Journal of Artificial Intelligence Research 78 (2023) 673–707 Submitted 03/2023; published 11/2023

On the Parallel Parameterized Complexity of MaxSAT Variants

Max Bannach MAX.BANNACH@ESA.INT
European Space Agency, Advanced Concepts Team, Noordwijk, The Netherlands

Malte Skambath MALTE.SKAMBATH@EMAIL.UNI-KIEL.DE
Department of Computer Science, Kiel University, Kiel, Germany

Till Tantau TANTAU@TCS.UNI-LUEBECK.DE
Institute for Theoretical Computer Science, Universität zu Lübeck, Lübeck, Germany

Abstract

In the maximum satisfiability problem (MAX-SAT) we are given a propositional formula in conjunctive normal form and have to find an assignment that satisfies as many clauses as possible. We study the *parallel* parameterized complexity of various versions of MAX-SAT and provide the first *constant-time* algorithms parameterized either by the solution size or by the allowed excess relative to some guarantee. For the *dual parameterized* version where the parameter is the number of clauses we are allowed to leave unsatisfied, we present the first parallel algorithm for MAX-2SAT (known as ALMOST-2SAT). The difficulty in solving ALMOST-2SAT in *parallel* comes from the fact that the iterative compression method, originally developed to prove that the problem is fixed-parameter tractable at all, is inherently *sequential.* We observe that a graph flow whose value is a parameter can be computed in parallel and develop a parallel algorithm for the vertex cover problem parameterized above the size of a given matching. Finally, we study the parallel complexity of MAX-SAT parameterized by the vertex cover number, the treedepth, the feedback vertex set number, and the treewidth of the input's incidence graph. While MAX-SAT is fixed-parameter tractable for all of these parameters, we show that they allow different degrees of possible parallelization. For all four we develop dedicated parallel algorithms that are *constructive*, meaning that they output an optimal assignment – in contrast to results that can be obtained by parallel meta-theorems, which often only solve the decision version.

1. Introduction

Maximum satisfiability problems ask us to find solutions for constraint systems that satisfy as many constraints as possible. The perhaps best-studied version is MAX-SAT, where the constraint system is a propositional formula in conjunctive normal form, and the goal is to find an assignment that satisfies the largest number of clauses possible. The problem is NP-complete even restricted to formulas with at most two literals per clause (Garey et al., 1976). It is also the canonical complete problem for the optimization class MaxSNP and central in the research of approximation algorithms (Papadimitriou & Yannakakis, 1991). Many real-world problems can be encoded as MAX-SAT instances, which led to the successful development of exact solvers, see Chapter 23 and 24 in (Biere et al., 2021). Following the positive example of SAT solvers, these tools became ever better over the last decades – regularly breaking theoretical barriers in practice. In search of an explanation for this phenomenon, theoreticians studied the parameterized complexity of MAX-SAT (Alon et al., 2011; Crowston et al., 2013; Dell et al., 2017; Iwata et al., 2014; Narayanaswamy et al., 2012; Reed et al., 2004), resulting in concepts like *parameterization above a guarantee* (Mahajan & Raman, 1999) or *dual parameterizations* (Razgon & O'Sullivan, 2009).

With membership in the class para-P (or FPT) of fixed-parameter tractable problems settled for many variants of MAX-SAT, a new question has surfaced both in theoretical and practical research over the last decade: Which problems admit *parallel* fpt-algorithms, that is, which problems lie in para-NC, the parameterized version of NC? The vertex cover problem is the poster child for such a problem as it lies even in para-AC^0, which is the smallest commonly studied parameterized class and can be thought of as "solvable with fpt-many parallel processing units in constant time" (Bannach et al., 2015). Many of the important tools underlying fpt-theory, such as search trees, graph decompositions, or kernelizations, have been adapted to the parallel setting by different research groups (Abu-Khzam & Kontar, 2020; Bannach et al., 2015; Chen & Flum, 2020; Pilipczuk et al., 2018).

1.1 Contributions of this Article

In this paper we study the parallel complexity of maximum satisfiability problems for various parameterizations. We show that the parallel fpt-toolkit can be used to establish parallel algorithms for MAX-SAT parameterized by the solution size or parameterized above some guarantee. We also develop dedicated algorithms for the problem parameterized by the structural parameters *treewidth*, *feedback vertex set number*, *treedepth*, and *vertex cover number* and observe an ever higher level of achievable parallelization. Our most technical contribution is a parallel algorithm for p_k-ALMOST-2SAT, which is MAX-2SAT for the dual parameterization where we try to satisfy at least $m - k$ clauses in a given 2CNF formula[1]. This problem has stubbornly resisted all known techniques in the parallel fpt-toolkit: First, one cannot use algorithmic meta-theorems that are often used to show membership in para-NC. The algorithmic meta-theorems for second-order logic (Bannach & Tantau, 2016) fail as the underlying incidence graphs generally do not have bounded treewidth, and those for first-order logic (Chen et al., 2017; Flum & Grohe, 2003; Pilipczuk et al., 2018) fail as the satisfiability of a 2CNF formula is not first-order definable. Second, the central tool for showing that it lies in para-P, namely iterative compression (Razgon & O'Sullivan, 2009; Reed et al., 2004), is – as the name suggests – highly *sequential*.

We develop new tools that go beyond the established toolkits and involve two ideas. First, we make a simple, but non-trivial, observation concerning the parallel computation of graph flows. While computing graph flows is P-complete (Goldschlager et al., 1982) and, thus, most likely *not* paralellizable and while even computing a 0-1-flow in parallel is a long standing open problem (Karp et al., 1986), we observe that computing a flow of *parameter value* k can be done in k consecutive rounds of a *parallel* Ford-Fulkerson (Ford & Fulkerson, 1956) step. The second idea is more complex, as we study a seemingly different

1. In the article m is always the number of clauses, n the number of variables, k a positive integer parameter, and "p-" indicates a parameterized problem with the index being the parameter. Variables occurring in problem names such as in dSAT are fixed constants

problem: vertex cover, but not with the sought size of the vertex cover as the parameter, but with the (smaller and hence less restrictive) parameter "integrality excess of the LP." An fpt-reduction from parameterized ALMOST-2SAT to this vertex cover version is well known (Razgon & O'Sullivan, 2009). To compute vertex covers for this "looser" parameter in parallel, we combine results by Iwata, Oka and Yoshida (Iwata et al., 2014) on the properties of the Hochbaum network underlying the linear program and apply the earlier-mentioned observations on graph flows.

Contribution I. We settle the parallel complexity of MAX-SAT for the canonical parameters k (solution size) and g (solution size minus $\lceil \frac{m}{2} \rceil$): p_k-MAX-SAT $\in$ para-AC^0, but p_g-MAX-SAT is para-TC^0-complete. If we assume that clauses have size exactly d (MAX-EdSAT), we show that an "above average version" lies in para-AC^0 as well – a version that is known to be para-NP-hard if the size of the clauses is unbounded.

Contribution II. We study variants of p_k-ALMOST-SAT (MAX-SAT parameterized dually) and present, for the first time, parallel algorithms for this problem on various classes of CNFs. The main achievement is a para-NC algorithm for the problem restricted to 2CNFs.

Contribution III. The structural parameters *vertex cover number*, *treedepth*, *feedback vertex set number*, and *treewidth* are classical graph parameters[2] that can be applied to formulas by considering a graph representation (such as the incidence or primal graph, formal definitions follow) of the formula. The parameters are partially ordered, meaning that graphs of bounded vertex cover number have bounded treedepth and so on. It is known that MAX-SAT is in para-P parameterized by any of these, but the sequential algorithms tend to hide beneficial properties gained by more restrictive parameterizations. We show that we obtain a higher level of parallelization for larger parameters (reaching from para-TC^0 and para-$\mathrm{TC}^{0\uparrow}$, over para-$\mathrm{TC}^{1\uparrow}$, up to para-$\mathrm{AC}^{2\uparrow}$). Additionally, our algorithms are constructive (they output an optimal assignment), which is in contrast to existing parallel meta-theorems.

Table 1 on the next page provides an overview of all results presented within this manuscript. As byproducts, we establish results that may be of independent interest: First, we present an alternative characterization of the "up-classes". Second, we lower the complexity of the feedback vertex set problem to para-$\mathrm{L}^{\uparrow}$, which is obtained "by iterating a para-L computation parameter-many times." Third, we obtain para-NC algorithms for problems that can be reduced to p_k-ALMOST-2SAT – including the odd cycle transversal problem (can we make a given graph bipartite by deleting k vertices?).

1.2 Related Work

A conference version of this article was presented at SAT 2022 (Bannach et al., 2022a). The parameterized complexity of MAX-SAT is an active field of research dating back to the pioneering work by Mahajan and Raman (Mahajan & Raman, 1999). Since then, parameterized algorithms for ever looser parameters have been found (Crowston et al., 2011, 2012; Gutin et al., 2013) or their existence has been refuted (Crowston et al., 2013). This research has also branched out into the study of preprocessing algorithms (Gaspers

2. Since the main content of this paper is not about graph theory, we refer the reader to the textbook of Diestel (2012) for an introduction and give only small preliminaries in Section 2. The book by Cygan et al. (2015) presents these concepts in the context of parameterization.

& Szeider, 2011, 2014), parameterized heuristics (Szeider, 2011), and algorithms utilizing structural decompositions (Dell et al., 2017; Grohe, 2006). However, to the best of our knowledge, not yet to parallel parameterized algorithms.

While research on parallel fixed-parameter algorithms dates back to the early 1990s to the study of the *space* complexity of parameterized problem (Cai et al., 1997) (via the inclusion chain $\mathrm{NC}^1 \subseteq \mathrm{L} \subseteq \mathrm{NL} \subseteq \mathrm{AC}^1$), a systematic study of parallel fixed-parameter algorithms started only in the last decade (Elberfeld et al., 2015). Since then, a toolbox has been compiled that contains algorithmic meta-theorems both for monadic second-order logic (Bannach & Tantau, 2016) and for first-order logic (Chen et al., 2017; Flum & Grohe, 2003; Pilipczuk et al., 2018).

1.3 Organization of this Paper

After some preliminaries in the next section, we study MAX-SAT parameterized by the solution size and parameterized above a guarantee in Section 3. We continue and study MAX-SAT variants with a dual parameterization in Section 4. The largest and technical most involved part here is a parallel algorithm for p_k-ALMOST-2SAT. Finally, we consider structural parameterizations of MAX-SAT in Section 5 and establish a connection between the level of parallelization we can achieve and the used parameter.

2. Background on Parameterized Problems and Classes

We start with the necessary definitions, especially about propositional logic and the optimization variant of the satisfiability problem. This section also provides a light introduction to the network theory underlying some of our results are based, as well as on parallel parameterized complexity.

2.1 Propositional Logic and MaxSAT

We assume an infinite supply of *propositional variables* $x_1, x_2, \dots$ and call a variable x or its negation $\neg x$ a *literal*. A *propositional formula in conjunctive normal form* (a CNF) ϕ is a conjunction of disjunctions of literals, for instance $\phi = (x_1 \lor x_2 \lor \neg x_2) \land (x_1) \land (x_1) \land (x_2 \lor x_2)$. We write $\mathrm{vars}(\phi)$ for the set of variables in ϕ and $\mathrm{clauses}(\phi)$ for the *multi*set of clauses, which are the sets of literals in the disjunctions, e.g., $\mathrm{clauses}(\phi) = \big\{\{x_1, x_2, \neg x_2\}, \{x_1\}, \{x_1\}, \{x_2\}\big\}$. We denote $|\mathrm{vars}(\phi)|$ by n and $|\mathrm{clauses}(\phi)|$ by m (so $n = 2$ and $m = 4$ in the example), and let $m_\emptyset$ be the number of empty clauses.

An *assignment* $\beta\colon \mathrm{vars}(\phi) \to \{0, 1\}$ maps every variable of ϕ to a truth value. It satisfies a literal ℓ if $\ell = x$ and $\beta(x) = 1$ or if $\ell = \neg x$ and $\beta(x) = 0$. Furthermore, it *satisfies* a clause C (denoted by $\beta \models C$) if it satisfies at least one literal in it; it *nae-satisfies* a clause if it additionally falsifies at least one literal ("not-all-equal-satisfies").

The MAX-SAT problem asks, given a CNF ϕ and a number k, whether there is an assignment β that satisfies at least k clauses. If β satisfies all m clauses, then $\beta \models \phi$, i.e., β is a model of ϕ. Variations are obtained by modifying the condition of a clause being satisfied, e.g., in MAX-NAE-SAT we seek an assignment that nae-satisfies at least k clauses.

Table 1: Variations of the maximum satisfiability problem studied within this paper (formal definitions of the problems will be given in the sections containing the references). The lower bounds are the trivial ones, while the upper bounds are proven in the referenced theorems or lemmas. If the result is marked as constructive, a corresponding optimal assignment can be produced (this either is proven directly, or the presented algorithm can be modified in an obvious way).

	Complexity Bound			*Constructive?*	
Problem	*Lower*	*Upper*	*Clause Size*		*Reference*
Can Be Solved Using Color Coding					
$\mathrm{p}_{k,t}$-MAX-δ-CIRCUIT-SAT	para-AC^0	para-AC^0	unbounded	✓	Theorem 6
p_k-MAX-SAT	para-AC^0	para-AC^0	unbounded	✓	Corollary 9
p_k-MAX-NAE-SAT	para-AC^0	para-AC^0	unbounded	✓	Corollary 9
$\mathrm{p}_{k,d,x}$-MAX-EXACT-SAT	para-AC^0	para-AC^0	$\leq d$	✓	Corollary 9
$\mathrm{p}_{k,d}$-MAX-DNF	para-AC^0	para-AC^0	$\leq d$	✓	Corollary 9
p_g-MAX-SAT-ABOVE-HALF	para-TC^0	para-TC^0	unbounded	✗	Theorem 11
Can Be Solved Using Algebraic Techniques					
p_g-MAX-$d_=$SAT-ABOVE-AVERAGE	para-AC^0	para-AC^0	$= d$	✗	Lemma 13
Can Be Solved Using Graph Flows					
p_k-ALMOST-NAE-2SAT	para-L	para-$\mathrm{NL}^{\uparrow}$	≤ 2	✓	Theorem 15
p_k-ALMOST-2SAT	para-NL	para-$\mathrm{NL}^{\uparrow}$	≤ 2	✓	Theorem 15
Can Be Solved Using Graph Extensions					
p_k-ALMOST-NAE-SAT(2)	para-L	para-L	unbounded	✗	Theorem 37
p_k-ALMOST-SAT(2)	para-L	para-L	unbounded	✗	Theorem 37
Can Be Solved Using Reduction to Vertex Cover					
p_k-ALMOST-DNF	para-AC^0	para-AC^0	unbounded	✓	Theorem 40
p_k-MIN-SAT	para-AC^0	para-AC^0	unbounded	✓	Lemma 41
Can Be Solved Using Dynamic Programming					
p_{vc}-PARTIAL-MAX-SAT	para-TC^0	para-TC^0	unbounded	✓	Theorem 42
p_{td}-PARTIAL-MAX-SAT	para-TC^0	para-$\mathrm{TC}^{0\uparrow}$	unbounded	✓	Theorem 42
$\mathrm{p}_{\mathrm{fvs}}$-PARTIAL-MAX-SAT	para-L	para-$\mathrm{TC}^{1\uparrow}$	unbounded	✓	Theorem 42
p_{tw}-PARTIAL-MAX-SAT	para-L	para-$\mathrm{AC}^{2\uparrow}$	unbounded	✓	Theorem 42

2.2 Graphs, Networks, and Flows

In this paper, graphs are pairs $G = (V, E)$ of finite sets of vertices and edges. In this context, n denotes $|V|$ and m denotes $|E|$. For undirected graphs, edges are two-element subsets of V, for directed graphs (digraphs) $E \subseteq V \times V$. A *walk in G of length p* is a sequence $(v_0, \ldots, v_p)$ of vertices $v_i \in V$ with $(v_i, v_{i+1}) \in E$ (or $\{v_i, v_{i+1}\} \in E$ for undirected graphs) for all $i \in \{0, \ldots, p-1\}$. A *path* is a walk in which all vertices (and hence all edges) are distinct. A *cycle* is a walk of length at least 3 in which all vertices are distinct expect for the first and last, which must be identical. For a set $S \subseteq V$ we write $G - S$ for the graph induced on the set $V \setminus S$. For an undirected graph G the *neighborhood* $N(v)$ of a vertex v is the set $\{u \in V \mid \{u, v\} \in E\}$, the *degree* of v is $|N(v)|$.

