Complexity and Algorithms for Well-Structured *k*-SAT Instances

Konstantinos Georgiou[1] and Periklis A. Papakonstantinou[1,2]

[1] University of Toronto, Dept. of Computer Science, Toronto, ON, M5S 3G4, Canada
[2] University of Toronto, Dept. of Mathematics, Toronto, ON, M5S 2E4, Canada
{cgeorg,papakons}@cs.toronto.edu

Abstract. This paper consists of two conceptually related but independent parts. In the first part we initiate the study of k-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 investigate the relation between the diameter of a formula and the tree-width and the path-width of its corresponding incidence graph. We show that under highly parallel and efficient transformations, diameter and path-width are equal up to a constant factor. Our main result is that the computational complexity of SAT, Max-SAT, #SAT grows smoothly with the diameter (as a function of the number of variables). Our focus is in providing space efficient and highly parallel algorithms, while the running time of our algorithms matches previously known results. Our results refer to any diameter, whereas for the special case where the diameter is $O(\log n)$ we show NL-completeness of SAT and NC^2 algorithms for Max-SAT and #SAT.

In the second part we deal directly with k-CNF formulas of bounded tree-width. We describe algorithms in an intuitive but not-so-standard model of computation. Then we apply constructive theorems from computational complexity to obtain deterministic time-efficient and simultaneously space-efficient algorithms for k-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 [9], Max-SAT is NP-hard to approximate within some constant [3], while #SAT is hard for #P [32]. The intractability of SAT, Max-SAT and #SAT soon led to the study of restricted versions based on hidden structures of formulas and in particular on the so-called width restrictions. In this work, first we introduce a natural structural width parameter directly defined on k-CNF formulas that we call diameter. We consider SAT, Max-SAT and #SAT and parameterize them with respect to diameter, giving space-efficient and parallel algorithms. Second, given the tree decomposition of the incidence graph of a formula, we show how to decide SAT in simultaneously efficient time and space.

H. Kleine Büning and X. Zhao (Eds.): SAT 2008, LNCS 4996, pp. 105–118, 2008.

Parameterizing SAT instances using width parameters follows the more general study of NP-hard graph problems initiated by Lipton and Tarjan [19]. Along these lines, Robertson and Seymour [24,25] introduced tree-width that has been widely used to parameterize the complexity of many NP-hard problems, see e.g. surveys [5,17]. When it comes to SAT, a CNF formula can be associated with many underlying graphs and for each one of them a number of width parameters can be defined e.g. tree-width, path-width, clique-width, branch-width and cluster-width (for a comparison see [20]). There are numerous works parameterizing SAT with respect to width parameters. In what follows, due to space limitations, our exposition is far from being complete.

Khanna and Motwani [18] considered Max-SAT for formulas of constant tree-width, while [2] exploits the same structural property for SAT. Deciding SAT has been proved fixed-parameter tractable with respect to branch-width by Alekhnovich and Razborov [1], and to tree-width by Gottlob *et al* [16] on primal graphs. Using DPLL procedures, Bacchus, Dalmao and Pitassi, [4] considered #SAT, while the same time-bound for #SAT was achieved by Samer and Szeider [27] extending [16]. Fixed-parameter tractability of SAT and #SAT has also been considered in e.g. [10,13,20,21,28]; see also [31] for a survey.

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 a formula ϕ fully coincides with the bandwidth (see [7] for a survey) of the intersection graph of ϕ. In this work, we consider ordered k-CNF instances of bounded diameter, and we do not deal with the independent and well-studied problem of finding the best ordering (equivalent to bandwidth minimization) which is NP-complete.

