1=1}}$ and $G_{F_{x_1=0}}$ with the **Algorithm Number ASP** ($G_{F_{x_1=1}}$) and by the **Algorithm Number ASP** ($G_{F_{x_1=0}}$) which calls the **Procedure NumberPath**.
W.l.o.g. let x_1 only occur as a positive literal in F. If we set $x_1 = 1$, the clauses c_1 and c_n are satisfied and we obtain a path $P_1 = x_2 - c_2 - \ldots - c_{n-1} - x_n$ with $n-1$ variables. According to Theorem 45 this path has exactly $N(P_1) = \psi_n$ many satisfying truth assignments. We further set $x_1 = 0$. With this assignment neither the clause c_1 nor the clause c_n are satisfied. Thus we have to set the variable x_2 so, that its assignment satisfies the clause c_1 and the variable x_n so, that its assignment satisfies the clause c_n. Since every variable only occurs as a pure literal, the clauses c_2 and c_{n-1} are also satisfied by the assignment of x_2 and x_n. We remove these and obtain a path $P_2 = x_3 - c_3 - \ldots - x_{n-1}$ of $n-3$ variables.
According to Theorem 45 this path has exactly $N(P_2) = \psi_{n-2}$ many models. In summary the number of models for F therefore is:

$$N(K) = N(P_1) + N(P_2) = \psi_n + \psi_{n-2}$$

□

4.3 The Case of k-Outerplanar Formulas and the Separator Theorem of Lipton & Tarjan

Aiming to treat #SAT for CNF formulas with k-outerplanar graphs, for $k \geq 2$, we now extend the concepts of the previous section.

Definition 8. *Let k-ASP be the class of* CNF *formulas with n variables whose graph is k-outerplanar, for a $k > 1$.*

A k-outerplanar graph can easily be partitioned into two subgraphs with at most $2k$ common separator vertices as shown by B.S. Baker. This is a result of the separator theorem of Lipton & Tarjan:

Lemma 18 ([3])**.** *Let G be any n-vertex k-outerplanar graph. The vertices of G can be partitioned into three sets A, B, C such that no edge joins a vertex in A with*

a vertex in B, neither A nor B contain more than $2n/3$ vertices, and C contains no more than $2k$ vertices. Such a separator set can be computed in linear time.

As we are interested in the number of all satisfying truth assignments of a k-outerplanar CNF formula F, we consider the following algorithm, which solves our problem:

Algorithm Number k-ASP (F)

INPUT: $F \in k$-ASP,
OUTPUT: $N(F) :=$ number of all models for F.
BEGIN

1. If F is 1-outerplanar, we compute the number of all models of F by the **Algorithm Number ASP**(F).
2. As long as there is a clause-vertex c_j in G_F which is adjacent to only one variable-vertex x_i, we set x_i so that its assignment satisfies the clause c_j, and we simplify the formula by removing all the clause-vertices satisfied by the assignment of x_i. Further we remove the vertex x_i and all edges incident with it from G_F.
3. If there is an isolated clause-vertex (i.e. an empty clause), the formula is unsatisfiable and the procedure terminates with the output $N(G_F) = 0$.
4. - Let $G_F = (V(G_F), E)$ be the k-outerplanar graph of F. We determine the separator set $S = \{x_1, \ldots, x_l\}$ of G_F, $l \leq 2k$ such that the vertex set $V(G_F)$ of G_F can be divided into three sets V_1, V_2 and S with the following feature: There is no edge joining a vertex of V_1 with a vertex of V_2 and $|V_1|, |V_2| \leq \frac{2n}{3}$.
   - Next we consider both subgraphs of G_F which we obtain by adding the separator vertices $x_1, \ldots, x_l$ to V_1 resp. V_2 and admitting all the edges between those vertices which were present between those variables in G_F. We obtain: $G_1 := G(V_1 \cup \{x_1, \ldots, x_l\})$ and $G_2 := G(V_2 \cup \{x_1, \ldots, x_l\})$. Here $V_1 \cup \{x_1, \ldots, x_l\}$ resp. $V_2 \cup \{x_1, \ldots, x_l\}$ is the vertex set of G_1 resp. G_2, and the edge set of G_1 resp. G_2 is E restricted to $V_1 \cup \{x_1, \ldots, x_l\}$ resp. $V_2 \cup \{x_1, \ldots, x_l\}$. Let F^1 and F^2 be the subformulas of F whose graphs correspond to G_1 resp. G_2.
   - Let $\phi_1, \ldots, \phi_{2^l}$ be all distinct truth assignments for the variables $x_1, \ldots, x_l$. For each fixed ϕ_j, $1 \leq j \leq 2^l$, the number of all models of $F^i_{\phi_j(x_1,\ldots,x_l)}$, for $i = 1, 2$, is computed by the **Algorithm Number k-ASP** $(F^i_{\phi_j(x_1,\ldots,x_l)})$. Here $F^i_{\phi_j(x_1,\ldots,x_l)}$ denotes the evaluation of F^i according to ϕ_j: All satisfied clauses in F_i are removed from F_i. Furthermore the variables $x_1, \ldots, x_l$ are removed from all remaining clauses.
   - Let $N(F^i_{\phi_j(x_1,\ldots,x_l)})$ be the number of all models of $F^i_{\phi_j(x_1,\ldots,x_l)}$, for $i = 1, 2$ and $1 \leq j \leq 2^l$. Then we have:

$$N(F) = \sum_{j=1}^{2^l} \left(N(F^1_{\phi_j(x_1,\ldots,x_l)}) \cdot N(F^2_{\phi_j(x_1,\ldots,x_l)}) \right)$$

END

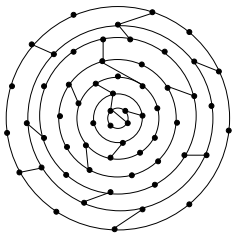


Figure 4.10: A 7-circular-levelplanar graph

Remark 6. *In the **Algorithm Number k-ASP** we assume that the separator set S consists of variable vertices only but this is not necessarily always the case. In case a clause vertex is contained in S we proceed as in **Algorithm Number ASP** by extending it to l, $l \leq 2k$, vertices.*

Theorem 47. ***Algorithm Number** k**-ASP** needs $O(n^{1.7(2k+1)})$ time to compute the number of all models for a k-outerplanar formula F with n variables, where $k \geq 1$ is a fixed integer.*

Proof. Let $T(n)$ be the running time needed to compute the number of all models of an arbitrary k-outerplanar formula F (with a constant k) with n variables by the separator theorem:

Then we obtain the following recurrence:

$$T_k(1) = O(1) \tag{4.1}$$

$$T_k(n) = 2^{2k+1} T_k\left(\frac{2}{3}n\right) + O(n) + O(2^{2k+1}), n > 1 \tag{4.2}$$

For fixed values of k equation (4.2) above reduces to

$$T_k(n) = 2^{2k+1} T_k\left(\frac{2}{3}n\right) + O(n)$$

whose solution

$$T_k(n) = O\left(n^{\log_{3/2}(2^{2k+1})}\right) = O\left(n^{\frac{2k+1}{\log_2(3/2)}}\right) = O\left(n^{1.7(2k+1)}\right)$$

is obtained from the Master Theorem for recurrence equations. Thus it holds: $T_k(n) = T_k^{'}(n) - O(1) = O\left(n^{1.7(2k+1)}\right)$. Therefore the running time for #SAT for k-outerplanar formulas is $O(n^{1.7(2k+1)})$. That means, for constant k, #SAT for k-outerplanar formulas is solvable in polynomial-time. □

Next we consider an important subclass of the class of k-outerplanar formulas, namely the class of k-circular-levelplanar formulas for which #SAT can be solved faster.

Definition 9. *Let G be a k-outerplanar graph that is constructed as follows: G consists of k circles (free of secants apart from the smallest circle inside), which are encapsulated into each other. Here edges do only exist between adjacent circles. We call such a graph k-circular-levelplanar(see figure (4.10)).*

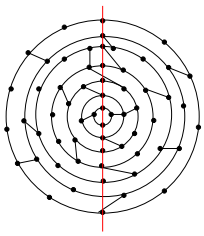


Figure 4.11: Separation of a 7-circular-levelplanar graph

Definition 10. *We call a* CNF *formula F k-circular-levelplanar if its graph G_F is k-circular-levelplanar.*

For this specific formula class we obtain the following:

Lemma 19. *Let F be a k-circular-levelplanar formula. Then F is satisfiable.*

Proof. In the first section we have already proven that a CNF formula whose graph is a ring (that is a circle without secants) is always satisfiable. Accordingly, we can satisfy each of the rings and the edges between the rings do not influence the satisfiability. □

Lemma 20. *Let G be a k-circular-levelplanar graph consisting of n vertices. Then the following holds: Partitioning G by a separator set of $|S| = 2k$, as shown in figure 4.11, into two subgraphs G_1 and G_2, for which holds, that G_1 as well as G_2 do not have more than $2n/3$ vertices and there is no edge joining a vertex from G_1 with a vertex from G_2, then G_1, G_2 are at most $(\lceil k/2 \rceil)$-circular-levelplanar.*

Proof. If we slice G such that every circle is being sliced at exactly two vertices as depicted in Figure 4.11, then G is obviously partitioned into two subgraphs G_1 and G_2, that are at most $(\lceil k/2 \rceil)$-circular-levelplanar. □

The next theorem states that k-circular-levelplanar formulas can be solved faster than k-outerplanar formulas.

Theorem 48. *For k-circular-levelplanar formulas, where $k \geq 1$, #SAT is solvable in time $O(k \cdot 16^k (2/3)^{5.13 \cdot \log_2 k} n^{5.13})$. So the class of k-circular-levelplanar formulas belongs to the class FPT.*

Proof. Let F be a k-circular-levelplanar CNF formula with n variables. Then we can solve #SAT for F faster using the **Algorithm Number k-ASP**.

Using Lemma 20 we get: In every separation step the level of the outerplanarity of F is bisected. After $\log_2 k$ separation steps we have separated F into 1-outerplanar formulas and every one of them does not have more than $(2/3)^{\log_2 k} n$ variables. Each of these formulas we can solve in $O(((2/3)^{\log_2 k} n)^{5.13})$ running time according to Theorem 44.

After $\log_2 k$ separation steps we have to treat

$$2 \cdot 2^{2k} \cdot 2 \cdot 2^{2(\lceil k/2 \rceil)} \cdot 2 \cdot 2^{2(\lceil k/4 \rceil)} \cdot \ldots \cdot 2 \cdot 2^{2 \cdot 2} = O(k \cdot 2^{4k})$$

many $(\lceil k/(2^{\log_2 k})\rceil)$-circular-levelplanar (that is 1-outerplanar) subformulas.

So we have $O(k\cdot 2^{4k})$ many 1-outerplanar formulas and thus a running time of $O(k\cdot 2^{4k}((2/3)^{\log_2 k}n)^{5.13}) = O(k\cdot 16^k(2/3)^{5.13\cdot\log_2 k}n^{5.13})$, to solve a k-circular-levelplanar CNF formula with n variables. □

Next we consider Knuth's nested formulas for which SAT can be decided in linear time [27]. We show that #SAT can be solved in polynomial-time $O(n^{8.5})$ for the class of nested formulas.

Definition 11 ([27]). *Let X be a finite alphabet linearly ordered by $<$; we think of the elements of X as boolean variables. The linear ordering of X can be extended to a linear preordering of all its literals in a natural way if we simply disregard the signs. If $X = \{a, b, c\}$ has the usual alphabetic order, we have*

$$a \equiv \overline{a} < b \equiv \overline{b} < c \equiv \overline{c}$$

If σ and τ are literals, we write $\sigma \leq \tau$ if $\sigma < \tau$ or $\sigma \equiv \tau$.
We say that a clause C straddles $C^{'}$ if there are literals σ, τ in C and $\psi^{'}$ in $C^{'}$, such that:

$$\sigma < \psi^{'} < \tau$$

Two clauses overlap if they straddle each other. A set of clauses in which no two clauses overlap is called nested. A nested formula is a CNF *formula which consists of nested clauses only. [27]*

In [27] it has been proved that the class of nested formulas can be SAT-decided in linear time:

Theorem 49 ([27]). *Let F be a nested formula. Then* SAT *can be decided in linear time for F.*

For our purposes the next result, derived by J. Kratochvil and M. Krivanek, turns out to be useful.

