Complexity and algorithms for well-structured k-SAT instances

Konstantinos Georgiou* and Periklis A. Papakonstantinou*†

*Department of Computer Science, University of Toronto,
10 King's College Road, Toronto, ON M5S 3G4, Canada
†Department of Mathematics, University of Toronto,
40 St. George Street, Toronto, ON, M5S 2E4, Canada

{cgeorg,papakons}@cs.toronto.edu

January 20, 2008

Abstract

This paper initiates the study of SAT instances of bounded diameter. The diameter of an ordered CNF formula is defined as the maximum difference between the index of the first and the last occurrence of a variable. We study the complexity of the satisfiability, the counting and the maximization problems for formulas of bounded diameter. We investigate the relation between the diameter of a formula, and the tree-width and the path-width of its corresponding incidence graph, and show that under highly parallel and efficient transformations, diameter and path-width are equal up to a constant factor.

Our main technical contribution is that the computational complexity of SAT, Max-SAT, #SAT grows smoothly with the diameter (as a function of the number of variables). Our main focus is in providing space efficient and highly parallel algorithms, while the running time of our algorithms matches previously known results. Among others, we show NL-completeness of SAT and NC^2 algorithms for Max-SAT, #SAT when diameter is $O(\log n)$. Given the tree decomposition of a formula, we further improve on the space efficiency to decide SAT as asked by Alekhnovich and Razborov [1].

1 Introduction

SAT, Max-SAT and #SAT are among the most fundamental and well-studied problems in theoretical computer science, all intractable in the most general case: SAT is NP-complete [14], Max-SAT is NP-hard to approximate within some constant [5], while #SAT is hard for #P [39]. The intractability of SAT, Max-SAT and #SAT soon led to the study of restricted versions. Many of them are based on hidden structures of formulas and in particular on the so-called width restrictions. In this direction, we introduce the diameter of a formula, a natural structural restriction which unlike width restrictions, it is defined directly on k-CNF formulas. For a CNF formula ordered as a sequence of clauses and for every variable consider the distance between the clause-indices where this variable appears. The maximum such distance is the diameter of the ordered formula. In this work, we consider SAT, Max-SAT and #SAT and parameterize them with respect to diameter.

Parameterizing SAT instances using width parameters follows the more general study of NP-hard graph problems. Lipton and Tarjan initiated this study [25] solving some NP-hard problems much more efficiently than brute-force algorithms do. Along these lines, Robertson and Seymour [31, 32] introduced tree-width, which received more attention than other width parameters. The value of tree-width (as all width parameters) is witnessed by decomposing the vertex set of a graph. In general it is NP-hard to compute such an optimal tree decomposition. However, the concept of tree-width has been widely used to parameterize the complexity of many NP-hard problems, see e.g. [7, 8, 4, 10] or [9, 23] for two surveys.

The diameter of an ordered formula formalizes the following idea: if we know that the distance between the first and last occurrence of any variable is bounded, we may be able to understand better the complexity of such restricted SAT-instances. We extend the definition to unordered formulas to be the smallest diameter over all clause orderings. Technically, the diameter of an unordered formula ϕ fully coincides with the bandwidth of an underlying graph of ϕ. Minimizing the bandwidth is a well-studied problem (for a survey see [12]) not considered in this work. From this point on we focus on the diameter of ordered formulas, unless mentioned otherwise.

Let us consider some typical values of the diameter of an ordered k-CNF formula. At one extreme it is easy to see that the subproblem of k-SAT instances of diameter n^ϵ, $\epsilon > 0$, is NP-complete. On the other hand we show that the satisfiability of CNF formulas of $\log n$ diameter is NL-complete. That is, formulas of diameter $\log n$ encode arbitrary NL computations. Hence, it is intuitively clear that the diameter does not break the problem into independent subproblems. A preliminary study for a similar problem was given in [20].

A CNF formula is associated with many underlying graphs, each associated with a number of width parameters such as tree-width, path-width, clique-width, branch-width, cluster-width (for a comparison see [26]). There are numerous works in the area, and our exposition below is far from being complete.

Khanna and Motwani [24] considered Max-SAT for formulas of constant tree-width, while [3] exploits the same structural property for SAT. Deciding SAT has been proved fixed-parameter tractable with respect to branch-width [1] by Alekhnovich and Razborov, and to tree-width by Gottlob *et al* [22] on primal graphs. Using DPLL procedures, Bacchus, Dalmao and Pitassi, [6] considered #SAT, while the same time-bound for #SAT was achieved by Samer and Szeider [34] extending [22]. Fixed-parameter tractability of SAT and #SAT has also been considered in e.g. [16, 18, 27, 35, 26]; see also [38] for a survey. In our paper we study tree-width and path-width on the incidence graph of formulas.

Even for unordered formulas the value of the diameter is provably less informative than the width parameters in the following sense. Path-width is always upper bounded by the diameter, although the two values can be off by almost a linear factor (Lemma 3). Despite this, we prove that by a highly efficient algorithm (Theorem 4), a formula of path-width $\mathsf{d}(n)$ can be viewed as a formula of diameter $O(\mathsf{d}(n))$. Hence we counter any undesirable properties of this parameter and we only keep its simplicity.

Bounding the distance in which a variable appears in an ordered formula generates natural theoretical questions. Generally speaking, in engineering applications one may wish to exploit the concept of "locality of reference" of the constraints. On the practical side, one can think of a situation where modeling a problem instance as a SAT instance, this is associated with a natural bound on the diameter. Whenever this bound is sufficiently small we present efficient time and space and highly parallel algorithms for SAT, Max-SAT and #SAT. Additional practical motivation is

given by Alekhnovich and Razborov [1]. For formulas of branch-width $w(n)$, they decide SAT in time $n^{O(1)}2^{O(w(n))}$ and in space $n^{O(1)}2^{O(w(n))}$ and ask whether it is possible to reduce the space to polynomial. A consequence of our study is an algorithm that works in time $n^{O(1)}2^{O(w(n)\log n)}$ and space $n^{O(1)}$.

Here is a summary of our main contributions.

1. We introduce the diameter (Definition 3), whose value is combinatorially off almost by linear factor from the path-width of the incidence graph of a formula (Lemma 3). We provide a highly parallel transformation that preserves the satisfiability of a formula and moreover it algorithmically preserves the solutions of the maximization and counting problems (Section 3.2).

2. We show that for bounded diameter formulas, SAT, MAX-SAT and #SAT, have similar space and time complexity by designing space bounded and parallel algorithms (Section 3). In particular, when the diameter is $O(\log n)$, we show that the decision problem is complete for NL, the maximization and counting problems have highly efficient parallel algorithms (NC^2) (Section 3.3).

3. The above results translate to algorithms for bounded tree-width through the path-width tree-width well known bound. When the tree-width is $O(\log n)$ we improve on this translation (and previous research) by providing a polynomial and highly parallel algorithms (AC^1). When the tree-width is $t(n)$, we build upon this algorithm and by invoking a sequence of deep theorems of structural complexity we give parallel algorithms of circuit depth $t(n)\log n$. In particular, this significantly reduces the space resources as asked in [1].