It is worth noting that the subproblem of k-SAT instances of diameter n^{ϵ}, $\epsilon > 0$, where n is the number of variables, is NP-complete. In contrast we show that k-CNF formulas of $\log n$ diameter encode arbitrary NL computations. Arbitrary NL-computations are objects exhibiting highly complex interactions between their parts. Hence, it is intuitively clear that by considering instances of bounded ordered diameter we do not break the problem into independent problems (a preliminary study for a similar problem was given in [14]). 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 2). Despite this, we prove that by a highly efficient algorithm (Theorem 2), a formula of path-width $\mathsf{d}(n)$ can be viewed as a formula of diameter $O(\mathsf{d}(n))$. Hence we (computationally) counter any undesirable properties of the diameter and we only keep its simplicity. For ordered instances of SAT, Max-SAT and #SAT of bounded diameter we design space-efficient and time-efficient algorithms, showing that the complexity of all three problems grows smoothly with respect to the diameter. If in particular the instances are of sufficiently small diameter, we present algorithms that in addition are highly parallel. A strong point of this work is that these algorithms appear to have quite intuitive descriptions. To the best of our knowledge this is the first work that simultaneously gives efficient time and space fixed parameter tractability bounds or even deals with parallelization issues for SAT, MAX-SAT and #SAT.

Additional motivation for the study of SAT with respect to simultaneously time and space tractability is explicitly given by Alekhnovich and Razborov [1]. Given instances of bounded branch-width $w(n)$ and given a decomposition, they decide SAT in time $n^{O(1)}2^{O(w(n))}$ and in space $n^{O(1)}2^{O(w(n))}$; they further ask whether it is possible to reduce the space to polynomial preserving time efficiency. The last part of our paper goes in a fashion independent to the study of diameter. A consequence of our study is a new algorithm that matches the same time-space bounds as in [1], and more importantly an algorithm that works in time $n^{O(1)}2^{O(w(n)\log n)}$ and space $n^{O(1)}$.

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 ϕ. In an input, an unordered (ordered) formula ϕ (ϕ_π) is represented in the standard way as a sequence (sequence in the given order) of clauses. We consistently use n to denote the number of variables in a formula. N is used to denote the size 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. DEPTH($f(N)$) is the class of languages decidable by a family of circuits in depth $f(N)$. DSPACE($f(N)$), NSPACE($f(N)$), DTIME($f(N)$) denotes the class of problems decidable in deterministic, non-deterministic space and deterministic time $f(N)$ respectively. For the function analogs of decision complexity classes we extend the notation introducing a leading F; e.g. FDSPACE($\log^2 N$). NC^i (AC^i) is the class of languages decidable by polynomial size circuits of depth $O(\log^i N)$ where the gates are of bounded (unbounded) fan-in. We denote by NL $=$ NSPACE($\log N$). Our notation is standard, see e.g. [11,34]. LOGCFL is the class of languages logspace reducible to Context Free Languages (see Section 2.5). When the input is a formula of n variables we abuse notation by writing CompClass($f(n)$) instead of 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 term* path-width *is used instead of tree-width.*

Determining the optimal tree (path) decomposition is NP-hard while the problem is approximable within factor $O(\log n)$ ($O(\log^2 n)$) [6]. Tree-width is closed under the operation of graph minors and wlog we may assume that the number of nodes of the tree decomposition (T, X) of a graph G is linear, and that up to logspace transformations the degree of T is at most 3. For a survey on tree-width we cite [5].

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 f witnessing $\mathcal{B}(G)$.*

The bandwidth problem is NP-complete [22] and remains intractable even if the input graph is a tree of maximum degree 3 [15]. The problem is polylogarithmic approximable due to Feige [12]. See [7] for a not-so-recent survey.

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. The* ordered diameter *is $\mathcal{D}(\phi_\pi) = \max_{x \in V}(l(x) - f(x))$ and the* unordered diameter *is $\Delta(\psi) = \min_\pi \mathcal{D}(\psi_\pi)$.*

In this work we associate a k-CNF formula ϕ with two graphs. The *incidence graph G_ϕ of ϕ* is a bipartite graph. G_ϕ has a distinct vertex for each clause and each variable. A variable-vertex u_x is connected to clause-vertex u_c whenever the variable x appears in the clause c. The *clause-graph C_ϕ of ϕ* (intersection graph) arises by associating each clause with a distinct vertex. An edge connects vertices whose clauses share a variable. In [31] it is shown that the tree-width of the incidence graph is always smaller than the corresponding width parameters on other graphs appearing in the literature.

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) $\mathcal{PW}(\phi) \le \log n \cdot \mathcal{TW}(\phi)$, (iii) $\mathcal{PW}(\phi) = O(\mathcal{D}(\phi_\pi))$.*

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

(ii) For every graph G on n vertices, $\mathcal{PW}(G) \leq \log n \cdot \mathcal{TW}(G)$.