Lemma 21 ([31]). *A formula $F = c_1 \wedge c_2 \wedge \ldots \wedge c_n$ is nested if the variables allow an ordering $X = \{x_1, x_2, \ldots, x_m\}$ such that the graph*

$$\begin{aligned} G_F^V &= (X \cup \{c_1, c_2, \ldots, c_n\}, \{\{x, c_i\}; x \in c_i \text{ or } \overline{x} \in c_i\} \\ &\quad \cup \{\{x_i, x_{i+1}\}; i = 1, 2, \ldots, m\}) \end{aligned}$$

allows a noncrossing drawing in the plane so that the circle $x_1, x_2, \ldots x_m$ bounds the outer face.

As an example consider Figure 4.12. Obviously the graph G_F of a nested formula F which we obtain from G_F^V by eliminating the edges between the variable-vertices in G_F^V is at most 2-outerplanar. This can easily be achieved by removing all the vertices from the outer face of G_F: Removing all the vertices of the outer face particularly means removing all the variable-vertices and hence we have only clause-vertices left. Since there is no edge between two clause-vertices the remaining graph only consists of single vertices without any edges and is thus clearly 1-outerplanar.
Thus an arbitrary nested formula is at most 2-outerplanar and according to Theorem (47) we have:

Corollary 7. *The counting problem* #SAT *can be solved in time $O(n^{8.5})$ for the class of nested formulas with the separator theorem of Lipton and Tarjan.*

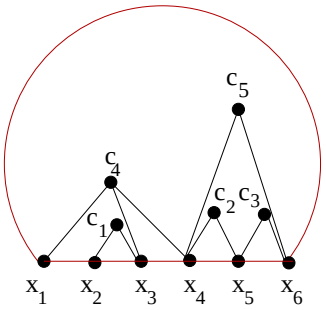


Figure 4.12: The graph of a nested formula

4.4 Solving #SAT for k-Outerplanar Formulas by a Nice Tree Decomposition

Recently, tree decompositions of graphs have received considerable interest in practical applications. In this section we present an algorithm which solves the #SAT problem for a k-outerplanar formula F in linear time by using the technique of a *nice tree decomposition* of its variable-clause graph G_F. Before we describe our algorithm we introduce some notation.
Let $G = (V, E)$ be a graph. A *tree decomposition* of G is a pair $(\{X_i | i \in I\}, T)$, where each X_i, $i \in I$, is a subset of V, called a *bag*, and T is a tree with the elements of I as nodes. The following three properties must hold:

1. $\bigcup_{i \in I} X_i = V$.
2. For every edge $\{u, v\} \in E$, there is an $i \in I$ such that $\{u, v\} \subseteq X_i$.
3. For all $i, j, k \in I$, if j lies on the path between i and k in T, then $X_i \cap X_k \subseteq X_j$.

The *width* of $(\{X_i | i \in I\}, T)$ equals $\max\{|Xi| | i \in I\} - 1$. The *treewidth* of G is the minimum w such that G has a tree decomposition of width w [9].
To better illustrate this definition, we distribute the vertices of G on the *bags* X_i, $i \in I$, which are the vertices of the tree $T = (I, F)$, such that the following holds:

- Each vertex of V is contained in at least one bag X_i, for a $i \in I$.
- The two vertices of each edge are together in at least one bag.
- For each vertex $v \in V$ the bags containing it form a subtree of T.

A tree decomposition intuitively represents how close a graph G is to a tree. Bodlaender observed that each k-outerplanar graph has a tree decomposition of width at most $3k - 1$ and his analysis implicitly leads to an $O(kn)$ time algorithm for computing such a tree decomposition [8].
A tree decomposition $(\{X_i | i \in I\}, T)$ with a particularly simple (and useful in regard of dynamic programming, cf. [1, 9]) structure is given by the following definition.
A tree decomposition is called a *nice tree decomposition* if the following conditions are satisfied: Every node i of the tree T has at most two children; if a node i has two children j and j', then $X_i = X_j = X_{j'}$; and if a node i has one child j, then either $|X_i| = |X_j| + 1$ and $X_j \subset X_i$ or $|X_i| = |X_j| - 1$ and $X_i \subset X_j$. More precisely, a *nice*

tree decomposition for a graph $G = (V, E)$, as introduced by H.L. Bodlaender and T. Kloks in [7], is a tree decomposition (X, T) of G with the following properties:

1. T is a rooted tree.
2. T has only four different types of nodes i:
   - **Leaf**: i has no children, it is the leaf of the tree T.
   - **Join**: i is a node with two children j, j' and $X_i = X_j = X_{j'}$.
   - **Forget**: i is a node with only one child j such that $X_i = X_j - \{v\}$, for some vertex $v \in X_j$.
   - **Introduce**: i is a node with only one child j such that $X_i = X_j \cup \{v\}$, for some vertex $v \notin X_j$.

Note that in a nice tree decomposition T one can always achieve that $|X_i| = 1$ for each leaf node i in T. Hence the bag X_i of each leaf node i consists of one vertex only. Assume there is a leaf node i in T whose corresponding bag X_i contains two vertices: $X_i = \{x_1, x_2\}$, then we can expand the tree T such that $X_{i_1} = \{x_1\}$ is now the only child of X_i. Now i is an introduce node and i_1 a leaf node in T whose bag contains exactly one vertex. So, in the following we assume that the bag of each leaf node in a nice tree decomposition contains exactly one vertex.
It is not hard to transform a given tree decomposition into a nice tree decomposition. A tree decomposition of width w with n nodes can be converted into a nice tree decomposition of width w with $O(n)$ nodes in time $O((n) \cdot poly(k))$ [26](Lemma 13.1.2, 13.1.3).

Now we are able to present an algorithm which solves the #SAT problem for a k-outerplanar formula F in linear time by using a *nice tree decomposition* of its variable-clause graph G_F. Tree decomposition based algorithms usually proceed in two phases. First, given some input graph, a tree decomposition of bounded width is constructed. Second, one solves the given problem (such as #SAT) using dynamic programming. Dynamic programming to solve the optimisation version of a problem again proceeds in two phases: first bottom-up from the leaves to the root and then top-down from the root to the leaves in order to construct the solution. To do this two-phase dynamic programming, however, one has to store all dynamic programming tables, each of them corresponding to a bag of the tree decomposition. Each bag X_i usually leads to a table that is exponential in its size [5]. Here our sole interest lies in the decision version of the #SAT problem, so performing only the first phase is sufficient, namely bottom-up from the leaves to the root, in the dynamic programming of our algorithm.

Algorithm NTD

Input: k-outerplanar Formula F with n variables; its corresponding variable-clause graph G_F.
Output: Number of models of F, if F is satisfiable. Else: 0.
begin

1. Compute a nice tree decomposition $(\{X_i | i \in I\}, T)$ of G_F of width w, where $w \leq 3k - 1$.
2. For each bag X_i, $i \in I$, compute all $2^{|X_i|}$ assignments α of values 1 or 0 to the vertices in X_i to gain all possible models for $G[X_i]$, the subgraph of G_F

which is induced by the vertices in the bags of all descendants of i. Store all the $2^{|X_i|}$ assignments α for the vertices of X_i in a table A_i, for all $i \in I$.

3. For each assignment α in A_i consider a numerator $n_i(\alpha) \in \mathbb{N}$ that stores the number of truth assignments for the vertices in $G[X_i]$ such that the following holds:
   - If $y \in X_i$ is a variable vertex, y is true iff $\alpha(y) = 1$.
   - If $y \in X_i$ is a clause vertex, y is satisfied iff $\alpha(y) = 1$.
   - All the clauses in $G[X_i]$ apart from the clauses in X_i are already satisfied.
4. Go bottom-up from the leaves to the root and for each node i in T adjust the numerators of A_i as follows:
   (a) In case $i \in T$ is a leaf node with $X_i = \{y\}$, we obtain the following for the numerator $n_i(\alpha)$ belonging to an assignment α in A_i:

$$n_i(\alpha) = \begin{cases} \mathbf{1}, \text{if } y \text{ is a variable vertex or a clause vertex with } \alpha(y) = 0 \\ \mathbf{0}, \text{if } y \text{ is a clause vertex and } \alpha(y) = 1. \end{cases}$$

   (b) In case $i \in T$ is a join node, then i has exactly two children j, j' such that $X_i = X_j = X_{j'}$. For the numerator $n_i(\alpha)$ of each truth assignment α in A_i we obtain: $n_i(\alpha) = n_j(\alpha) \cdot n_{j'}(\alpha)$, where $n_j(\alpha)$ resp. $n_{j'}(\alpha)$ are the numerators in A_j resp. $A_{j'}$ for the truth assignment α.
   (c) In case $i \in T$ is an introduce node, then $X_i = X_j \cup \{y\}$, where j is the only child of i and $y \notin X_j$. Let $\alpha \in A_i$ be a truth assignment to the vertices in X_i, then we have to distinguish the following two cases:
      - If y is a clause vertex, then $n_i(\alpha) = 0$, if additionally:
        - $\alpha(y) = 0$ although the assignment α to the variable vertices in X_i satisfies clause y.
        - $\alpha(y) = 1$ although the assignment α to the variable vertices in X_i does not satisfy clause y.

        Otherwise $n_i(\alpha) = n_j(\beta)$, where the assignment β assigns the same truth values to the vertices in X_j as α.
      - If y is a variable vertex, $n_i(\alpha) = 0$ if there is a clause vertex c for which holds $\alpha(c) = 0$, although $\alpha(y)$ satisfies c. Otherwise let C_i be the set of all clause vertices in X_i which are satisfied by $\alpha(y)$ alone. Then $n_i(\alpha)$ is the sum of all numerators $n_j(\beta)$ in A_j, where β assigns the same truth values as α to the vertices in $X_j - C_i$ and is free to assign any value to the vertices in C_i.
   (d) In case $i \in T$ is a forget node, that is $X_i = X_j - \{y\}$, where j is the only child of X_i and $y \in X_j$, we have to distinguish the following two cases:
      - If y is a clause vertex, then $n_i(\alpha) = 0$, for all truth assignments α in A_i that assign $y = 0$. Further, $n_i(\alpha) = n_j(\beta)$, where β assigns the same values to the vertices in X_i as α and sets $y = 1$.

- If y is a variable vertex, then for each numerator $n_i(\alpha)$ of a truth assignment α in A_i holds: $n_i(\alpha) = n_j(\beta) + n_j(\gamma)$, where β, γ are truth assignments in A_j which assign the same values as α to the vertices in X_i and set $\beta(y) = 1$, $\gamma(y) = 0$.

5. Let r be the root of T, then $G[X_r] = G_F$ and we get the total number of models for F by summing up all the numerators in A_r

end

Now, the analysis of the running time of Algorithm NTD is quite easily accomplished: Let F be a k-outerplanar formula, where $k \in \mathbb{N}$ is a constant, with variable-clause graph G_F and let n be the number of variables in F. Then we can compute a nice tree decomposition $(\{X_i | i \in I\}, T)$ of G_F of width w in linear time [26](Lemma 13.1.2, 13.1.3), where $w \leq 3k - 1$. There are $O(n)$ nodes in T and Algorithm NTD obviously needs constant time at each node of T, that is at each X_i, $i \in I$, because each bag X_i has size $\leq 3k - 1$. Hence the running time of Algorithm NTD is $O(n)$.

To prove the correctness of Algorithm NTD it is sufficient to prove the correctness of the computation of the numerators $n_i(\alpha)$, for each of the four different types of nodes i in T. Note that every numerator $n_i(\alpha)$ of a node $i \in T$ and an arbitrary assignment α to the vertices of X_i corresponds to the number of all possible assignments to the variables in $G[X_i]$, the subgraph of G_F induced by all vertices in the bags of all descendants of i, which do not belong to variables in X_i.

- In case $i \in T$ is a leaf node with $X_i = \{y\}$, we have to distinguish the following two cases: If y is a variable vertex, then α_1, α_2 are the only two truth assignments, where $\alpha_1(y) = 1$ and $\alpha_2(y) = 0$. As there are no clauses to satisfy we obtain $n_i(\alpha_1) = n_i(\alpha_2) = 1$ because both assignments are possible. If y is a clause vertex, then let α_1, α_2 be two truth assignments, where $\alpha_1(y) = 1$ and $\alpha_2(y) = 0$. As there are no variable vertices in X_i the clause y is not satisfied yet, hence $n_i(\alpha_1) = 0$ and $n_i(\alpha_2) = 1$, because α_2 assigns 0 to y.

- In case $i \in T$ is a join node then it is possible that there are vertices in $G[X_j]$ which do not occur in $G[X_{j'}]$ and vice versa. Assume that there is a vertex x in the subgraph $G[X_j]$, that does not occur in X_j. Then x does not occur in any ancestor bag of X_j and not in $G[X_{j'}]$, either, because for each vertex $x \in V(F)$ the bags containing it form a subtree of T. Hence the number of truth assignments consistent to α must be the product of the corresponding numerators in X_j and in $X_{j'}$, because we have to combine the truth assignments of $G[X_j]$ and $G[X_{j'}]$.

