

Computing Kernels in Parallel: Lower and Upper Bounds

Max Bannach Till Tantau

Institute for Theoretical Computer Science,
Universität zu Lübeck
Lübeck, Germany
{bannach,tantau}@tcs.uni-luebeck.de

Abstract

Parallel fixed-parameter tractability studies how parameterized problems can be solved in parallel. A surprisingly large number of parameterized problems admit a high level of parallelization, but this does not mean that we can also efficiently compute small problem kernels in parallel: known kernelization algorithms are typically highly sequential. In the present paper, we establish a number of upper and lower bounds concerning the sizes of kernels that can be computed in parallel. An intriguing finding is that there are complex trade-offs between kernel size and the depth of the circuits needed to compute them: For the vertex cover problem, an exponential kernel can be computed by AC^0 -circuits, a quadratic kernel by TC^0 -circuits, and a linear kernel by randomized NC -circuits with derandomization being possible only if it is also possible for the matching problem. Other natural problems for which similar (but quantitatively different) effects can be observed include tree decomposition problems parameterized by the vertex cover number, the undirected feedback vertex set problem, the matching problem, or the point line cover problem. We also present natural problems for which computing kernels is inherently sequential.

1 Introduction

The core objective of parameterized complexity has classically been to determine which problems can be solved in “FPT time,” meaning time $f(k) \cdot n^c$ for instances of size n , where c is a constant, f is an arbitrary computable function (usually at least exponential), and k is a hopefully small instance parameter. Over the last 25 years, theoreticians in the field have been very successful at determining which problems admit algorithms of this kind and practitioners have been very successful at implementing them. In both cases, the focus has traditionally been on finding *sequential* algorithms, but in recent years interest in *parallel* algorithms has sparked, leading to the new field of parallel fixed parameter tractability.

In classical sequential FPT algorithms, *kernelization algorithms* play a key role. They shrink the input to a small but difficult core (called the *kernel*), leading to the following design principle of modern parameterized algorithms: Firstly, in polynomial time, a kernelization algorithm computes a kernel that is, secondly, solved using an exponential (or worse) time algorithm – yielding a total running time of the form $f(k) + n^c$. Regarding the parallelization of these two algorithmic steps, it turns out that the second one is usually the easier one: the kernel is typically processed using the search tree technique or just by “brute force,” both of which allow natural parallelizations. In contrast, kernelization algorithms are typically described in a very sequential way, namely “apply these reduction rules over and over again.” This means that designing parallel fixed-parameter algorithms effectively means designing parallel kernelization algorithms – which is exactly what this paper addresses.

Our Contributions. We start our systematic investigation of parallel kernelization by linking the parameterized analogues of the NC -hierarchy to kernel computation using NC -circuits. Such a link is already known for FPT and kernels computed in polynomial time. We establish a circuit version of the well-known result that all algorithms running in time $f(k) \cdot n^c$ can also be implemented with running time $g(k) + n^c$: We can turn any circuit family of size $f(k) \cdot n^c$ and depth $f(k) + c \log^i n$ into one of size $g(k) + n^c$ and depth $c' \log^i n$ (note that we can remove the parameter dependence from the depth).

The bulk of the paper consists of a series of lower and upper bounds on the size of kernels that can be computed by circuits of certain depths. We show that for natural problems like the vertex cover problem

intriguing trade-offs arise: the faster our algorithm, the worse our kernel. For p -VERTEX-COVER we show that a simple exponential kernel can be computed in AC^0 , a quadratic kernel can be computed in TC^0 , and a linear kernel can be computed in randomized NC. Other problems for which we establish similar results include the tree width, path width, and tree depth problems parameterized by the vertex cover number of the input graph.

On the negative side, we also establish a number of lower bounds for the parallel computation of small kernels. We show that a classical $2k$ kernel for the vertex cover problem can only be computed in parallel if the maximum matching problem for bipartite graphs is in NC, for which RNC^2 and $quasi-NC^2$ are the best known upper bounds; that classic reduction rules for feedback vertex set are P-complete (but an exponential kernel can be computed in AC^2); that for the point line cover problem we cannot (absolutely, without any assumptions) compute any kernel in AC^0 (but we can compute a quadratic one in TC^0); and that kernels for generalized versions of Horn satisfiability, linear programming, and maximum flow cannot be computed in polylogarithmic time unless $NC = P$. The later results in fact presents three *natural* FPT-complete problems, which demonstrate the limits of fixed parameter parallelization.