2 Definitions and preliminary results

2.1 Notation and terminology

All logarithms are of base 2. All propositional formulas are in CNF. A k-CNF is a CNF where each clause has at most k literals, for a constant $k \in \mathbb{N}$. We denote by ϕ_π a total ordering of the clauses of the (unordered) formula ϕ. We assume that in an input, an unordered (ordered) formula ϕ (ϕ_π) is represented in the standard way as a sequence (respecting the order) of clauses. We consistently use n to denote the number of variables in a formula. N is used to denote length of given inputs. The diameter of an ordered formula is always expressed as a function of the number of variables, and it is denoted by $\mathsf{d}(n)$. All circuit families are logspace or logtime uniform. $\mathrm{DEPTH}(f(N))$ is the class of languages decidable by a family of circuits in depth $f(N)$. $\mathrm{DSPACE}(f(N))$, $\mathrm{NSPACE}(f(N))$, $\mathrm{DTIME}(f(N))$ denote the class of problems decidable in deterministic, non-deterministic space and deterministic time $f(N)$ respectively. For the function analogs of complexity classes we extend the notation by a leading F; e.g. $\mathrm{FDSPACE}(\log^2 N)$. NC^i and AC^i are classes of languages decidable by polynomial size circuits of depth $O(\log^i N)$ where the gates are of bounded and unbounded fan-in respectively; NC and AC denote the respective countable unions. We denote by $\mathrm{NL} = \mathrm{NSPACE}(\log N)$. Our notation is standard, see e.g. [41, 17]. LOGCFL is the class of languages logspace reducible to Context Free Languages (see also Section 2). DET is the class of languages logspace reducible to the problem of computing the determinant of an $n \times n$ integer matrix with n-bit long integer entries. #L is wrt NL the analog of #P wrt NP; #L is introduced in [2]. When the input is a formula of n variables we abuse notation by

writing $\textsc{CompClass}(f(n))$ instead of $\textsc{CompClass}(f(N))$. Since $N > n$ our containment results are slightly better than what our notation suggests. We use the term "highly parallel algorithms" to refer to circuits that are both of polynomial size and of small depth e.g. logarithmic or a square of a logarithm.

2.2 Structural parameters of graphs

Definition 1. Let $G = (V, E)$ be an undirected graph. A *tree decomposition* of G is a tuple (T, X), where $T = (W, F)$ is a tree, and $X = \{X_1, \ldots, X_{|W|}\}$ with $X_i \subseteq V$ such that:

1. $\bigcup_{s=1}^{|T|} X_s = V$
2. For all $\{i, j\} \in E$, there exist $t \in W$, such that both $i, j \in X_t$.
3. For all $i \in V$, the subset $\{t : i \in X_t\}$ of W forms a subtree of T.

The quantity $\max_{t \in W} |X_t| - 1$ is called the width of (T, X). The *tree-width* of G, denoted by $\mathcal{TW}(G)$, is the minimum width over all tree decompositions of G. The path decomposition is defined similarly; T has to be a path and the tree-width is then called as path-width.

Determining the optimal tree (path) decomposition is NP-hard while the problem is approximable within factor $O(\log n)$ ($O(\log^2 n)$) [11]. Tree-width is closed under the operation of graph minors and we may assume up to logspace transformations that in a tree decomposition (T, X) of a graph G, the degree of T is at most 3. For a survey on tree-width we cite [9].

The diameter of a formula is related to the bandwidth of graphs.

Definition 2. For a graph $G = (V, E)$, let $f : V \to \{1, 2, \ldots, |V|\}$ be an injective map. The *bandwidth* of G, $\mathcal{B}(G)$ is defined as $\min_f \max_{ij \in E} |f(i) - f(j)|$. In the minimum bandwidth problem, we compute $\arg\min_f \mathcal{B}(G)$.

The bandwidth problem is NP-complete [28] and remains intractable even if the input graph is a tree of maximum degree 3 [21]. The problem is polylogarithmic approximable due to Feige [19]. More on bandwidth can be found in the survey [12].

For every graph G, a relation (see e.g. [30]) between tree-width, path-width and bandwidth is

$$\mathcal{TW}(G) \leq \mathcal{PW}(G) \leq \mathcal{B}(G).$$

2.3 Structural parameters of formulas

Definition 3. Let V be the set of variables of an ordered formula ϕ_π. For $x \in V$, let $f(x), l(x)$ be the index of the clause that x appears for the first and last time respectively. $\mathcal{D}(\phi_\pi)$ is $\max_{x \in V}(l(x) - f(x))$. We also set $\Delta(\psi) = \min_\pi \mathcal{D}(\psi_\pi)$, the diameter of ψ.

We associate a k-CNF formula ϕ with two graphs. The clause-graph C_ϕ of ϕ arises by representing each clause by a vertex, and introducing edges between vertices whose corresponding clauses share common variables. The incidence graph G_ϕ of ϕ is a bipartite graph. G_ϕ has a vertex for each clause and a vertex for each variable (ignoring polarities). A variable-vertex u_x is connected to clause-vertex u_c whenever the variable x appears in the clause c. Note that C_ϕ is a graph minor of G_ϕ.

For a formula ϕ , we further define tree-width $\mathcal{TW}(\phi)$, path-width $\mathcal{PW}(\phi)$ and bandwidth $\mathcal{B}(\phi)$ of ϕ to be

$$\mathcal{TW}(\phi) = \mathcal{TW}(G_\phi),\ \mathcal{PW}(\phi) = \mathcal{PW}(G_\phi),\ \mathcal{B}(\phi) = \mathcal{B}(C_\phi)$$

2.4 Relations between $\mathcal{TW}(\phi), \mathcal{PW}(\phi), \mathcal{B}(\phi)$ and $\Delta(\phi)$

Lemma 1. *For any ordered k-*CNF *formula ϕ_π, the following are true: (i) $\mathcal{B}(\phi) = \Delta(\phi)$, (ii) $\mathcal{PW}(\phi) \leq \log n \cdot \mathcal{TW}(\phi)$, (iii) $\mathcal{PW}(\phi) = O(\mathcal{D}(\phi_\pi))$.*

Proof. (i) Follows directly from the definitions 2 and 3.

(ii) For every graph G, $\mathcal{PW}(G) \leq \log n \cdot \mathcal{TW}(G)$, where n is the number of vertices of G.

(iii) Consider some k-CNF ordered formula ϕ_π on n variables with $\mathcal{D}(\phi_\pi) = \mathsf{d}(n)$ and set $r = \lceil m/(\mathsf{d}(n)+1) \rceil$. We decompose G_ϕ to a path of width $(k+1)\cdot \mathsf{d}(n)$. Define the path $P = v_1, v_2, \ldots, v_r$. For every i, X_i, that v_i is associated with, consists of the following two types of vertices: clause-vertices v_{c_i} corresponding to clauses c_i, for $i = (i-1)\cdot(\mathsf{d}(n)+1)+1$ to $i\cdot(\mathsf{d}(n)+1)$; variable-vertices v_x, for all variables x that are involved in clauses with vertices already in X_i. We claim that P is valid path decomposition of G_ϕ. Indeed, properties (1),(2) of definition 1 are trivially satisfied. As for the third one, consider any variable x and the associated vertex u_x of G_ϕ. By construction we only have to consider variable-vertices.

It remains to preclude the possibility that there exist indices $i < s < j$, such that u_x is in both X_i, X_j and $u_x \notin X_s$. But then, in ϕ_π, x does not appear in any of the $\mathsf{d}(t)+1$ clauses in X_s, and therefore the $\mathcal{D}(\phi_\pi) > (j-i-1)\cdot(\mathsf{d}(n)+1)$. Finally, since ϕ is k-CNF formula, for every i, $|X_i| \leq \mathsf{d}(n) + k\cdot\mathsf{d}(n)$. □

The previous lemma does not preclude the possibility that $\Delta(\phi), \mathcal{PW}(\phi)$ are related up to (say) some constant factor.

On the contrary, it is easy to devise instances where the path-width and the (unordered formula) diameter are off by linear factor. Such an example is the formula $\phi = (x_0 \vee x_1) \wedge (x_0 \vee x_2) \wedge \ldots (x_0 \vee x_n)$ with $\mathcal{D}(\phi) = n-1$. G_ϕ admits the following path decomposition (T, X): $T =< v_1, v_2, \ldots, v_n >$, $X_i = \{u_{c_i}, u_{x_0}, u_{x_i}\}$.