- In case $i \in T$ is an introduce node, then $X_i = X_j \cup \{y\}$, where j is the only child of i and $y \notin X_j$. Let $\alpha \in A_i$ be a truth assignment to the vertices in X_i. Then we have to distinguish the following two cases:

  - If y is a clause vertex, then $n_i(\alpha) = 0$, if:

    * $\alpha(y) = 0$, although the assignment α to the variable vertices in X_i satisfies clause y.

  - $*$ $\alpha(y) = 1$, although the assignment α to the variable vertices in X_i does not satisfy clause y.

  This is due to the fact that the clause y does not occur in any descendant bag of X_i. So this condition is sufficient since the variables in $G[X_j]$ have no influence on whether y is satisfied or not. Otherwise $n_i(\alpha) = n_j(\beta)$, where β assigns the same truth values to the vertices in X_j as α.

- If y is a variable vertex, then $n_i(\alpha) = 0$ if there is a clause vertex c for which holds $\alpha(c) = 0$, although $\alpha(y)$ satisfies c. Otherwise let C_i be the set of all clause vertices in X_i which are satisfied by $\alpha(y)$ alone. Then $n_i(\alpha)$ is the sum of all numerators $n_j(\beta)$ in A_j, where β assigns the same truth values to the vertices in $X_j - C_i$ as α and is free to assign any value to the vertices in C_i.
  This results from the fact that the clause vertices in C_i are all satisfied by $\alpha(y)$, that is the other variables in X_i have no influence on whether or not the clauses in C_i are satisfied as long as $y = \alpha(y)$ is assigned. This is why we are now free to assign any truth values to clauses in C_i. Note that C_i may also be the empty set, in this case we obtain $n_i(\alpha) = n_j(\beta)$, where β assigns the same truth values to the variables in X_j as α.

- Assume $i \in T$ is a forget node, i.e. $X_i = X_j - \{y\}$, where j is the only child of X_i and $y \in X_j$. Then we have to distinguish the following two cases:
  - If y is a clause vertex, then $n_i(\alpha) = 0$, for all truth assignments α in A_i that assign $y = 0$. Further, $n_i(\alpha) = n_j(\beta)$, where β assigns the same values as α to the vertices in X_i and sets $y = 1$.
    As a consequence of the definition of a tree decomposition y does not occur in any ancestor bag of X_j, thus y can only be satisfied by the variable vertices in $G[X_j]$. That is why we do not count truth assignments α in A_i which do not satisfy the clause y. As X_i and X_j have the same vertices apart from vertex y, we consider all truth assignments in A_j which assign $y = 1$ and set the numerator of A_i equal to the numerator of A_j for these assignments.
  - If y is a variable vertex, then each numerator $n_i(\alpha)$ of a truth assignment α in A_i is the sum of two numerators of A_j: $n_i(\alpha) = n_j(\beta) + n_j(\gamma)$, where β, γ are truth assignments in A_j which assign the same values to the vertices in X_i as α and $\beta(y) = 1$, $\gamma(y) = 0$.
    This numeration is clear, since the variable vertex y does not occur in any ancestor bag of X_i (that is y occurs only in $G[X_j]$) as a consequence of the definition of a tree decomposition and thus does not have any influence on the satisfiability of clauses in X_i and on clauses in the ancestor bags of X_i in T. Hence, for each assignment α of truth values to the variables of X_i, we have to consider the assignments β and γ and calculate the sum of their numerators.

Remark 7. *By using a nice tree decomposition of the variable-clause G_F of an outerplanar formula F we can also solve* SAT *for F in linear time. The corresponding algorithm is much shorter than Algorithm ASP as presented in the first section of this chapter, but does not exploit the structure of this formula class.*

Chapter 5

Outlook and Open Problems

We have provided a more systematical insight into MHF as destination class and thus illustrated that MHF has a central relevance in CNF. However, some challenging questions remain unanswered and should be investigated in the future. As already mentioned in the introduction there is an interesting connection between MHF-SAT and unrestricted SAT presented in [43]: If there is some $\alpha < \frac{1}{2}$ such that each MHF $M = P \wedge H$, where P has $k \leq 2n$ variables, can be solved in time $O(\|M\|2^{\alpha k})$, then there is some $\beta \leq 2\alpha < 1$ such that SAT for an arbitrary CNF-formula F can be decided in time $O(\|F\|2^{\beta n})$. Here $||F||$ denotes the length of F. Although, recently there has been some progress in finding non-trivial bounds for SAT for arbitrary CNF formulas [17, 18], it would require a significant breakthrough in our understanding of SAT to obtain upper time bounds of the form $O(2^{(1-\epsilon)n})$, for some $\epsilon > 0$. So the question remains whether an $\alpha < \frac{1}{2}$ exists such that each MHF $M = P \wedge H$, where P has $k \leq 2n$ variables, can be solved in time $O(\|M\|2^{\alpha k})$.

A further question is whether there exists a better polynomial-time reduction from CNF-SAT to MHF-SAT than the reduction presented in [43] where the number of variables is doubled in worst-case. So, is there a clever polynomial-time reduction from CNF-SAT to MHF-SAT for which the number of variables of an arbitrary CNF formula F is n and the number of variables of its corresponding SAT-equivalent formula $F' \in$ MHF is $< 2n$ in the worst-case? This problem appears extremely challenging.

We have presented the algorithm COUNT which solves SAT for formulas in $LMH^k_- F^{d+}$. Experimental results lead to the strong conjecture that its running time is better than $O((\sqrt[3]{3})^n)$, where n is the number of variables. Unfortunately, we were not able to analytically calculate its running time, so this continues to pose a challenging task for future work.

The question whether SAT can be solved for unrestricted formulas in MHF faster than in time $O((\sqrt[3]{3})^n)$ should be analysed in the future, too.

In this thesis we have presented several polynomial-time MHF subclasses. So subsequently the question arises whether one can find further interesting polynomial-time MHF subclasses which we have not considered yet.

We have only considered the satisfiability problem for MHF. Obviously MHF-XSAT as well as MHF-NAE-SAT remain NP-complete as one can easily see by applying the polynomial-time reduction presented in [43]. So from the point of

view of exact algorithmics, it would be desirable to gain progress for XSAT restricted to mixed Horn formulas beyond the so far best bound of $O(2^{0.2325 \cdot n})$, for unrestricted CNF-XSAT over n variables, provided by Byskov et al. [12].

We have illustrated that SAT remains NP-complete for the class k-BLMHF, $k \geq 3$, and we have presented an algorithm which solves SAT for the class k-BLMHF in time $O((\sqrt[k]{k})^n)$. The algorithm k-BLMHF verifies for each minimal hitting set of H, whether the partial truth assignment, which results from setting all variables in the hitting set to 0, can be extended to a model of F by checking the remaining 2-CNF part in linear time [2]. The number of minimal hitting sets is maximal if the Horn part of F consists of disjoint clauses only. So the running time of the algorithm k-BLMHF is dominated by this subclass of k-BLMHF, for which the number of minimal hitting sets of H is $k^{\lceil n/k \rceil}$ yielding the running time $O(p(n)(\sqrt[k]{k})^n)$, for a polynom p. Considering mixed Horn formulas where the Horn part is negative monotone and linear the question arises whether we can find an upper bound for the maximal number of minimal hitting sets of the Horn part which is smaller than $3^{n/3}$ where n is the number of variables. If so we could adapt algorithm k-BLMHF for these formulas and solve SAT faster than in $O(3^{n/3})$.
Linear formulas overlap only sparsely and there is some evidence that linear formulas form the algorithmically hard kernel for CNF-SAT, making this class specifically interesting for other variants of SAT, too. Nevertheless we were not able to prove this hypothesis, so it remains another challenging task for the future to investigate whether linear formulas indeed form the algorithmically hard kernel for CNF-SAT.
XSAT has been shown to be NP-complete when restricted to certain linear k-uniform and l-regular CNF classes, for $k, l \geq 3$. However the complexity status for XSAT restricted to the classes k-LCNF$^l_+$, for arbitrary values of $k, l \geq 3$ is left open for future work. Using some connections to finite projective planes we were able to show that XSAT remains NP-complete for linear and l-regular formulas which in addition are l-uniform whenever $l = q + 1$, where q is a prime power. Thus XSAT is most likely NP-complete for the other values of $l \geq 3$, too.
In [32] it is shown that whereas unrestricted k-SAT is NP-complete, for $k \geq 3$, it behaves trivially (i.e. all formulas are satisfiable) if each clause has length exactly k and no variable occurs in more than $f(k)$ clauses; it becomes NP-complete if variables are allowed to occur at most $f(k) + 1$ times. Here $f(k)$ asymptotically grows as $\lfloor 2^k/(e \cdot k) \rfloor$; meanwhile this bound has been improved by other authors. Thus, one could investigate whether a similar result also exists for XSAT on LCNF.
Moreover, we have proven that NAE-SAT remains NP-complete for k-LCNF$_+$ as well as for LCNF$^l_+$ so it would be desirable to investigate the computational complexity of k-LCNF$^l_+$-NAE-SAT, for $k, l \geq 3$.
Finally from the point of view of exact algorithmics, it would be desirable to gain progress for XSAT restricted to linear formulas beyond the so far best bound of $O(2^{0.2325 \cdot n})$ for unrestricted CNF-XSAT over n variables, provided by Byskov et al. [12]. Note that for NAE-SAT such progress seems to be hard to achieve since NAE-SAT can be shown to be as hard as SAT itself for unrestricted CNF formulas [28, 33]. It remains to investigate whether NAE-SAT or even SAT can be solved in less than 2^n steps for unrestricted linear formulas.
There are several other problems left open for future work. Concerning NAE-SAT we were not able to find unsatisfiable, positive monotone, linear and k-uniform formulas for $k \geq 5$, which one could use as padding backbone formulas to prove the NP-completeness of k-LCNF$_+$-NAE-SAT. It seems to be a quite complex but also

challenging combinatorial task to find such formulas.
In addition, we have shown that XSAT remains NP-complete for XLCNF_+ and XLCNF. It is not hard to see that this result sharpens the long-standing NP-hardness result for clique packing of a graph maximizing the number of covered edges of Hell and Kirkpatrick [24]. Recently Chataigner et al. have provided nice approximation (hardness) results regarding the clique packing problem [13]. It could be interesting to investigate in the future whether similar approximation results can be gained for XSAT on LCNF respectively XLCNF.
The complexity of XSAT restricted to k-uniform exact linear formulas, for $k \geq 7$, is left for future work. The same lack is present for the case where the number of occurrences is bounded, meaning that the XSAT-complexity of XLCNF^l_+ and k-XLCNF^l_+ are unsolved, for $k, l \geq 3$.
Our results imply the NP-completeness for some subversions of the well-known combinatorial optimization problems Set Partitioning and Exact Hitting Set on regular and linear hypergraphs. Recall that Set Partitioning takes as input a finite hypergraph with vertex set M and a set of hyperedges $\mathcal{M}$ (i.e. subsets of M). It asks for a subfamily $\mathcal{T}$ of $\mathcal{M}$ such that each element of M occurs in exactly one member of $\mathcal{T}$. It is evident that monotone XSAT coincides with Set Partitioning when the clauses overtake the roles of vertices in M and the variables are regarded as the hyperedges in $\mathcal{M}$ so that a variable contains all clauses in which it occurs. Exact Hitting Set, however, is nothing but monotone XSAT only translated to the hypergraph (or set system) terminology. Therefore Exact Hitting Set remains NP-complete for linear, l-regular hypergraphs. A simple dualization argument implies the same for Set Partitioning on that specific class of hypergraphs.

SAT as well as NAE-SAT behave polynomial-time solvable on the exact linear formula class. In 1996, T. Eiter [20] mentioned a problem called symmetrical intersecting monotone UNSAT (SIM-UNSAT), which is computationally equivalent to a problem called IM-UNSAT that in turn forms the hard core of several interesting combinatorial problems arising in different areas. In [20] Eiter poses the question concerning the computational complexity of SIM-UNSAT (resp. IM-UNSAT) which has been open for 15 years. As far as we know this question has not been answered yet. Instances of SIM-UNSAT have the form $C \cup C^\gamma$, where C is a set of pairwise intersecting *monotone* clauses. The question is whether such an instance is unsatisfiable. Observe that solving NAE-SAT for exact linear formulas in polynomial-time means to solve SIM-UNSAT in polynomial time restricted to the special case, where the monotone clauses intersect pairwise in exactly one variable. In that respect, we have given a partial answer to Eiter's problem. However, the general case clearly remains open.