(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.

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

Lemma 1 does not preclude the possibility that $\Delta(\phi), \mathcal{PW}(\phi)$ are related up to (say) some constant factor. Combinatorially, things are the worst possible regarding the diameter. We show that even when each variable appears a small constant number of times the gap between tree-width (path-width) and diameter is off by almost linear factor. For this we use theorem 1, p.204 from [29].

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

Lemma 2. *There exists a family formulas ϕ with n variables each one appearing only 3 times, for which $\Delta(\phi) = \Omega(n/\log n)$, $\mathcal{PW}(\phi) = O(\log n)$ and $\mathcal{TW}(\phi) = 1$.*

Proof. We determine a 3-CNF formula ϕ with positive literals, by defining its incidence graph G_ϕ. 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 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. 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 n the number of variable-vertices in G_ϕ. Observe that $T = G_\phi$ and $n = \Theta(n')$.

By definition $\mathcal{TW}(T) = 1$, and by Lemma 1, $\mathcal{PW}(T) \leq \log n'$. Next we argue about the bandwidth of C_ϕ. It is easy 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), the resulting graph consists of two disconnected complete trees. Every vertex has 4 children, and height at least $\lfloor \frac{\log n' - 1}{2} \rfloor$. Theorem 1 then implies that $\mathcal{B}(C_\phi) = \Omega(n'/\log n')$. □

Despite Lemma 2, 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 1 (see below) that Theorem 2 is constructive.

Theorem 2. *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* $\phi \in \mathrm{SAT}$ *iff* ${\phi_\pi}' \in \mathrm{SAT}$*. 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 in X_i or in the clauses of 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 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 $n + n + 2k \cdot t \cdot \mathsf{d}(n)$.

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, and for k-CNF formulas, we define

Definition 4 (Computational Problems). $\mathrm{SAT}(\mathsf{d}(n)), \mathrm{Max\text{-}SAT}(\mathsf{d}(n))$ *and* $\#\mathrm{SAT}(\mathsf{d}(n))$ *are the restrictions of* SAT, Max-SAT *and* #SAT *respectively, where the instances* ϕ_π *are ordered formulas and obey* $\mathcal{D}(\phi_\pi) \leq \mathsf{d}(n)$*.*

2.5 NAuxPDAs: A Practical Model of Computation

A non-deterministic auxiliary pushdown automaton (NAuxPDA) is a generalization of a space-bounded Turing Machine (TM) extended by an unbounded stack. Cook [8] showed that every NAuxPDA bounded to work in space $s(n)$ and arbitrary time can be simulated by a TM in time $2^{O(s(n))}$. Sudborough [30] showed that LOGCFL ($\subseteq \mathrm{AC}^1 \subseteq \mathrm{NC}^2$) is characterized by NAuxPDAs that run simultaneously in logarithmic space and polynomial time. 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) [23] is a paradigm which simplifies the presentation of algorithms, something that recently made possible to obtain complex polynomial time algorithms whose translations into TMs are extremely complicated and unnatural. The transformation of an NAuxPDA to a TM or to parallel algorithms (e.g. circuits or PRAMs) is possible and explicit through strongly non-trivial translation theorems, see Section 4, although the resulting TM can be conceptually complicated. Among others, application of these theorems shows that ND-DnC algorithms that have simple and elegant descriptions can find practical applications through their transformations. An example of such an application is demonstrated in Section 4.

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

3.1 Algorithms for $\mathsf{d}(n) = \Omega(\log n)$

This section is devoted to $\mathsf{d}(n) = \Omega(\log n)$. We show that SAT can be decided within non-deterministic space $O(\mathsf{d}(n))$, whereas for Max-SAT and #SAT it suffices to use deterministic space $O(\mathsf{d}(n)^2)$. Moreover, all three problems can be solved in (deterministic) time $2^{O(\mathsf{d}(n))}$. The time-bounded and space-bounded algorithms for Max-SAT and #SAT are obtained independently. Under the current knowledge in computational complexity we do not know how FDSPACE($\mathsf{d}^2(n)$) compares to FDTIME($2^{O(\mathsf{d}(n))}$).