Recall that 3-SAT remains NP-complete even when a variable appears at most 3 times in the formula. That is, there is a large, hard set of formulas with a sparsity condition on the number of occurrences of variables which in particular means that these formulas have large diameter, path-width and tree-width, unless e.g. NP is contained in quasi-polynomial time. This raises the question whether large diameter happens only when the instances are either non-sparse or when considering computationally hard sets. The following example shows that this is not true in the strongest possible form. This example suggests that all three parameters of a formula, i.e. tree-width, path-width and diameter, can be off by as much as (almost) the upper bound of their relation ($\mathcal{PW}(\phi) \leq (\log n)\cdot\mathcal{TW}(\phi)$ and $\mathcal{TW}(\phi) \leq n\cdot\Delta(\phi)$), in a formula which is trivially satisfiable. In the following Lemma 3, we use a family of formulas with every variable appearing only in 3 clauses. For this we use theorem 1, p.204 from [36].

Theorem 2 (Smithline '95)**.** *For the complete k-ary tree T of height h, $\mathcal{B}(T) = \lceil k(k^h-1)/(k-1)(2h) \rceil$*

Lemma 3. *There exists a family formulas ϕ with n variables where each variable appears at most three times, for which $\Delta(\phi) = \Omega(n/\log n)$, $\mathcal{PW}(\phi) = \Theta(\log n)$ and $\mathcal{TW}(\phi) = 1$.*

Proof. We will determine a 3-CNF formula ϕ with positive literals, by defining its incidence graph $G_\phi = (V, E)$. We start with the rooted complete binary tree T of height $\log n'$, where $\log n'$ is even (the root has level 0). Label all nodes of T in a arbitrary breadth-first-search manner starting from the root. At an even level, associate vertex i with a new variable x_i; at an odd level, associate vertex j with a new clause c_j (see Figure 1 for an example). Define clause c_j to be the conjunction of the parental-node $x_{\lfloor j/2 \rfloor}$ and the two children-nodes x_{2j}, x_{2j+1}. Set ϕ to be the conjunction of all clauses and observe that $T = G_\phi$ and n to be the number of variable-vertices in G_ϕ; i.e. $n = \Theta(n')$.

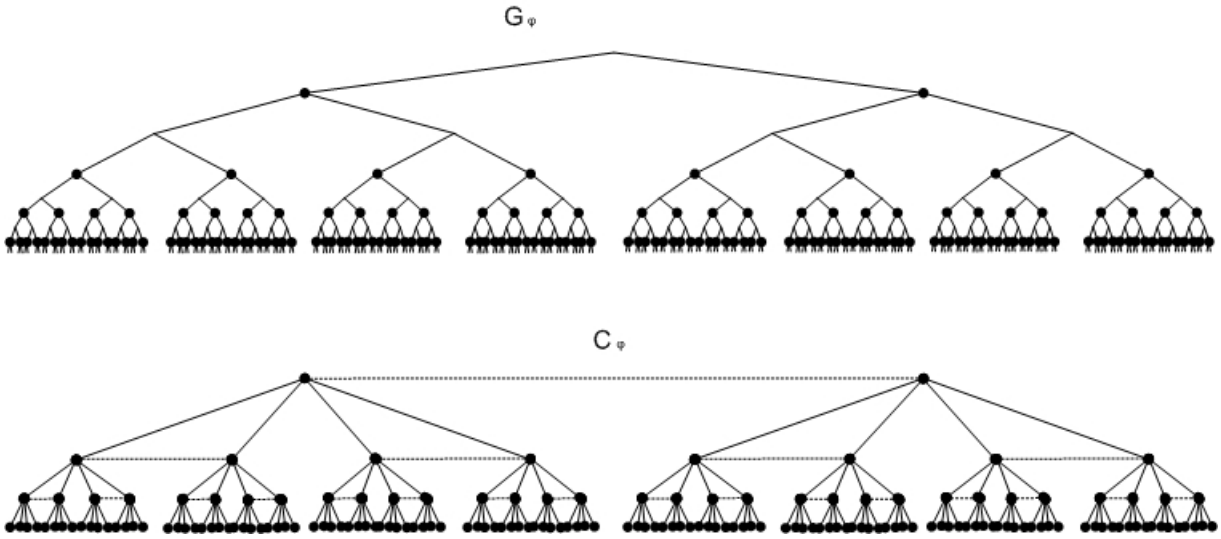

Figure 1: An example of G_ϕ with height 8. Vertices corresponding to clauses have been highlighted as thick nodes. The remaining nodes correspond to variable-vertices (e.g. the root and the leaves of G_ϕ). In C_ϕ, dashed edges connect clauses that appeared in G_ϕ at the same level (i.e., edges that connect clause-vertices sharing in G_ϕ a common ancestor). The removal of these vertices yields two complete trees of height 3

By definition $\mathcal{TW}(T) = 1$, and by Lemma 1, $\mathcal{PW}(T) \leq \log n'$. One can easily see that this bound is tight for complete trees. It remains to argue about the bandwidth of the clause graph C_ϕ. It is not difficult to see that if we remove edges from C_ϕ that connect clauses that appeared in T at the same level (i.e., edges that connect clause-vertices sharing in T a common ancestor), then the resulting graph consists of two disconnected complete trees, with every vertex having 4 children, and height at least $\lfloor \frac{\log n' - 1}{2} \rfloor$. Theorem 2 then implies that $\mathcal{B}(C_\phi) = \Omega(n'/\log n')$. □

Despite Lemma 3, we capitalize on the fact that the notions of diameter and path-width are the same up to some constant and up to a logspace transformation. It is also essential for Corollary 11 that Theorem 4 is constructive.

Theorem 4. *For any* k-CNF *formula* ϕ*, there exists an ordered* k-CNF *formula* ${\phi_\pi}'$ *with* $\Delta(\phi') \leq \mathcal{D}({\phi_\pi}') = \Theta(\mathcal{PW}(\phi))$ *such that* $\phi \in$ SAT *iff* ${\phi_\pi}' \in$ SAT*. Moreover, given the path decomposition of* ϕ*,* ${\phi_\pi}'$ *can be computed in logarithmic space with respect to the size of* ϕ*.*

Proof. Consider the path decomposition $X_1, \ldots, X_t$ of C_ϕ with $|X_i| = \mathsf{d}(n)$. We identify the vertices in the block X_i by the corresponding clauses and variables. We construct ${\phi_\pi}'$ as the output of the following iterative procedure.

For every block X_i do the following: (copy-step) output all the clauses of X_i in some order; (intercalate-step) for every variable x involved in X_i, output the renaming of x, $x \leftrightarrow x'$; finally replace all appearances of x in $X_{i+1}, \ldots X_t$ by x'. We call every clause introduced in the intercalate-step intercalary. ${\phi_\pi}'$ is the conjunction of the of the clauses ordered as the output suggests. By construction ϕ is satisfiable iff ${\phi_\pi}'$ is satisfiable.

It is clear that the previous procedure can be implemented in logarithmic space: instead of renaming all subsequent occurrences of x, just count its previous occurrences. In a reasonable renaming, the indices of the variables do not exceed $n + n + 2k \cdot t \cdot \mathsf{d}(n)$.