Appendix A

Notation and Definitions

Let CNF denote the set of duplicate-free conjunctive normal form formulas over propositional variables $x \in \{0,1\}$. A *positive (negative)* literal is a (negated) variable. The *complement* of a literal l is its negation $\bar{l}$. Each formula $C \in$ CNF is considered as a clause set, and each clause $c \in C$ is represented as a literal set, so we specifically do not allow that clauses contain a literal more than once. For a formula C, clause c we denote the set of variables contained (neglecting negations) in C resp. in c by $V(C)$ resp. $V(c)$. $L(C)$ is the set of all literals in C.

The satisfiability problem (SAT) asks whether an input formula $C \in$ CNF has a *model*, that is a truth assignment $t : V(C) \to \{0,1\}$ assigning at least one literal in each clause of C to 1. It is straightforward to extend a truth assignment to the literal set $L(C)$ (for which we will not introduce a distinct notion) as follows: If $x \in V(C)$ is assigned 1, then x, considered as a positive literal, is also assigned 1 and $\overline{x}$ is assigned 0. The counting problem #SAT asks how many models an input formula $C \in$ CNF has. If there is no truth assignment satisfying C then the output should be 0, else, if C is satisfiable, then the exact number of satisfying truth assignments should be the output.

A graph is *planar* if it can be embedded in the plane in such a way that its edges intersect only at their endpoints. An embedding of a graph $G = (V, E)$ is *1-outerplanar* (from now on simply outerplanar), if it is planar, and all vertices lie on the exterior face. For $k \geq 2$, an embedding of a graph $G = (V, E)$ is *k-outerplanar* if it is planar and if all vertices on the outer face are deleted, a $(k-1)$-outerplanar embedding of the resulting graph is obtained. A graph is called *k-outerplanar* if it has a k-outerplanar embedding.

Given a CNF formula F with variable set $V(F) = \{v_1, \ldots, v_n\}$ and clause set $C(F) = \{c_1, \ldots, c_m\}$, we define the corresponding *variable-clause graph* G_F as follows: The vertex set of G_F is $V(G_F) = C(F) \cup V(F)$, the edge set of G_F is $E(G_F) = \{\{c_i, v_j\} : v_j \in c_i \vee \overline{v_j} \in c_i\}$. We call $F \in$ CNF *k-outerplanar*, for $k \geq 1$, if its variable-clause graph is k-outerplanar.
Formulas $F = P \wedge H$ of a positive monotone 2-CNF part P and a Horn part H are called mixed Horn formulas (MHF).

Similarly, NAE-SAT (resp. XSAT) is the variant of SAT asking for a truth assignment setting at least one (resp. exactly one) literal in each clause to 1 and at least one (resp. all other) literal(s) to 0; such a truth assignment is called a *nae-model* (resp. *x-model*). We assume throughout that clauses do not contain

complemented pairs of literals such as $x, \overline{x}$. This does not affect generality because these clauses are always satisfiable w.r.t. to SAT and NAE-SAT and can be removed from a formula.
A CNF formula C is called *linear* if C contains no pair of complementary unit clauses and for all $c_1, c_2 \in C : c_1 \neq c_2$ we have $|V(c_1) \cap V(c_2)| \leq 1$. C is called *exact linear* if C is linear and we have $|V(c_1) \cap V(c_2)| = 1$ for all $c_1, c_2 \in C : c_1 \neq c_2$. A *positive monotone* formula has no negated variables. A formula is *k-uniform* if all its clauses have length exactly k. A CNF formula C is called *l-regular*, for $l \in \mathbb{N}, l \geq 2$ if each variable of C occurs exactly l times in C. We denote with $w(x)$, for a $x \in V(C)$ the number of clauses in C containing x. Hence each l-regular formula C can be characterized by $w(x) = l$, for all $x \in V(C)$. Let LCNF denote the linear formula class, XLCNF the exact linear formula class. Further let $k{-}A$, $(\geq k){-}A$ or $(\leq k){-}A$ denote formulas of the class $A \in \{\mathrm{CNF}, \mathrm{LCNF}, \mathrm{XLCNF}\}$ for which additionally holds that their clauses have length exactly k, at least k or at most k. Then the classes A^l, $A^{\geq l}$ or $A^{\leq l}$ contain formulas of the class $A \in \{\mathrm{CNF}, \mathrm{LCNF}, \mathrm{XLCNF}\}$, for which additionally holds that all variables in these formulas occur exactly l times, at least l times or at most l times. A_+ denotes the positive monotone formulas of the class $A \in \{\mathrm{CNF}, \mathrm{LCNF}, \mathrm{XLCNF}\}$. Let C be a satisfiable formula. A variable $y \in V(C)$ is called a *backbone variable* of C if y has the same value in each model of C. Let C be an x-satisfiable formula. A variable $y \in V(C)$ is called an *x-backbone variable* of C, if y has the same value in each x-model of C. We call a positive monotone, k-uniform and exact linear formula, with the additional property that it is k-regular, a *k-block formula*.

Appendix B

Abbreviations of Some Formula Classes

Abbreviation	Explanation
MHF	Mixed Horn formulas (MHF) $F = P \wedge H$ where P consists of 2-clauses and H of Horn clauses.
$\mathrm{MH_kF^+}$	MHF with k-uniform Horn clauses and a positive monotone P part.
$\mathrm{MH_k^-F^+}$	MHF with k-uniform, negative monotone Horn clauses and a positive monotone P part.
$\mathrm{MH_k^-F^{d+}}$	MHF with k-uniform, negative monotone Horn clauses and G_P consists of disjoint edges only.
$\mathrm{LMH_k^-F^+}$	MHF with k-uniform, negative monotone and linear Horn clauses and a positive monotone P part.
k-BLMHF	k-Boundary-Linear mixed Horn formulas $M = G \wedge H \in$ MHF: G consists of 2-clauses, H consists of linear Horn clauses for which holds: All clauses are negative monotone, k-uniform, $k \geq 3$, and there is an ordering $c_1, c_2 \ldots, c_{\lvert H \rvert}$ of the clauses of H and an ordering of all literals in each clause, such that H has overlappings in boundary variables only.
k-BLMHF^{d+}	MHF where P consists of disjoint, positive monotone 2-clauses only and H is negative monotone, k-uniform, for $k \geq 3$, linear and has overlappings in boundary variables only.
$\mathrm{MH\_F^{\triangle}}$	MHF with a negative monotone Horn part H and a positive monotone 2-CNF part P for which holds that the corresponding variable graph G_P consists of disjoint triangles only.
$MH_k^- F^{\triangle}$	MHF whose Horn part H is k-uniform, $k \geq 3$, negative monotone and whose corresponding graph G_P consists of disjoint triangles only.
$MH_{\_d} F^{\triangle}$	MHF whose Horn part H is negative monotone and consists of disjoint clauses only and for whose positive 2-CNF part P holds that the corresponding graph G_P consists of disjoint triangles only.
$\mathrm{XLMH^-}F$	MHF where H is negative monotone, exact linear and P consists of positive monotone 2-clauses only.
$XLMH\_F^{d+}$	MHF where H is negative monotone, exact linear and G_P consists of disjoint edges only.
$XLMH\_F^{\triangle}$	MHF where H is negative monotone, exact linear and G_P consists of disjoint triangles only.

Abbreviation	Explanation
CNF	Class of formulas in conjunctive normal form.
CNF_+	Positive monotone CNF formulas.
CNF_+^l	Positive monotone and l-regular CNF formulas.
k-CNF_+	Positive monotone and k-uniform CNF formulas.
k-$\mathrm{CNF}_+^{\leq l}$	Positive monotone and k-uniform CNF formulas for which additionally holds: each variable occurs $\leq l$ times.
k-CNF_+^l	Positive monotone, k-uniform and l-regular CNF formulas.
LCNF	Linear CNF formulas.
LCNF_+^l	Linear, positive monotone and l-regular formulas.
$(\leq l)$-$\mathrm{LCNF}_+^{\geq l}$	Positive monotone, linear formulas where each clause has length $\leq l$ and each variables occurs $\geq l$ times.
$(\leq l)$-LCNF_+^l	Positive monotone and l-regular linear formulas whose clauses have all length $\leq l$.
XLCNF	Exact linear CNF fomulas.
XLCNF_+	Exact linear and positive monotone CNF fomulas.
k-XLCNF	Exact linear, k-uniform CNF fomulas.
k-XLCNF_+	Exact linear, k-uniform and positive monotone CNF fomulas.
XLCNF^l	Exact linear, l-regular CNF fomulas.
XLCNF_+^l	Exact linear, l-regular and positive monotone CNF fomulas.
k-XLCNF^l	Exact linear, l-regular and k-uniform CNF fomulas.
k-XLCNF_+^l	Exact linear, l-regular, positive monotone and k-uniform CNF fomulas.

Appendix C

Abstract/Zusammenfassung

Abstract
The Boolean conjunctive normal form (CNF) satisfiability problem, called SAT for short, gets as input a CNF formula and has to decide whether this formula admits a satisfying truth assignment. As is well known, the remarkable result by S. Cook in 1971 established SAT as the first and genuine complete problem for the complexity class NP [15]. Thus SAT resides at the heart of the NP $\neq P$ conjecture of complexity theory. In this thesis we consider SAT for a subclass of CNF, the so called Mixed Horn formula class (MHF). A formula $F \in$ MHF consists of a 2-CNF part P and a Horn part H. We propose that MHF has a central relevance in CNF because many prominent NP-complete problems, e.g. Feedback Vertex Set, Vertex Cover, Dominating Set and Hitting Set, can easily be encoded as MHF. Furthermore, we show that SAT remains NP-complete for some interesting subclasses of MHF. We also provide algorithms for some of these subclasses solving SAT in a better running time than $O(2^{0.5284n}) = O((\sqrt[3]{3})^n)$ which is the best bound for MHF so far. One of these subclasses consists of formulas, where the Horn part is negative monotone and the variable graph corresponding to the positive 2-CNF part P consists of disjoint triangles only. Regarding the other subclass consisting of certain k-uniform linear mixed Horn formulas, we provide an algorithm solving SAT in time $O((\sqrt[k]{k})^n)$, for $k \geq 4$. Additionally, we consider mixed Horn formulas $F = P \wedge H \in MHF$ for which holds: H is negative monotone, $|c| \leq 3$, for all $c \in H$, and P consists of positive monotone 2-clauses. We solve SAT in running time $O(1.325^n)$ for this formula class by using the autarky principle, that means we can provide a better running time than the so far best one of $O(p(n) \cdot 1.427^n)$ by S. Kottler, M. Kaufmann and C. Sinz [29] for this class of mixed Horn formulas. Afterwards we consider mixed Horn formulas $F = P \wedge H \in MHF$ for which holds: G_P consists of disjoint triangles, edges and isolated vertices and H consists of Horn clauses which have at most three literals, but are not necessarily negative monotone and $V(P) = V(H)$. We can solve SAT in running time $O(1.41^n)$ for this formula class by applying the autarky principle.