Table 1 summarizes which trade-offs are established in this paper between the parallel time needed to compute kernels and their sizes.

Table 1: An overview of problems studied in this paper, showing which kernel size can be achieved in certain layers of the NC-hierarchy. An explicit function represents the best bound the authors are aware of, pointed out in this work or (for the P-column) in cited works; $f(k)$ corresponds to kernels originating from Theorem 2.3; and “–” means that there is no kernel of any size (either absolutely or unless $TC^0 = L$ for -1 , unless $TC^0 = NL$ for -2 , unless $TC^0 = P$ for -3 , unless $NC = P$ for -4 , unless $P \subseteq RNC$ for -5 , and unless $NC^1 = P$ for -6). For problems parameterized by the vertex cover number, S is the given vertex cover; the δ in the first column can be any fixed positive integer.

Problem	Kernel size achievable in				
	AC^0	TC^0	NC	RNC	P
p -VERTEX-COVER	$2^{\sqrt[\delta]{k}}$	$k^2 + 2k$	$k^2 + 2k$	$2k$	$2k - c \log k$
p -MATCHING	$2^{\sqrt[\delta]{k}}$	$6k^2$	$6k^2$	1	1
p_{vc} -TREE-WIDTH	$2^{\sqrt[\delta]{ S }}$	$ S ^3$	$ S ^3$	$ S ^3$	$ S ^3$
p_{vc} -PATH-WIDTH	$2^{\sqrt[\delta]{ S }}$	$ S ^3$	$ S ^3$	$ S ^3$	$ S ^3$
p_{vc} -TREE-DEPTH	$2^{\sqrt[\delta]{ S }}$	$ S ^3$	$ S ^3$	$ S ^3$	$ S ^3$
p -POINT-LINE-COVER	–	k^2	k^2	k^2	k^2
p -FEEDBACK-VERTEX-SET	–	-1	$f(k)$	$f(k)$	$2k^2 + k$
p -STRONG-BACKDOOR-2CNF-SAT	–	-2	$f(k)$	$f(k)$	$f(k)$
p -STRONG-BACKDOOR-HORN-SAT	–	-3	-3	-5	$f(k)$
p -MIXED-INTEGER-PROGRAMMING	–	-6	-4	-5	$f(k)$
p -MAX-FLOW-QUANTITIES	–	-6	-4	-5	$f(k)$

Related Work. Parameterized complexity is a rapidly growing field, see [14, 15, 18] for an introduction, in which parallelization is a recent research direction. Early research in the late 1990s was done by Cai, Chen, Downey, and Fellows [9] who studied parameterized logarithmic space. A structural study of parameterized logspace and parameterized circuit classes was started around 2015 by Elberfeld et al. [16]; see also the references therein. The parameterized version of the NC-hierarchy we use in this paper was introduced in [2]. Chen and Flum studied lower bounds in this context and especially provide some details and alternative characterizations for parameterized AC^0 . There is a huge body of literature on polynomial-time algorithms for computing small kernels, but the authors are not aware of results concerning how quickly these kernels can be computed in parallel.

Organization of This Paper. We review basic terminology in Section 2, where we also establish the link between parameterized parallel complexity and parallel kernel computation. Each of the following sections studies a different well-known parameterized problem and establishes trade-offs between kernel size and speed. We start with the vertex cover and the matching problem in Section 3, followed by the

feedback vertex set problem in Section 4, structural parameterizations for tree width, path width, and tree depth in Section 5, the p -POINT-LINE-COVER problem in Section 6, and finally generalized versions of Horn satisfiability, linear programming, and maximum flow in Section 7.