Example 1. Consider the clauses

$$c_1 = \overline{x}_1 \vee x_2 \vee x_3, c_2 = x_1 \vee \overline{x}_4 \vee x_5, c_3 = \overline{x}_2 \vee \overline{x}_6 \vee x_1, c_4 = \overline{x}_7 \vee \overline{x}_8 \vee \overline{x}_1, c_5 = x_5 \vee x_8 \vee x_9$$

and the formula $\phi = c_1 \wedge c_2 \wedge c_3 \wedge c_4 \wedge c_5$. A valid tree decomposition of G_ϕ is (T, X), with $T =< v_1, v_2, v_3, v_4 >$ and

$$\begin{aligned} X_1 &= \{u_{c_1}, u_{c_2}, u_{x_1}, u_{x_2}, u_{x_3}, u_{x_4}, u_{x_5}\} \\ X_2 &= \{u_{c_2}, u_{c_3}, u_{x_1}, u_{x_2}, u_{x_4}, u_{x_5}, u_{x_6}\} \\ X_3 &= \{u_{c_2}, u_{c_4}, u_{x_1}, u_{x_4}, u_{x_5}, u_{x_7}, u_{x_8}\} \\ X_4 &= \{u_{c_4}, u_{c_5}, u_{x_1}, u_{x_5}, u_{x_7}, u_{x_8}, u_{x_9}\} \end{aligned}$$

Now suppose that we are given only the path decomposition (T, X) with width 7. A possible execution of the algorithm of Theorem 4 constructs ${\phi_\pi}'$ as follows

Initially set ${\phi_\pi}' := \varnothing$
(copy-step) ${\phi_\pi}' := {\phi_\pi}' \wedge (\overline{x}_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \overline{x}_4 \vee x_5)$
(intercalate-step) ${\phi_\pi}' := {\phi_\pi}' \wedge (x_1 \leftrightarrow x_{10}) \wedge (x_2 \leftrightarrow x_{11}) \wedge (x_3 \leftrightarrow x_{12}) \wedge (x_4 \leftrightarrow x_{13}) \wedge (x_5 \leftrightarrow x_{14})$
(copy-step) ${\phi_\pi}' := {\phi_\pi}' \wedge (x_{10} \vee \overline{x}_{13} \vee x_{14}) \wedge (\overline{x}_{11} \vee \overline{x}_6 \vee x_{10})$
(intercalate-step) ${\phi_\pi}' := {\phi_\pi}' \wedge (x_{10} \leftrightarrow x_{15}) \wedge (x_{11} \leftrightarrow x_{16}) \wedge (x_{13} \leftrightarrow x_{17}) \wedge (x_{14} \leftrightarrow x_{18}) \wedge (x_6 \leftrightarrow x_{19})$
(copy-step) ${\phi_\pi}' := {\phi_\pi}' \wedge (x_{15} \vee \overline{x}_{17} \vee x_{18}) \wedge (\overline{x}_7 \vee \overline{x}_8 \vee \overline{x}_{15})$
(intercalate-step) ${\phi_\pi}' := {\phi_\pi}' \wedge (x_{15} \leftrightarrow x_{20}) \wedge (x_{17} \leftrightarrow x_{21}) \wedge (x_{18} \leftrightarrow x_{22}) \wedge (x_7 \leftrightarrow x_{23}) \wedge (x_8 \leftrightarrow x_{24})$
(copy-step) ${\phi_\pi}' := {\phi_\pi}' \wedge (\overline{x}_{23} \vee \overline{x}_{24} \vee \overline{x}_{20}) \wedge (x_{22} \vee x_{24} \vee x_9)$
(intercalate-step) ${\phi_\pi}' := {\phi_\pi}' \wedge (x_{20} \leftrightarrow x_{25}) \wedge (x_{22} \leftrightarrow x_{26}) \wedge (x_{23} \leftrightarrow x_{27}) \wedge (x_{24} \leftrightarrow x_{28}) \wedge (x_9 \leftrightarrow x_{29})$

Each intercalate-step follows the step of copying a set of clauses in some block. Note that in a intercalate-step, we rename all variables involved in the previous block, either as part of clauses or as variable-vertices. Moreover, each renaming of a variable affects also subsequent appearances. For example, variable x_1, renamed to x_{10} in the first intercalate-step, is treated as x_{10} in X_2, because of which it is once again renamed. Finally, we emphasize that the the previous execution deviates from the actual execution that requires bounded space resources. Our example only serves as a intuitive clarification of the high level of the idea.

Now, we calculate the ordered diameter of ϕ'_π. We distinguish between variables introduced in the copy-step and the intercalate-step. By the renamings, it is immediate that for any variable x of a clause introduced at the copy-step, the maximum distance between occurrences of x is at most $(2k + 1) \cdot \mathsf{d}(n)$.

For variables introduced in the intercalate-step we rely on the definition of path-width. Consider such a variable x introduced between blocks X_i, X_{i+1}. Variable x is (i) either a renaming of a former variable, or (ii) it is brand new variable that replaces y. Case (i) is easy to handle. For case (ii), the clause c of X where y appeared, either appears in X_{i+1} or not. If it does not appear, then by the definition of path-width, c does not appear in any subsequent block. Finally, if c appears in X_{i+1} then it will be renamed again when we consider the next block. In every case $\mathcal{D}(\phi_\pi{}') \leq (2k+1) \cdot \mathsf{d}(n)$. □

Motivated by the previous observations, we define

Definition 4 (Computational Problems)**.** $\text{SAT}(\mathsf{d}(n)), \text{MAX-SAT}(\mathsf{d}(n))$ and $\#\text{SAT}(\mathsf{d}(n))$ are the restrictions of SAT, MAX-SAT and #SAT respectively, where the instances ϕ_π are ordered formulas and obey $\mathcal{D}(\phi_\pi) \leq \mathsf{d}(n)$. Moreover, we will restrict our study to k-CNF formulas.

2.5 NAuxPDAs, Non-Deterministic Divide and Conquer and LOGCFL

A non-deterministic auxiliary pushdown automaton (NAuxPDA) is a generalization of a space-bounded Turing Machine (TM) extended by an unbounded stack. Cook [13] showed that every NAuxPDA bounded to work in space $s(n)$ and arbitrary time can be simulated by a computer program (deterministic TM) in time $2^{O(s(n))}$. Sudborough [37] showed that LOGCFL ($\subseteq \text{AC}^1$) is characterized by NAuxPDAs that run simultaneously in logarithmic space and polynomial time, i.e. one can express highly parallel algorithms as NAuxPDAs. Using NAuxPDAs one can simulate a special form of non-deterministic recursion and from there even a special form of divide and conquer. Non-deterministic Divide and Conquer (ND-DnC) [29] is a paradigm which simplifies the presentation of algorithms, something that recently made possible to obtain complex polynomial and subexponential algorithms whose translations into computer programs (DTM) are extremely complicated and unnatural. The transformation of an NAuxPDA to a TM or to parallel algorithms (e.g. circuits or PRAMs) is possible through strongly non-trivial translation theorems, see Section 4. The resulting TM (or parallel algorithm) can be conceptually complicated, still the transformation is explicit. For example, ND-DnC algorithms that have simple and elegant descriptions, can find practical applications through their transformations. As an application, we demonstrate in Section 4 a simple example.

3 Solving SAT($\mathsf{d}(n)$), Max-SAT($\mathsf{d}(n)$), #SAT($\mathsf{d}(n)$)

3.1 Algorithms and the main theorem

For $\mathsf{d}(n) \geq \log n$ we show that the satisfiability problem can be decided within non-deterministic space $O(\mathsf{d}(n))$, whereas for the maximization and counting problems it suffices to use deterministic space $O(\mathsf{d}^2(n))$. Moreover, all three problems can be solved in (deterministic) time $2^{O(\mathsf{d}(n))}$. The time-bounded and space-bounded algorithms (for maximization and counting) are obtained independently. Under the current knowledge in computational complexity we do not know how $\text{FDSPACE}(\mathsf{d}^2(n))$ compares to $\text{FDTIME}(2^{O(\mathsf{d}(n))})$. We deal with the case $\mathsf{d}(n) = O(\log n)$ in Section 3.3.

Theorem 5. $\text{SAT}(\mathsf{d}(n)) \in \text{NSPACE}(\mathsf{d}(n))$, $\text{MAX-SAT}(\mathsf{d}(n))$, $\#\text{SAT}(\mathsf{d}(n)) \in \text{FDSPACE}(\mathsf{d}^2(n))$. *Also,* $\text{MAX-SAT}(\mathsf{d}(n))$, $\#\text{SAT}(\mathsf{d}(n)) \in \text{FDTIME}(2^{O(\mathsf{d}(n))})$, $\mathsf{d}(n) = \Omega(\log n)$.