Thereafter we consider some interesting subclasses of MHF for which SAT can be solved in polynomial-time. Furthermore, we present an algorithm which solves SAT for mixed Horn formulas with a linear, negative monotone and k-uniform Horn part and a P part which consists of positive monotone and disjoint 2-clauses only. Experimental results lead to the strong conjecture that its running time is better

than $O((\sqrt[3]{3})^n)$, where n is the number of variables.
In addition, we investigate the computational complexity of some prominent variants of SAT, namely not-all-equal SAT (NAE-SAT) and exact SAT (XSAT) restricted to the class of *linear* CNF formulas. Clauses of a linear formula pairwise have at most one variable in common. We show that NAE-SAT and XSAT are NP-complete for monotone and linear formulas where clauses have length greater or equal k, $k \geq 3$. We also prove the NP-completeness of XSAT for CNF formulas which are l-regular meaning that every variable occurs exactly l times, where $l \geq 3$ is a fixed integer. On that basis, we can provide the NP-completeness of XSAT for the subclass of linear and l-regular formulas. This result is transferable to the monotone case. Moreover, we provide an algorithm solving XSAT for the subclass of monotone, linear and l-regular formulas faster than the so far best algorithm from J. M. Byskov et al. for CNF-XSAT with a running time of $O(2^{0.2325n})$ [12]. Using some connections to finite projective planes, we can also show that XSAT remains NP-complete for linear and l-regular formulas that in addition are l-uniform whenever $l = q + 1$, where q is a prime power. Thus XSAT most likely is NP-complete for the other values of $l \geq 3$, too. Apart from that, we are interested in exact linear formulas: Here *each* pair of distinct clauses has exactly one variable in common. We show that NAE-SAT is polynomial-time decidable restricted to exact linear formulas. Reinterpreting this result enables us to give a partial answer to a long-standing open question mentioned by T. Eiter in [20]: Classify the computational complexity of the symmetrical intersecting unsatisfiability problem (SIM-UNSAT). Then we show the NP-completeness of XSAT for monotone and exact linear formulas, which we can also establish for the subclass of formulas whose clauses have length at least k, $k \geq 3$. This is somehow surprising since both SAT and not-all-equal SAT are polynomial-time solvable for exact linear formulas [42]. However, for $k \in \{3, 4, 5, 6\}$ we can show that XSAT is polynomial-time solvable for the k-uniform, monotone and exact linear formula class. An additional contribution of this thesis is the investigation of the computational complexity of the counting problem #SAT for k-outerplanar formulas, which in general is #P-complete. A CNF formula is called *k-outerplanar* if its variable-clause graph has a k-outerplanar embedding. First of all we provide an algorithm solving SAT in linear time for the 1-outerplanar formula class. Thereafter we present an algorithm which also solves #SAT in linear time for a given 1-outerplanar formula, whose graph has either no cycles or consists of disjoint cycles without chords. For 1-outerplanar formulas over n variables whose graphs may have cycles we solve #SAT in time $O(n^{5.13})$ using the separator theorem of Lipton & Tarjan [35]. More generally, we show that #SAT for k-outerplanar graphs, $k > 1$, can be solved in time $O(n^{1.7(2k+1)})$ with this prominent separator theorem of Lipton & Tarjan. For formulas having a *k-circular-levelplanar* graph we solve #SAT in time $O(k \cdot 16^k \cdot (\frac{2}{3})^{5.13 \cdot \log_2 k} n^{5.13})$ by the separator theorem of Lipton & Tarjan establishing its fixed-parameter tractability with respect to the parameter k. While we need polynomial-time to solve #SAT for a k-outerplanar formula F using the separator theorem of Lipton and Tarjan, we can actually solve #SAT in linear time using the technique of a nice tree decomposition of width at most $3k - 1$ for the variable-clause graph of a k-outerplanar formula, as is introduced by H.L. Bodlaender and T. Kloks in [7]. We present an algorithm which uses dynamic programming bottom-up from the leaves to the root in a nice tree decomposition of width at most $3k - 1$ of G_F to solve the #SAT problem in linear time for a k-outerplanar formula F. Finally, we are able to solve #SAT for Knuth's nested

formulas in polynomial-time $O(n^{8.5})$ by the separator theorem of Lipton & Tarjan and in linear time by the nice tree decomposition technique.

Zusammenfassung
Bei dem Boole'schen Erfüllbarkeitsproblem für Formeln in konjunktiver Normalform (KNF), kurz SAT genannt, ist eine KNF Formel gegeben und man muss entscheiden, ob diese Formel eine erfüllende Belegung besitzt. Wie bereits bekannt ist, hat S. Cook in seinem bemerkenswerten Ergebnis von 1971 SAT als das erste authentische Problem für die Klasse NP herauskristallisiert. In dieser Arbeit betrachten wir SAT für eine Teilklasse von KNF, die sogenannten Mixed-Horn-Formeln (MHF). Eine Formel $F \in$ MHF besteht aus einem 2-KNF Teil P und einem Horn Teil H. Wir zeigen, dass MHF eine zentrale Rolle in KNF spielt, weil viele prominente Probleme, wie z.B. Feedback Vertex Set, Vertex Cover, Dominating Set und Hitting Set leicht als MHF kodiert werden können. Des Weiteren zeigen wir, dass SAT für einige interessante Teilklassen von MHF NP-vollständig bleibt. Sodann stellen wir Algorithmen für zwei solcher Teilklassen vor, die SAT in einer besseren Laufzeit als $O(2^{0.5284n}) = O((\sqrt[3]{3})^n)$ lösen, was soweit die beste Schranke für MHF ist. Eine dieser Teilklassen besteht aus Formeln, bei denen der Horn Teil negativ monoton ist und der zugehörige Variablengraph des entsprechenden positiv monotonen 2-KNF-Teils P nur aus disjunkten Dreiecken besteht. Für die zweite Teilklasse, die aus gewissen k-uniformen, linearen Mixed-Horn-Formeln besteht, geben wir einen Algorithmus an, der SAT in Zeit $O((\sqrt[k]{k})^n)$ für $k \geq 4$ löst. Zusätzlich betrachten wir Mixed-Horn-Formeln $F = P \wedge H \in MHF$, für die gilt: H ist negativ monoton, $|c| \leq 3$, für alle $c \in H$, und P besteht nur aus positiven 2-Klauseln. Wir können SAT in der Laufzeit $O(1.325^n)$ für diese Formelklasse lösen, indem wir das Autarkie-Prinzip benutzen. Das bedeutet, wir erzielen eine bessere Laufzeit als die bisher beste Laufzeit $O(p(n) \cdot 1.427^n)$ von S. Kottler, M. Kaufmann und C. Sinz [29] für diese Klasse von Mixed-Horn-Formeln. Anschließend widmen wir uns denjenigen Mixed-Horn-Formeln $F = P \wedge H \in MHF$, für die gilt: G_P besteht aus disjunkten Dreiecken, Kanten und isolierten Knoten und H besteht aus Horn Klauseln, die höchstens drei Literale besitzen, aber nicht notwendigerweise negativ monoton sind. Ferner soll gelten $V(P) = V(H)$. Wir können SAT in der Laufzeit $O(1.41^n)$ für diese Formelklasse lösen, indem wir ebenfalls das Autarkie-Prinzip benutzen. Danach betrachten wir noch einige interessante Teilklassen von MHF, für die wir SAT in Polynomzeit lösen können. Schließlich präsentieren wir einen Algorithmus, der SAT für Mixed-Horn-Formeln mit einem linearen, negativ monotonen und k-uniformen Horn-Teil und einem P -Teil, bestehend nur aus positiv monotonen und disjunkten 2-Klauseln, löst. Experimentelle Ergebnisse führen zu der starken Vermutung, dass dessen Laufzeit besser ist als $O((\sqrt[3]{3})^n)$, wobei n die Anzahl der Variablen einer solchen Formel bezeichnet. Als Nächstes erforschen wir die Komplexität einiger prominenter Varianten von SAT, nämlich NAE-SAT und XSAT, eingeschränkt auf die Klasse der *linearen* Formeln. Je zwei Klauseln einer linearen Formel haben höchstens eine Variable gemeinsam. Wir zeigen, dass NAE-SAT und XSAT für monotone und lineare Formeln, deren Klauseln länger oder gleich k sind, $k \geq 3$, NP-vollständig sind. Ferner beweisen wir, dass XSAT für KNF-Formeln, die l-regulär sind, d.h. in denen jede Variable genau l Mal vorkommt, wobei $l \geq 3$ eine feste ganze Zahl ist, NP-vollständig ist. Darauf aufbauend können wir auch die NP-Vollständigkeit von XSAT für die Teilklasse der linearen und l-regulären Formeln zeigen. Dieses Ergebnis ist auch auf den monotonen Fall übertragbar. Ein weiteres Resultat dieser Arbeit ist ein Algorithmus, der

XSAT für die Klasse der monotonen, linearen und l-regulären Formeln schneller löst als der bis dato schnellste Algorithmus von J. M. Byskov für CNF-XSAT mit einer Laufzeit von $O(2^{0.2325n})$ [12]. Indem wir einige Verbindungen zu den projektiven Ebenen nutzen, können wir auch zeigen, dass XSAT NP-vollständig bleibt für die Klasse der linearen und l-regulären Formeln, die zusätzlich l-uniform sind, wobei $l = q + 1$ und q eine Primzahlpotenz ist. Daher ist es sehr wahrscheinlich, dass XSAT auch NP-vollständig ist für alle anderen Werte von $l \geq 3$. Wir interessieren uns im Weiteren für die exakt linearen Formeln: Je zwei Klauseln haben hier genau eine Variable gemeinsam. Wir zeigen, dass NAE-SAT für die Klasse der exakt linearen Formeln in Polynomzeit gelöst werden kann. Wenn wir dieses Ergebnis neu deuten, dann ermöglicht es uns eine teilweise Antwort auf eine lange offen stehende Frage, die von T. Eiter in [20] gestellt worden ist: Klassifizieren der Berechnungskomplexität des SIM-UNSAT-Problems.
Als nächstes zeigen wir die NP-Vollständigkeit von XSAT für monotone und exakt lineare Formeln, deren Klauseln mindestens Länge k, $k \geq 3$, haben. Dies ist ein überraschendes Ergebnis, da sowohl SAT als auch NAE-SAT beide in Polynomzeit für exakt lineare Formeln gelöst werden können [42]. Für $k \in \{3, 4, 5, 6\}$ können wir jedoch zeigen, dass XSAT für die k-uniformen, monotonen und exakt linearen Formeln in Polynomzeit gelöst werden kann.
Ein zusätzlicher Beitrag dieser Arbeit ist die Untersuchung der Berechnungskomplexität des Aufzählungsproblems #SAT für die k-außenplanaren Formeln, das im Allgemeinen #P-vollständig ist. Eine KNF-Formel heißt *k-außenplanar*, wenn deren Variablen-Klauseln-Graph eine k-außenplanare Einbettung besitzt. Als Erstes stellen wir einen Algorithmus vor, der SAT für die 1-außenplanaren Formeln in linearer Zeit löst. Dann präsentieren wir einen Algorithmus, welcher #SAT ebenfalls in linearer Zeit löst für eine 1-außenplanare Formel, deren Graph entweder keine Kreise oder nur disjunkte Kreise ohne Sekanten besitzt. Für 1-außenplanare Formeln mit n Variablen, deren Graphen auch Kreise enthalten können, lösen wir #SAT in Zeit $O(n^{5.13})$, indem wir das Separator-Theorem von Lipton & Tarjan [35] benutzen. Allgemein zeigen wir, dass #SAT für k-außenplanare Graphen, $k > 1$, mit Hilfe des bekannten Separator-Theorems von Lipton & Tarjan in der Laufzeit $O(n^{1.7(2k+1)})$ gelöst werden kann. Für Formeln, die einen *k-zirkulär-levelplanaren* Graphen besitzen, lösen wir #SAT bereits in der Zeit $O(k \cdot 16^k \cdot (\frac{2}{3})^{5.13 \cdot \log_2 k} n^{5.13})$ mit Hilfe des Separator-Theorems von Lipton & Tarjan und unterstreichen damit deren Zugehörigkeit zu der Klasse FPT bezüglich des Parameters k. Während wir #SAT für eine k-außenplanare Formel F mit Hilfe des Separator-Theorems von Lipton & Tarjan in Polynomzeit lösen, können wir das Aufzählungsproblem #SAT sogar schon in Linearzeit lösen mit Hilfe der Technik einer *schönen Baumzerlegung* der Weite $\leq 3k-1$ des Variablen-Klauseln Graphen einer k-außenplanaren Formel, wie von H.L. Bodlaender und T. Kloks in [7] vorgestellt. Wir stellen einen Algorithmus vor, der mittels der dynamischen Programmierung von unten nach oben, d.h. von den Blättern zu der Baumwurzel in einer schönen Baumzerlegung der Weite $\leq 3k - 1$ von G_F das Problem #SAT für eine k-außenplanare Formel F in Linearzeit löst. Schließlich sind wir auch im Stande, #SAT für Knuths *Nested* Formeln in Polynomzeit $O(n^{8.5})$ (mit Hilfe des Separator-Theorems) und in Linearzeit (mit Hilfe der Technik der Baumzerlegung) zu lösen.

Appendix D

Acknowledgements

An dieser Stelle möchte ich die Gelegenheit nutzen, um all denjenigen Menschen zu danken, die zum Erfolg dieser Arbeit beigetragen haben.
Mein Dank gilt in erster Linie meinem Doktorvater Prof. Dr. Ewald Speckenmeyer, unter dessen sachkundiger Betreuung diese Arbeit entstanden ist. Er hat mir die Chance gegeben, diese Arbeit an seinem Lehrstuhl zu verfassen und mich dabei stets unterstützt.

Ein besonderer Dank gilt meinem Kollegen PD Dr. Stefan Porschen, der sich stets sehr viel Mühe gegeben hat, mir weiter zu helfen. Er hat mir ständig neue Anregungen gegeben und mich sehr motiviert. Auch gab er sich stets sehr viel Mühe, alle meine Fragen zu beantworten und war auch sonst immer sehr hilfsbereit. Er war auch immer offen für Diskussionen jeglicher Art und hat viel zu der angenehmen Büroatmosphäre beigetragen.