2 Parameterized Parallel Complexity Classes and Kernelization

We use standard terminology of parameterized complexity theory, see for instance [18]. A *parameterized problem* is a tuple (Q, κ) consisting of a *language* $Q \subseteq \Sigma^*$ and a *parameterization* $\kappa: \Sigma^* \rightarrow \mathbb{N}$. The complexity of κ should not exceed the power of the classes that we consider, and since we study small parameterized circuit classes, we require κ to be computable by DLOGTIME-uniform constant-depth AC-circuits or, equivalently, to be first-order computable. We denote parameterized problems by a leading “ p –” as in p -VERTEX-COVER, and, whenever the parameterization κ is not clear from the context, we add it as an index as in p_{vc} -TREE-WIDTH. A parameterized problem (Q, κ) is *fixed-parameter tractable* (or in FPT) if there is a computable function $f: \mathbb{N} \rightarrow \mathbb{N}$ and a constant c such that we can decide $x \in Q$ in time $f(\kappa(x)) \cdot |x|^c$ for all $x \in \Sigma^*$. In this paper we study the parallel complexity of parameterized problems, that is, the parameterized counter part of the NC-hierarchy. Formally we study the following classes, see for instance [2, 11] for a detailed discussion:

Definition 2.1. *For each $i > 0$, a parameterized problem (Q, κ) is in DLOGTIME-uniform para-AC ^{i} if there exists a computable function $f: \mathbb{N} \rightarrow \mathbb{N}$, a constant $c \in \mathbb{N}$, and a family of AC-circuits $(C_{n,k})_{n,k \in \mathbb{N}}$ such that:*

1. *For all $x \in \Sigma^*$ we have $C_{|x|, \kappa(x)}(x) = 1 \iff x \in Q$.*
2. *The depth of each $C_{n,k}$ is at most $f(k) + c \log^i n$.*
3. *The size of each $C_{n,k}$ is at most $f(k) \cdot n^c$.*
4. *There is a deterministic Turing machine that on input of $\text{bin}(i) \# \text{bin}(k) \# \text{bin}(n)$, where $\text{bin}(x)$ is the binary encoding of x , outputs the i th bit of a suitable encoding of $C_{n,k}$ in at most $f(k) + c \log n$ steps.*

The class para-AC⁰ is defined as above, but with circuits of *constant depth*. Additionally, we define for all $i \geq 0$ the class para-AC ^{i} \uparrow with circuits of depth $f(k) \cdot \log^i n$. In particular, para-AC⁰ \uparrow -circuits have depth $f(k)$. Recall that AC-circuits are defined over the standard base of NOT-, OR-, and AND-gates and that the last two may have unlimited fan-in. The same definition works for NC-circuits (all gates have bounded fan-in) and TC-circuits (additional threshold gates are allowed). It is known that the parameterized classes inherit their inclusion structure from their classical counterparts [2]:

$$\text{para-AC}^0 \subsetneq \text{para-TC}^0 \subseteq \text{para-NC}^1 \subseteq \text{para-AC}^1 \subseteq \text{para-TC}^1 \subseteq \dots \subseteq \text{para-NC} \subseteq \text{FPT}.$$

A Parallel Analogue of “FPT = Kernels Computable in Polynomial Time”. One of the most fruitful aspects of parameterized complexity is the concept of *kernelization*. Let $f: \mathbb{N} \rightarrow \mathbb{N}$ be a computable function. A *kernelization* of a parameterized problem (Q, κ) is a self-reduction $K: \Sigma^* \rightarrow \Sigma^*$ such that for every $x \in \Sigma^*$ we have $x \in Q \iff K(x) \in Q$ and $|K(x)| \leq f(\kappa(x))$. The images of K are called *kernels* and as they later need to be processed by at least exponential-time algorithms, we are interested in kernels that are as small as possible – while they still need to be efficiently computable, meaning in polynomial time from the view point of FPT theory. The following result is well-known and gives a deep connection between parameterized complexity and kernelization:

Fact 2.2 (for instance [18]). *A decidable parameterized problem (Q, κ) is in FPT if, and only if, it admits a polynomial-time computable kernelization.*

The following theorem shows that the same relation also connects the AC-hierarchy with its parameterized counterpart. Note that in the theorem the AC ^{i} -circuits are really “normal AC ^{i} -circuits,” meaning that their size is just polynomial in the input length.