We think of digraphs $G = (V, E)$ with two designated vertices $s, t \in V$ as *networks*, and we always assume that in networks between any two different vertices u and v at most one edge is present (either (u, v) or (v, u)) – if this is not the case we may simply subdivide each edge. A 0-1-*flow* from s to t in G is a mapping $f\colon E \to \{0, 1\}$ such that for all $v \in V \setminus \{s, t\}$ we have $\sum_{(u,v)\in E} f(u, v) = \sum_{(v,w)\in E} f(v, w)$. The *value* $|f|$ of a flow is defined as the amount $|f| = \sum_{(s,v)\in E} f(s, v) - \sum_{(w,s)\in E} f(w, s)$ of flow leaving the source (or, equivalently, arriving at the target). For a flow f in a network G, the *residual graph* $R_f = (V, E_f)$ contains all edges of G that are not part of the flow and all reversed edges of the flow:

$$E_f = \{(u, v) \in E \mid f(u, v) = 0\} \cup \{(v, u) \in V \times V \mid f(u, v) = 1\}.$$

2.3 Classic Parameterized Problems and Complexity Classes

A *parameterized problem* is a set $Q \subseteq \Sigma^* \times \mathbb{N}$, where $\mathbb{N}$ is the set of non-negative integers. In an instance (w, k) we call w the *input* (typically a CNF in this paper) and k the *parameter.* For instance, p_k-MAX-SAT $= \{(\phi, k) \mid \phi$ has an assignment satisfying at least k clauses$\}$. We indicate the parameter as a subscript to the leading "p".

A *parameterized function* is a mapping $F\colon \Sigma^* \times \mathbb{N} \to \Sigma^* \times \mathbb{N}$ such that the output parameter is bounded in terms of the input parameter, i.e., there is a function $b\colon \mathbb{N} \to \mathbb{N}$ with $k' \leq b(k)$ whenever $F(w, k) = (w', k')$. The *characteristic function* χ_Q of a parameterized problem Q maps $(w, k) \in Q$ to $(1, 0)$ and $(w, k) \notin Q$ to $(0, 0)$.

In parameterized complexity theory, the class para-P (also known as FPT) takes the role of P in classical complexity theory. A parameterized problem Q is in para-P if there is an algorithm that decides whether $(w, k) \in Q$ holds in time $f(k) \cdot n^{O(1)}$ for some computable function f. A *parallel* parameterized algorithm is able to decide the same question by a logarithmic-time-uniform[3] family of unbounded fan-in circuits of depth $O(\log^i n)$ for some fixed i (note that the depth does not depend on k) and size $f(k) \cdot n^{O(1)}$. The problem is then in the class para-AC^i or, in the presence of *threshold gates*, para-TC^i. Define para-NC as the union of all these para-AC^i classes or, equivalently, the union of all para-TC^i classes.

2.4 Up-Classes

The "up-arrow notation" was originally introduced in the context of parameterized circuit classes (Bannach et al., 2015) to denote circuits that arise from taking a circuit of a certain depth (like $\log n$) and then allow "parameter-dependent-many layers" of such circuits (resulting in a depth like $f(k) \log n$). In this paper, we define the notation as the "closure of a parameterized function class under parameter-dependent-many iterations of linear functions," which yields the same circuit classes, but also yields natural "up-versions" of para-L and para-NL. In detail, we take a parameterized function class and allow the functions in it to be applied to an input not just once, but rather "parameter-dependent-many times." One must be a bit careful, though, to ensure that the intermediate results do not get too large:

3. Details about uniformity will not be of importance in our study. We refer the interested reader to (Bannach et al., 2015; Barrington et al., 1990; Chen & Flum, 2016) and abbreviate "logarithmic-time-uniform" with "uniform" in the following.

We need to require that the function we apply iteratively causes only a linear increase in the output size. For this, let us call a parameterized function F *linear* if $|F(w,k)| \leq f(k) \cdot |w|$ for some computable f.

Definition 1. *Let* para-FC *be a class of parameterized functions. A parameterized function F lies in* para-FC$^{\uparrow}$ *if there are*

1. *an* initial function $I \in$ para-FC*,*
2. *a* linear iterator function $L \in$ para-FC*, and*
3. *a computable* iteration number function $r\colon \mathbb{N} \to \mathbb{N}$*,*

such that $F(w,k) = L^{r(k)}(I(w,k))$, where L^r is the r-fold composition (or iteration) of L with itself. A problem lies in para-C$^{\uparrow}$ *if its characteristic function lies in* para-FC$^{\uparrow}$*.*

As mentioned above, our motivation for this (new) definition of up-classes is that it naturally yields the classes para-L$^{\uparrow}$ and para-NL$^{\uparrow}$ based on para-FL and para-FNL, the parameterized versions of FL and FNL. These latter classes contain all functions $F\colon \Sigma^* \to \Sigma^*$ such that a Turing machine (deterministic for L, non-deterministic for NL) with a read-only input tape and a write-only output tape produces $F(w)$ on input $w \in \Sigma^*$ using only $O(\log |w|)$ cells on its work tape (in the non-deterministic case, all halting computations must lead to $F(w)$ on the output tape). It is worth noting that both FL and FNL are closed under composition (the Immerman-Szelepcsényi Theorem is needed for FNL) and that they only contain functions F with $|F(w)| \leq |w|^{O(1)}$. The parameterized function classes are defined analogously, only they contain parameterized functions $F\colon \Sigma^* \times \mathbb{N} \to \Sigma^* \times \mathbb{N}$ and the machines may use $f(|w|) + O(\log |w|)$ cells on the work tape on input (w,k) for some computable function $f\colon \mathbb{N} \to \mathbb{N}$. Note that the maximum length of $F(w)$ is now $f'(|w|)\cdot|w|^{O(1)}$ for some computable function f'. These classes are closed under composition.

Example 2 (Computing Feedback Vertex Sets Using Logspace-Up-Classes)**.** To help readers get a better feeling for our definition of the up-classes, we argue that *on input of a pair (G,k) a size-k feedback vertex set in G can be computed in* para-FL$^{\uparrow}$*, if one exists.*

According to the definition, we have to provide two functions $I, L \in$ para-FL such that I maps inputs (G,k) to initial instances (w_0,k_0) and such that L outputs a new instance (w_{i+1},k_{i+1}) on its output tape while reading (w_i,k_i) from its input tape. Each time the machine underlying the function L is run, it can freely access the output of the previous iteration (in a read-only fashion).

We develop a bounded search tree algorithm of depth k. To that end, I simply translates (G,k) to a list with a single element $\big(((G,\emptyset),k)\big)$. We will keep the invariant that the second part of $(G,\emptyset)$ is a partial feedback vertex set of G. The iterator function L takes such a list, processes each item in it, and outputs a potentially larger list.

In detail, for a list $\big(((G_1,S_1),k_1),\dots,((G_\ell,S_\ell),k_\ell)\big)$ the following logspace operations are performed on all tuples $((G_j,S_j),k_j)$: First, all degree-1 vertices are removed; secondly, all paths of degree-2 vertices are contracted. If the resulting graph G'_j contains a vertex v with a self-loop, any solution has to contain v. Therefore, the machine maps (G_j,k_j) to $((G'_j - \{v\}, S_j \cup \{v\}), k_j - 1)$ in the output list. Otherwise the minimum degree of G'_j is 3 and a well-known fact states that any size-k_j feedback vertex set of G'_j has to contain one of the $3k_j$ vertices $v_1, \dots, v_{3k_j}$ of highest degree (Cygan et al., 2015, Lemma 3.3). Thus, the machine branches on these vertices (and simulates the corresponding bounded search tree) by mapping (G_j, k_j) to $\big(((G'_j - \{v_1\}, S_j \cup \{v_1\}), k_j - 1), \dots, ((G'_j - \{v_{3k_j}\}, S_j \cup \{v_{3k_j}\}), k_j - 1)\big)$.

We can check in logspace whether one of the instances in the current list is a forest, in which case a solution S was found. On the other hand, after at most k iterations all parameter values fall to 0 and, hence, the machine recognizes that it deals with a no-instance.

Observe that in any iteration, the output list is larger than the input list by a factor of at most $3k$ and, hence, the function L is linear in the sense of Definition 1.

It is, of course, important that our new (and arguably more complex) definition of up-classes is still compatible with the original definition in the literature (Bannach et al., 2015), where para-$\mathrm{AC}^{i\uparrow}$ is defined as the class of problems decidable by circuits of depth $f(k) \cdot O(\log^i n)$ and size $f(k) \cdot n^{O(1)}$. The following lemma shows that, indeed, the two definitions yield the same classes, and it implies the chain of inclusions shown in Figure 1.

Lemma 3. *A problem Q is in* para-$\mathrm{AC}^{i\uparrow}$ *(in the sense of Defintion 1) iff Q can be decided by a family of Boolean circuits of depth $f(k) \cdot O(\log^i n)$ and size $f(k) \cdot n^{O(1)}$ for some computable function f.*

Proof. If $Q \in$ para-$\mathrm{AC}^{i\uparrow}$ via functions I, L, and f, where I is implemented by circuit family C_I and L by C_L, both of depth $O(\log^i n)$, the claimed family of circuits simply consists of C_I followed by $f(k)$ copies of C_L. For the other direction, let $C = (C_{n,k})_{n,k \in \mathbb{N}}$ be a family of depth $f(k) \cdot O(\log^i n)$ that decides Q. The circuit family C_I then maps an input (w, k) to (C', k) where C' is the following partially evaluated circuit: It is $C_{|w|,k}$ with (only) the input gates evaluated to the corresponding bits of w. The iteration function L then takes a partially evaluated circuit and does $O(\log^i n)$ "rounds of evaluation," which just means that any gate whose inputs have all been evaluated gets evaluated itself. Clearly, a single round of evaluating gates can be done by an AC^0 circuit, so $O(\log^i n)$ rounds can be done by an AC^i circuit. Putting it all together, we see that after applying C_i and then $f(k)$ times the iteration function, we map the input to the fully evaluated circuit $C_{|w|,k}$ and can, thus, obtain the desired output from the output gate in the last iteration. $\square$

$$\begin{tikzcd}
\text{para-AC}^0 \arrow[rr, dashed] \arrow[dr, dashed] & & \text{para-TC}^0 \arrow[rr] \arrow[dr] & & \text{para-NC}^1 \arrow[rr] \arrow[dr] & & \text{para-L} \arrow[rr] \arrow[dr] & & \text{para-NL} \arrow[rr] \arrow[dr] & & \text{para-AC}^1 \arrow[rr] \arrow[dr] & & \text{para-AC}^2 \\
& \text{para-AC}^{0\uparrow} \arrow[rr] & & \text{para-TC}^{0\uparrow} \arrow[rr] & & \text{para-NC}^{1\uparrow} \arrow[rr] & & \text{para-L}^{\uparrow} \arrow[rr] & & \text{para-NL}^{\uparrow} \arrow[rr] & & \text{para-AC}^{1\uparrow} \arrow[ur] &
\end{tikzcd}$$

Figure 1: Inclusions among parallel parameterized complexity classes within para-P. An arrow from A to B means $A \subseteq B$, and a dashed arrow indicates $A \subsetneq B$. The inclusions between the two rows follow from arguments for the up-classes (Bannach et al., 2015) and the other inclusions follow from the standard inclusion chain $\mathrm{AC}^0 \subsetneq \mathrm{TC}^0 \subseteq \mathrm{L} \subseteq \mathrm{NL} \subseteq \mathrm{AC}^1 \subseteq \mathrm{AC}^2 \subseteq \mathrm{P}$.

The following lemma, needed later on, provides evidence that the proposed definition of up-classes is "well-behaved" (the technical report (Bannach et al., 2022b) contains the simple-but-technical proof, which does not lie at the heart of the present paper):

Lemma 4 ((Bannach et al., 2022b))**.** para-$\mathrm{FNL}^{\uparrow\uparrow}$ = para-$\mathrm{FNL}^{\uparrow}$.

3. MaxSAT Variants Parameterized by Solution Size

A natural parameterization of a problem such as MAX-SAT is to take as parameter k the size of the sought solution. It is well-known that the corresponding problem p_k-MAX-SAT is in para-P (Mahajan & Raman, 1999). We prove in Section 3.1 that the problem lies in para-AC^0 and that this result generalizes to a broader range of problems. It is also known that a version with a less restrictive parameter is in para-P as well (Mahajan & Raman, 1999): p_g-MAX-SAT-ABOVE-HALF asks whether there is an assignment that satisfies at least $\lceil \frac{m-m_\emptyset}{2} \rceil + g$ clauses, where m is the total number of clauses and $m_\emptyset$ the number of empty clauses in the input. We show in Section 3.2 that this problem is strictly harder than p_k-MAX-SAT, as it is complete for para-TC^0.

3.1 Maximum Bounded-Circuit Satisfiability

We consider four variants of MAX-SAT, where we maximize the number of clauses

- for p_k-MAX-SAT in which at least one literal is true;
- for p_k-MAX-NAE-SAT in which at least one literal is true and one is false;
- for $\mathrm{p}_{k,d,x}$-MAX-EXACT-SAT in which exactly x of the d literals are true;
- for $\mathrm{p}_{k,d}$-MAX-DNF in which all of the d literals are true.

All of these problems are special cases of Problem 5 below. For its definition, we say that a Boolean function $f\colon \{0,1\}^n \to \{0,1\}$ is *t-robust* if for every $x \in \{0,1\}^n$ with $f(x) = 1$ there is a set of at most t indices such that $f(y) = 1$ for any $y \in \{0,1\}^n$ that equals x on these indices. For instance, a clause on d literals is 1-robust, while a *term* (a conjunction of literals) is d-robust. We are interested in the following promise problem:

Problem 5 ($\mathrm{p}_{k,t}$-MAX-δ-CIRCUIT-SAT)**.**

Instance: *Integers k and t, AC-circuits $C_1, \ldots, C_m$, all connected to the same n input variables $x_1, \ldots, x_n$, all with a single output gate, and all of depth at most δ.*
Parameter: $k + t$
Question: *Is there an assignment from the input variables to $\{0,1\}$ such that at least k circuits evaluate to 1?*
Promise: *All circuits compute a t-robust function.*

Theorem 6. $\mathrm{p}_{k,t}$-MAX-δ-CIRCUIT-SAT $\in$ para-AC^0.

Proof. We start with some observations. First, a para-AC^0 circuit can, given an assignment to the input variables, evaluate every input circuit $C_1, \ldots, C_n$ as these are all of constant depth at most δ. Second, a para-AC^0 circuit can also check whether at least k of these circuits evaluate to 1 (since para-AC^0 circuits can simulate threshold gates with parameter bounded thresholds (Bannach et al., 2015)). Third, since it is promised that all circuits are t-robust, we need to set at most t variables correctly in order to let a circuit evaluate to 1. Hence, if there is a solution that satisfies k circuits, we have to find the correct truth value for at most tk variables – of course, we do not know of which tk variables.