Ein weiterer Dank gilt Gabriele Eslamipour, der Sekretärin des Lehrstuhls. Sie hat mir oft mit bürokratischen Angelegenheiten weitergeholfen und auch einen grossen Teil des anfallenden Papierkriegs für mich übernommen. Außerdem hat sie auch viel zu der freundlichen Atmosphäre des Lehrstuhls beigetragen.

Weiterhin möchte ich mich bei Prof. Dr. Hubert Randerath dafür bedanken, dass er mich ebenfalls oft motiviert hat, und für die zahlreichen netten Gespräche. Des Weiteren möchte ich mich bei Michael Belling dafür bedanken, dass er mir immer so gut bei der Literatursuche weiter geholfen hat. Ein weiterer Dank gilt den beiden studentischen Hilfskräften Mattias Gärtner und Sascha Zur. Die beiden haben für mich im Laufe der Jahre so einige Algorithmen implementiert und getestet. Was ihr Programmierkönnen anbelangt, so sind die beiden für mich Koryphäen auf diesem Gebiet.

Ein ganz besonderer Dank gilt meinem Vater, Alexander Schmidt, der mich stets unterstützt hat, wo er nur konnte, besonders nach der Geburt meiner Tochter Lena. Er hat es ermöglicht, dass ich diese Arbeit so schnell nach der Geburt fertig stellen konnte, indem er sich extra Urlaub genommen hat, um als Babysitter einzuspringen.

Ferner, danke ich Franz Wankum dafür, dass er sich soviel Zeit genommen hat, mir zu helfen.

Weiter möchte ich mich bei meinen Freundinnen Ina Meider, Anika Dewald, Julia Hemmers, Christine Textoris und Simone Schulz bedanken. Zum Schluss gilt ein ganz lieber Dank meinem Freund, Tobias Giesbers, für dessen unermüdliche

Unterstützung und Motivation. Er hat von Anfang an an mich geglaubt und war immer für mich da, wenn ich ihn gebraucht habe.

Appendix E

Publications

- Porschen, S., Schmidt, T., Speckenmeyer, E.: *On Some Aspects of Mixed Horn Formulas*, Lecture Notes in Computer Science, Vol. 5584, 86-100 (2009).

- Porschen, S., Schmidt, T.: *On Some* SAT -*Variants over Linear Formulas*, Lecture Notes in Computer Science, Vol. 5404, 449-460 (2009).
- Porschen, S., Schmidt, T., Speckenmeyer, E.: *Complexity Results for Linear* XSAT-*Problems*, To appear in: Lecture Notes in Computer Science (2010).

Appendix F

Shares

chapter	section	result	my share
2	1	all results	100%
2	2	Theorem 1	90 %
2	3	all results	100%
2	4	all results	100%
2	5	all results	100%
3	1	all results	20%
3	2	Theorem 26	80%
3	2	Theorem 27	80%
3	3	Theorem 28	100%
3	3	Theorem 29	60%
3	3	Theorem 30	100%
3	3	Theorem 31	85%
3	3	Theorem 32	100%
3	4	all results	100%
3	5.1	Theorem 36	90%
3	5.1	Theorem 37	100%
3	5.2	Lemma 10-13	100%
3	5.2	Lemma 14	10%
3	5.2	Theorem 38	100%
3	5.2	Lemma 15, Lemma 16	100%
3	5.2	Theorem 39	70%
3	5.2	Theorem 40	100%
3	6	Corollary 5	30%
4	1	all results	70%
4	2	all results	100%
4	3	all results	100%
4	4	all results	100%

Appendix G

Experimental Results

By using a computer program which was specially written for our purposes we have calculated the difference $N(t) - 2^{0.2325n}$ for distinct inputs of α_0, l and n. Here $N(t)$ denotes the number of leaves in the tree t which simulates the recursion steps of Algorithm AVRG (chapter 3, section 5).

$\alpha_0 = 3$, $l = 3$:
n=150 difference=-17179398398.9106133392978835162605 5
n=160 difference=-143419535138.8001205521540839746327
n=170 difference=-776190071366.6202699761889988944210
n=180 difference=-3947516748158.419126133287898188307
n=190 difference=-19840324816222.79086516798470544802
n=200 difference=-99481476731956.95013654501146593483

$\alpha_0 = 3$, $l = 4$:
n=50 difference=278022292.5522295645391465995353016
n=60 difference=278010766.0991868953422364758155183
n=70 difference=277950979.8273689245088223593602016
n=80 difference=277644913.9963886985982761991489793
n=90 difference=276090151.2723643789554605886980495
n=100 difference=268231379.6814312663771203845579025
n=110 difference=228632028.8617168466362781897388057
n=120 difference=29492451.4537733322613007408854642 9
n=130 difference=-970670518.7198511342917203855151021
n=140 difference=-5989765616.627850087034335582941807
n=150 difference=-31163532862.9106133392978835162605 5
n=160 difference=-157381399356.8001205521540839746327
n=170 difference=-790078836377.6202699761889988944210
n=180 difference=-3961165491203.419126133287898188307
n=190 difference=-19853185235592.79086516798470544802
n=200 difference=-99491747431200.95013654501146593483

$\alpha_0 = 4$, $l = 4$:
n=50 difference=-3048.4477704354608534004646984004 50
n=60 difference=-15423.900813104657763524184481664 42
n=70 difference=-78155.172631075491177640639798363 94

n=80 difference=-394330.00361130140172380085102072 78
n=90 difference=-1982861.72763562104453941130195048 6
n=100 difference=-9954998.31856873362287961544209751 2
n=110 difference=-49932295.13828315336372181026119430
n=120 difference=-250318789.54622666773869925911453 57
n=130 difference=-1254603827.71985113429172038551510 2
n=140 difference=-6287280567.62785008703433558294180 7
n=150 difference=-31505582578.91061333929788351626055
n=160 difference=-157869626389.800120552154083974632 7
n=170 difference=-791046121034.620269976188998894421 0
n=180 difference=-3963699093749.419126133287898188307
n=190 difference=-19860842738872.79086516798470544802
n=200 difference=-99516154358120.95013654501146593483

$\alpha_0 = 5$, $l = 4$:
n=50 difference=24684949310.55222956453914659953530
n=60 difference=24684938128.09918689534223647581552
n=70 difference=24684879709.82736892450882235936020
n=80 difference=24684578822.99638869859827619914898
n=90 difference=24683043412.27236437895546058869805
n=100 difference=24675256690.68143126637712038455790
n=110 difference=24635921688.86171684663627818973881
n=120 difference=24437746423.45377333226130074088546
n=130 difference=23441077748.28014886570827961448490
n=140 difference=18434573138.37214991296566441705819
n=150 difference=-6694041613.91061333929788351626054 6
n=160 difference=-132750659955.80012055215408397463 27
n=170 difference=-764874341816.620269976188998894421 0
n=180 difference=-3933926027589.419126133287898188307
n=190 difference=-19818748898728.79086516798470544802
n=200 difference=-99431923675812.95013654501146593483

$\alpha_0 = 6$, $l = 4$:
n=50 difference=-2767.44777043546085340046469840 0450
n=60 difference=-14088.90081310465776352418448166442
n=70 difference=-73498.17263107549117764063979836394
n=80 difference=-377253.00361130140172380085102072 78
n=90 difference=-1919700.72763562104453941130195048 6
n=100 difference=-9753082.31856873362287961544209751 2
n=110 difference=-49221307.13828315336372181026119430
n=120 difference=-247762812.54622666773869925911453 57
n=130 difference=-1246512652.71985113429172038551510 2
n=140 difference=-6259169315.62785008703433558294180 7
n=150 difference=-31405394816.91061333929788351626055
n=160 difference=-157553815496.80012055215408397463 27
n=170 difference=-789954696068.620269976188998894421 0
n=180 difference=-3959823103831.419126133287898188307
n=190 difference=-19848646479432.79086516798470544802
n=200 difference=-99474096792694.95013654501146593483

$\alpha_0 = 7$, $l = 4$:
n=50 difference=42335587649.5522295645391465995353 0
n=60 difference=42335576049.0991868953422364758155 2
n=70 difference=42335516073.8273689245088223593602 0
n=80 difference=42335207966.9963886985982761991489 8
n=90 difference=42333648049.2723643789554605886980 5
n=100 difference=42325779221.6814312663771203845579 0
n=110 difference=42286094032.8617168466362781897388 1
n=120 difference=42086737167.4537733322613007408854 6
n=130 difference=41086076851.2801488657082796144849 0
n=140 difference=36066089534.3721499129656644170581 9
n=150 difference=10883426546.0893866607021164837394 5
n=160 difference=-115356348788.8001205521540839746327
n=170 difference=-748100693525.6202699761889988944210
n=180 difference=-3919540003439.419126133287898188307
n=190 difference=-19812471510899.79086516798470544802
n=200 difference=-99453184849943.95013654501146593483

$\alpha_0 = 8$, $l = 4$:
n=50 difference=63247530366.5522295645391465995353 0
n=60 difference=63247518576.0991868953422364758155 2
n=70 difference=63247457660.8273689245088223593602 0
n=80 difference=63247147771.9963886985982761991489 8
n=90 difference=63245579072.2723643789554605886980 5
n=100 difference=63237671996.6814312663771203845579 0
n=110 difference=63197903396.8617168466362781897388 1
n=120 difference=62998179969.4537733322613007408854 6
n=130 difference=61996007328.2801488657082796144849 0
n=140 difference=56970039152.3721499129656644170581 9
n=150 difference=31772852551.0893866607021164837394 5
n=160 difference=-94524503560.8001205521540839746326 8
n=170 difference=-727490890499.6202699761889988944210
n=180 difference=-3899484929624.419126133287898188307
n=190 difference=-19794558270912.79086516798470544802
n=200 difference=-99443373521734.95013654501146593483

$\alpha_0 = 9$, $l = 4$:
n=50 difference=73058858537.5522295645391465995353 0
n=60 difference=73058846394.0991868953422364758155 2
n=70 difference=73058784834.8273689245088223593602 0
n=80 difference=73058471606.9963886985982761991489 8
n=90 difference=73056895042.2723643789554605886980 5
n=100 difference=73048954430.6814312663771203845579 0
n=110 difference=73009064476.8617168466362781897388 1
n=120 difference=72808995932.4537733322613007408854 6
n=130 difference=71805575341.2801488657082796144849 0
n=140 difference=66776045372.3721499129656644170581 9
n=150 difference=41566131134.0893866607021164837394 5

n=160 difference=-84767870625.8001205521540839746326 8
n=170 difference=-717863479256.6202699761889988944210
n=180 difference=-3890232765562.419126133287898188307
n=190 difference=-19786613576177.79086516798470544802
n=200 difference=-99439873508303.95013654501146593483

$\alpha_0 = 10, l = 4$:
n=50 difference=76558871949.55222956453914659953530
n=60 difference=76558859736.09918689534223647581552
n=70 difference=76558797736.82736892450882235936020
n=80 difference=76558483600.99638869859827619914898
n=90 difference=76556901403.27236437895546058869805
n=100 difference=76548949013.68143126637712038455790
n=110 difference=76509031055.86171684663627818973881
n=120 difference=76308821573.45377333226130074088546
n=130 difference=75305063553.28014886570827961448490
n=140 difference=70273975967.37214991296566441705819
n=150 difference=45060279733.08938666070211648373945
n=160 difference=-81290087594.8001205521540839746326 8
n=170 difference=-714426191877.620269976188998894421 0
n=180 difference=-3886962315112.419126133287898188307
n=190 difference=-19783762708507.79086516798470544802
n=200 difference=-99438070313374.95013654501146593483

$\alpha_0 = 4, l = 5$:
n=150 difference=-26104425008.91061333929788351626055
n=160 difference=-152462438276.800120552154083974632 7
n=170 difference=-785622710276.620269976188998894421 0
n=180 difference=-3958228195855.419126133287898188307
n=190 difference=-19855244428140.79086516798470544802
n=200 difference=-99510171719659.95013654501146593483

$\alpha_0 = 5, l = 5$:
n=150 difference=-27994293031.91061333929788351626055
n=160 difference=-154343623848.800120552154083974632 7
n=170 difference=-787476877923.620269976188998894421 0
n=180 difference=-3960005086488.419126133287898188307
n=190 difference=-19856783943635.79086516798470544802
n=200 difference=-99511032254194.95013654501146593483