Theorem 2.3. *A decidable parameterized problem (Q, κ) is in para-AC^i if, and only if, it admits a kernelization computable by a DLOGTIME-uniform family of AC^i -circuits.*

Proof. Let $f: \mathbb{N} \rightarrow \mathbb{N}$ be a computable function and $c \in \mathbb{N}$ be a constant as in Definition 2.1, let furthermore (Q, κ) be a parameterized problem. Assume for the first direction that a kernelization K of (Q, κ) can be computed by a DLOGTIME-uniform family $(C_n)_{n \in \mathbb{N}}$ of AC^i -circuits. Then we construct a family $(C_{n,k})_{n,k \in \mathbb{N}}$ of para-AC^i -circuits as follows: Circuit $C_{n,k}$ uses C_n as a first black box, which is possible due to the depth and size definitions, and reduces the input to an instance of size at most $f(\kappa(x))$. Then the circuit essentially applies naive “brute force” in the form of a big OR-gate that checks if any element of Q of length at most $f(\kappa(x))$ equals the computed kernel (we need the decidability of Q at this point to ensure that the circuit family is uniform).

For the other direction let us assume $(Q, \kappa) \in \text{para-AC}^i$ witnessed by a DLOGTIME-uniform family $(C_{n,k})_{n,k \in \mathbb{N}}$ of para-AC^i -circuits, and let us first assume $i > 0$. We may further assume that the computable function f used in the definition of $(C_{n,k})_{n,k \in \mathbb{N}}$ is a monotone increasing function with $f(x) > x$ for all $x \in \mathbb{N}$, and that there is a Turing machine M_f that computes $f(x)$ on input $\text{bin}(x)$ in time $O(\log f(x))$. To see this, observe that, since f is computable, there is a Turing machine M'_f that computes $f(x)$ on input $\text{bin}(x)$ in some time $T(x)$ such that T is a monotone increasing function and such that $T(x) > x$ for all $x \in \mathbb{N}$. We may replace f with $g(x) = 2^{T(x)}$, which fulfils the above requirements. The resulting family $(C'_{n,k})_{n,k \in \mathbb{N}}$ is still a family of para-AC^i -circuits that accepts the same language.

For every $n \in \mathbb{N}$ we define $\tilde{k} \in \mathbb{N}$ to be the maximum k such that $f(k) \leq c \log^i n$. We will use \tilde{k} in the following construction and, hence, \tilde{k} must be computable by a Turing machine in time $O(\log n)$ on input $\text{bin}(n)$ to ensure uniformity. This is the case, as an appropriate Turing machine can first compute the value $c \log^i n$ (this is possible since $\log n$ is a $\log \log n$ -bit number, i is a constant, and $c \log^i n$ is thus a $2^i \log \log n \in O(\log \log n)$ -bit number) and can then perform binary search to find \tilde{k} . The later is possible since f is monotone increasing and since $\tilde{k} \leq c \log^i n$ as we have $f(x) > x$. Therefore, the Turing machine has to test only $\log(c \log^i n) \in O(\log \log n)$ possible k . Finally, for a fixed $k \leq c \log^i n$ the Turing machine can simulate M_f on input $\text{bin}(k)$ for $\log(c \log^i n) \in O(\log \log n)$ steps and either obtains the value $f(k)$ or, if M_f does not finish, can conclude that $f(k) > c \log^i n$.

We now construct a family of AC^i -circuits that compute a kernelization of (Q, κ) . Each circuit C_n consists of \tilde{k} subcircuits $C_n^0, \dots, C_n^{\tilde{k}}$ that are evaluated in parallel. The circuit C_n^j first checks on input x whether or not $\kappa(x) = j$, which is possible since κ can be computed by AC^0 -circuits (by definition). If this test is affirmative, the circuit uses $C_{n,j}$ to solve the problem and outputs a trivial kernel, that is, a trivial yes- or no-instance of Q . Otherwise C_n^j just sets a flag that states that it is not responsible for this instance. Note that there is a constant c' such that $C_{n,j}$ has, by definition, depth at most $f(j) + c \log^i n \leq (c+1) \log^i n \leq c' \log^i n$ and size at most $f(j) \cdot n^c \leq c \log^i n \cdot n^c \leq n^{c'}$. If any C_n^i produces a kernel, then C_n just presents this kernel as result. If, otherwise, all C_n^j state that they are not responsible, we have $\kappa(x) > \tilde{k}$ and $f(\kappa(x)) > c \log^i n$ and, thus, we already deal with a kernel, that is, C_n can just present the input as output.