To find them, we use the well-known color coding technique (Alon et al., 1995) and its constant-detph derandomization (Bannach et al., 2015). The proof hinges on universal coloring families and the fact that we can compute them quickly in parallel:

Definition 7 (Universal Coloring Families)**.** *For natural numbers n, k, and c, an (n,k,c)-universal coloring family is a set Λ of functions $\lambda\colon \{1,\dots,n\} \to \{1,\dots,c\}$ such that for every subset $S \subseteq \{1,\dots,n\}$ of size $|S| = k$ and for every mapping $\mu\colon S \to \{1,\dots,c\}$ there is at least one function $\lambda \in \Lambda$ with $\forall s \in S\colon \mu(s) = \lambda(s)$.*

Fact 8 (Theorem 3.2 in (Bannach et al., 2015))**.** *There is a uniform family $(C_{n,k,c})_{n,k,c\in\mathbb{N}}$ of* AC*-circuits without inputs such that each $C_{n,k,c}$*

1. *outputs an (n,k,c)-universal coloring family (coded as a sequence of function tables),*
2. *has constant depth (independent of n, k, or c), and*
3. *has size at most $O(n \log c \cdot c^{k^2} \cdot k^4 \log^2 n)$.*

To solve $\mathrm{p}_{k,t}$-MAX-δ-CIRCUIT-SAT, we use a $(n,tk,2)$-universal coloring family Λ. Intuitively, we "color" the variables with two colors, which we interpret as assigning truth values to them. Clearly, if the input is a no-instance, no coloring will satisfy k circuits and we can correctly reject. On the other hand, assume there is some assignment that satisfies at least k circuits. Then, by the above observations, there are tk variables $y_1,\dots,y_{tk}$ that, if set correctly, satisfy the same k circuits. By Definition 7, there is at least one $\lambda \in \Lambda$ that realises exactly this correct assignment and, hence, by testing all colorings of Λ in parallel, we can decide whether at least one assignment satisfies k or more circuits. ☐

Corollary 9. *The problems* p_k-MAX-SAT*,* p_k-MAX-NAE-SAT*,* $\mathrm{p}_{k,d,x}$-MAX-EXACT-SAT*, and* $\mathrm{p}_{k,d}$-MAX-DNF *are in* para-AC^0*.*

3.2 Maximum Satisfiability Above Guarantee

The solution size is a restrictive parameter for problems such as MAX-SAT, because *every* instance has *relatively large* solutions. In particular, if ϕ is a CNF with m clauses of which $m_\emptyset$ are empty, then ϕ has an assignment that satisfies at least $\lceil \frac{m-m_\emptyset}{2} \rceil$ clauses: Pick an arbitrary assignment β and observe that either β or its bitwise complement satisfies half of the clauses (Mahajan & Raman, 1999). Hence, parallel algorithms should be able to deal with large k and, thus, require a parametrization that can be small even if k is large.

We start with a problem of the form $Q = \{(w,k) \mid \mathrm{opt}(w) \gtreqless k\}$, where $\mathrm{opt}(w)$ is some property to be evaluated. The new problem has the form $Q' = \big\{\big((w,\pi),g\big) \mid \pi \text{ is an easily checkable proof for } \mathrm{opt}(w) \geq \gamma(\pi), \text{ and } \mathrm{opt}(w) \gtreqless \gamma(\pi)+g\big\}$. Here, $\gamma(\pi)$ is called the *guaranteed lower bound proved by* π or just the *guarantee.* For $Q = \mathrm{p}_k$-MAX-SAT the situation is particularly easy, as we can take as proof π a tautology (since there is nothing to prove in this case) and set $\gamma(\pi) = \lceil \frac{m-m_\emptyset}{2} \rceil$. Note that Q' is conceptually harder than Q: An fpt-algorithm for Q' must find a (possibly large) optimal solution, but may only use time $f(g) \cdot n^{O(1)}$ for a (possibly small) difference g.

Problem 10 (p_g-MAX-SAT-ABOVE-HALF)**.**

Instance: *A* CNF *ϕ with m clauses of which $m_\emptyset$ are empty, and a difference $g \in \mathbb{N}$.*
Parameter: g
Question: *Is there an assignment that satisfies at least $\lceil \frac{m-m_\emptyset}{2} \rceil + g$ clauses?*

Algorithms for above-guarantee parameterizations have led to a number of algorithmic breakthroughs, for instance in the design of algorithms for ALMOST-2SAT (Narayanaswamy et al., 2012), linear-time fpt-algorithms (Iwata et al., 2014), or stricter parameterizations of VERTEX-COVER (Garg & Philip, 2016). One of them is due to Mahajan and Raman (1999):

$$\mathrm{p}_g\text{-MAX-SAT-ABOVE-HALF} \in \text{para-P}.$$

The following theorem sharpens this result by showing that p_g-MAX-SAT-ABOVE-HALF is para-TC^0-complete. This pinpoints the intuition that above-guarantee parameterizations are harder than their counterparts, as we obtain that p_g-MAX-SAT-ABOVE-HALF is strictly harder than p_k-MAX-SAT (since we have *unconditionally* para-$\mathrm{AC}^0 \subsetneq$ para-TC^0).

Theorem 11. p_g-MAX-SAT-ABOVE-HALF *is* $\leq_{\mathrm{tt}}^{\text{para-AC}^0}$*-complete for* para-$\mathrm{TC}^0$.

Proof. We prove containment with a parallel version of the algorithm from Mahajan and Raman (1999). The following reduction rules are easily seen to be safe (empty clauses cannot be satisfied and do not count towards the lower bound; any assignment satisfies exactly one of the two unit clauses):

Rule 1 (Empty Clauses)**.** *If there are empty clauses, remove them.*

Rule 2 (Unit Pair)**.** *If there are two unit clauses* (x) *and* $(\neg x)$*, remove both.*

An exhaustive application of Rule 2 can be carried out in TC^0 by counting for every variable $x \in \mathrm{vars}(\phi)$ the number of unit clauses that contain x or $\neg x$, respectively.

Rule 3 (Trivial Decision)**.** *Reduce to a trivial yes-instance if there are at least* $\lceil m/2 \rceil + g$ *unit clauses or at least* $4g + 4$ *non-unit clauses.*

Rule 3 is safe if Rules 1 and 2 cannot be applied: The amount of unit clauses alone would constitute a solution and every CNF on m clauses with $m_\emptyset = 0$ and p non-unit clauses has an assignment that satisfies at least $\lceil m/2 \rceil + p/4 - 1$ clauses (Mahajan & Raman, 1999, Proposition 8). Hence, every such formula with $4g + 4$ non-unit clauses is a yes-instance.

Finally, assume we have a formula ϕ with m clauses and parameter g, to which the Rules 1–3 cannot be applied. Then $m \leq (\lceil m/2 \rceil + g - 1) + (4g + 4 - 1) = \lceil m/2 \rceil + 5g + 2$ and, thus, $\lfloor m/2 \rfloor \leq 5g + 2$ and $\lceil m/2 \rceil \leq 5g + 3$. Hence, the problem has reduced to the question whether there is an assignment that satisfies at least $k = \lceil m/2 \rceil + g \leq 6g + 3$ clauses, which we can answer with Corollary 9 as k depends only on the parameter g.

For hardness we perform a truth-table reduction from a parameterized version of the majority problem: p_0-MAJORITY (the majority problem ask whether a binary string contains more 1s than 0s; the trivial parameter does nothing, implying that the problem is complete for para-TC^0 since MAJORITY is truth-table complete for TC^0). In truth-table reductions we are allowed to produce polynomial many instances of the target problem, query an oracle to solve them all at once, and then build a Boolean combination of the results.

Given an instance $w = b_1 \dots b_n$ of p_0-MAJORITY, we build a formula $\phi_0 = \bigwedge_{i=1}^n C_i$ with:

$$C_i = \begin{cases} (x) & \text{if } b_i = 1, \\ (\neg x) & \text{else.} \end{cases}$$

From ϕ_0 we build $n+1$ instances of p_g-MAX-SAT-ABOVE-HALF: Set $g = 1$ and obtain $\phi_1, \dots, \phi_n$ from ϕ_0 by setting $\phi_i := \phi_0 \wedge \bigwedge_{j=1}^{i}(x)$.

Observe that $(\phi_0, 1) \notin \text{p}_g$-MAX-SAT-ABOVE-HALF iff w contains the same amount of 0s as 1s. Then observe that, if $(w, 0) \in \text{p}_0$-MAJORITY, we have for *all* $i \in \{0, \dots, n\}$ that $(\phi_i, 1) \in \text{p}_g$-MAX-SAT-ABOVE-HALF. On the other hand, if $(w, 0) \notin \text{p}_0$-MAJORITY, then there is an index $\ell \in \{0, \dots, n\}$ such that $(\phi_0, 1), \dots, (\phi_{\ell-1}, 1)$ are members of the set p_g-MAX-SAT-ABOVE-HALF; but $(\phi_\ell, 1)$ is *not*. We conclude:

$$\text{p}_0\text{-MAJORITY} \leq_{\text{tt}}^{\text{para-AC}^0} \text{p}_g\text{-MAX-SAT-ABOVE-HALF}. \qquad \square$$

This result is tight in the sense that relaxing the parameterization further leads to an intractable problem: Let $r_1, \dots, r_m$ be the number of literals in the clauses of a CNF ϕ, then $\text{E}[\phi] := \sum_{i=1}^{m}(1 - 2^{-r_i})$ is the expected number of clauses satisfied by a random assignment. It is well-known that an assignment that satisfies at least $\text{E}[\phi]$ clauses can be found in polynomial time (Crowston et al., 2013). However, the problem p_g-MAX-SAT-ABOVE-AVERAGE, which asks whether we can satisfy *at least* $\text{E}[\phi] + g$ *clauses,* is intractable:

Fact 12 (Crowston et al.)**.** *p_g-MAX-SAT-ABOVE-AVERAGE is para-NP-complete.*

This result requires clauses of arbitrary size. If all clauses contain *exactly* d distinct and non-complementary literals, the problem becomes fixed-parameter tractable (Alon et al., 2011). Note that $\text{E}[\phi] = (1 - 2^d)m$ holds in this case. The corresponding algorithm is quite simple and can directly be parallelized (however, it requires non-trivial results about algebraic representations of formulas that were proven by Alon et al. (2011); see also Section 9.2 in the textbook by Cygan et al. (2015) for details).

Lemma 13. *p_g-MAX-$d_=$SAT-ABOVE-AVERAGE $\in$ para-AC^0.*

Proof. Let ϕ with $\text{vars}(\phi) = \{x_1, \dots, x_n\}$ and $\text{clauses}(\phi) = \{C_1, \dots, C_m\}$ be the input, where each C_i contains exactly d distinct and non-complementary literals, and let us denote by $\text{vars}(C_i)$ the variables that occur as literals in C_i (i.e., $|\text{vars}(C_i)| = d$). We identify the truth value "true" with 1 and "false" with -1 (and not with 0, as it standard) and then consider the following polynomial:

$$X(x_1, \dots, x_n) = \sum_{1 \leq i \leq m} \Big(1 - \textstyle\prod_{x_j \in \text{vars}(C_i)} \big(1 + \text{sign}(x_j, C_i)x_j\big)\Big),$$

where $\text{sign}(x_j, C_i) = -1$ if $x_j \in C_i$ and $\text{sign}(x_j, C_i) = 1$ if $\neg x_j \in C_i$. As observed by Alon et al. (2011), each product $\prod_{x_j \in \text{vars}(C_i)} \big(1 + \text{sign}(x_j, C_i)x_j\big)$ equals 2^d if C_i is falsified by an assignment $x_1, \dots, x_n$ and equals 0 when it is satisfied. This in turn means that every satisfied clause contributes 1 towards the sum in $X(x_1, \dots, x_n)$, while each falsified clause contributes $1 - 2^d$. Thus, $X(x_1, \dots, x_n) = m - 2^d(m - s) = 2^d(s - (1 - 2^{-d})m) = 2^d(s - \text{E}[\phi])$ where s is the number of clauses satisfied by $x_1, \dots, x_n$ and $\text{E}[\phi]$ is the expected number of satisfies clauses. Thus, $X(x_1, \dots, x_n) \geq g \cdot 2^d$ iff $x_1, \dots, x_n$ is an assignment that satisfies g clauses more than the expected number of clauses satisfied by a random assignment. Note that in $g \cdot 2^d$ the number g is the parameter and d is a constant.

Since the size of the clauses is a fixed constant d, we can write X as sum of at most $(2^d + 1)m$ monomials, which can be produced on input ϕ by a para-FAC^0 circuit. We now apply the following reduction rule that follows directly from the results of Alon et al.:

Rule 4 (see Lemma 9.19 and Lemma 9.12 in the textbook by Cygan et al.)**.** *If there are at least* $4 \cdot 9^d \cdot 4^d \cdot g^2$ *monomials, reduce to a trivial yes-instance.*

Since d is a constant and g a parameter, a para-AC^0 circuit can check whether or not the rule can be applied. If it applies, we are done. Otherwise at most $O(g^2)$ variables appear in the monomial representation of X and we can solve the problem via "brute-force". □

4. Dual Parameterizations for Variants of MaxSAT

We saw that MAX-SAT can be solved in parallel when parameterized by the solution size. However, since MAX-SAT instances always only have large solutions, we moved on to seeking solutions of size $\lceil \frac{m-m_\emptyset}{2} \rceil + g$ and then of size $\mathrm{E}[\phi] + g$ for parameter g. We saw that the complexity increases, but also that parallel parameterized algorithms are still possible for most variants. Now, we consider *dual parameterizations* where the sought solution size is $m - k$. The corresponding problem is called p_k-ALMOST-SAT or, if the input formula comes from a family Φ, p_k-ALMOST-Φ. These problems can also be seen as *distance to satisfiability* (can we delete k clauses to make the formula satisfiable?) and are even harder as, in order to solve them, we must be able to decide Φ for inputs with $k = 0$:

Observation 14. *If* Φ *is a family of propositional formulas such that deciding satisfiability for* Φ *is hard for a complexity class* $\mathcal{C}$*, then* p_k-ALMOST-Φ *is hard for* para-$\mathcal{C}$*.*

Hence we have that p_k-ALMOST-3SAT is para-NP-hard, p_k-ALMOST-HORN is para-P-hard, and p_k-ALMOST-2SAT is para-NL-hard. However, the observation does *not* provide any hint on upper bounds. For instance, it is not clear whether p_k-ALMOST-2SAT $\in$ para-NL. Since we are interested in parallel algorithms, we study families of formulas that can be decided in subclasses of P: p_k-ALMOST-NAE-2SAT and p_k-ALMOST-2SAT in Section 4.1 (NAE-2SAT $\in$ L and 2SAT $\in$ NL), p_k-ALMOST-NAE-SAT(2) and p_k-ALMOST-SAT(2) in Section 4.2 (NAE-SAT(2) $\in$ L and SAT(2) $\in$ L), and p_k-ALMOST-DNF in Section 4.3 (DNF $\in \mathrm{AC}^0$).

4.1 Dual Parameterization for Krom Formulas

Our first result about dual parameterizations is the technically most involved part:

Theorem 15. p_k-ALMOST-NAE-2SAT *and* p_k-ALMOST-2SAT *both lie in* para-$\mathrm{NL}^{\uparrow}$.

The proof of the theorem is based on the equivalence between p_k-ALMOST-2SAT and another member of the family of above-guarantee problems (see Section 3.2):

Problem 16 (p_g-VC-ABOVE-MATCHING)**.**
Instance: *A graph* $G = (V, E)$*, a matching* $M \subseteq E$*, a difference* $g \in \mathbb{N}$*.*
Parameter: g
Question: *Is there a set* $S \subseteq V$ *with* $|S| \leq |M| + g$ *and* $e \cap S \neq \emptyset$ *for every* $e \in E$*?*

While it is known that p_k-VERTEX-COVER $\in$ para-AC^0 (Bannach et al., 2015, Theorem 4.5), we will need the rest of this section to prove the following theorem:

Theorem 17. p_g-VC-ABOVE-MATCHING $\in$ para-$\mathrm{NL}^{\uparrow}$*.*

Theorem 15 follows directly with the following lemma, which shows that the required well-known reductions (Cygan et al., 2013; Narayanaswamy et al., 2012) can, firstly, be implemented in para-FAC^0 and, secondly, the last reduction can also compute the necessary matching as part of its output.