3.1.1 The satisfiability problem SAT($\mathsf{d}(n)$)

Solve-SAT (Algorithm 1) decides the problem in non-deterministic space $O(\mathsf{d}(n))$.

Algorithm 1 Solve-SAT

The input is an ordered k-CNF formula ϕ_π which $\mathcal{D}(\phi_\pi) = \mathsf{d}(n)$.

- Initially, consider a window (ordered subformula) W of length $\mathsf{d}(n)$ containing the first $\mathsf{d}(n)$ clauses of ϕ_π. Guess values for all variables in W and if the guess does not satisfy W then reject.
- Iteratively do the following.
- Slide the current position of the window W one clause to the right and free the space of the variables of the first clause of W.
- Guess (and store in the freed space) truth values for the variables of the new clause in the updated W. If the updated W is not satisfied or if the new values are inconsistent with those stored in the memory then reject. Otherwise, if there are more clauses in ϕ_π to the right of W then iterate; else accept.

This algorithm uses space $O(\mathsf{d}(n))$. We can standardize the way the truth assignment is stored. Reserve one bit for the variable of each occurrence of a literal in W repeating the value for variables which appear more than once; i.e. in total we have $k \cdot \mathsf{d}(n)$ space.

Correctness follows immediately. Suppose that ϕ_π is satisfiable, let τ be a satisfying thruth assignment, and then observe that we can construct a computational branch where the algorithm guesses the values of τ. For the other direction, suppose that there is a computational branch which accepts. For such a computational branch the following invariant is true for the main loop: at the end of the i-th iteration the subformula from the first to the $(i + \mathsf{d}(n))$-th clause is satisfied by the chosen truth values (which are consistently chosen).

3.1.2 The maximization problem Max-SAT($\mathsf{d}(n)$)

We reduce Max-SAT($\mathsf{d}(n)$) in deterministic space $O(\mathsf{d}(n))$ to the computation of the longest path problem in DAGs (Directed Acyclic Graphs). This is a significant improvement over the natural dynamic programming time-bounded algorithm.

We define DAG-Longest-Path to be the optimization problem where given a DAG $G = (V, E)$ and $w : E \to \mathbb{N}$, the goal is to output the (edge-weighted) length of a longest dipath.

Proposition 6. DAG-Longest-Path $\in$ FDEPTH($\log^2 N$). *Furthermore, this family of circuits has size polynomial in N.*

Here is a brief justification. Power the adjacency matrix using repeated squaring. This way we compute all walks of length N in depth $O(\log^2 N)$. Actually, something slightly stronger holds. Since matrix powering is in DET [15] we have that the decision version of DAG-Longest-Path is in DET $\subseteq$ NC^2.

Solve-MaxSAT (Algorithm 2) makes use of a space-efficient routine. This is the space simulation of the above longest path algorithm. It is well-known (see e.g. [41]) that DEPTH($s(N)$) $\subseteq$ DSPACE($s(N)$), $s(N) \geq \log_2 N$. That is, DAG-Longest-Path $\in$ FDSPACE($\log^2 n$), and furthermore the proof of the inclusion gives us an explicit space-efficient algorithm.

Algorithm 2 Solve-MaxSAT

The input is an ordered k-CNF formula ϕ_π with $\mathcal{D}(\phi_\pi) = \mathsf{d}(n)$. First we show how to reduce to DAG-Longest-Path in space $\mathsf{d}(n)$ and then we compose in the standard way two space efficient algorithms.

- The graph consists of blocks of vertices. Each block is associated with a a window (ordered subformula) W of length $\mathsf{d}(n)+1$, where W starts from a distinct position (clause) in the ordered ϕ_π. The i-th block is associated with the window which starts from the i-th clause of ϕ_π. Each of the vertices of each block is associated with a distinct, satisfying truth assignment for this window. We also introduce a fresh starting vertex s and assume it is associated with an empty subformula.
- There is an edge from a vertex v in block i to every other vertex u in block $j > i$ whenever the partial truth assignments of the two vertices are consistent (i.e. both extend to the same truth assignment). The weight of the edge (v, u) is the number of clauses in the window associated with u, satisfied by the partial truth assignment of u which are not (already) satisfied by the partial truth assignment of v. Let us call the constructed graph as H_{ϕ_π} (see Figure 2 for an example).
- Simulate the $O(\log^2 N)$ space-bounded algorithm for input of length N, on the constructed edge-weighted graph by composing the space bounded computations in the standard way.

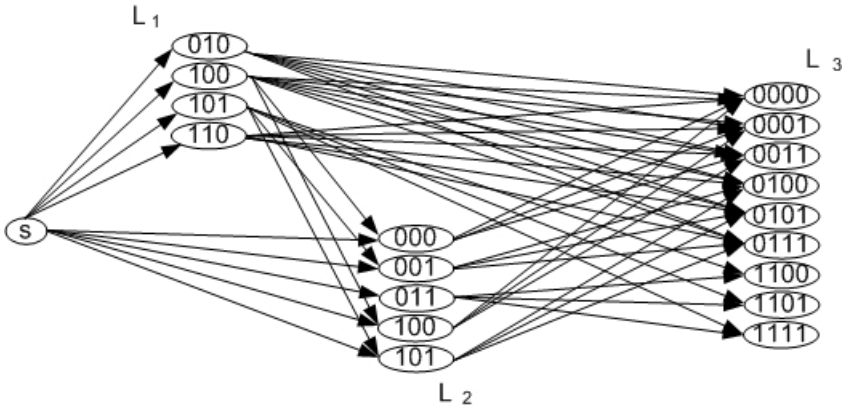

Figure 2: Example of H_{ϕ_π}, given $\phi_\pi = c_1 \wedge c_2 \wedge c_3 \wedge c_4$, where $c_1 = (x_1 \vee x_2)$, $c_2 = (\overline{x}_2 \vee \overline{x}_3)$, $c_3 = (\overline{x}_3 \vee x_4)$, $c_4 = (\overline{x}_5 \vee x_6)$. The vertices of H_{ϕ_π} are identified as partial assignments. Note that ϕ_π has diameter 1, and each one of the blocks L_1, L_2, L_3 is a partial assignment on two clauses: L_1, L_2, L_3 are partial assignments for the variables (x_1, x_2, x_3), (x_2, x_3, x_4) and (x_3, x_4, x_5, x_6) respectively. For example, node 0100 of L_3 is the partial assignment $x_3 = 0$, $x_4 = 1$, $x_5 = 0$,and $x_6 = 0$. For simplicity, we omitted edges from vertex s to L_3. Moreover, edges from s to any block has weight 2, edges from L_1 to L_2 and from L_2 to L_3 have weight 1, and edges from L_1 to L_3 have weight 2.

Solve-MaxSAT uses deterministic space $O\big(\log^2(n^{O(1)}2^{O(\mathsf{d}(n))}) + \mathsf{d}(n)\big) = O(\mathsf{d}^2(n))$. The time-bounded algorithm is obtained if instead simulating the space bounded algorithm we do dynamic programming to solve DAG-Longest-Path.

Correctness is transparent. Let us denote by $R(u, v)$ the relation that the partial truth assign-

ment of u is consistent with the partial truth assignment of v, and u is in a smaller-indexed block than v. Then, we observe that R is transitive and moreover R is represented by the edges in H_{ϕ_π}. Use R to argue for consistency of truth assignments as needed. Any longest path contains s. We finish by an easy induction on the index of blocks for paths starting from s.

3.1.3 The counting problem #SAT($\mathsf{d}(n)$)

The algorithm for #SAT($\mathsf{d}(n)$) proceeds by a logspace reduction (Reduce-#SAT) (Algorithm 3) to the problem of counting paths in a DAG. This graph is layered and unlabeled.

Algorithm 3 Reduce-#SAT

The input is an ordered k-CNF formula ϕ_π with $\mathcal{D}(\phi_\pi) = \mathsf{d}(n)$.