For the remaining case, $i = 0$, we perform the same construction, but choose \tilde{k} such that $f(\tilde{k}) \leq n^c$, that is, we bound the subcircuits by size and not by depth. \square

The theorem also holds if we replace AC^i with NC^i or TC^i . The only exception is NC^0 , as this class may not be powerful enough to compute κ .

Application: Improve the Work of Parallel Algorithms. When we study the performance of parallel algorithms, we usually do not only measure the time of the algorithm (as we would in the sequential case), but also its *work* (the total number of computational steps performed by the algorithm). This is important as a parallel algorithm may need polynomially many processors to reach its promised runtime: For instance, an algorithm that runs in time $O(\log n)$ with $O(n^2)$ work will need at least time $O(n^2/p)$ on a machine with p processors – which is bad if there exists a linear time sequential algorithm and $p < n$. In the circuit model the parallel time of an algorithm corresponds to the depth of the circuit, and the work to its size. While the layers of the AC - and para-AC -hierarchy measure the time of parallel algorithms quite precisely, they only require the size of the circuits to be polynomial or to be bounded by $f(k) \cdot n^c$, respectively. Using Theorem 2.3, we can improve the work of any parameterized parallel algorithm from $f(k) \cdot n^c$ to $g(k) + n^{c'}$ while, at the same time, reducing the depth of the circuit from $f(k) + c \log^i n$ to $c' \log^i n$.

Lemma 2.4. *Let (Q, κ) be a parameterized problem with $(Q, \kappa) \in \text{para-AC}^i$. Then there are a computable function $g: \mathbb{N} \rightarrow \mathbb{N}$ and a constant c' such that there is a DLOGTIME-uniform family $(C'_{n,k})_{n,k \in \mathbb{N}}$ of para-ACⁱ-circuits that decides (Q, κ) and in which every $C'_{n,k}$ has depth at most $c' \log^i n$ and size at most $g(k) + n^{c'}$.*

Proof. Since $(Q, \kappa) \in \text{para-AC}^i$, there is a DLOGTIME-uniform family $(C_{n,k})_{n,k \in \mathbb{N}}$ of para-ACⁱ-circuits that decides (Q, κ) . Let $f: \mathbb{N} \rightarrow \mathbb{N}$ and $c \in \mathbb{N}$ be as in Definition 2.1. By Theorem 2.3 there is a constant c' and a DLOGTIME-uniform family $(C_n)_{n \in \mathbb{N}}$ of ACⁱ-circuits such that every C_n has depth at most $c' \log^i n$ and size at most $n^{c'}$ and produces a kernel of size at most $f(\kappa(x))$. We construct the desired family $(C'_{n,k})_{n,k \in \mathbb{N}}$ as follows: The circuit $C'_{n,k}$ first applies the circuit C_n to an input x and obtains an instance x' of size at most $f(\kappa(x))$, then the circuit uses a constant number of AC layers to check $x' \in Q$ by testing in parallel for all $w \in Q$ with $|w| \leq f(\kappa(x))$ whether $w = x'$ holds.

Therefore, the depth of $C'_{n,k}$ equals (up to a constant) the depth of C_n , and the size of $C'_{n,k}$ is the sum of the size of C_n and the size of the “brute force” circuit applied at the end, that is, there is a computable function $g: \mathbb{N} \rightarrow \mathbb{N}$ such that size of $C'_{n,k}$ can be bounded by $g(\kappa(x)) + n^{c'}$. \square

Note that the function g from the lemma may grow exponentially faster than f , as the circuit from the lemma internally solves an instance x' with $|x'| \leq f(\kappa(x))$ and $\kappa(x') \leq f(\kappa(x))$. A direct application of Lemma 2.4 is therefore only of theoretical interest. It shows, however, that we can always search for parameterized parallel algorithms that run in polylogarithmic time and whose work is polynomial plus an *additive* term depending only on the parameter.