Theorem 3. *SAT($\mathsf{d}(n)$) $\in$ NSPACE($\mathsf{d}(n)$), Max-SAT($\mathsf{d}(n)$), #SAT($\mathsf{d}(n)$) $\in$ FDSPACE($\mathsf{d}(n)^2$); Max-SAT($\mathsf{d}(n)$),#SAT($\mathsf{d}(n)$) $\in$ FDTIME($2^{O(\mathsf{d}(n))}$).*

The satisfiability problem SAT($\mathsf{d}(n)$)
Solve-SAT (Algorithm 1) shows that SAT($\mathsf{d}(n)$) $\in$ NSPACE($\mathsf{d}(n)$). We can standardize the way the truth assignment is stored. Reserve one bit for the

---

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.

---

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.

For the correctness it is easy to see that there is a computational branch which accepts iff there exists a satisfying truth assignment for ϕ_π. Details omitted from proofs are given in the full version of the paper.

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

We define DAG-Longest-Path to be the optimization problem where given a DAG (Directed Acyclic Graphs) $G = (V, E)$ and $w : E \rightarrow \mathbb{N}$, the goal is to output the (edge-weighted) length of a longest dipath. We reduce Max-SAT($\mathsf{d}(n)$) in deterministic space $O(\mathsf{d}(n))$ to DAG-Longest-Path. This is a significant improvement over the natural dynamic programming time-bounded algorithm.

Lemma 3. DAG-Longest-Path $\in$ FDEPTH($\log^2 N$). *Furthermore, this family of circuits has size polynomial in N. In particular, the problem is in* P.

Here is a brief justification. Power the adjacency matrix using repeated squaring, over the semiring $\mathbb{N}$ with operations $(\max, +)$ instead of $(+, \cdot)$. This way we compute all walks of length N in depth $O(\log^2 N)$.

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. [34]) 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 working 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 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 v, u are consistent. The weight of the edge (v, u) is the number of clauses in the window associated with u, satisfied by u and not (already) by v. Let us call the constructed graph as H_{ϕ_π}.
- Solve DAG-Longest-Path for H_{ϕ_π}.

The reduction works in space $O(\mathsf{d}(n))$ since we can enumerate all pairs of vertices in H_{ϕ_π} in that space. Hence, Solve-MaxSAT requires 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 we do matrix powering using repeated squaring (or dynamic programming) to solve DAG-Longest-Path.

Correctness is transparent. Let us denote by $R(u, v)$ the relation that u is consistent with 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}. R can be used to prove consistency of truth assignments. Any longest path contains s. We finish by an easy induction on the index of blocks for paths starting from s.

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.

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.

Lemma 4. *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.

It is not hard to see why this mapping is a function (e.g. by considering the first time that two paths split) and why it is injective. Similarly, we define an inverse 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 [33]. 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 $+, \cdot$.

Theorem 4. *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+h}*. 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 2

The equivalence of Theorem 2 extends to Max-SAT and #SAT. The details are given in the full version of this paper. For #SAT we observe that in the reduction of Theorem 2, ϕ and ϕ' have the same number of satisfying assignments. For Max-SAT the connection is less straightforward. We modify Theorem 2 and the graph H_{ϕ_π} in Solve-MaxSAT. In the proof of Theorem 2 we omit occurrences of a clause in multiple blocks X_i's. Furthermore, it is possible to mark on ϕ' the beginning and the end of each copy-step using "dummy" clauses. Given the transformed bounded diameter formula we construct $H_{\phi_\pi'}$ by defining windows according to the previously introduced dummy clauses. Also, we omit all windows of intercalary clauses but we use their induced relations to connect the vertices.

3.3 Diameter $O(\log n)$: Parallel Algorithms and Low Complexity Classes

When $\mathsf{d}(n) = O(\log n)$ the corresponding problems are deeply buried inside P. The proof of Lemma 5 follows the lines of the standard Cook-Levin reduction modified with systematic rewritings to avoid diameter blow-up.

Lemma 5. SAT($\log n$) *is* NL*-complete under many-to-one logspace reductions.*

As a corollary of Theorem 3 and its proof (in particular Lemma 3 and Theorem 4) we obtain,

Lemma 6. Max-SAT($\log n$)*,* #SAT($\log n$) *are in the function analog of* NC^2*.*

Let us consider SAT, Max-SAT and #SAT for formulas of path-width $O(\log n)$. Results of this section and of Section 3.2 derive the following corollary.