- We construct a layered directed graph. Each layer (block) is associated with a distinct position of a window (ordered subformula) W of length $\mathsf{d}(n)$; the i-th layer is associated with the window which starts from the i-th clause of ϕ_π. Each of the vertices of each layer is associated with a distinct, satisfying truth assignment for this window. We denote by L_i the subset of vertices of the i-th layer.
- There is an edge from a vertex v in layer i to every other vertex u in layer $i+1$ whenever the partial truth assignments of the two vertices are consistent.
- Add two fresh designated vertices s, t. Add an edge from s to every vertex in L_1. Let L_h be the last layer. Add an edge from each vertex $v \in L_h$ to t. Let us denote by F_{ϕ_π} the constructed graph.

Proposition 7. *The number of s-t dipaths in F_{ϕ_π} equals the number of satisfying truth assignments of ϕ_π.*

Proof. We define a mapping from the set of truth assignments of ϕ_π to the set of s-t paths in F_{ϕ_π}.Let τ be a satisfying truth assignment for ϕ_π. By definition τ satisfies all windows. For each of the corresponding partial truth assignment there exists a vertex in the corresponding layer. Since all of them extend to the same τ they are in particular consistent and thus by construction there is a directed path in F_{ϕ_π} from a vertex in the first to a vertex in the last layer.

First we show that this mapping is a function. Suppose that there are two paths associated one associated with τ_1 and the other with τ_2. Starting from s let us consider the first time that the two paths split at vertex v at the i-th layer. The associated two distinct vertices in the $(i+1)$-th layer correspond to distinct truth assignments; i.e. $\tau_1 \neq \tau_2$.

Now we show that the mapping is injective. Let τ_1, τ_2 be two satisfying truth assignments. Say that the x is a variable where $\tau_1(x) \neq \tau_2(x)$ and say that x appears in the i-th window. Then, by construction there are two distinct vertices in L_i one associated with τ_1 and the other with τ_2. That is, there are necessarily two distinct paths associated with each truth assignment.

To finish the proof we define the inverse of the above mapping. We leave the details to the reader. Technically we first show that if we consider the relation Q consisting of all pairs (variable, truth value) obtained from the associations of the vertices of the paths, then it is a function. To that end we define a relation R on pairs of nodes of the graph as in Section 3.1.2. Then, we show that this is a satisfying truth assignment. We conclude by showing that this mapping is an injective function. □

From this point on there are two ways to count the number of s-t paths. One is to reduce to an arithmetic circuit by mapping vertices in F_{ϕ_π} to $+$ gates and then apply the results in [40]. The other way is to deal with the problem directly. The later is even cleaner. The number of layers including s and t is $2+h$, where $h = m - \mathsf{d}(n)$. We conclude the proof of the following by repeated squaring in the semiring $\mathbb{N}$ with operations $+, \cdot$.

Proposition 8. *Let $A \in \mathbb{N}^{|V(F_{\phi_\pi})|\times|V(F_{\phi_\pi})|}$ be the adjacency matrix of F_{ϕ_π}. The number of s-t paths in F_{ϕ_π} equals the single non-zero entry of A^{1+k}. Moreover this can be computed by a polysize circuit of depth $O(\log^2 N)$.*

3.2 Strong, constructive extensions of the equivalence of Theorem 4

The equivalence of Theorem 4 extends to the maximization and counting problems. For #SAT we observe that in the reduction of Theorem 4, ϕ and ϕ' have the same number of satisfying assignments. For MAX-SAT the connection is less straightforward. We modify Theorem 4 and the graph H_{ϕ_π} in Solve-MaxSAT. We modify the proof of Theorem 4 in two ways. Let us identify all renamings of the initially (as given in the path-width decomposition) same clause as equivalent. The first modification in the proof of Theorem 4 is that the "copy-step" outputs only one representative among the equivalent ones; i.e. each (up to equivalence) clause is copied only once. Furthermore, we use "dummy" clauses (e.g. $x \vee \neg x$) in order to mark the beginning and the end of the copy-step in ϕ'. Finally, we modify Solve-MaxSAT in the way it constructs $H_{\phi_\pi{}'}$. Now, the windows are of variable length each determined by the marks introduced by the dummy clauses. From, $H_{\phi_\pi{}'}$ we exclude the intercalary windows, but we use the intercalary clauses when constructing the graph identifying variables by their equivalence class of renamings.

3.3 Diameter $O(\log n)$: Parallel algorithms & low complexity classes

When the diameter is $O(\log n)$ the corresponding problems are deeply buried inside P. Recall that the same holds for formulas of path-width $O(\log n)$.

Lemma 9. SAT($\log n$) *is* NL*-complete under many-to-one logspace reductions.*

Proof. Containment in NL follows by Theorem 5. We are not going to give the details of the proof since hardness follows the lines of the standard Cook-Levin reduction (e.g. [17]). We only point-out the differences and modifications with the NP-completeness of SAT. Let us briefly recall the main parts of the NP-completeness proof.

- We fix an arbitrary language $L \in$ NP and a non-deterministic machine M that decides L. M has only one (read-write) tape and it is standarized to run in time n^k for some constant $k \in \mathbb{N}$ and to halt with empty tape.

- The reduction deals with the encoding into a CNF formula the verification of a valid, accepting computation of M on a given input $w \in \{0,1\}^n$. A satisfying truth assignment corresponds to an accepting computation and a falsifying assignments correspond to rejecting or invelid computations. Recall that a computation is a sequence of succesively valid configurations, where a configuration is determined by the memory context, the position of the head and the state of the machine. The main technical part of the proof relies on the fact that given two configurations, if the machine-head is at cell i in one configuration then

in the next configuration things can only differ in a small neighborhood of three memory cells. Hence, we associate each of the n^k bits of each of the n^k configurations with a distinct propositional variable and express the constraints for a valid computation. There are also some details one has to take care of before writing the formula checking valid computations (for a detailed and careful exposition see [17]).

For the NL-hardness we also fix an arbitrary language in NL and a logspace bounded non-deterministic TM that decides it. Now, the machine has an extra read-only tape therefore the configuration is determined by the input, the head on the input, the working tape, the head on the working tape and the state of the machine. The technical modification of the NP-completeness proof involves considering: (i) partial configurations and (ii) systematic rewritings to avoid blow-up in the diameter of the formula. The partial configuration is determined by the working tape, the head of the working tape and the machine's state. Note that this partial configuration corresponds to a configuration of the machine in the NP-completeness proof. Fox an input x consider the computation of the NL-machine as a sequence of configurations and then project it to the sequence of partial configurations. Intuitively our reduction works by enforcing validity of extensions of partial computations to computations. Fix an input x and consider the computation that takes n^k steps. The two head indices, and the working tape are of size $O(\log n)$. We associate each of the bits of the partial configuration of each partial configuration with a distinct propositional variable. We also have a sequence of the values of the index to the read-only input tape and furthermore we associate each bit in each configuration of this index with a propositional variable. To avoid the formula diameter blow-up we work with partial configurations and particular values of the index to the input tape. Let us say that we are in the procedure of writing what it means for the computation to go from the i-th to the $(i+1)$-th step. We proceed as follows: We first write the subformula which semantically states that "if the value of the head to the input tape in the i-th step is 1, then check the corresponding bit in the read-ony tape the machine transitions and the states and update accordingly for the $(i+1)$th step the bits of the partial configuration and the value of the input-head index". The number of states, and the size of the alphabet is constant and since two partial configuration are of size $O(\log n)$ we have that this subformula is of size $O(\log n)$. Now we rewrite (i.e. give new names using equivalences) all variables involved in this step of the reduction and repeat the same procedure when the input-head index has value 2 and so on until the input-head index is n. Following the technical part of the NP-completeness of SAT and with these rewritings we construct a formula of diameter $O(\log n)$ which is satisfiable iff $M(x)$ accepts. Note that by padding with dummy clauses we can make the diameter $\log n$. □

As a corollary of Theorem 5 and its proof (in particular propositions 6 and 8) we obtain as a corollary that MAX-SAT($\log n$), #SAT($\log n$) are in the function analog of NC^2. In fact we can do slightly better as the following lemma suggests.