Lemma 18.

$$\text{p}_k\text{-ALMOST-NAE-2SAT} \leq_{\mathrm{m}}^{\text{para-AC}^0} \text{p}_k\text{-ALMOST-2SAT} \leq_{\mathrm{m}}^{\text{para-AC}^0} \text{p}_g\text{-VC-ABOVE-MATCHING}.$$

Proof. Let (ϕ, k) be the input of p_k-ALMOST-NAE-2SAT. We generate a new formula ψ by replacing every clause $(\ell \vee \ell') \in \text{clauses}(\phi)$ with $(\ell \vee \ell') \wedge (\neg\ell \vee \neg\ell')$. Clearly, ψ is satisfiable iff ϕ has an assignment that makes exactly one literal in every clause true (if ϕ has an nae-assignment). Since any assignment satisfies at least one of $(\ell \vee \ell')$ and $(\neg\ell \vee \neg\ell')$, deleting a clause in ϕ is equivalent to deleting a clause in ψ. Thus:

$$(\phi, k) \in \text{p}_k\text{-ALMOST-NAE-2SAT} \Leftrightarrow (\psi, k) \in \text{p}_k\text{-ALMOST-2SAT}.$$

For the next reduction, we first establish the following:

$$\text{p}_k\text{-ALMOST-2SAT} \leq_{\mathrm{m}}^{\text{para-AC}^0} \text{p}_k\text{-VARIABLE-DELETION-2SAT},$$

where the latter problem contains all pairs (ϕ, k) such that ϕ is a 2CNF formula in which we can delete k variables together with all clauses containing them in order to make ϕ satisfiable. We replace each $x \in \text{vars}(\phi)$ with copies $x_1, \dots, x_m$ such that each copy occurs in at most one clause, i.e., if clause C_i originally contains variable x, it will contain x_i in the new formula. We add equality constraints to ensure that all copies obtain the same value:

$$\bigwedge_{x \in \text{vars}(\phi)} \bigwedge_{i=1}^{m} \bigwedge_{j=i+1}^{m} \big((\neg x_i \vee x_j) \wedge (x_i \vee \neg x_j)\big).$$

It is easy to see that the resulting formula is satisfiable iff ϕ is satisfiable. Furthermore, deleting a variable x_i has exactly the same effect as deleting the clause C_i from ϕ.

Finally, we show $\text{p}_k\text{-VARIABLE-DELETION-2SAT} \leq_{\mathrm{m}}^{\text{para-AC}^0} \text{p}_g\text{-VC-ABOVE-MATCHING}$. Let again (ϕ, k) be the input. We construct an undirected graph that contains for every variable x two vertices x^+ and x^- that are connected by an edge. Furthermore, every clause (recall that these are binary) is represented by an edge between the vertices of the corresponding literals. The resulting graph has a perfect matching, namely $M = \{\{x^+, x^-\} \mid x \in \text{vars}(\phi)\}$. Thus, if ϕ has n variables, any vertex cover in G needs to have size at least n. In fact, if ϕ is satisfiable, there will be a vertex cover of size n (the satisfying assignment). Deleting a variable x and all clauses containing x from ϕ is equivalent to selecting both, x^+ and x^-, to the vertex cover. Hence, ϕ can be made satisfiable by deleting at most k variables iff G contains a vertex cover of size $n + k$. Note that we do not have to compute the perfect matching but rather obtain it directly from the construction and, hence, this shows that we can map (ϕ, k) to $((G, M), k)$.

The whole reduction chain can be carried out by a para-FAC^0 function: We preserve the parameter (k is always mapped directly to k) and perform otherwise only simple projections. In fact, we only rename variables and add some fixed additional clauses. ∎

4.1.1 A Parallel Algorithm to Compute 0-1-Flows

To prove Theorem 17 (which states p_g-vc-above-matching $\in$ para-NL$^{\uparrow}$) we develop an algorithm (in the rest of this section) that relies heavily on repeated flow computations. Maximum flows can be computed in polynomial time with, say, the Ford–Fulkerson algorithm (Ford & Fulkerson, 1956). However, computing the value of a weighted maximum flow is P-complete (Goldschlager et al., 1982), and whether we can compute a 0-1-flow in parallel is a long standing open problem (Karp et al., 1986). It is worth noting that a maximum 0-1-flow can be computed in randomized NC via a reduction to the maximum matching problem in bipartite graphs (Karp et al., 1986). Unfortunately, this reduction is *not* parameter-preserving and, thus, we may not apply parameterized matching algorithms (Bannach & Tantau, 2018).

Our objective in this section is to show that a flow of value k can be computed in parallel; more precisely, that there is a function in para-FNL$^{\uparrow}$ mapping $((G, s, t), k)$ to a 0-1-flow of value k from s to t, if it exists, and otherwise to a maximum flow (formally, the output of a parameterized function must be a pair where the second component is a new parameter value, but we will not need this here and just silently assume that this value is set to, say, 0).

Computing Paths in FNL. It is well-known that the reachability problem in digraphs is the canonical complete problem for NL and, thus, it may seem trivial that we should be able to compute paths in FNL. However, being able to tell whether there is a path form s to t is not the same as actually finding such a path: For instance, it is known that in tournaments (digraphs with exactly one edge between any pair of vertices) reachability lies in AC^0, the distance problem is NL-complete, and constructing a path longer than the shortest path by a factor of $1+\epsilon$ can be done in deterministic logarithmic space (Nickelsen & Tantau, 2005) – meaning that reachability and path construction can have vastly different complexities. Nevertheless:

Lemma 19. *There is a function in* FNL *that maps* (G, s, t) *to a shortest path from* s *to* t*, provided it exists.*

Proof. Let (G, s, t) with $G = (V, E)$ be given as input. Since the distance problem is complete for both NL and coNL, an NL-machine can compute the distance d from a given vertex $v \in V$ to t in G. Furthermore, if $d < \infty$, the machine can also compute all vertices $u \in V$ that are one step nearer to t, i.e., that have distance $d - 1$. Finally, for each $v \in V$ it can chose one such u (say, the lexicographical smallest) and form a graph $H = (V, E')$ where E' contains all these edges (v, u). Then H is a forest with out-degree at most 1 and with a unique s-t-path (if one exists in G). The machine may deterministically traverse and output this path. Note that the result is independent of the nondeterministic choices that were made during the computation. □

Computing 0-1-Flows in para-FNL$^{\uparrow}$. The most important operation in the Ford–Fulkerson algorithm is the computation of an *augmenting path.* An iterated application of Lemma 19 therefore allows us to compute a small flow:

Theorem 20. *There is a parameterized function in* para-FNL$^{\uparrow}$ *that maps* $((G, s, t), k)$ *to a flow from* s *to* t *in* G *of value* k*, if it exists, or to a maximum flow otherwise.*

Proof. To show that a function lies in para-$\mathrm{FNL}^{\uparrow}$, we need to specify the initial function, the iteration function, and the number of iterations, see Definition 1.

The initial function does very little: It just maps $((G, s, t), k)$ to $((G, s, t, f_0), k)$ where f_0 is the empty flow ($f_0(e) = 0$ for all $e \in E$). The interesting part is the iteration function in para-FNL (actually, it will even lie in FNL), which implements a single step of the Ford–Fulkerson algorithm: It gets (G, s, t, f) as input, where f is some flow in G from s to t, and will output (G, s, t, f'), where f' is a flow of value $|f| + 1$ (provided such a flow exists, otherwise $f' = f$). If there is no path from s to t in the residual network R_f (an NL-machine can easily check this), set $f' = f$. Otherwise, use Lemma 19 to compute such a path (called an *augmenting path*) and output the flow f' that corresponds to f augmented by the path (initially set $f' = f$ and then for every edge (u, v) on the path set $f'(u, v) = 1$ if $(u, v) \in E$; and $f'(v, u) = 0$ otherwise).

By setting $r(k) = k$, we get a value-k or maximum-value flow from s to t. □

Let p_k-FLOW $= \bigl\{\bigl((G, s, t), k\bigr) \mid$ there is a 0-1-flow f from s to t in G with $|f| \leq k\bigr\}$ be the corresponding parameterized decision problem. Then we have:

Corollary 21. p_k-FLOW $\in$ para-$\mathrm{NL}^{\uparrow}$.

The following corollary observes that instead of starting with the empty flow we can also start with an arbitrary flow f and augment it k times:

Corollary 22. *There is a parameterized function in* para-$\mathrm{FNL}^{\uparrow}$ *that maps* $((G, s, t, f), k)$, *where* f *is an* s-t*-flow in* G, *to an* s-t*-flow* f' *in* G *of value* $|f| + k$, *if it exists, or to a maximum flow otherwise.*

Remark 23. While Theorem 20 and Corollaries 21 and 22 only speak about 0-1-flows, it is easy to see that the same techniques can be used to compute flows in networks with *fixed constant capacities:* Just replace each edge with capacity c by c parallel edges and divide each of these edges with a fresh vertex afterwards. In particular, Corollary 22 can also be used to augment *half-integral flows* in networks with fixed maximum capacity.

4.1.2 Linear Programs for Vertex Cover and Matching

To prove Theorem 17, we will study a more general problem and obtain the theorem as corollary: Instead of using matchings as proofs for lower bounds for the vertex cover problem, we use fractional solutions of LP-relaxations. Let us fix some notations: For a linear program Π let $\mathrm{vars}(\Pi)$ be the set of variables occurring in Π. A *solution* for Π is an assignment $\alpha\colon \mathrm{vars}(\Pi) \to \mathbb{Q}$ that satisfies all inequalities, and the *solution value* (or just *value*) $|\alpha|$ of α is the value of the optimization function under α. An *optimal solution* is an assignment that has the minimum (or maximum) solution value over all possible assignments. We say an assignment α is *integral* if $\alpha\colon \mathrm{vars}(\Pi) \to \mathbb{N}$ for all $x \in \mathrm{vars}(\Pi)$; α is *half-integral* if all $\alpha(x)$ are half-integral, meaning $\alpha(x) = i/2$ for some $i \in \mathbb{N}$; otherwise α is *fractional*. Let $\mathrm{opt}_{\mathbb{Q}}(\Pi)$, $\mathrm{opt}_{\mathbb{N}/2}(\Pi)$, and $\mathrm{opt}_{\mathbb{N}}(\Pi)$ denote the optimal value of a fractional, half-integral, and integral solution for Π, respectively. We are interested in the following two linear programs:

Definition 24 (Linear program $\Pi_{\mathrm{VC}}(G)$ to find a vertex cover of a graph $G = (V, E)$)**.**
Minimize $\sum_{v \in V} x_v$ *subject to* $\quad x_u + x_v \geq 1 \quad$ *for all* $\{u, v\} \in E$,
$\quad 0 \leq x_v \leq 1 \quad$ *for all* $v \in V$.

Definition 25 (Linear program $\Pi_{\mathrm{M}}(G)$ to find a matching of a graph $G = (V, E)$)**.**

$$\text{Maximize } \sum_{e \in E} y_e \text{ subject to } \quad \begin{array}{ll} \sum_{v \in e} y_e \leq 1 & \text{for all } v \in V, \\ 0 \leq y_e \leq 1 & \text{for all } e \in E. \end{array}$$

A vertex cover of G naturally corresponds to an integral solution $\alpha_{\mathbb{N}}$ of $\Pi_{\mathrm{VC}}(G)$ and a matching corresponds to an integral solution $\beta_{\mathbb{N}}$ of $\Pi_{\mathrm{M}}(G)$ (the index "$\mathbb{N}$" emphasizes that the solution is integral). In particular, $\mathrm{opt}_{\mathbb{N}}(\Pi_{\mathrm{VC}}(G))$ and $\mathrm{opt}_{\mathbb{N}}(\Pi_{\mathrm{M}}(G))$ are the sizes of a minimum vertex cover and a maximum matching of G, respectively. The programs are dual to each other, which implies that their optimal fractional solutions have the same value.

Fact 26 (Nemhauser & Trotter, 1975)**.** *Let $G = (V, E)$ be a graph. Then $\Pi_{\mathrm{VC}}(G)$ and $\Pi_{\mathrm{M}}(G)$ have solutions α and β, respectively, with the following properties:*

1. $\mathrm{opt}_{\mathbb{Q}}(\Pi_{\mathrm{VC}}(G)) = |\alpha| = |\beta| = \mathrm{opt}_{\mathbb{Q}}(\Pi_{\mathrm{M}}(G))$,
2. *α and β are* half-integral*,*
3. *there is an optimal* integral *solution γ for $\Pi_{\mathrm{VC}}(G)$ such that for $v \in V$ with $\alpha(x_v) \neq 1/2$ we have $\gamma(x_v) = \alpha(x_v)$ (that is, γ equals α on its integral part).*

Fact 26 implies that the following (in)equalities hold, where $\alpha_{\mathbb{N}}$ and $\alpha_{\mathbb{N}/2}$ are arbitrary integral and half-integral solutions for $\Pi_{\mathrm{VC}}(G)$ and $\beta_{\mathbb{N}}$ and $\beta_{\mathbb{N}/2}$ correspondingly for $\Pi_{\mathrm{M}}(G)$:

$$\begin{array}{ccccccc} |\beta_{\mathbb{N}}| \leq \mathrm{opt}_{\mathbb{N}}(\Pi_{\mathrm{M}}(G)) & \leq & & & & & \overbrace{\mathrm{opt}_{\mathbb{N}}(\Pi_{\mathrm{VC}}(G))}^{(**)} \leq |\alpha_{\mathbb{N}}| \\ & & \overbrace{\mathrm{opt}_{\mathbb{N}/2}(\Pi_{\mathrm{M}}(G))}^{(*)} & = & \mathrm{opt}_{\mathbb{N}/2}(\Pi_{\mathrm{VC}}(G)) & & \\ |\beta_{\mathbb{N}/2}| & \leq & & & & \leq & |\alpha_{\mathbb{N}/2}| \end{array} \tag{1}$$

The parameter of p_g-vc-above-matching is the difference between the upper left value $|\beta_{\mathbb{N}}|$, which is the size of some matching of G, and $(**)$, which is the size of a minimum vertex cover of G. When working with linear programs, it is natural to work with a different ("better") parameter, namely the difference between the lower left value $|\beta_{\mathbb{N}/2}|$ and $(**)$:

Problem 27 (p_g-vc-above-relaxed-matching)**.**

Instance: *A graph $G = (V, E)$, a half-integral solution $\beta_{\mathbb{N}/2}$ for $\Pi_{\mathrm{M}}(G)$, and a number g.*
Parameter: g
Question: *Is there a set $S \subseteq V$ with $|S| \leq |\beta_{\mathbb{N}/2}| + g$ and $e \cap S \neq \emptyset$ for every $e \in E$?*

4.1.3 An FPT-Algorithm for Solving VC Above Half-Integral Matching

Let us briefly review how one usually shows p_g-vc-above-relaxed-matching $\in$ para-P:

Step 0: Computing an Optimal Half-Integral Solution. Compute an optimal half-integral solution α for $\Pi_{\mathrm{VC}}(G)$ in polynomial-time ($|\alpha|$ has the value $(*)$ in (1)).

Step 1: Reduction to the All-1/2-Solution. We turn α into an "all-1/2-solution" such that $\alpha(x_v) = 1/2$ holds for all vertices. To achieve this, we use Fact 26, which tells us that vertices $v \in V$ with $\alpha(x_v) = 0$ are not part of an optimal vertex cover while vertices with $\alpha(x_v) = 1$ are. Thus, we can delete all these vertices and continue with the same parameter g (the integrality excess does not change). Note that α restricted to the new graph (which we still call G) is constantly $1/2$, which we denote as $\alpha \equiv 1/2$.

Step 2: Making the All-1/2-Solution Unique. Now $\alpha \equiv 1/2$ is an optimal solution, but there may be other optimal half-integral solutions. (For instance, the all-1/2-solution is an optimal solution for any even cycle, but so is $\alpha(i) = (i \bmod 2)$.) We can check in polynomial time whether α is the unique optimal solution as follows: Test for every x_v whether $\mathrm{opt}_{\mathbb{Q}}(\Pi_{\mathrm{VC}}(G)) = \mathrm{opt}_{\mathbb{Q}}(\Pi_{\mathrm{VC}}(G - \{v\})) + 1$. If so, there is an optimal solution other than α that assigns 1 to x_v. We remove v from G using Fact 26, leave g untouched, and repeat until the all-1/2-solution is the only optimal solution.