Corollary 1 (Bounded path-width). *Consider* k-CNF *instances of path-width* $O(\log n)$ *where the path decomposition is given. For these instances* SAT *is complete for* NL*, and* Max-SAT, #SAT *are in the function analog of* NC^2*.*

4 Improved Results for k-CNFs 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. To that end our treatment in this section is independent to the results obtained for the diameter. We obtain an AC^1 algorithm for $\log n$ tree-width. Furthermore, by applying strongly non-trivial results from complexity theory, we provide simultaneous space and time efficiency as asked in [1] (even for the weaker notion of the tree-width of the primal graph).

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. 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 k-CNF formula ϕ and a tree decomposition (T, X) of width $t(n)$ and of degree at most 3 (see Section 2.2). 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
- Set τ_u = Recurse-Treewidth-SAT$[u]$ and τ_w = Recurse-Treewidth-SAT$[w]$.
- 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 using space $t(n)$ and time $n^{O(1)}$ (wlog the number of nodes in the decomposition is linear to the number of nodes in the graph). 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.

The proof of completeness is easy and does not even rely on tree decomposition properties. For the soundness we use the tree decomposition properties and a little preparation is necessary.

Lemma 7. *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 set 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' the subtree T_y is extended 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. We need to show that Q is a function and that 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 7 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 5. k-SAT *with tree decompositions of width* $O(\log n)$ *is in* LOGCFL.

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 [8] (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 5). In fact, when $t(n) = \omega(\log n)$ we improve on [1] as well. To that end we successively apply non-trivial results from [26] and simple well-known results from structural complexity. It is worth noting that each theorem we apply is constructive and thus we successively transform Solve-Treewidth-SAT. The following theorem is a corollary of three successive transformations in [26] Theorem 3, p.375 and Theorem 5(2),5(3) p.379.

Theorem 6 (Ruzzo '81). *NAuxPDAs working 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.*

Theorem 6 gives a family of circuits of size $2^{O(t(n))}$ and depth $O(t(n) \log n)$ deciding SAT instances of tree-width $t(n)$. Apart from these parallel algorithms we have the following as an immediate consequence of the depth bound.

Theorem 7. 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 most 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.

Acknowledgments

We would like to thank Paul Medvedev for bringing to our attention the equivalence between the CNF-diameter and the bandwidth of the intersection 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. Alekhnovich, A., Razborov, A.: Satisfiability, branch-width and Tseitin tautologies. In: FOCS, pp. 593–603 (2002)
2. Amir, E., Mcilraith, S.: Solving satisfiability using decomposition and the most constrained subproblem. In: LICS workshop on Theory and Applications of Satisfiability Testing. Electronic Notes in Discrete Mathematics (2001)
3. Arora, S., Safra, S.: Probabilistic checking of proofs: A new characterization of NP. J. ACM 45, 70–122 (1998)
4. Bacchus, F., Dalmao, S., Pitassi, T.: Algorithms and complexity results for #SAT and bayesian inference. In: FOCS, pp. 340–351 (2003)
5. Bodlaender, H.L.: A tourist guide through treewidth. Acta Cybernet 11(1-2), 1–21 (1993)
6. Bodlaender, H.L., Gilbert, J.R., Hafsteinsson, H., Kloks, T.: Approximating treewidth, pathwidth, frontsize, and shortest elimination tree. J. Algorithms 18(2), 238–255 (1995)
7. Chinn, P.Z., Chvátalová, J., Dewdney, A.K., Gibbs, N.E.: The bandwidth problem for graphs and matrices - A survey. J. Graph Theory 6(3), 223–254 (1982)
8. Cook, S.A.: Characterizations of pushdown machines in terms of time-bounded computers. J. ACM 18(1), 4–18 (1971)
9. Cook, S.A.: The complexity of theorem-proving procedures. In: STOC, pp. 151–158 (1971)
10. Courcelle, B., Makowsky, J.A., Rotics, U.: On the fixed parameter complexity of graph enumeration problems definable in monadic second-order logic. Discrete Appl. Math. 108(1-2), 23–52 (2001)
11. Du, D.-Z., Ko, K.-I.: Theory of Computational Complexity. Wiley-Interscience, New York (2000)
12. Feige, U.: Approximating the bandwidth via volume respecting embeddings. J. Comput. Syst. Sci 60(3), 510–539 (2000)
13. Fischer, E., Makowsky, J.A., Ravve, E.R.: Counting truth assignments of formulas of bounded tree-width or clique-width. Discrete Appl. Math. 155(14), 1885–1893 (2007)