3 Parallel Kernels for Vertex Cover and Matching

The parameterized vertex cover problem is a prime example used to demonstrate many different kernelization techniques, and an outrider in the race for small kernels. In this section we revisit the problem from the point of view of circuit complexity and establish a link between circuit complexity and kernel size. An early result in this context is due to Cai et al. [9] which, translated into the terminology of the present paper, implies that a kernel for p -VERTEX-COVER can be computed in logarithmic space and, hence, in AC¹. Elberfeld et al. [16] later noticed that the kernel of size $k^2 + 2k$ computed by Cai et al. can actually also be computed in TC⁰. This result was later once more refined by showing that the same kernel can be computed in para-AC⁰ [2]. Together with Theorem 2.3 this implies that a kernel of size $f(k)$ can be computed in AC⁰ for some computable function f . In fact, we can improve the bound in this case to $2^{\delta \sqrt{k}}$ for any fixed $\delta > 0$:

Lemma 3.1. *For every $\delta \in \mathbb{N}$ there is a DLOGTIME-uniform family of AC⁰-circuits that, on input of a tuple (G, k) , outputs a p -VERTEX-COVER kernel with at most $2^{\delta \sqrt{k}}$ vertices.*

Proof. Let I be the input instance and let $n = |I|$ be the size of its encoding. The circuit first checks if we have $k \leq \log^\delta(n)$. If not, we have $2^{\delta \sqrt{k}} > n$ and the instance is already the desired kernel. Otherwise the circuit can simulate threshold gates up to k using standard hashing techniques, as AC⁰-circuits can simulate polylogarithmic threshold gates [33]. Since the TC⁰-circuit from Elberfeld et al. [16] only uses threshold gates up to k , it follows that the AC⁰-circuit under construction can simulate this TC⁰-circuit, which completes the proof. \square

The central observation in the proof of Lemma 3.1 is that the threshold-gates in the corresponding family of TC⁰-circuits only “count up to the parameter.” We will use exactly the same trick for other TC⁰-kernelizations, but will then only formulate it as corollary. Summarizing the statements from above, we can compute an exponential kernel for p -VERTEX-COVER in AC⁰ and a quadratic kernel in TC⁰. However, the best known kernelizations for p -VERTEX-COVER are able to produce *linear kernels* – and a reasonable next step is to implement them in parallel as well. Unfortunately, this is a way more challenging task, as both the classical $3k$ kernel based on crown decomposition [14] and the $2k$ kernel due to Chen et al. [10] require the computation of sufficiently large matchings. We can state this more precisely for the latter observation, by showing that the core part of the kernelization is NC-equivalent to computing maximum matchings in bipartite graphs. The kernelization of Chen et al. is based on the following fact, known as the Nemhauser–Trotter Theorem:

Fact 3.2 ([32]). Let $G = (V, E)$ be a graph and $I = \{x_v \mid v \in V\}$ be a set of variables. For every optimal solution $\beta: I \rightarrow \mathbb{R}$ for the following linear program (LPVC)

$$\begin{aligned} \min \quad & \sum_{v \in V} x_v \\ \text{s.t.} \quad & x_u + x_v \geq 1 \quad \text{for all } \{u, v\} \in E \\ & x_v \geq 0 \quad \text{for all } v \in V \end{aligned}$$

let $V_0 = \{v \mid \beta(x_v) < 1/2\}$, $V_{1/2} = \{v \mid \beta(x_v) = 1/2\}$, $V_1 = \{v \mid \beta(x_v) > 1/2\}$ be a partition of V . There is a minimum vertex cover S of G that satisfies $V_1 \subseteq S \subseteq V_1 \cup V_{1/2}$.

Chen et al. have shown that one can obtain the desired kernel from a solution of LPVC by discarding the vertices of V_0 and by taking the vertices of V_1 into the solution. The remaining $2k$ vertices of $V_{1/2}$ constitute the kernel [10]. The following theorem shows that solving LPVC is tightly linked to the maximum matching problems for bipartite graphs.