Step 3: Branching. Suppose we knew that some vertex $v \in V$ is part of an optimal vertex cover of G. Then $\mathrm{opt}_{\mathbb{N}}(G - \{v\}) = \mathrm{opt}_{\mathbb{N}}(G) - 1$ while $\mathrm{opt}_{\mathbb{Q}}(G - \{v\}) = \mathrm{opt}_{\mathbb{Q}}(G) - 1/2$. This means that the integrality excess of $G - \{v\}$ is reduced by $1/2$ compared to G. Of course, we do not know which vertices are part of an optimal vertex cover, but we can find them using *branching:* Pick an arbitrary edge $\{u, v\} \in E$ and recursively run the whole algorithm (starting from Step 1 once more) for $G - \{u\}$ and $G - \{v\}$, but now for the parameter $g - 1/2$ (the parameter should be an integer, but it is convenient for the recursion to allow integers divided by 2 as parameters in this setting).

It is now easy to see that the depth of the search tree of the above algorithm is $2g$, so the total runtime is $4^g \cdot n^{O(1)}$.

4.1.4 A Parallel Algorithm for Solving VC Above Half-Integral Matching

In this section we parallelize the different steps sketched above for solving Problem 27. This yields the following theorem, of which Theorem 17 is a corollary and whose formal proof, where all arguments get assembled, is given at the section's end:

Theorem 28. p_g-vc-above-relaxed-matching $\in$ para-$\mathrm{NL}^{\uparrow}$.

While steps 1 and 3 are easy to parallelize (search trees can be traversed in parallel), steps 0 and 2 are not. They either involve open problems (like computing optimal solutions for $\Pi_{\mathrm{VC}}(G)$ in parallel) or are very sequential (like the iterative removal of vertices in step 2).

Parallelizing Step 0: Computing an Optimal Half-Integral Solution. Given a half-integral solution β of $\Pi_{\mathrm{M}}(G)$, we wish to compute an optimal half-integral solution α of $\Pi_{\mathrm{VC}}(G)$. A para-P-machine could ignore β and solve the linear program, but we only have a para-$\mathrm{NL}^{\uparrow}$-machine. The idea we use was developed by Iwata, Oka, and Yoshida (2014) in the context of linear-time algorithms: One can encode an (optimal) solution of $\Pi_{\mathrm{M}}(G)$ into a (maximum) flow in the so-called *Hochbaum network.* More crucially, we can obtain an (optimal) solution for $\Pi_{\mathrm{M}}(G)$ *and* $\Pi_{\mathrm{VC}}(G)$ from a (maximum) flow in this network.

In detail, for a graph $G = (V, E)$ the *Hochbaum network* is the digraph $H = (V', E')$ with V' consisting of $V_1 = \{v_1 \mid v \in V\}$ and $V_2 = \{v_2 \mid v \in V\}$ plus the two vertices s and t. The edge set is $E' = \{(s, v_1) \mid v \in V\} \cup \{(u_1, v_2) \mid \{u, v\} \in E\} \cup \{(v_2, t) \mid v \in V\}$, i.e., from s we get to all vertices in V_1, then we can cross from u_1 to v_2 exactly if $\{u, v\} \in E$ (and then also from v_1 to u_2), and from all vertices in V_2 we can get to t.

Fact 29 (Hochbaum, 2002; Iwata et al., 2014)**.** *Let $G = (V, E)$ be a graph and $H = (V', E')$ be its Hochbaum network.*

1. *If β is a solution of $\Pi_{\mathrm{M}}(G)$, then $f_\beta(s, v_1) = f_\beta(v_2, t) = \sum_{w \in N(v)} \beta(y_{\{v,w\}})$ and $f_\beta(u_1, v_2) = f_\beta(v_1, u_2) = \beta(y_{\{u,v\}})$ is an s-t-flow with $|f_\beta| = 2|\beta|$ in H.*

2. *If f is an s-t-flow in H, then $\beta_f(y_{\{u,v\}}) = \frac{1}{2}\big(f(u_1, v_2) + f(v_1, u_2)\big)$ is a solution for $\Pi_{\mathrm{M}}(G)$ with $|\beta_f| = |f|/2$.*

Note that, in particular, β is an optimal solution of $\Pi_{\mathrm{M}}(G)$ iff f_β is maximal and, *vice versa,* f is a maximal flow iff β_f is an optimal solution. Figure 2 illustrates these definitions and the interplay between solutions for $\Pi_{\mathrm{M}}(G)$ and flows in the corresponding Hochbaum network. Since the translation between flows and solutions is computationally easy, we freely switch between flows and solutions for $\Pi_{\mathrm{M}}(G)$ as needed.

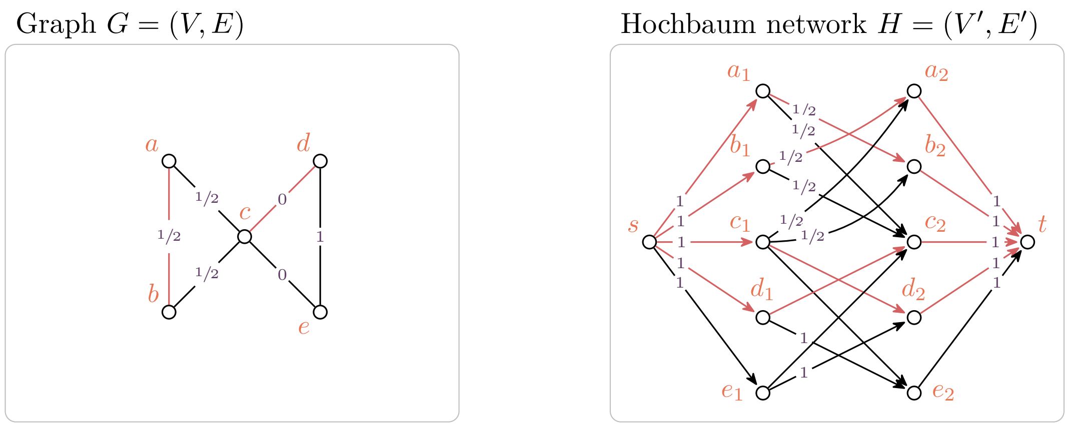

Figure 2: The left side shows a graph $G = (V, E)$ on five vertices $V = \{a, b, c, d, e\}$. On the edges a *half-integral* solution β of $\Pi_{\mathrm{M}}(G)$ of value $\mathrm{opt}_{\mathbb{N}/2}(\Pi_{\mathrm{M}}(G)) = 2.5$ is illustrated. The two red edges constitute an optimal *integral* solution for $\Pi_{\mathrm{M}}(G)$. The right side shows the Hochbaum network $H = (V', E')$ corresponding to G. The edges are labeled with a maximum flow f_β of value $|f_\beta| = 5$ that corresponds to β. The integral solution (the red maximum matching) corresponds to the flow of value four that sends one unit over every red edge (which is not maximal).

Lemma 30. *There is a function in* para-FNL$^\uparrow$ *that maps $((G, \beta), g)$, consisting of a graph G, a half-integral solution β of $\Pi_{\mathrm{M}}(G)$, and a number g, to an* optimal *half-integral solution β' of G, provided such a solution with $|\beta'| \leq |\beta| + g$ exists.*

Proof. The mapping of (G, β) to the Hochbaum network H and the flow f_β from Fact 29 can easily be done in (even deterministic) logarithmic space. The flow f_β does not have to be a maximal flow, but we can turn it into a maximal flow using Corollary 22: This corollary states that a para-FNL$^\uparrow$-machine can map f_β to a flow f' of value $|f_\beta| + 2g + 1$, if such a flow exists, or to a maximum flow f' otherwise.

If a flow of value $|f_\beta| + 2g + 1$ exists, Fact 31 tells us that $\beta_{f'}$ is a solution of $\Pi_{\mathrm{M}}(G)$ of value $|\beta_{f'}| = |f'|/2 = |f_\beta|/2 + g + 1/2 = |\beta| + g + 1/2$. In particular, $|\beta| + g < |\beta_{f'}| \leq \mathrm{opt}_{\mathbb{Q}}(\Pi_{\mathrm{M}}(G))$ and, thus, no optimal solution with $|\beta| + g \geq |\beta'|$ exists. Hence, we output the error symbol.

If there is no flow of value $|f_\beta| + 2g + 1$, we know that f' is a maximum s-t-flow in H and, by Fact 29, we can output $\beta_{f'}$ as optimal solution for $\Pi_{\mathrm{M}}(G)$. $\square$

Lemma 30 provides a reduction rule for p_g-VC-ABOVE-RELAXED-MATCHING: We can map $((G,\beta), g)$ to $((G,\beta'), g - |\beta'| + |\beta|)$ such that β' is optimal. Since we will often use triples (G,H,f) where $G=(V,E)$ is a graph, $H=(V',E')$ is its Hochbaum network, and f is a maximum flow in H, let us call such a triple a *graph-Hochbaum-flow triple.*

We now have a way of computing an optimal solution β' for $\Pi_{\mathrm{M}}(G)$, but for the next steps of our algorithm, we need a half-integral solution α of $\Pi_{\mathrm{VC}}(G)$. Fortunately, there is another observation that shows how a maximum flow f can be used to derive an optimal solution α_f for $\Pi_{\mathrm{VC}}(G)$ (note that this solution is trivially half-integral):

Fact 31 (Hochbaum, 2002; Iwata et al., 2014)**.** *Let (G,H,f) be a graph-Hochbaum-flow triple. Let $X \subseteq V'$ be the set of vertices reachable from s in the residual network R_f. Then*

$$\alpha_f(x_v) = \begin{cases} 0 & \text{if } v_1 \in X \text{ and } v_2 \notin X, \\ 1 & \text{if } v_1 \notin X \text{ and } v_2 \in X, \\ 1/2 & \text{otherwise,} \end{cases}$$

is an optimal solution for $\Pi_{\mathrm{VC}}(G)$.

Lemma 32. *There is a function in* FNL *that maps (G,β), consisting of a graph and an* optimal *half-integral solution of $\Pi_{\mathrm{M}}(G)$, to an optimal half-integral solution α of $\Pi_{\mathrm{VC}}(G)$.*

Proof. Use Fact 29 to obtain an optimal flow f_β in the Hochbaum network from β and use Fact 31 to obtain α_{f_β} from this flow. $\square$

Together, Lemmas 30 and 32 clearly allow us to perform Step 0 of the computation (namely the computation of an optimal solution α of $\Pi_{\mathrm{VC}}(G)$) using a para-$\mathrm{FNL}^{\uparrow}$-machine.

Parallelizing Step 1: Reduction to the All-1/2-Solution. The next step turns the half-integral solution α into an all-1/2-solution by deleting all vertices v with $\alpha(x_v) \neq 1/2$. Clearly, this can be done in parallel. Note that here we really need an optimal solution α of $\Pi_{\mathrm{VC}}(G)$ rather than a solution β of $\Pi_{\mathrm{M}}(G)$: Only α tells us which vertices can be removed.

Parallelizing Step 2: Making the All-1/2-Solution Unique. The sequential method described in Section 4.1.3 for implementing Step 2 is exactly that: highly *sequential.* It is not difficult to construct a graph for which the number of iterations used by this method is linear in the graph size – just consider a large matching: The all-1/2-solution is an optimal solution, but in each iteration only one edge will be removed from the graph. Even worse, after the removal of a vertex it might be necessary to recompute the optimal solution α.

For a parallel algorithm, we need some further insights from the work of Iwata et al. (2014). Let us start with some definitions, which adapt their ideas to our context:

Definition 33. *Let (G,H,f) be a graph-Hochbaum-flow triple. A set $S \subseteq V'$ is* loose *if the following holds:*

1. *S is a strongly connected component of the residual graph $R_f = (V', E'_f)$.*

2. *$\{v \in V \mid v_1 \in S\}$ and $\{v \in V \mid v_2 \in S\}$ are disjoint.*

We call a loose set removable *if the following holds additionally:*

3. *There are no edges leaving S in R_f, i.e., no edges $(x, y) \in E'_f$ with $x \in S$ and $y \notin S$.*

Definition 34. *Let (G, H, f) be a graph-Hochbaum-flow triple and $S \subseteq V'$ be a removable set.* Removing *S yields the following triple (G^{-S}, H^{-S}, f^{-S}):*

1. *$G^{-S} = G - \{v \in V \mid v_1 \in S \vee \exists w \in N(v)[w_1 \in S]\}$,*
2. *$H^{-S} = H - S$,*
3. *f^{-S} is the flow induced on the vertices of H^{-S}.*

Intuitively, (G^{-S}, H^{-S}, f^{-S}) should also be a graph-Hochbaum-flow triple and this is the case, at least if $\alpha \equiv 1/2$ is an optimal solution:

Fact 35 (Iwata et al., 2014, Corollary 4.2 and the subsequent discussion)**.** *Let (G, H, f) be a graph-Hochbaum-flow triple such that $\alpha \equiv 1/2$ is an optimal solution of $\Pi_{\mathrm{VC}}(G)$.*

1. *If there is no removable set S, then $\alpha \equiv 1/2$ is the only optimal solution for $\Pi_{\mathrm{VC}}(G)$.*
2. *If there is a removable set S, then (G^{-S}, H^{-S}, f^{-S}) is a graph-Hochbaum-flow triple and G^{-S} has the same integrality excess as G.*

While the fact tells us which vertices we should remove from G, it does not tell us which will be part of the vertex cover. This can easily be fixed, however: When S is removed, we can set $\beta(x_v) = 0$ for all $v_1 \in S$ and $\beta(x_v) = 1$ for all $v \in V$ for which there is a $w \in N(v)$ with $w_1 \in S$, see the discussion after Lemma 4.6 in the work of Iwata et al. (2014).

Using Fact 35, an NL-machine can test whether $\alpha \equiv 1/2$ is the only optimal solution of $\Pi_{\mathrm{VC}}(G)$ by looking for a removable S. Furthermore, the machine can iteratively remove such sets until the all-1/2-solution is the only optimal half-integral solution. This may seem similarly sequential as the repetitive removal of vertices in Step 2, but it turns out that we can remove everything in a single run:

Lemma 36. *There is a function in* FNL *that gets a graph-Hochbaum-flow triple (G, H, f) as input and outputs the graph-Hochbaum-flow triple (G^-, H^-, f^-) resulting from iteratively removing removable sets as long as they exist.*

Proof. Consider the acyclic digraph D of all strongly connected components C of H. Some of these components will be loose sets (see Definition 33) and if they are also sinks in D, they are one of the (initial) removable sets of H. Note that removing one of these loose sinks does not change the fact that the other loose sinks are (still) removable sets in the resulting graph-Hochbaum-flow triple. Removing loose sinks from H and D may produce new loose sinks, but these sets were already loose sets in the original H (Definition 33 is "local" in the sense that only properties of vertices within the strongly connected component are relevant).

This leads to a rule for determining the set Q of all vertices that will (eventually) be removed as part of the iterative removal of removable sets: Q contains all vertices that are an element of a loose set from which only loose sets are reachable in D. This test can be implemented by an FNL-machine and the claim follows with (G^{-Q}, H^{-Q}, f^{-Q}). ☐

Parallelizing Step 3: Branching. As mentioned earlier, the branching step is easy to parallelize, as the two children in the search tree can be explored in parallel. Branching also fits nicely into our framework of the up-class para-FNL$^{\uparrow}$, which arises from parameter-dependent-many iterations of a linear function in para-FNL: In each iteration a list of instances is on the input tape and this list is mapped to at most twice as many new instances on the output tape, but with a reduction of the parameter in all these instances.

Proof of Theorem 28. Let $((G,\beta),g)$ be given as input, where $G=(V,E)$ is an undirected graph, β is a half-integral solution of $\Pi_{\mathrm{M}}(G)$, and g is a parameter.

To show that a problem is in para-NL$^{\uparrow}$, we must specify an initial function and an iteration function, both in para-FNL. In our case the initial function simply maps $((G,\beta),g)$ to the single-element list $\big(((G,\beta),g)\big)$. This list, which will change after each application of the iteration function, will satisfy the following invariant: The original instance $((G,\beta),g)$ is a positive instance iff at least one instance in the list is a positive instance. Clearly, after the application of the initial function, this invariant is true.