$\alpha_0 = 6, l = 5$:
n=150 difference=-29410410581.91061333929788351626055
n=160 difference=-155763604464.800120552154083974632 7
n=170 difference=-788909983971.620269976188998894421 0
n=180 difference=-3961466954250.419126133287898188307
n=190 difference=-19858345376173.79086516798470544802
n=200 difference=-99512925158437.95013654501146593483

$\alpha_0 = 7, l = 5$:

n=150 difference=-30164300960.9106133392978835162 6055
n=160 difference=-156523178598.8001205521540839746327
n=170 difference=-789685123957.6202699761889988944210
n=180 difference=-3962299176162.419126133287898188307
n=190 difference=-19859333619601.79086516798470544802
n=200 difference=-99514340190981.95013654501146593483

$\alpha_0 = 8$, $l = 5$:
n=150 difference=-30563855164.9106133392978835162 6055
n=160 difference=-156926544141.8001205521540839746327
n=170 difference=-790099506895.6202699761889988944210
n=180 difference=-3962745029772.419126133287898188307
n=190 difference=-19859869126271.79086516798470544802
n=200 difference=-99515092009943.95013654501146593483

$\alpha_0 = 9$, $l = 5$:
n=150 difference=-30754766558.9106133392978835162 6055
n=160 difference=-157120224513.8001205521540839746327
n=170 difference=-790299735179.6202699761889988944210
n=180 difference=-3962960660064.419126133287898188307
n=190 difference=-19860128051364.79086516798470544802
n=200 difference=-99515489960484.95013654501146593483

$\alpha_0 = 10$, $l = 5$:
n=150 difference=-30851721311.9106133392978835162 6055
n=160 difference=-157218058626.8001205521540839746327
n=170 difference=-790400431055.6202699761889988944210
n=180 difference=-3963070314152.419126133287898188307
n=190 difference=-19860257379251.79086516798470544802
n=200 difference=-99515680167816.95013654501146593483

$\alpha_0 = 4$, $l = 6$:
n=150 difference=-30944051253.9106133392978835162 6055
n=160 difference=-157310493731.8001205521540839746327
n=170 difference=-790493216942.6202699761889988944210
n=180 difference=-3963164158322.419126133287898188307
n=190 difference=-19860354213397.79086516798470544802
n=200 difference=-99515785553729.95013654501146593483

$\alpha_0 = 5$, $l = 6$:
n=150 difference=-31106998058.9106133392978835162 6055
n=160 difference=-157472403847.8001205521540839746327
n=170 difference=-790652385784.6202699761889988944210
n=180 difference=-3963316104939.419126133287898188307
n=190 difference=-19860487220768.79086516798470544802
n=200 difference=-99515869010798.95013654501146593483

$\alpha_0 = 6$, $l = 6$:
n=150 difference=-31240152067.9106133392978835162 6055

n=160 difference=-157605964255.800120552154083974 6327
n=170 difference=-790787014458.620269976188998894 4210
n=180 difference=-3963453511804.419126133287898188307
n=190 difference=-19860631824783.79086516798470544802
n=200 difference=-99516032348580.95013654501146593483

$\alpha_0 = 7, l = 6$:
n=150 difference=-31315035367.91061333929788351626055
n=160 difference=-157681612913.800120552154083974 6327
n=170 difference=-790864813449.620269976188998894 4210
n=180 difference=-3963536534063.419126133287898188307
n=190 difference=-19860729426482.79086516798470544802
n=200 difference=-99516165343598.95013654501146593483

$\alpha_0 = 8, l = 6$:
n=150 difference=-31356343271.91061333929788351626055
n=160 difference=-157723444968.800120552154083974 6327
n=170 difference=-790907917260.620269976188998894 4210
n=180 difference=-3963582920373.419126133287898188307
n=190 difference=-19860783612939.79086516798470544802
n=200 difference=-99516239913552.95013654501146593483

$\alpha_0 = 9, l = 6$:
n=150 difference=-31377200535.91061333929788351626055
n=160 difference=-157744668086.800120552154083974 6327
n=170 difference=-790929881942.620269976188998894 4210
n=180 difference=-3963606893348.419126133287898188307
n=190 difference=-19860812335381.79086516798470544802
n=200 difference=-99516281019288.95013654501146593483

$\alpha_0 = 10, l = 6$:
n=150 difference=-31388726123.91061333929788351626055
n=160 difference=-157756349338.800120552154083974 6327
n=170 difference=-790941969213.620269976188998894 4210
n=180 difference=-3963620025836.419126133287898188307
n=190 difference=-19860827570484.79086516798470544802
n=200 difference=-99516301758034.95013654501146593483

$\alpha_0 = 4, l = 7$:
n=150 difference=-31403846695.91061333929788351626055
n=160 difference=-157771388987.800120552154083974 6327
n=170 difference=-790956860387.620269976188998894 4210
n=180 difference=-3963634566592.419126133287898188307
n=190 difference=-19860841417262.79086516798470544802
n=200 difference=-99516313961096.95013654501146593483

$\alpha_0 = 5, l = 7$:
n=150 difference=-31429676485.91061333929788351626055
n=160 difference=-157797017804.800120552154083974 6327

n=170 difference=-790981986333.6202699761889988944210
n=180 difference=-3963658520776.419126133287898188307
n=190 difference=-19860862493404.79086516798470544802
n=200 difference=-99516328361362.95013654501146593483

$\alpha_0 = 6$, $l = 7$:
n=150 difference=-31450386438.91061333929788351626055
n=160 difference=-157817834895.800120552152083974 6327
n=170 difference=-791003038545.6202699761889988944210
n=180 difference=-3963680066720.419126133287898188307
n=190 difference=-19860885422836.79086516798470544802
n=200 difference=-99516354278186.95013654501146593483

$\alpha_0 = 7$, $l = 7$:
n=150 difference=-31463558929.91061333929788351626055
n=160 difference=-157831141570.800120552152083974 6327
n=170 difference=-791016677702.6202699761889988944210
n=180 difference=-3963694561247.419126133287898188307
n=190 difference=-19860901691203.79086516798470544802
n=200 difference=-99516374949845.95013654501146593483

$\alpha_0 = 8$, $l = 7$:
n=150 difference=-31470828230.91061333929788351626055
n=160 difference=-157838529419.800120552152083974 6327
n=170 difference=-791024364497.6202699761889988944210
n=180 difference=-3963702896139.419126133287898188307
n=190 difference=-19860911490605.79086516798470544802
n=200 difference=-99516388055991.95013654501146593483

$\alpha_0 = 9$, $l = 7$:
n=150 difference=-31474703568.91061333929788351626055
n=160 difference=-157842478470.800120552152083974 6327
n=170 difference=-791028458343.6202699761889988944210
n=180 difference=-3963707329774.419126133287898188307
n=190 difference=-19860916773617.79086516798470544802
n=200 difference=-99516395275421.95013654501146593483

$\alpha_0 = 10$, $l = 7$:
n=150 difference=-31476965767.91061333929788351626055
n=160 difference=-157844779471.800120552152083974 6327
n=170 difference=-791030843677.6202699761889988944210
n=180 difference=-3963709921083.419126133287898188307
n=190 difference=-19860919752854.79086516798470544802
n=200 difference=-99516399116894.95013654501146593483

$\alpha_0 = 4$, $l = 8$:
n=150 difference=-31480775309.91061333929788351626055
n=160 difference=-157848546612.800120552152083974 6327
n=170 difference=-791034524137.6202699761889988944210

n=180 difference=-3963713417660.419126133287898188307
n=190 difference=-19860922872400.79086516798470544802
n=200 difference=-99516401481601.95013654501146593483

$\alpha_0 = 5, l = 8$:
n=150 difference=-31486890858.91061333929788351626055
n=160 difference=-157854608561.8001205521540839746327
n=170 difference=-791040463447.6202699761889988944210
n=180 difference=-3963719084684.419126133287898188307
n=190 difference=-19860927934572.79086516798470544802
n=200 difference=-99516405192601.95013654501146593483

$\alpha_0 = 6, l = 8$:
n=150 difference=-31491789398.91061333929788351626055
n=160 difference=-157859538163.8001205521540839746327
n=170 difference=-791045464803.6202699761889988944210
n=180 difference=-3963724210737.419126133287898188307
n=190 difference=-19860933361187.79086516798470544802
n=200 difference=-99516411331658.95013654501146593483

$\alpha_0 = 7, l = 8$:
n=150 difference=-31494911828.91061333929788351626055
n=160 difference=-157862705101.8001205521540839746327
n=170 difference=-791048723320.6202699761889988944210
n=180 difference=-3963727703681.419126133287898188307
n=190 difference=-19860937293114.79086516798470544802
n=200 difference=-99516416218519.95013654501146593483

$\alpha_0 = 8, l = 8$:
n=150 difference=-31496782707.91061333929788351626055
n=160 difference=-157864605418.8001205521540839746327
n=170 difference=-791050689912.6202699761889988944210
n=180 difference=-3963729810566.419126133287898188307
n=190 difference=-19860939738682.79086516798470544802
n=200 difference=-99516419319706.95013654501146593483

$\alpha_0 = 9, l = 8$:
n=150 difference=-31497796996.91061333929788351626055
n=160 difference=-157865644244.8001205521540839746327
n=170 difference=-791051776182.6202699761889988944210
n=180 difference=-3963730987734.419126133287898188307
n=190 difference=-19860941131012.79086516798470544802
n=200 difference=-99516421174486.95013654501146593483

$\alpha_0 = 10, l = 8$:
n=150 difference=-31498418187.91061333929788351626055
n=160 difference=-157866277810.8001205521540839746327
n=170 difference=-791052438868.6202699761889988944210
n=180 difference=-3963731708297.419126133287898188307

n=190 difference=-19860941942059.79086516798470544802
n=200 difference=-99516422178065.95013654501146593483

$\alpha_0 = 4$, $l = 9$:
n=150 difference=-31499667109.9106133392978835162 6055
n=160 difference=-157867504188.8001205521540839746327
n=170 difference=-791053622924.6202699761889988944210
n=180 difference=-3963732810494.419126133287898188307
n=190 difference=-19860942892625.79086516798470544802
n=200 difference=-99516422819638.95013654501146593483

$\alpha_0 = 5$, $l = 9$:
n=150 difference=-31501580598.9106133392978835162 6055
n=160 difference=-157869399269.8001205521540839746327
n=170 difference=-791055478913.6202699761889988944210
n=180 difference=-3963734585594.419126133287898188307
n=190 difference=-19860944489396.79086516798470544802
n=200 difference=-99516424053911.95013654501146593483

$\alpha_0 = 6$, $l = 9$:
n=150 difference=-31503156237.9106133392978835162 6055
n=160 difference=-157870984916.8001205521540839746327
n=170 difference=-791057084729.6202699761889988944210
n=180 difference=-3963736237414.419126133287898188307
n=190 difference=-19860946234659.79086516798470544802
n=200 difference=-99516425976355.95013654501146593483

$\alpha_0 = 7$, $l = 9$:
n=150 difference=-31504158624.9106133392978835162 6055
n=160 difference=-157872006407.8001205521540839746327
n=170 difference=-791058140900.6202699761889988944210
n=180 difference=-3963737361293.419126133287898188307
n=190 difference=-19860947504156.79086516798470544802
n=200 difference=-99516427546012.95013654501146593483

$\alpha_0 = 8$, $l = 9$:
n=150 difference=-31504760306.9106133392978835162 6055
n=160 difference=-157872619807.8001205521540839746327
n=170 difference=-791058780736.6202699761889988944210
n=180 difference=-3963738052228.419126133287898188307
n=190 difference=-19860948295419.79086516798470544802
n=200 difference=-99516428539263.95013654501146593483

$\alpha_0 = 9$, $l = 9$:
n=150 difference=-31505123997.9106133392978835162 6055
n=160 difference=-157872990394.8001205521540839746327
n=170 difference=-791059167083.6202699761889988944210
n=180 difference=-3963738471432.419126133287898188307
n=190 difference=-19860948773829.79086516798470544802

n=200 difference=-99516429135920.950136545011465934 83

$\alpha_0 = 10, l = 9$:
n=150 difference=-31505336950.9106133392978835162 6055
n=160 difference=-157873208230.8001205521540839746 327
n=170 difference=-791059393041.6202699761889988944 210
n=180 difference=-3963738715902.4191261332878981883 07
n=190 difference=-19860949058461.790865167984705448 02
n=200 difference=-99516429496172.950136545011465934 83