Lemma 10. *The decision version of* MAX-SAT($\log n$) $\in \mathrm{DET} \subseteq \mathrm{NC}^2$ *and* #SAT($\log n$) $\in$ #L *which is contained in the function analog of* $\mathrm{DET} \subseteq \mathrm{NC}^2$.

The containment of MAX-SAT($\log n$) follows from the algorithm (reduction) in the proof of Theorem 5 and by the fact that the decision version of DAG-LONGEST-PATH is in DET. The containment of #SAT($\log n$) follows by observing that the non-deterministic branches of the window algorithm for SAT($\mathtt{d}(n)$) correspond to distinct truth assignments.

Let us turn our attention to the satisfiability, maximization and counting problems when the path-width is $O(\log n)$. Using results presented in this section and in Section 3.2 we have the following corollary for formulas of $\log n$-bounded path-width.

Corollary 11 (Bounded path-width)**.** *Consider* k-CNF *instances of path-width* $O(\log n)$ *where the path-width decomposition is given in the input. For these instances the satisfiability problem is complete for* NL*, and the maximization and counting problems are in the function analog of* NC^2*.*

4 Improved results for formulas of bounded tree-width

Since tree-width is at worst $\log n$ smaller than path-width, the statements of Section 3 hold for tree-width when the value of the parameter is off by $\log n$ factor. Here, we improve on this corollary when it comes to SAT. In particular, we obtain an AC^1 algorithm for $\log n$ tree-width and by applying strongly non-trivial results from complexity theory, we provide simultaneous space and time efficiency as asked in [1].

4.1 Dealing directly with tree-width for SAT

Given a tree decomposition of formula of tree-width $t(n)$ we design an algorithm that in particular, when $t(n) = O(\log n)$ shows SAT $\in$ LOGCFL. This algorithm is one of the simplest applications of the Non-Deterministic Divide and Conquer paradigm [29]. For notational succinctness, in this section only, n corresponds to the total number of variables and clauses in a formula.

Algorithm 4 Solve-Treewidth-SAT

The input is a CNF formula ϕ and a tree decomposition (T, X) of width $t(n)$. Initially we make a call to Recurse-Treewidth-SAT$[r]$, where r is an arbitrary root of T. If the call returns then accept.

Recurse-Treewidth-SAT[root node v]

- Guess a truth assignment τ for the clauses and the variables corresponding to v. If τ does not satisfy the clauses associated with v then reject.
- If v is a leaf then return τ. Else, let u, w be the children of v (wlog v has two children - Section 2.2).
- Call Recurse-Treewidth-SAT$[u]$ which returns τ_u and then call Recurse-Treewidth-SAT$[w]$ which returns τ_w.
- If τ is not consistent with τ_u and τ_w then reject. Else, return τ.

Solve-Treewidth-SAT can be implemented on an NAuxPDA within $t(n)$ NAuxPDA-working memory and $n^{O(1)}$ NAuxPDA-time. When the tree-width is $t(n)$ then there are at most $t(n)$ clauses and variables whose truth values are checked at each level of the recursion. Moreover, the algorithm visits each node twice.

For the completeness direction suppose that ϕ is satisfiable, and let τ be a satisfying truth assignment. This is the trivial direction and does not even rely on tree decomposition properties.

For the soundness direction we use the tree decomposition properties and a little preparation is necessary.

Proposition 12. *Let ϕ be a k-*CNF *and (T, X) a tree decomposition of G_ϕ. Construct (T, X') by extending the association of each node u to be associated with all nodes corresponding to variables that appear in the clauses associated with u. Then, X' witnesses a tree-width constant times bigger than X.*

Proof. It is obvious that each set in X' is at most k times bigger than the corresponding one in X. (T, X') is a tree decomposition: Axioms (1) and (2) are easily satisfied; hence we check whether axiom (3) is satisfied too. For clause-vertices everything is as in X. For a variable-vertex y let the subtree $T_y = \{t \in T : y \in X_t\}$ and the set $T'_y = \{t \in T : y \in X'_t\}$. Let $v \in T'_y$ such that $v \notin T_y$, where $y \in C$ for a clause C. By property (2) of the definition there exists a node $u \in T_y$ which is associated with C. Moreover, there exists a path $P_{u,v}$ connecting v and u s.t. C is associated with every vertex in $P_{u,v}$. By construction of X' the vertex associated with y is also associated with every vertex in $P_{v,u}$. That is, in $X'i$ the subtree T_y extends to include v. □

We continue with the soundness direction. Fix an input ϕ where the algorithm accepts. Fix an arbitrary accepting computational branch. We define the binary relation Q to be the (variable, truth value) pairs that the algorithm assigned to variables in this computational branch. Below we show (i) Q is a function and (ii) it is a satisfying truth assignment.

Consider any two nodes u, v of the tree decomposition where at v we have $(x, True) \in Q$ and at u we have $(x, False) \in Q$. By Proposition 12 there exists $\{i, j\} \in T$ in the u-v path, such that $x \in X_i$ and $x \in X_j$ which contradicts the consistency check of the algorithm.

The proof of correctness finishes by defining and applying transitive relation R referring to consistent extensions of partial truth assignments.

When $t(n) = O(\log n)$ algorithm Solve-Treewidth-SAT runs in logspace and polytime which establishes the following strong theorem.

Theorem 13. SAT *of tree-width $O(\log n)$ is in* LOGCFL ($\subseteq \mathrm{AC}^1 \subseteq \mathrm{NC}^2$).

4.2 Alekhnovich and Razborov's question

Given a tree decomposition of width $t(n)$, the refutation algorithm of [1] runs in time and in space $O(n^{O(1)}2^{O(t(n))})$. By applying on Solve-Treewidth-SAT the deterministic time simulation of [13] (Theorem 1, p.7) we obtain an algorithm that runs in time $2^{O(t(n))}$ and space $2^{O(t(n))}$, $t(n) = \Omega(\log n)$, which matches the time-space bounds in [1] (note that when $t(n) = O(\log n)$ we have the very strong result of Theorem 13). In fact, when $t(n) = \omega(\log n)$ we improve on [1] as well. To that end we successively apply strongly non-trivial results from [33] and simple well-known results from structural complexity. It is worth noting that each theorem we apply is constructive and thus we successively transforms Solve-Treewidth-SAT. The following theorem is a corollary of three successive transformations in [33] Theorem 3, p.375 and Theorem 5(2),5(3) p.379.

Theorem 14 (Ruzzo '81)**.** *An NAuxPDA computing in space $s(n)$ and time $z(n)$ can be simulated by a family of circuits of size $2^{O(s(n))}$ and depth $O(s(n) \log z(n))$. Furthermore, this transformation between algorithms is given explicitly.*

By Theorem 14 we have a family of circuits of size $2^{O(t(n))}$ and depth $O(t(n) \log n)$ that decides SAT instances of tree-width $t(n)$. Apart from these parallel algorithms we have as an immediate consequence (just relying on the depth bound) the following.

Theorem 15. SAT *instances consisting of a* k-CNF *formulas together with tree decompositions of width* $t(n)$ *can be decided in space* $O(t(n)\log n)$ *and thus simultaneously in time* $2^{O(t(n)\log n)}$. *Furthermore, if the decomposition is not given we decide in time* $2^{O(t(n)\log n)}$ *and space* $n^{O(1)}$.

5 Open questions

Our work raises many questions which are left open. We consider as more fundamental the following four.