The iteration function gets a list $\big(((G_1,\beta_1),g_1),\dots,((G_l,\beta_l),g_l)\big)$ as input and will output a new list of such pairs that is at most twice as long (which will ensure that the iteration function is linear, see Definition 1). When processing the pairs, the iteration function may notice that one of the pairs is a positive instance. Because of the invariant, the iteration function can now immediately output "yes" (formally, it outputs $(1,0)$ and further iterations do nothing except for copying this tuple to their output tape). It may also happen that the list becomes empty (at the latest after $2g+1$ iterations), in which case the invariant implies that the original instance was a negative instance and the iterations function immediately outputs "no" in the form of $(0,0)$ (and once more further iterations do not modify this).

We now describe how the iteration function processes a pair $((G_i,\beta_i),g_i)$ in the list, i.e., which new pairs are added to the output list (if any). For Definition 1, we have to implement the iterator function in para-FNL, but Lemma 4 allows us to use para-FNL$^{\uparrow}$-transformation instead, as long as the initial functions are linear (which they are).

Step 0, first part. Apply Lemma 30 to $((G_i,\beta_i),g_i)$. This will yield a new instance $((G_i,\beta'),g')$ such that β' is an optimal solution of $\Pi_{\mathrm{M}}(G_i)$ – or an error symbol, in which case we know that $((G_i,\beta_i),g_i)$ was a no-instance and we can skip it.

Step 0, second part. Apply Lemma 32 to obtain an optimal solution α for $\Pi_{\mathrm{VC}}(G_i)$.

Step 1. Remove all vertices $v\in V$ from G_i with $\alpha(v)\neq 1/2$, yielding the graph G'.

Step 2. Compute the graph-Hochbaum-flow triple (G',H',f') and apply Lemma 36 to it. This yields a graph-Hochbaum-flow triple (G^-,H^-,f^-) such that (i) the integrality excess of G^- is the same as that of G and, hence, $((G^-,\beta_{f^-}),g')$ is an element of p_g-VC-ABOVE-RELAXED-MATCHING iff $((G',\beta_{f'}),g')$ is, and (ii) $\alpha\equiv 1/2$ is the only optimal half-integral solution of $\Pi_{\mathrm{VC}}(G^-)$.

Step 3. If in $((G^-,\beta_{f^-}),g')$ the graph G^- contains no edges and $g'\geq 0$, we have found a yes-instance and can stop. Likewise, if $g'<0$, we have a no-instance and can also stop. Otherwise, we branch by picking an arbitrary edge $e=\{u,v\}$ in G^- and, starting with u, consider the graph $G_u=G^- - \{u\}$. In the corresponding Hochbaum network H_u the vertices u_1 and u_2 will be missing. Consider the flow f_u that is obtained from f^- by removing any flow through u_1 or u_2. Then $|f_u| \geq |f^-| - 2$ and $|f_u|$ can be at most 2 below the value of a maximum flow in H_u. Lemma 30 allows us to restore the maximality by computing a maximum flow f'_u in H_u. We add $((G_u, \beta_{f'_u}), g' - 1/2)$ to the list. Then we repeat the whole process with v and also add $((G_v, \beta_{f'_v}), g' - 1/2)$ to the list.

To see that the branching is correct and upholds the invariant, suppose $((G^-, \beta_{f^-}), g')$ is a yes-instance, i.e., the integrality excess of G^- is at most g'. For the edge $\{u, v\}$ one of the vertices must be in a minimal vertex cover – suppose it is u. Then

$$\mathrm{opt}_{\mathbb{N}}(\Pi_{\mathrm{VC}}(G^-)) = \mathrm{opt}_{\mathbb{N}}(\Pi_{\mathrm{VC}}(G^- - \{u\})) + 1,$$
$$\mathrm{opt}_{\mathbb{Q}}(\Pi_{\mathrm{VC}}(G^-)) = \mathrm{opt}_{\mathbb{Q}}(\Pi_{\mathrm{VC}}(G^- - \{u\})) + 1/2.$$

To see the last equality, observe that if we had

$$\mathrm{opt}_{\mathbb{Q}}(\Pi_{\mathrm{VC}}(G^-)) = \mathrm{opt}_{\mathbb{Q}}(\Pi_{\mathrm{VC}}(G^- - \{u\})) + 1,$$

then any optimal solution α for $\Pi_{\mathrm{VC}}(G^- - \{u\})$ could be augmented to an optimal solution for $\Pi_{\mathrm{VC}}(G^-)$ by setting $\alpha(u) = 1$, contradicting the assumption that $\alpha \equiv 1/2$ is the only optimal solution of $\Pi_{\mathrm{VC}}(G^-)$. The two equalities taken together show that the integrality excess of $\Pi_{\mathrm{VC}}(G^- - \{u\})$ is, indeed, 1/2 less than that of G^-. ☐

4.2 Dual Parameterization When Every Variables Occur at Most Twice

A formula ϕ is in CNF(2) if it is a CNF and every variable occurs at most twice (variables may occur positively and negatively, and clauses may be arbitrarily large). Johannsen showed that the satisfiability problem and the nae-satisfiability problem for CNF(2) formulas are complete for L (Johannsen, 2004). We extend this result and observe that the logspace algorithms can be modified such that they solve the corresponding maximization problem: Given a CNF(2) formula ϕ, they output the maximum number of simultaneously satisfiable clauses. Combined with Observation 14 we obtain:

Theorem 37. p_k-ALMOST-NAE-SAT(2) *and* p_k-ALMOST-SAT(2) *are complete for* para-L.

Lemma 38. *There is a function in* FL *that maps* CNF(2) *formulas* ϕ *to the maximum number of simultaneously satisfiable clauses of* ϕ.

Proof. We follow the proof by Johannsen and first count and remove all empty clauses (these can never be satisfied), then we represent ϕ as a *tagged graph* $G(\phi)$. Such a graph is a triple (V, E, T) in which

- $V = \mathrm{clauses}(\phi)$ is the set of vertices,
- $E = \big\{\{C_i, C_j\} \mid \exists x \in \mathrm{vars}(\phi)[x \in C_i \wedge \neg x \in C_j]\big\}$ is a *multiset* of undirected edges that connects clauses that contain complementary literals, and
- $T = \{C_i \mid C_i \text{ contains a pure literal}\} \subseteq V$ is a set of tagged vertices (a literal is *pure* if the negated literal is not present in the formula or, equivalently, if the occurrences of the literal's variable are either all positive or all negated).

Note that the graph is a multigraph, i.e., if clauses share multiple complementary literals, they are connected by multiple edges.

Johannsen observed that the satisfiability problem of ϕ is equivalent to the following orientation problem of $G(\phi)$ (Johannsen, 2004, Proposition 1): *Can the edges of $G(\phi)$ be directed such that there is no untagged sink?* The intuition is that tagged clauses can greedily be satisfied by setting the pure literal they contain, and that a variable $x \in \text{vars}(\phi)$ can be used to satisfy exactly one of the two clauses it connects – orienting an edge $\{C_i, C_j\}$ as $C_i \to C_j$ thus means to set x such that it satisfies C_i but has no effect on C_j.

Any connected component of $G(\phi)$ that contains a tagged vertex v can be oriented in this way (just perform a depth-first search from v and orient all edges towards the root of the dfs-tree). If a connected component contains a cycle, we can satisfy all vertices on that cycle by orienting it as a directed cycle. Then we can virtually contract the cycle, tag the resulting vertex, and use the previous argument. Hence, Johannsen concluded: ϕ is satisfiable iff $G(\phi)$ does not contain a connected component without a tagged vertex that is a tree. Since computing connected components and testing whether a component is a tree can be done in logarithmic space, it follows that $\text{SAT}(2) \in \text{L}$.

If ϕ is satisfiable, the FL function that we wish to construct simply outputs m, the number of clauses. So assume that ϕ is unsatisfiable. By the above argument, $G(\phi)$ then contains connected components $T_1, \dots, T_k$ $(k \geq 1)$ that are trees and that do not contain tagged vertices (these are the unsatisfiable cores of ϕ). To make ϕ satisfiable, we have to delete at least one clause per core, thus, we can satisfy at most $m - k$ clauses.

On the other hand, deleting any clause C in a tree T_i will make all literals contained in C pure and, thus, will tag all neighbors of C in $G(\phi)$. Hence, by deleting an arbitrary clause from T_i we can make the remaining clauses of T_i satisfiable. In conclusion, we can always satisfy at least $m - k$ clauses and, thus, we can output $m - k$ (taking into account the clauses removed in the preprocessing step). □

Lemma 39. *There is a function in* FL *that maps* $\text{CNF}(2)$ *formulas* ϕ *to the maximum number of simultaneously nae-satisfiable clauses of* ϕ.

Proof. The proof is similar to the proof of Lemma 38: On input ϕ, we first count and remove all empty and unit clauses (these can never be nae-satisfied). Then we construct a tagged graph $H(\phi) = (V, E, T)$ as follows, where the d_x are fresh dummy vertices:

- $V = \text{clauses}(\phi) \cup \{d_x \mid x \in \text{vars}(\phi) \text{ appears positive and negative}\}$;
- $E = \big\{\{C_i, C_j\} \mid C_i \text{ and } C_j \text{ contain a common literal}\big\} \cup \big\{\{C, d_x\} \mid x \in C \text{ or } \neg x \in C\big\}$;
- $T = \{C \mid C \in \text{clauses}(\phi) \text{ contains a variable that does not occur in another clause}\}$.

Johannsen observed that ϕ is nae-satisfiable iff *the edges of $H(\phi)$ can be colored with two colors such that each untagged vertex is adjacent to edges of both colors.* (The intuition is that edges correspond to literals and colors represent truth values of these literals; tagged vertices can always be nae-satisfied with their private literal.)

Since a tagged graph can be colored in the described way iff (i) every untagged vertex has degree at least two and (ii) every connected component without tagged vertices is *not* a simple odd length cycle (Johannsen, 2004, Lemma 9), we get $\text{NAE-SAT}(2) \in \text{L}$ (both criteria can easily be checked in logarithmic space).

As in Lemma 38, if ϕ is satisfiable, the FL function we construct simply outputs m. So assume otherwise. Then there are connected components $O_1, \dots, O_k$ in $H(\phi)$ that do not contain a tagged vertex and that are simple odd length cycles (these are the nae-unsatisfiable cores of ϕ). Clearly, any assignment can satisfy at most $m - k$ clauses.

However, deleting an arbitrary clause C from an odd cycle O_i will tag all the neighbors of C (either the neighbor is another clause that now has a private variable, or it is a dummy vertex that now corresponds to a variable that occurs only once). Hence, we can satisfy at least $m - k$ clauses and can, thus, output $m - k$ (taking into account the amount of clauses we have removed in the preprocessing step). $\square$

4.3 Dual Parameterization for Formulas in Disjunctive Normal Form

Testing whether a DNF is satisfiable can be done in polynomial time (even in AC^0), in contrast, deciding whether we can satisfy k terms simultaneously (i.e., MAX-DNF) is NP-complete (Escoffier & Paschos, 2005). In this section we study MAX-DNF with a dual parameterization: p_k-ALMOST-DNF asks whether a given DNF ϕ has an assignment that satisfies at least $m - k$ terms.

Theorem 40. p_k-ALMOST-DNF $\in$ para-AC^0.

The proof of the theorem boils down to the following reduction and the subsequent lemma. Construct a CNF ψ from ϕ by simply negating every term, i.e., if $(\ell_1 \wedge \cdots \wedge \ell_d)$ is a term in ϕ, we add $(\neg\ell_1 \vee \cdots \vee \neg\ell_d)$ as clause to ψ. Observe that every assignment that satisfies a term in ϕ does *not* satisfy the corresponding clause in ψ. Hence, there is an assignment satisfying at least $m - k$ terms in ϕ if there is an assignment that satisfies *at most* k clauses in ψ. In other words, we have reduced p_k-ALMOST-DNF to p_k-MIN-SAT.

Lemma 41. p_k-MIN-SAT $\in$ para-AC^0.

Proof. The following reduction from p_k-MIN-SAT to p_k-VERTEX-COVER by Marathe and Ravi (1996) is computable in para-AC^0 and is parameter-preserving. It takes an input (ϕ, k) and constructs a vertex cover instance $(G(\phi), k)$ as follows: The vertex set of $G(\phi)$ is clauses(ϕ) and two clauses are connected by an edge if they contain complementary literals. Observe that any variable that occurs both, positively and negatively, satisfies at least one clause. In other words, every edge in $G(\phi)$ connects two clauses such that any assignment satisfies at least one of them. Therefore, we have $(\phi, k) \in \mathrm{p}_k$-MIN-SAT iff the edges of $G(\phi)$ can be covered by at most k vertices, i.e., if $(G(\phi), k) \in \mathrm{p}_k$-VERTEX-COVER. The claim follows as p_k-VERTEX-COVER $\in$ para-AC^0 (Bannach et al., 2015, Theorem 4.5). $\square$

5. Structural Parameterizations for Partial MaxSAT Variants

The most general incarnation of MAX-SAT is the *partially weighted version.* The input is a CNF ϕ and a function $\omega\colon$ clauses$(\phi) \to \mathbb{N} \cup \{\infty\}$, in which we call clauses C *soft* if $\omega(C) < \infty$ and *hard* otherwise. The goal is to find among all assignments $\beta\colon$ vars$(\phi) \to \{0, 1\}$ that satisfy *all* hard clauses the one that *maximises* the sum of the satisfied soft clauses. We refer to the decision version, in which a target sum is given, as PARTIAL-MAX-SAT.

The usual approach to identify tractable fragments of PARTIAL-MAX-SAT is to use *structural parameters*, an overview is provided by Dell et al. (2017). Structural parameters are defined over the *incidence graph* of the input formula ϕ, which is the bipartite graph on vertex set $\text{vars}(\phi) \cup \text{clauses}(\phi)$ that contains an edge between $x \in \text{vars}(\phi)$ and $C \in \text{clauses}(\phi)$ if either $x \in C$ or $\neg x \in C$.

Natural parameters are the *vertex cover number*, the *treedepth*, the *feedback vertex set number*, or the *treewidth* of the incidence graph, see Figure 3 for an overview of how these parameters are related. It is well-known that PARTIAL-MAX-SAT is in para-P parameterized by any of these, which follows quite directly from optimization versions of Courcelle's Theorem (Courcelle, 1990). By the parallel version of this theorem (Bannach & Tantau, 2016) it follows that PARTIAL-MAX-SAT lies in para-$\text{AC}^{2\uparrow}$ if parameterized by *both*, the structural parameter and the solution size.

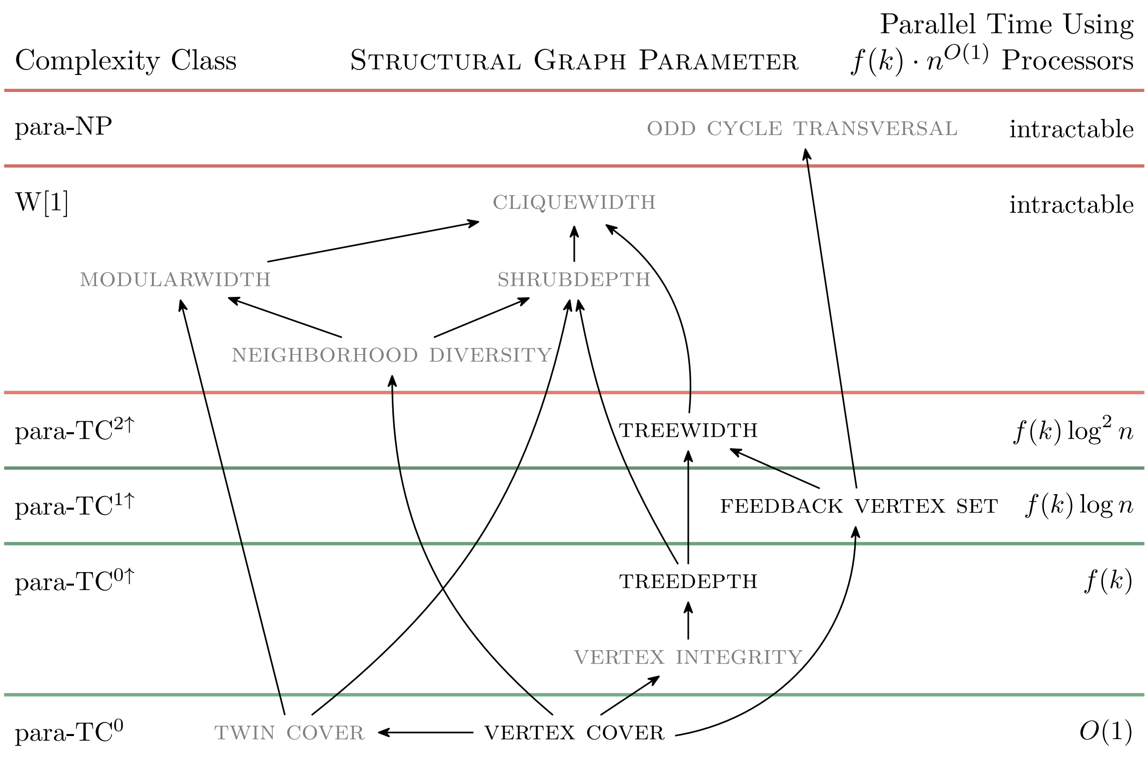

Figure 3: A Hasse diagram of the major structural graph parameters. An arrow from A to B here means that for any graph G the parameter B is upper-bounded by a function in A. Each node corresponds to the complexity of PARTIAL-MAX-SAT parameterized by the corresponding value of the input's incidence graph. The darker parameters are the ones mentioned in Theorem 42.