Theorem 3.3. *Computing a solution for LPVC is NC-equivalent to computing a maximum matching in bipartite graphs.*

Proof. The first direction is essentially the standard way of efficiently solving LPVC: Given an instance of LPVC we construct a bipartite graph $H = (\{v_1, v_2 \mid v \in V\}, \{\{u_1, v_2\}, \{u_2, v_1\} \mid \{u, v\} \in E\})$ and compute a minimum vertex cover S of it. One can show that the following assignment is an optimal solution for LPVC [14]:

$$\beta(x_v) = \begin{cases} 0 & \text{for } |\{v_1, v_2\} \cap S| = 0, \\ 1/2 & \text{for } |\{v_1, v_2\} \cap S| = 1, \text{ and} \\ 1 & \text{for } |\{v_1, v_2\} \cap S| = 2. \end{cases}$$

Since H is bipartite, computing a minimum vertex cover is equivalent to computing a maximum matching due to König's Theorem [28]. More precisely: To obtain the vertex cover S , we compute a maximum matching in H and this matching constitutes an optimal solution to the dual program of LPVC. Due to the Complementary Slackness Theorem, we can derive an optimal solution for the primal program from an optimal solution of the dual program by solving a linear system of equations, which is possible in NC [25]. Note that the matrices of both LPVC and its dual are totally unimodular, as the incidence matrix of a bipartite graph is totally unimodular, and since the transpose of a totally unimodular matrix is so as well. Therefore, Cramer's Rule states that the solution that we obtain for the dual program with the algorithm from above is integral as well. This completes this part of the proof.

For the other direction the input is a bipartite graph $G = (V, E)$ in which we search for a maximum matching. Let β be an optimal real solution of LPVC for G . We can transform β into a (still optimal) half-integral solution β' by simple rounding:

$$\beta'(x_v) = \begin{cases} 0 & \text{if } \beta(x_v) < 1/2, \\ 1/2 & \text{if } \beta(x_v) = 1/2, \text{ and} \\ 1 & \text{if } \beta(x_v) > 1/2. \end{cases}$$

This well-known fact is based on [32], and can be shown by the following procedure that successively transforms β into refined optimal solutions, ending at β' . To refine β we define the sets $V_+ = \{x_v \mid 0 < \beta(x_v) < 1/2\}$ and $V_- = \{x_v \mid 1/2 < \beta(x_v) < 1\}$. We now define for a suitable small $\epsilon > 0$ the two assignments

$$\beta_+(x_v) = \begin{cases} \beta(x_v) & \text{if } x_v \notin V_+ \cup V_-, \\ \beta(x_v) + \epsilon & \text{if } x_v \in V_+, \text{ and} \\ \beta(x_v) - \epsilon & \text{if } x_v \in V_-, \end{cases} \quad \text{and} \quad \beta_-(x_v) = \begin{cases} \beta(x_v) & \text{if } x_v \notin V_+ \cup V_-, \\ \beta(x_v) - \epsilon & \text{if } x_v \in V_+, \text{ and} \\ \beta(x_v) + \epsilon & \text{if } x_v \in V_-. \end{cases}$$

Observe that both, β_+ and β_- , are still feasible solutions, as for any edge $\{u, v\}$ the constraint $x_u + x_v \geq 1$ is still satisfied (either one of the variables is already 1, or they are both 1/2, or we add ϵ to at least one of them). Further observe that, compared to β , the value of the target function changes by $\epsilon|V_+| - \epsilon|V_-|$ and $\epsilon|V_-| - \epsilon|V_+|$, respectively. Since β is optimal, neither β_+ nor β_- may reduce the value of the target function compared to β ; consequently we have $|V_+| = |V_-|$ and β_+ and β_- are both optimal solutions. Conclusively observe that, by repeating this process successively, we will end up at β' .

To conclude this part of the proof, we will now turn β' into an integral solution. To achieve this, we construct an auxiliary graph G' by deleting all vertices with value 1 in G (as these must be in the vertex cover). Since all vertices with value 0 are now isolated, we may remove them too. We end up with a bipartite graph G' with n' vertices, which are all assigned with the value $1/2$ by β' . We claim β' is an optimal solution for LPVC on G' . For a contradiction assume otherwise, that is, assume there is an assignment γ with $\sum_{v \in V(G')} \gamma(x_v) < \sum_{v \in V(G')} \beta'(x_v)$. We can infer a new assignment β'' for G by “plugging” γ into β' :

$$\beta''(x_v) = \begin{cases} \beta'(x_v) & \text{if } x_v \notin V(G'); \\ \gamma(x_v) & \text{if } x_v \in V(G'). \end{cases}$$