14. Flouris, M., Lau, L.C., Morioka, T., Papakonstantinou, P.A., Penn, G.: Bounded and ordered satisfiability: connecting recognition with Lambek-style calculi to classical satisfiability testing. In: Math. of language 8, pp. 45–56 (2003)
15. Garey, M.R., Graham, R.L., Johnson, D.S., Knuth, D.E.: Complexity results for bandwidth minimization. SIAM J. Appl. Math. 34(3), 477–495 (1978)
16. Gottlob, G., Scarcello, F., Sideri, M.: Fixed-parameter complexity in AI and nonmonotonic reasoning. Artificial Intelligence 138(1–2), 55–86 (2002)
17. Hlineny, P., Oum, S., Seese, D., Gottlob, G.: Width parameters beyond tree-width and their applications. The Computer Journal 8, 216–235 (2007)
18. Khanna, S., Motwani, R.: Towards a syntactic characterization of PTAS. In: STOC, pp. 329–337. ACM, New York (1996)
19. Lipton, R.J., Tarjan, R.E.: Applications of a planar separator theorem. SIAM J. of Comp. 9(3), 615–627 (1980)
20. Nishimura, N., Ragde, P., Szeider, S.: Solving #SAT using vertex covers. In: Biere, A., Gomes, C.P. (eds.) SAT 2006. LNCS, vol. 4121, pp. 396–409. Springer, Heidelberg (2006)
21. Oum, S., Seymour, P.: Approximating clique-width and branch-width. J. Combin. Theory Ser. B 96(4), 514–528 (2006)
22. Papadimitriou, C.H.: The NP-completeness of the bandwidth minimization problem. Computing 16(3), 263–270 (1976)
23. Papakonstantinou, P.A., Penn, G., Vahlis, Y.: Polynomial-time and parallel algorithms for fragments of Lambek Grammars (unpublished manuscript)
24. Robertson, N., Seymour, P.D.: Graph minors I. Excluding a forest. J. of Comb. Theory (Ser. B) 35, 39–61 (1983)
25. Robertson, N., Seymour, P.D.: Graph minors. II. algorithmic aspects of tree-width. J. of Algorithms 7, 309–322 (1986)
26. Ruzzo, W.L.: On uniform circuit complexity. J. Comput. System Sci. 22(3), 365–383 (1981) Special issue dedicated to Michael Machtey
27. Samer, M., Szeider, S.: A fixed-parameter algorithm for #SAT with parameter incidence treewidth. CoRR, abs/cs/061017 (2006) informal publication
28. Samer, M., Szeider, S.: Algorithms for propositional model counting. In: Dershowitz, N., Voronkov, A. (eds.) LPAR 2007. LNCS (LNAI), vol. 4790, pp. 484–498. Springer, Heidelberg (2007)
29. Smithline, L.: Bandwidth of the complete k-ary tree. Discrete Math. 142(1-3), 203–212 (1995)
30. Sudborough, I.H.: On the tape complexity of deterministic context-free languages. J. Assoc. Comput. Mach. 25(3), 405–414 (1978)
31. Szeider, S.: On fixed-parameter tractable parameterizations of SAT. In: Giunchiglia, E., Tacchella, A. (eds.) SAT 2003. LNCS, vol. 2919, pp. 188–202. Springer, Heidelberg (2004)
32. Valiant, L.G.: The complexity of computing the permanent. Theoret. Comput. Sci. 8(2), 189–201 (1979)
33. Valiant, L.G., Skyum, S., Berkowitz, S., Rackoff, C.: Fast parallel computation of polynomials using few processors. SIAM J. Comput. 12(4), 641–644 (1983)
34. Vollmer, H.: Introduction to Circuit Complexity - A Uniform Approach. Springer, Heidelberg (1999)