In the remainder of this section we develop handcrafted algorithms for all four structural parameters that (i) work independently of the solution size (it does *not* have do be a parameter), (ii) work with arbitrary weights, and (iii) are constructive in the sense that an optimal assignment is output. Figure 3 reveals intriguing connections between these parameters to the degree of parallelism we can achieve – connections that remain hidden in the study of sequential para-P algorithms.

Theorem 42.

1. $\mathrm{p_{vc}}$-PARTIAL-MAX-SAT $\in$ para-TC^0,
2. $\mathrm{p_{td}}$-PARTIAL-MAX-SAT $\in$ para-$\mathrm{TC}^{0\uparrow}$,
3. $\mathrm{p_{fvs}}$-PARTIAL-MAX-SAT $\in$ para-$\mathrm{TC}^{1\uparrow}$,
4. $\mathrm{p_{tw}}$-PARTIAL-MAX-SAT $\in$ para-$\mathrm{AC}^{2\uparrow}$.

For each item of the theorem, we prove a lemma in the following.

Lemma 43. *There is a uniform family of constant-depth* TC *circuits of size $f(k) \cdot n^{O(1)}$ that, on input (ϕ, ω, k), either reports that the incidence graph of ϕ has no vertex cover of size k, or outputs the assignment of an optimal solution for* PARTIAL-MAX-SAT *on (ϕ, ω).*

Proof. First construct the incidence graph, which is easy in AC^0. Subsequent compute a vertex cover of size at most $k^2 + 2k$ or decide that there is no vertex cover of size at most k at all as follows: Run the parallel version of the Buss kernel with parameter k (Bannach et al., 2015); if the kernelization produces a trivial no-instance (if it "rejects"), then there is no vertex cover of size k and we may reject as well. Otherwise, we have a kernel with $k^2 + k$ vertices and a set of at most k vertices that were classified by the Buss kernel as being necessary for any vertex cover (i.e., high-degree vertices). Together we obtain the desired vertex cover X of size at most $k^2 + 2k$.

Let $V_1 = \mathrm{vars}(\phi) \cap X$ and $V_2 = \mathrm{vars}(\phi) \setminus V_1$; $S_1 = \{C \in \mathrm{clauses}(\phi) \mid \omega(C) < \infty\} \cap X$ and $S_2 = \{C \in \mathrm{clauses}(\phi) \mid \omega(C) < \infty\} \setminus S_1$; and $H_1 = \{C \in \mathrm{clauses}(\phi) \mid \omega(C) = \infty\} \cap X$ and $H_2 = \{C \in \mathrm{clauses}(\phi) \mid \omega(C) = \infty\} \setminus H_1$. The circuit runs the following steps in sequence:

1. Brute-force (i.e., test in parallel) all possible partial assignments for the variables in V_1. Discard assignments that leave a clause in H_2 unsatisfied.
2. Brute-force (i.e., test in parallel) which clauses of $\tilde{S}_1 \subseteq S_1$ shall be satisfied.
3. Verify that the current partial solution can be extended to an assignment that satisfies $\tilde{S}_1 \cup H_1$. Since $|\tilde{S}_1 \cup H_1| \leq k^2 + 2k$, we can check if there is an assignment satisfying all clauses of this subformula using Theorem 6.

Note that the first step already determines the truth value of all clauses in $S_2 \cup H_2$. It remains to determine the sum of the weights of all satisfied clauses, which is easy in TC^0. Observe that all steps are constructive, i.e., at this point we have a list of roughly 2^{k^2+k} assignments and their weights – we just have to output the one with the largest weight. ☐

Lemma 44. *There is a uniform family of* TC *circuits of depth $f(k)$ and size $f(k) \cdot n^{O(1)}$ that, on input (ϕ, ω, k), either reports that the treedepth of the incidence graph of ϕ exceeds k, or that outputs the assignment of an optimal solution for* PARTIAL-MAX-SAT *on (ϕ, ω).*

Proof. A *treedepth decomposition* of an undirected graph $G = (V, E_G)$ is a rooted forest $F = (V, E_F)$ (on the same vertex set) such that G is a subgraph of the closure of F. The *treedepth* of G is the minimum depth any treedepth decomposition of G must have, see the textbook by Nesetril and de Mendez (2012) for a detailed introduction to these notations.

A uniform family of para-$\mathrm{FAC}^{0\uparrow}$ circuits is known that maps a pair (G, k) either to $\perp$ (in which case the treedepth of G exceeds k) or to a treedepth decomposition F of depth at most $O(2^k)$ (Bannach & Tantau, 2016, Theorem 5). Furthermore, if access to a depth-$f(k)$ treedepth decomposition is provided, para-$\mathrm{FAC}^{0\uparrow}$ circuits can perform depth-first and breadth-first searches on G and, thus, can compute connected component (Bannach & Tantau, 2016, Lemma 6).

Since para-$\mathrm{FAC}^{0\uparrow} \subseteq$ para-$\mathrm{FTC}^{0\uparrow}$, we can assume that we have access to a depth-k' treedepth decomposition F of the incidence graph of ϕ (with $k' \in O(2^k)$), and that we can compute connected components in the incidence graph. For a CNF ψ, a variable $x \in \mathrm{vars}(\psi)$, and a bit $i \in \{0, 1\}$ let us denote by $\psi|_{x \mapsto i}$ the formula obtained by deleting all clauses from ψ that are satisfied by setting x to i, and by removing all remaining occurrences of x from the remaining clauses. The claim is proven by running the Davis–Putnam–Logemann–Loveland (DPLL) algorithm with a variable selection heuristic based on F. Clearly, the algorithm from Listing 1 correctly solves PARTIAL-MAX-SAT.

Listing 1: An algorithm that outputs an optimal PARTIAL-MAX-SAT solution on input of a weighted CNF (ϕ, ω) and a depth-k' treedepth decomposition F of the incidence-graph of ϕ.

1 ***function*** DPLL-TD(ϕ, ω, F)
2   ***if*** *ϕ contains a hard empty clause* ***then return*** $-\infty$
3   ***if*** *ϕ is empty or contains only empty soft clauses* ***then return*** 0
4   ***if*** *the incidence graph of ϕ is unconnected* ***then***
5         $\phi_1, \dots, \phi_\ell \leftarrow$ *the connected subformulas*
6   ***for*** $i \in \{1, \dots, \ell\}$ ***pardo***
7             $\sigma_i \leftarrow$ DPLL-TD(ϕ_i, ω, F)
8   ***return*** $\sum_{i=1}^{\ell} \sigma_i$
9   ***else***
10         $x \leftarrow$ *the variable in* $\mathrm{vars}(\phi)$ *closest to the root of* F
11   ***for*** $i \in \{0, 1\}$ ***pardo***
12             $\delta_i \leftarrow$ *the sum of soft clauses satisfied by setting x to i*
13             $\sigma_i \leftarrow$ DPLL-TD$(\phi|_{x \mapsto i}, \omega, F)$
14   ***return*** $\max(\sigma_0 + \delta_0, \sigma_1 + \delta_1)$

We are left with the task of arguing that a circuit family of depth $f(k)$ and size $f(k) \cdot n^{O(1)}$ can implement this algorithm. All operations can be computed by para-$\mathrm{TC}^{0\uparrow}$ circuits: Computing the connected components in line 5 can be done in depth $f(k)$ since the treedepth is bounded; the sum of multiple binary numbers in line 8 can be computed in constant depth (using threshold gates) by a result of Chandra et al. (1984); and all remaining operations are either simple arithmetic or the computation of projections.

Since the depth of F is k', the recursion depth of the algorithm is $O(k') \subseteq O(2^k)$. Finally, since F has at most $n + m = |\text{vars}(\phi)| + |\text{clauses}(\phi)|$ leaves, the total number of explored subformulas is bounded by $O(2^{2^k}(n+m))$.

We get the claim by adapting the algorithm such that it does not only return the maximum solution, but also the corresponding assignment. In detail, we assume a total order on the variables of ϕ, i.e., $\text{vars}(\phi) = \{x_1, \dots, x_n\}$, and represent an assignment $\beta\colon \text{vars}(\phi) \to \{0,1\}$ as a bit mask $\beta \in \{0,1\}^n$. At the end of the recursion, i.e., in lines 2 and 3, we return an assignment $x \mapsto 0$ in the form of $\beta = 0^n$. Getting such an assignment from the recursive call in line 13, we can obtain a corresponding assignment β_i by setting the bit corresponding to x to i. Finally, after the recursion into connected components following line 5, we return the *bitwise or* of all obtained assignments in line 8 (note that, since the formula was disconnected, these assignments modified pairwise different bits). $\square$

Lemma 45. *There is a uniform family of* TC *circuits of depth $f(k) \cdot \log n$ and size $f(k) \cdot n^{O(1)}$ that, given (ϕ, ω, k), either reports that ϕ's incidence graph has no size-k feedback vertex set, or outputs the assignment of an optimal solution for* PARTIAL-MAX-SAT *on (ϕ, ω).*

Proof. A *tree decomposition* of an undirected graph $G = (V_G, E_g)$ is a tuple (T, ι), where $T = (V_T, E_T)$ is a tree and $\iota\colon V_T \to 2^{V_G}$ a mapping from nodes of T to subsets of vertices of G, which we call *bags*. A tree decomposition has to satisfy the following constraints, see for instance Chapter 7 in the textbook of Cygan et al. for a detailed introduction:

- The set $\{x \mid v \in \iota(x)\}$ is non-empty and connected in T for every $v \in V_G$.
- For every $\{u,v\} \in E_G$ there is a $y \in V_T$ with $\{u,v\} \subseteq \iota(y)$.

The *width* of (T, ι) is the size of the largest bag minus one, and the *treewidth* of G is the minimum width any tree decomposition of G must have.

Clearly, a graph with a feedback vertex set X of size at most k has treewidth at most $k+1$: Remove X from G and obtain a tree, then consider the tree as tree decomposition and add X to every bag. Hence, by Example 2 (and since para-FL$^{\uparrow}$ $\subseteq$ para-TC$^{1\uparrow}$), we can either conclude that the incidence graph of ϕ has no feedback vertex set of size k, output this and stop; or we can compute a tree decomposition of width at most $k+1$. In order to make the description of the following dynamic program simpler, we bring (T, ι) into a standard form called *balanced nice tree decomposition.* In this form, T is a rooted tree of depth $f(k) \cdot O(\log n)$ such that every node x has one of the following types:

Leaf Nodes They have no children.

Introduce Nodes They have exactly one child y with $\iota(x) = \iota(y) \cup \{v\}$ for a $v \in V_G$.

Forget Nodes They have exactly one child y with $\iota(x) = \iota(y) \setminus \{v\}$ for a $v \in V_G$.

Join Nodes They have exactly two children y and z with $\iota(x) = \iota(y) = \iota(z)$.

A uniform family of para-FNC$^{1\uparrow}$ circuits that maps arbitrary width-k tree decomposition to balanced nice tree decompositions of width at most $8k+3$ is known (Bannach & Tantau, 2016, Lemma 9). Since para-FNC$^{1\uparrow}$ $\subseteq$ para-TC$^{1\uparrow}$ we may assume that (T, ι) is in this special form. Let us, to keep the notation intuitive, denote the size of the largest bag of the transformed decomposition with k (even though it did, of course, grow a little).

Our task is to describe a family of para-FTC$^{1\uparrow}$ circuits that obtains as input a tuple $\bigl((\phi,\omega,T,\iota),k\bigr)$ (where (T,ι) is a width-$(k+1)$ tree decomposition of the incidence graph of ϕ and k is the parameter) and outputs an assignment that satisfies all hard clauses while maximising the sum of the weights of satisfied soft clauses (or detects that such an assignment does not exist). The idea of the following algorithm is a dynamic program that bubbles up the tree decomposition, assigning *configuration sets* to the nodes of T. We think of T as being layered (with $f(k)\cdot\log n$ layers). Thus, all we have to do is to design TC circuits of depth $f'(k)$ (independent of n) and size $f'(k)\cdot n^{O(1)}$ for some computable function f', which compute the configuration sets of a node in T, given the configuration sets of its children.

Assume that there is a total order on the vertices of the incidence graph (for instance, take the lexicographical order induced by ϕ). A *configuration* is a triple (μ,σ,β), where $\mu\in\{0,1\}^k$ is a bit mask, $\sigma\in\mathbb{N}$ a weight, and $\beta\in\{0,1\}^n$ another bit mask (for a node $x\in V_T$ we interpret the bit masks as $\mu\colon\iota(x)\to\{0,1\}$ and $\beta\colon \mathrm{vars}(\phi)\to\{0,1\}$). For instance, say we have $\mathrm{vars}(\phi)=\{x_1,\dots,x_{100}\}$ and assume the incidence graph has width 6. We encode an assignment β as bit mask $\beta\in\{0,1\}^{100}$ with the ith bit set iff x_i is assigned to 1. For a node $x\in V_T$ of the tree decomposition we represent the local information $\mu\colon\iota(x)\to\{0,1\}$ as another bit mask $\mu\in\{0,1\}^5$, where the ith bit corresponds to the information stored for the lexicographical ith element of $\iota(x)$, e.g., if $\iota(x)=\{x_1,x_3,x_{42}\}$ we would store the information at the following positions:

$$\left(\begin{smallmatrix}\textit{bit position:} & 1 & 2 & 3 & 4 & 5\\ \textit{information:} & x_1 & x_3 & x_{42} & 0 & 0\end{smallmatrix}\right).$$

Note that we always set unused positions to the default value 0. During the execution of the dynamic program, we will encounter new nodes of the tree decomposition that differ by at most one element of the previous one (i.e., we introduce a vertex to the bag). To reuse a previous μ, we have to *shift* its content accordingly. For instance, if y is another bag with $\iota(y)=\iota(x)\cup\{x_2\}$, then for y we would store the corresponding data at the following positions, shifting data to the right from position 2 ongoing:

$$\left(\begin{smallmatrix}\textit{bit position:} & 1 & 2 & 3 & 4 & 5\\ \textit{information:} & x_1 & x_2 & x_3 & x_{42} & 0\end{smallmatrix}\right).$$

We say two configurations (μ_1,σ_1,β_1) and (μ_2,σ_2,β_2) are *equivalent* if $\mu_1=\mu_2$. Furthermore, a configuration is *better* than another if $\mu_1=\mu_2$ and $\sigma_1>\sigma_2$. A *configuration set* is a set of pairwise non-equivalent configurations. Note that such a set contains at most 2^k elements. For a node $x\in V_T$, a configuration (μ,σ,β) fulfills the following invariant:

1. For all $v\in\iota(x)\cap\mathrm{vars}(\phi)$ we have $\beta(v)=\mu(x)$.
2. For all $C\in\iota(x)\cap\mathrm{clauses}(\phi)$ we have $\beta\models C$ iff $\mu(C)=1$.
3. The assignment β satisfies all hard clauses in the subtree rooted at x.
4. The sum of the weights of soft clauses satisfied by β in the subtree rooted at x is σ.