1. Study interrelations of SAT, MAX-SAT and #SAT for different bounds of the diameter; e.g. can we reduce #SAT($\mathsf{d}(n)$) to SAT($\mathsf{d}^2(n)$)?
2. Investigate structural complexity implications by assuming SAT instances of bounded diameter to be either in P or NP-complete.
3. Improve the result of Section 4.2 by reducing the exponent in the running time.
4. Finally, we are optimistic that our research will find empirical applications.

Acknowledgements

We would like to thank Paul Medvedev for bringing to our attention the equivalence between the CNF-diameter and the bandwidth of the clause graph, and Mohammad Moharrami for explaining tree-width-related concepts. We also thank Steve Cook for discussions on combinatorial circuits, and Phuong Nguyen and Matei David for useful suggestions on the presentation of this work.

References

[1] A. Alekhnovich and A. Razborov. Satisfiability, branch-width and tseitin tautologies. In *FOCS*, pages 593–603, 2002.

[2] C. Àlvarez and B. Jenner. A very hard log-space counting class. *Theoret. Comput. Sci.*, 107(1):3–30, 1993. Structure in complexity theory (Barcelona, 1990).

[3] E. Amir and S. Mcilraith. Solving satisfiability using decomposition and the most constrained subproblem. In *LICS workshop on Theory and Applications of Satisfiability Testing*, 2001.

[4] S. Arnborg and A. Proskurowski. Linear time algorithms for NP-hard problems restricted to partial k-trees. *Discrete Appl. Math.*, 23(1):11–24, 1989.

[5] S. Arora and S. Safra. Probabilistic checking of proofs: A new characterization of NP. *J. ACM*, 45:70–122, 1998.

[6] F. Bacchus, S. Dalmao, and T. Pitassi. Algorithms and complexity results for #SAT and bayesian inference. In *FOCS*, pages 340–351. IEEE Computer Society, 2003.

[7] M. Bern, E. Lawler, and A. Wong. Linear time computation of optimal subgraphs of decomposable graphs. *J. Algorithms*, 8:216–235, 1987.

[8] H. L. Bodlaender. Dynamic programming on graphs with bounded treewidth. In *Automata, languages and programming (Tampere, 1988)*, volume 317 of *Lecture Notes in Comput. Sci.*, pages 105–118. Springer, Berlin, 1988.

[9] H. L. Bodlaender. A tourist guide through treewidth. *Acta Cybernet.*, 11(1-2):1–21, 1993.

[10] H. L. Bodlaender. Treewidth: Algorithmic techniques and results. In *Mathematical foundations of computer science 1997 (Bratislava)*, volume 1295 of *Lecture Notes in Comput. Sci.*, pages 19–36. Springer, Berlin, 1997.

[11] H. L. Bodlaender, J. R. Gilbert, H. Hafsteinsson, and T. Kloks. Approximating treewidth, pathwidth, frontsize, and shortest elimination tree. *J. Algorithms*, 18(2):238–255, 1995.

[12] P. Z. Chinn, J. Chvátalová, A. K. Dewdney, and N. E. Gibbs. The bandwidth problem for graphs and matrices - A survey. *J. Graph Theory*, 6(3):223–254, 1982.

[13] S. A. Cook. Characterizations of pushdown machines in terms of time-bounded computers. *J. ACM*, 18(1):4–18, 1971.

[14] S. A. Cook. The complexity of theorem-proving procedures. In *STOC*, pages 151–158, 1971.

[15] S. A. Cook. A taxonomy of problems with fast parallel algorithms. *Inform. and Control*, 64(1-3):2–22, 1985.

[16] B. Courcelle, J. A. Makowsky, and U. Rotics. On the fixed parameter complexity of graph enumeration problems definable in monadic second-order logic. *Discrete Appl. Math.*, 108(1-2):23–52, 2001.

[17] Ding-Zhu Du and Ker-I Ko. *Theory of Computational Complexity*. Wiley-Interscience Series in Discrete Mathematics and Optimization. Wiley-Interscience, New York, 2000.

[18] J.A. Makowsky E. Fischer and E.R. Ravve. Counting truth assignments of formulas of bounded tree-width or clique-width. *Discrete Appl. Math.*, 155(14):1885–1893, 2007.

[19] U. Feige. Approximating the bandwidth via volume respecting embeddings. *J. Comput. Syst. Sci*, 60(3):510–539, 2000.

[20] M. Flouris, L. C. Lau, T. Morioka, P. A. Papakonstantinou, and G. Penn. Bounded and ordered satisfiability: connecting recognition with Lambek-style calculi to classical satisfiability testing. In *Math. of language 8*, 2003.

[21] M. R. Garey, R. L. Graham, D. S. Johnson, and D. E. Knuth. Complexity results for bandwidth minimization. *SIAM J. Appl. Math.*, 34(3):477–495, 1978.

[22] G. Gottlob, F. Scarcello, and M. Sideri. Fixed-parameter complexity in AI and nonmonotonic reasoning. *Artificial Intelligence*, 138(1–2):55–86, 2002.

[23] P. Hlineny, S. Oum, D. Seese, and G. Gottlob. Width parameters beyond tree-width and their applications. *The Computer Journal*, 8:216–235, 2007.

[24] S. Khanna and R. Motwani. Towards a syntactic characterization of PTAS. In *STOC*, pages 329–337. ACM, 1996.

[25] R. J. Lipton and R. E. Tarjan. Applications of a planar separator theorem. *SIAM J. of Comp.*, 9(3):615–627, 1980.

[26] N. Nishimura, P. Ragde, and S. Szeider. Solving #SAT using vertex covers. In *SAT*, volume 4121 of *Lecture Notes in Comput. Sci.*, pages 396–409, Berlin, 2006. Springer.

[27] S. Oum and P. Seymour. Approximating clique-width and branch-width. *J. Combin. Theory Ser. B*, 96(4):514–528, 2006.

[28] C. H. Papadimitriou. The NP-completeness of the bandwidth minimization problem. *Computing*, 16(3):263–270, 1976.

[29] P. A. Papakonstantinou, G. Penn, and Y. Vahlis. Subexponential-time algorithms for unidirectional Lambek Grammars. Unpublished manuscript.

[30] A. Parra, P. Scheffler, and T. Berlin. Treewidth equals bandwidth for AT-free claw-free graphs, Aug. 1995.

[31] N. Robertson and P. D. Seymour. Graph minors I. Excluding a forest. *J. of Comb. Theory (Ser. B)*, 35:39–61, 1983.

[32] N. Robertson and P. D. Seymour. Graph minors. II. algorithmic aspects of tree-width. *J. of Algorithms*, 7:309–322, 1986.

[33] W. L. Ruzzo. On uniform circuit complexity. *J. Comput. System Sci.*, 22(3):365–383, 1981. Special issue dedicated to Michael Machtey.

[34] M. Samer and S. Szeider. A fixed-parameter algorithm for #SAT with parameter incidence treewidth. *CoRR*, abs/cs/0610174, 2006. informal publication.

[35] M. Samer and S. Szeider. Algorithms for propositional model counting. In *LPAR*, volume 4790 of *Lecture Notes in Computer Science*, pages 484–498. Springer, 2007.

[36] L. Smithline. Bandwidth of the complete k-ary tree. *Discrete Math.*, 142(1-3):203–212, 1995.

[37] I. H. Sudborough. On the tape complexity of deterministic context-free languages. *J. Assoc. Comput. Mach.*, 25(3):405–414, 1978.

[38] S. Szeider. On fixed-parameter tractable parameterizations of SAT. In *SAT*, volume 2919 of *Lecture Notes in Computer Science*, pages 188–202. Springer, 2003.

[39] L. G. Valiant. The complexity of computing the permanent. *Theoret. Comput. Sci.*, 8(2):189–201, 1979.

[40] L. G. Valiant, S. Skyum, S. Berkowitz, and C. Rackoff. Fast parallel computation of polynomials using few processors. *SIAM J. Comput.*, 12(4):641–644, 1983.

[41] H. Vollmer. *Introduction to Circuit Complexity - A Uniform Approach*. Springer Verlag, 1999.