$\alpha_0 = 4, l = 10$:
n=150 difference=-31505842929.9106133392978835162 6055
n=160 difference=-157873702889.8001205521540839746 327
n=170 difference=-791059866355.6202699761889988944 210
n=180 difference=-3963739148386.4191261332878981883 07
n=190 difference=-19860949415169.790865167984705448 02
n=200 difference=-99516429715210.950136545011465934 83

$\alpha_0 = 5, l = 10$:
n=150 difference=-31506573113.9106133392978835162 6055
n=160 difference=-157874426149.8001205521540839746 327
n=170 difference=-791060574553.6202699761889988944 210
n=180 difference=-3963739826826.4191261332878981883 07
n=190 difference=-19860950035585.790865167984705448 02
n=200 difference=-99516430218188.950136545011465934 83

$\alpha_0 = 6, l = 10$:
n=150 difference=-31507161589.9106133392978835162 6055
n=160 difference=-157875020981.8001205521540839746 327
n=170 difference=-791061180336.6202699761889988944 210
n=180 difference=-3963740449297.4191261332878981883 07
n=190 difference=-19860950694671.790865167984705448 02
n=200 difference=-99516430951533.950136545011465934 83

$\alpha_0 = 7, l = 10$:
n=150 difference=-31507550054.9106133392978835162 6055
n=160 difference=-157875415861.8001205521540839746 327
n=170 difference=-791061590814.6202699761889988944 210
n=180 difference=-3963740889159.4191261332878981883 07
n=190 difference=-19860951183178.790865167984705448 02
n=200 difference=-99516431537573.950136545011465934 83

$\alpha_0 = 8, l = 10$:
n=150 difference=-31507789874.9106133392978835162 6055
n=160 difference=-157875659955.8001205521540839746 327
n=170 difference=-791061843763.6202699761889988944 210
n=180 difference=-3963741163304.4191261332878981883 07
n=190 difference=-19860951499497.790865167984705448 02
n=200 difference=-99516431922984.950136545011465934 83

$\alpha_0 = 9$, $l = 10$:
n=150 difference=-31507930805.91061333929788351626055
n=160 difference=-157875805217.8001205521540839746327
n=170 difference=-791061995115.6202699761889988944210
n=180 difference=-3963741325696.419126133287898188307
n=190 difference=-19860951685299.79086516798470544802
n=200 difference=-99516432159689.95013654501146593483

$\alpha_0 = 10$, $l = 10$:
n=150 difference=-31508018957.91061333929788351626055
n=160 difference=-157875895988.8001205521540839746327
n=170 difference=-791062090200.6202699761889988944210
n=180 difference=-3963741427849.419126133287898188307
n=190 difference=-19860951799033.79086516798470544802
n=200 difference=-99516432298176.95013654501146593483

Appendix H

Bibliography

Bibliography

[1] J. Alber, H. L. Bodlaender, H. Fernau, T. Kloks, R. Niedermeier, *Fixed parameter algorithms for dominating set and related problems on planar graphs*, Algorithmica 33(4) (2002), p. 461493.

[2] B. Aspvall, M. R. Plass, R.E. Tarjan, *A linear-time algorithm for testing the truth of certain quantified Boolean formulas*, Inform. Process. Lett. 8 (1979) p. 121-123.

[3] B.S. Baker, *Approximation algorithms for* NP*-complete problems on planar graphs*, J. Assoc. Comput. Mach. 41 (1994), p. 153-180.

[4] C. Berge, *Hypergraphs*, North-Holland, Amsterdam, (1989).

[5] N. Betzler, R. Niedermeier, J. Uhlmann, *Tree decompositions of graphs: Saving Memory in Dynamic Programming*, Proc. CTW-04 (2004).

[6] D. Bienstock, C. L. Monma, *On the complexity of embedding planar graphs to minimize certain distance measures*, Algorithmica 5 (1990), p.93-109.

[7] H. L. Bodlaender, T. Kloks, *Efficient and constructive algorithms for the pathwidth and treewidth of graphs*, Journal of Algorithms 21 (1996), p. 358-402.

[8] H. L. Bodlaender, *A partial k-arboretum of graphs with bounded treewidth*, Theoretical Computer Science 209 (1998), p. 46-52.

[9] H. L. Bodlaender, *Treewidth: Algorithmic techniques and results*, Proceedings 22nd MFCS, Springer-Verlag LNCS 1295 (1997), p. 1936.

[10] H. L. Bodlaender, F. V. Fomin, *Tree decompositions with small cost*, Proceedings 8th SWAT, Springer-Verlag LNCS 2368 (2002), p. 378387.

[11] M. Böhm, E. Speckenmeyer, *A Fast Parallel SAT-Solver – Efficient Workload Balancing*, Annals of Mathematics and Artificial Intelligence, Vol. 17, p. 381-400 (1996).

[12] J. M. Byskov, B. Ammitzboll Madsen and B. Skjernaa, *New Algorithms for Exact Satisfiability*, Theoretical Comp. Sci. 332 (2005) p. 515-541.

[13] F. Chataigner, G. Manic, Y. Wakabayashi, R. Yuster, *Approximation algorithms and hardness results for the clique packing problem*, Discrete Appl. Math. 157 (2009) 1396-1406.

[14] E. M. Clarke, O. Grumberg, D. A. Peled, *Model Checking*, The MIT Press (2000).

[15] S. A. Cook, *The Complexity of Theorem Proving Procedures*, Proceedings of the 3rd ACM Symposium on Theory of Computing (1971), p. 151-158.

[16] B. Courcelle, J.A. Makowsky, U. Rotics, *On the Fixed Parameter Complexity of Graph Enumeration Problems Definable in Monadic Second Order Logic*, Discrete applied mathematics 108 (2001), p. 23-52.

[17] E. Dantsin, A. Wolpert, *Algorithms for SAT based on search in Hamming balls*, ECCC Report No. 17 (2004).

[18] E. Dantsin, A. Wolpert, *A faster clause-shortening algorithm for SAT with no restriction on clause length*, J. Satisfiability, Boolean Modeling and Computation 1 (2005) p. 49-60.

[19] R. G. Downey, M. R. Fellows, *Parameterized Complexity*, Springer-Verlag, New York (1999).

[20] T. Eiter, *Open Problems in Satisfiability*, available at http://www.ece.uc.edu/ franco/Sat-workshop/sat-workshop-open-problems.html.

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

[22] J. Gu, P. W. Purdom, J. Franco, B. W. Wah, *Algorithms for the Satisfiability (SAT) Problem: A Survey*, in: "D. Du, J. Gu, P. M. Pardalos (Eds.), Satisfiability Problem: Theory and Applications, DIMACS Workshop, March 11-13, 1996," DIMACS Series, vol. 35, pp. 19-151, American Mathematical Society, Providence, Rhode Island, 1997.

[23] P. Hall, *On representatives of subsets*, J. London Math. Soc., 10 (1935), p. 26-30.

[24] P. Hell, D. G. Kirkpatrick, *On the complexity of general k-factor problems*, SIAM J. Comput. 12 (1983) p. 601-609.

[25] R. M. Karp, *Reducibility Among Combinatorial Problems*, Complexity of Computer Computations, Proc. Sympos. IBM Thomas J. Watson Res. Center, Yorktown Heights, N.Y.. New York: Plenum (1972), p.85-103.

[26] T. Kloks, *Treewidth. Computations and Approximations*, Volume 842 of Lecture Notes in Computer Science. Springer-Verlag, Berlin (1994)

[27] D. E. Knuth, *Nested Satisfiability*, Acta Informatica, 28 (1990), p. 1-6.

[28] D. E. Knuth, *Axioms and Hulls*, LNCS vol. 606, Springer, New York, 1992.

[29] S. Kottler, M. Kaufmann, C. Sinz, *A New Bound for an* NP-*Hard Subclass of 3-SAT Using Backdoors*, Proc. SAT 08, LNCS vol. 4996 (2008), pp. 161-167.

[30] D. König, *Graphen und Matrizen*, Math. Fiz. Lapok, 38 (1931), p. 116-119.

[31] J. Kratochvil, M. Krivanek, *Satisfiability of co-nested formulas*, Acta Informatica, 30 (1993) p. 397-403.

[32] J. Kratochvil, P. Savicky, Z. Tusa, *One more occurrence of variables makes satisfiability jump from trivial to* NP-*complete*, SIAM J. Comput. 22 (1993), p. 203-210.

[33] D. Le Berre, L. Simon, *The Essentials of the SAT 2003 Competition*, Proceedings of the 6th International Conference on Theory and Applications of Satisfiability Testing (SAT'03), Lecture Notes in Computer Science, Vol. 2919, Springer-Verlag (2004), pp. p. 172-187.

[34] D. Lichtenstein, *Planar formulae and their uses*, SIAM Journal on Computing 11 (1982),p. 329-343.

[35] R. J. Lipton, R.E. Tarjan *A separator theorem for planar graphs*, SIAM J. Appl. Math. Vol. 36, No. 2 (1979).

[36] A. Maheshwari, N. Zeh, *External Algorithms for Outerplanar Graphs*, Lecture Notes in Computer Science Vol. 1741 (1999), p. 307-316.

[37] B. Monien, E. Speckenmeyer, O. Vornberger, *Upper Bounds for Covering Problems*, Methods of Operations Research 43 (1981), p. 419-431.

[38] R. Palisse, *A short proof of Fisher's inequality*, Discrete Math. 111 (1993), p. 421-422.

[39] S. Porschen, T. Schmidt, *On Some SAT-Variants over Linear Formulas*, Proc. SOFSEM 2009, LNCS, Vol. 5404 (2009), p. 449-460.

[40] S. Porschen, T. Schmidt, E. Speckenmeyer, *Some Aspects of Mixed Horn Formulas*, Proc. SAT 09, LNCS vol. (2009).

[41] S. Porschen, E. Speckenmeyer, X. Zhao, *Linear* CNF *formulas and satisfiability*, Discrete Applied Mathematics (2008), p. 1-23.

[42] S. Porschen, E. Speckenmeyer, and X. Zhao, *Linear* CNF *formulas and satisfiability*, Discrete Appl. Math. 157 (2009) p. 1046-1068.

[43] S. Porschen, E. Speckenmeyer, *Satisfiability of Mixed Horn Formulas*, Discrete Appl. Math. 155 (2007), p. 1408-1419.

[44] S. Porschen, E. Speckenmeyer, *A* CNF *class generalizing Exact Linear Formulas*, Proc. SAT 2008, LNCS, Vol. 4996 (2008), p. 231-245.

[45] B. Randerath, E. Speckenmeyer, E. Boros, P.Hammer, A. Kogan, K. Makino, B. Simeone, O. Cepek, *A Satisfiability Formulation of Problems on Level Graphs*, ENDM, Vol. 9, (2001).

[46] H. J. Ryser, *Combinatorial Mathematics*, Carus Mathematical Monographs 14, Mathematical Association of America (1963).

[47] T. J. Schaefer, *The complexity of satisfiability problems*, Proc. STOC 1978. ACM (1978), p. 216-226.

[48] D. Scheder, *Unsatisfiable Linear* k-CNF*s Exist, for every* k, Preprint arXiv:math cs.DM/0708.2336 v1 (2007).

[49] G. Stalmarck and M. Säflund, *Modeling and verifying systems and software in propositional logic*, B. K. Daniels, editor, Safety of Computer Control Systems (SAFECOMP90), Pergamon Press (1990), p. 31-36.

[50] H. Stamm-Wilbrandt, *Programming in propositional Logic or Reductions: Back to the Roots (Satisfiability)*, Technical Report, Universität Bonn (1991).

[51] L. Valiant, *The complexity of enumeration and reliability problems*, SIAM J. Comput. 9 (1979), p. 410-421.

Appendix I

Erklärung

Ich versichere, dass ich die von mir vorgelegte Dissertation selbständig angefertigt, die benutzten Quellen und Hilfsmittel vollständig angegeben und die Stellen der Arbeit - einschließlich Tabellen, Karten und Abbildungen -, die anderen Werken im Wortlaut oder dem Sinn nach entnommen sind, in jedem Einzelfall als Entlehnung kenntlich gemacht habe; dass diese Dissertation noch keiner einer anderen Fakultät oder Universität zur Prüfung vorgelegen hat; dass sie - abgesehen von unten angegebenen Teilpublikationen - noch nicht veröffentlicht worden ist sowie, dass ich eine solche Veröffentlichung vor Abschluss des Promotionsverfahrens nicht vornehmen werde. Die Bestimmungen dieser Promotionsordnung sind mir bekannt. Die von mir vorgelegte Dissertation ist von Prof. Dr. Ewald Speckenmeyer betreut worden.

Köln, den 27.04.2010

Tatjana Schmidt