Since Leaf Nodes have no children, there is not much to do for them. The configuration set contains a single configuration $(0^k,0,0^n)$.

The circuit for Introduce Nodes x obtains as input a configuration set S and an introduced vertex v, and outputs the following configuration set S': If $v \in \text{clauses}(\phi)$, construct for every $(\mu, \sigma, \beta) \in S$ a single new configuration $(\mu', \sigma', \beta) \in S'$ obtained by shifting μ according to the ordering of the vertices in $\iota(x)$, setting $\mu'(v) = 1$ iff $\beta \models v$ (and $\mu'(v) = 0$ otherwise), and setting $\sigma' = \sigma + \omega(v)$ if v is a satisfied soft clause, otherwise setting $\sigma' = \sigma$. If $v \in \text{vars}(\phi)$, we add for every $(\mu, \sigma, \beta) \in S$ configurations $(\mu_0, \sigma_0, \beta_0)$ and $(\mu_1, \sigma_1, \beta_1)$ to S', where we obtain $(\mu_i, \sigma_i, \beta_i)$ for $i \in \{0, 1\}$ from (μ, σ, β) by shifting μ, by setting $\mu_i(v) = i$, and by updating β_i appropriate. Subsequently, we check for every clause $C \in \iota(x)$ with $\mu(C) = 0$ whether $\beta_i \models C$, in which case we set $\mu_i(C) = 1$ and update σ_i as needed.

For Forget Nodes x that forget a vertex v, we construct a new configuration set S' from S by initially setting $S' = S$. Then in all configurations we remove v from μ by setting the corresponding bit to 0 and by shifting μ according to the new ordering in $\iota(x)$. If v was a hard clause, we remove all configurations with $\mu(v) = 0$ (i.e., configurations in which v was not satisfied). Finally, S' may now contain some equivalent configurations, in which case we keep just the one that is best.

When joining two bags in a Join Node x, we obtain two configuration sets S_1 and S_2 as input, and have to construct a new configuration set S' that fulfils the invariant. Recall that the children of x have the same bag as x. Define for every $X \subseteq \text{vars}(\phi) \cap \iota(x)$ and $i \in \{1, 2\}$:

$$S_i^X = \big\{(\mu, \sigma, \beta) \in S_i \mid \text{for all } v \in \text{vars}(\phi) \cap \iota(x) \text{ we have } \mu(v) = 1 \Leftrightarrow v \in X\big\}.$$

Build the sets $S'(X)$ by joining S_1^X and S_2^X in the following sense: Take every configuration $(\mu_1, \sigma_1, \beta_1) \in S_1^X$ and every $(\mu_2, \sigma_2, \beta_2) \in S_2^X$ and build the new configuration:

$$(\mu_1 \curlyvee \mu_2, \sigma_1 + \sigma_2 - \epsilon, \beta_1 \curlyvee \beta_2).$$

Here, "$\curlyvee$" is the *bitwise or* operation and ϵ the sum of soft clauses in $\iota(x)$ that are satisfied by both assignments, i.e., by β_1 *and* β_2. The configuration set S' is obtained by, firstly, collecting all $S'(X)$ and, secondly, by removing equivalent configurations from it.

A standard induction shows that, after the process has finished, any configuration stored in the root bag of T contains an assignment β that satisfies all hard clauses. Furthermore, the one with maximum σ corresponds to an optimal solution for PARTIAL-MAX-SAT.

The statement follows as all four operations can be implemented by TC circuits of depth $f(k)$ and size $f(k) \cdot n^{O(1)}$. For the part that manipulates μ, this follows as we have at most 2^k configurations and since $|\mu| = k$. Operations modifying σ just have to perform simple addition and subtraction, which is possible even in a constant number of TC layers of polynomial size. Regarding β, we perform only trivial bit projections. □

Lemma 46. *There is a uniform family of* AC *circuits of depth $f(k) \cdot \log^2 n$ and size $f(k) \cdot n^{O(1)}$ that, given (ϕ, ω, k), either reports that the treewidth of the incidence graph of ϕ exceeds k, or outputs the assignment of an optimal solution for* PARTIAL-MAX-SAT *on (ϕ, ω).*

Proof. An optimal tree decomposition can be computed in para-$\text{FAC}^{2\uparrow}$ (Bannach & Tantau, 2016). Afterwards the proof is equivalent to the proof of Lemma 45. □

6. Conclusion and Outlook

We presented a comprehensive list of parallel fixed-parameter algorithms for variations of MAX-SAT. As highlight we presented the first parallel algorithms for p_k-ALMOST-NAE-2SAT and p_k-ALMOST-2SAT, which implies parallel fpt-algorithms for various problems such as the odd cycle transversal problem.

The central method for proving that the latter problem is fixed-parameter tractable – the iterative compression method – seems to be inherently sequential. Interestingly, our parallel algorithm builds on another method that seems inherently sequential in general, namely the computation of maximum flows. However, using properties of the Hochbaum network allowed us to break the computation of a maximum flow into a series of small flow computations, which we can perform in parallel using fpt-many parallel processing units.

We remark that from a complexity-theoretic point of view, p_k-ALMOST-2SAT is a harder problem than p_k-ALMOST-NAE-2SAT as the former is easily seen to be hard for para-NL while the latter is easily seen to lie in para-WL (for a discussion of these classes, see the work of Elberfeld et al., 2015), which suggests that the problems have different complexity. As open problem we thus leave the question of whether p_k-ALMOST-NAE-2SAT $\in$ para-$\mathrm{L}^{\uparrow}$ holds (which would imply that the odd cycle transversal problem lies in this class, too). While we know of no complexity-theoretic assumption that would contradict this, our proofs make heavy use of finding augmenting paths in networks and these networks seem to be inherently directed.

References

Abu-Khzam, F. N., & Kontar, K. A. (2020). A Brief Survey of Fixed-Parameter Parallelism. *Algorithms*, *13*(8), 197.

Alon, N., Gutin, G. Z., Kim, E. J., Szeider, S., & Yeo, A. (2011). Solving MAX-r-SAT Above a Tight Lower Bound. *Algorithmica*, *61*(3), 638–655.

Alon, N., Yuster, R., & Zwick, U. (1995). Color-Coding. *J. ACM*, *42*(4), 844–856.

Bannach, M., Skambath, M., & Tantau, T. (2022a). On the parallel parameterized complexity of maxsat variants. In *25th International Conference on Theory and Applications of Satisfiability Testing, SAT 2022, August 2-5, 2022, Haifa, Israel*, pp. 19:1–19:19.

Bannach, M., Skambath, M., & Tantau, T. (2022b). On the parallel parameterized complexity of maxsat variants. Tech. rep. 2202.01280, arXiv.

Bannach, M., Stockhusen, C., & Tantau, T. (2015). Fast Parallel Fixed-Parameter Algorithms via Color Coding. In *10th International Symposium on Parameterized and Exact Computation, IPEC 2015, September 16-18, 2015, Patras, Greece*, Vol. 43 of *LIPIcs*, pp. 224–235. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

Bannach, M., & Tantau, T. (2016). Parallel Multivariate Meta-Theorems. In *11th International Symposium on Parameterized and Exact Computation, IPEC 2016, August 24-26, 2016, Aarhus, Denmark*, Vol. 63 of *LIPIcs*, pp. 4:1–4:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

Bannach, M., & Tantau, T. (2018). Computing Kernels in Parallel: Lower and Upper Bounds. In *13th International Symposium on Parameterized and Exact Computation,* *IPEC 2018, August 20-24, 2018, Helsinki, Finland*, Vol. 115 of *LIPIcs*, pp. 13:1–13:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

Barrington, D. A. M., Immerman, N., & Straubing, H. (1990). On Uniformity within NC^1. *Journal of Computer and System Sciences*, *41*(3), 274–306.

Biere, A., Heule, M., van Maaren, H., & Walsh, T. (Eds.). (2021). *Handbook of Satisfiability, Second Edition*. IOS Press.

Cai, L., Chen, J., Downey, R. G., & Fellows, M. R. (1997). Advice Classes of Parameterized Tractability. *Annals of Pure and Applied Logic*, *84*(1), 119–138.

Chandra, A. K., Stockmeyer, L. J., & Vishkin, U. (1984). Constant Depth Reducibility. *SIAM J. Comput.*, *13*(2), 423–439.

Chen, Y., & Flum, J. (2016). Some Lower Bounds in Parameterized AC^0. In *41st International Symposium on Mathematical Foundations of Computer Science, MFCS 2016, August 22-26, 2016 - Kraków, Poland*, pp. 27:1–27:14.

Chen, Y., & Flum, J. (2020). Parameterized Parallel Computing and First-Order Logic. In *Fields of Logic and Computation III - Essays Dedicated to Yuri Gurevich on the Occasion of His 80th Birthday*, pp. 57–78.

Chen, Y., Flum, J., & Huang, X. (2017). Slicewise Definability in First-Order Logic with Bounded Quantifier Rank. In *26th EACSL Annual Conference on Computer Science Logic, CSL 2017, August 20-24, 2017, Stockholm, Sweden*, Vol. 82 of *LIPIcs*, pp. 19:1–19:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

Courcelle, B. (1990). The Monadic Second-Order Logic of Graphs. I. Recognizable Sets of Finite Graphs. *Inf. Comput.*, *85*(1), 12–75.

Crowston, R., Fellows, M. R., Gutin, G. Z., Jones, M., Rosamond, F. A., Thomassé, S., & Yeo, A. (2011). Simultaneously Satisfying Linear Equations Over $\mathbb{F}_2$: MaxLin2 and Max-r-Lin2 Parameterized Above Average. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2011, December 12-14, 2011, Mumbai, India*, pp. 229–240.

Crowston, R., Gutin, G. Z., Jones, M., Raman, V., & Saurabh, S. (2013). Parameterized Complexity of MaxSat Above Average. *Theor. Comput. Sci.*, *511*, 77–84.

Crowston, R., Gutin, G. Z., Jones, M., & Yeo, A. (2012). A New Lower Bound on the Maximum Number of Satisfied Clauses in MaxSAT and Its Algorithmic Applications. *Algorithmica*, *64*(1), 56–68.

Cygan, M., Fomin, F. V., Kowalik, L., Lokshtanov, D., Marx, D., Pilipczuk, M., Pilipczuk, M., & Saurabh, S. (2015). *Parameterized Algorithms*. Springer.

Cygan, M., Pilipczuk, M., Pilipczuk, M., & Wojtaszczyk, J. O. (2013). On Multiway Cut Parameterized Above Lower Bounds. *ACM Transactions on Computation Theory*, *5*(1), 3:1–3:11.

Dell, H., Kim, E. J., Lampis, M., Mitsou, V., & Mömke, T. (2017). Complexity and Approximability of Parameterized MAX-CSPs. *Algorithmica*, *79*(1), 230–250.

Diestel, R. (2012). *Graph Theory, 4th Edition*, Vol. 173 of *Graduate texts in mathematics*. Springer.

Elberfeld, M., Stockhusen, C., & Tantau, T. (2015). On the Space and Circuit Complexity of Parameterized Problems: Classes and Completeness. *Algorithmica*, *71*(3), 661–701.

Escoffier, B., & Paschos, V. T. (2005). Differential Approximation of MinSAT, MaxSAT and Related Problems. In *Computational Science and Its Applications - ICCSA 2005, International Conference, Singapore, May 9-12, 2005, Proceedings, Part IV*, pp. 192–201.

Flum, J., & Grohe, M. (2003). Describing Parameterized Complexity Classes. *Information and Computation*, *187*(2), 291–319.

Ford, L. R., & Fulkerson, D. R. (1956). Maximal Flow Through a Network. *Canadian Journal of Mathematics*, *8*, 399–404.

Garey, M. R., Johnson, D. S., & Stockmeyer, L. J. (1976). Some Simplified NP-Complete Graph Problems. *Theor. Comput. Sci.*, *1*(3), 237–267.

Garg, S., & Philip, G. (2016). Raising The Bar For Vertex Cover: Fixed-Parameter Tractability Above a Higher Guarantee. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pp. 1152–1166. SIAM.

Gaspers, S., & Szeider, S. (2011). Kernels for Global Constraints. In *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011*, pp. 540–545.

Gaspers, S., & Szeider, S. (2014). Guarantees and Limits of Preprocessing in Constraint Satisfaction and Reasoning. *Artif. Intell.*, *216*, 1–19.

Goldschlager, L. M., Shaw, R. A., & Staples, J. (1982). The Maximum Flow Problem is Log Space Complete for P. *Theoretical Computer Science*, *21*, 105–111.

Grohe, M. (2006). The Structure of Tractable Constraint Satisfaction Problems. In *Mathematical Foundations of Computer Science 2006, 31st International Symposium, MFCS 2006, Stará Lesná, Slovakia, August 28-September 1, 2006, Proceedings*, pp. 58–72.

Gutin, G. Z., Jones, M., Scheder, D., & Yeo, A. (2013). A new Bound for 3-Satisfiable MaxSat and its Algorithmic Application. *Inf. Comput.*, *231*, 117–124.

Hochbaum, D. S. (2002). Solving Integer Programs over Monotone Inequalities in three Variables: A Framework for Half Integrality and Good Approximations. *European Journal of Operational Research*, *140*(2), 291–321.

Iwata, Y., Oka, K., & Yoshida, Y. (2014). Linear-Time FPT Algorithms via Network Flow. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pp. 1749–1761. SIAM.

Johannsen, J. (2004). Satisfiability Problems Complete for Deterministic Logarithmic Space. In *STACS 2004, 21st Annual Symposium on Theoretical Aspects of Computer Science, Montpellier, France, March 25-27, 2004, Proceedings*, pp. 317–325.

Karp, R. M., Upfal, E., & Wigderson, A. (1986). Constructing a Perfect Matching is in Random NC. *Combinatorica*, *6*(1), 35–48.

Mahajan, M., & Raman, V. (1999). Parameterizing above Guaranteed Values: MaxSat and MaxCut. *J. Algorithms*, *31*(2), 335–354.

Marathe, M. V., & Ravi, S. S. (1996). On Approximation Algorithms for the Minimum Satisfiability Problem. *Inf. Process. Lett.*, *58*(1), 23–29.

Narayanaswamy, N. S., Raman, V., Ramanujan, M. S., & Saurabh, S. (2012). LP can be a cure for Parameterized Problems. In *29th International Symposium on Theoretical Aspects of Computer Science, STACS 2012, February 29th - March 3rd, 2012, Paris, France*, pp. 338–349.

Nemhauser, G. L., & Trotter, L. E. (1975). Vertex Packings: Structural Properties and Algorithms. *Mathematical Programming*, *8*(1), 232–248.

Nesetril, J., & de Mendez, P. O. (2012). *Sparsity - Graphs, Structures, and Algorithms*, Vol. 28 of *Algorithms and combinatorics*. Springer.

Nickelsen, A., & Tantau, T. (2005). The Complexity of Finding Paths in Graphs with Bounded Independence Number. *SIAM Journal on Computing*, *34*(5), 1176–1195.

Papadimitriou, C. H., & Yannakakis, M. (1991). Optimization, Approximation, and Complexity Classes. *J. Comput. Syst. Sci.*, *43*(3), 425–440.

Pilipczuk, M., Siebertz, S., & Torunczyk, S. (2018). Parameterized Circuit Complexity of Model-Checking on Sparse Structures. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018, Oxford, UK, July 09-12, 2018*, pp. 789–798. ACM.

Razgon, I., & O'Sullivan, B. (2009). Almost 2-SAT is Fixed-Parameter Tractable. *Journal of Computer and System Sciences*, *75*(8), 435–450.

Reed, B. A., Smith, K., & Vetta, A. (2004). Finding Odd Cycle Transversals. *Operations Research Letters*, *32*(4), 299–301.

Szeider, S. (2011). The Parameterized Complexity of k-flip Local Search for SAT and MaxSAT. *Discret. Optim.*, *8*(1), 139–145.