Observe that this is a feasible solution for LPVC on G , since for all edges $\{u, v\}$ we have:

$$\beta''(x_u) + \beta''(x_v) = \begin{cases} \gamma(x_u) + \gamma(x_v) \geq 1 & \text{if } u, v \in V(G'); \\ \beta'(x_u) + \beta'(x_v) \geq 1 & \text{if } u, v \notin V(G'); \\ \beta'(x_u) + \gamma(x_v) \geq 1 & \text{if } u \notin V(G') \text{ and } v \in V(G'). \end{cases}$$

The first two lines follow by the fact that γ and β' are feasible; the last line follows by the construction of G' , as an edge $\{u, v\}$ with $u \notin V(G')$ and $v \in V(G')$ only appears if we have $\beta'(x_u) = 1$ (we have only deleted isolated vertices and vertices with value 1, and here u was deleted and is not isolated). By the construction of β'' , we end up with $\sum_{v \in V(G)} \beta''(x_v) < \sum_{v \in V(G)} \beta'(x_v)$, which is a contradiction as β' is an optimal solution for LPVC on G . Consequently, β' must be an optimal solution for LPVC on G' as well.

Since β' assigns $1/2$ to all vertices in G' , a minimal vertex cover of G' has size at least $n'/2$. Therefore, G' has to consist of two equally sized shores, as otherwise the smaller one would be a vertex cover of size smaller than $n'/2$. We can, thus, greedily select one shore into the vertex cover, that is, we set β' for one shore to 1 and for the other to 0. The obtained optimal integral solution of LPVC can be turned, as in the first direction, into a solution for the dual program in NC, i. e., into a maximum matching of G . \square

The parallel complexity of the maximum matching problem is still not fully resolved. The currently best parallel algorithms run in RNC² [31] or quasi-NC² [17]. From the theorem we can deduce that we can compute the Nemhauser–Trotter-based $2k$ -vertex kernel for p -VERTEX-COVER in RNC and quasi-NC; and we can deduce that we cannot compute this kernel in NC without improving the parallel complexity of the maximum matching problem – which is a longstanding open problem.

Corollary 3.4. *There is a DLOGTIME-uniform family of NC-circuits of polylogarithmic depth that, on input of a graph $G = (V, E)$ and an integer k , outputs a kernel of p -VERTEX-COVER with at most $2k$ vertices. The circuits of the family either use randomness and have size $|V|^c$, or are deterministic and of size $|V|^{c \log |V|}$.*

Note that other kernels that are based on the Nemhauser–Trotter Theorem, such as the one by Soleimanfallah and Yeo [34], or the one by Lampis [30], also do not bypass Theorem 3.3. A natural goal is, thus, to compute linear kernels for p -VERTEX-COVER in NC – most likely using an algorithm that does not rely on a LPVC relaxation. Table 1 summarizes the complexity of computing kernels of certain size for p -VERTEX-COVER.

Since p -MATCHING turns out to be an obstruction for parallel kernelization, it is a natural question in the light of this paper, whether or not we are able to compute polynomial kernels for the matching problem in NC. Note that the problem is in para-AC⁰, and hence we can compute a size- $f(k)$ kernel in AC⁰; and since MATCHING \in RNC we can compute a size-1 kernel in RNC.

Lemma 3.5. *There is a DLOGTIME-uniform family of TC⁰-circuits that, on input of a tuple (G, k) , outputs a p -MATCHING kernel with at most $O(k^2)$ vertices.*

Proof. The circuit first computes a set $S = \{v \in V \mid |N(v)| > 2k\}$ of “high-degree” vertices. If we have $|S| \geq k$, the circuit can output a trivial yes-instance since for such a set S we can greedily match any vertex $v \in S$ with a vertex $u \in N(v) \setminus S$, reducing the available matching mates of all other vertices in S by at most two – and since they have degree at least $2k$, there are still enough mates left to match every vertex of S .

If the circuit has not finished yet, we compute a set S' consisting of S and $2k$ arbitrary neighbors of every vertex in S (take the lexicographically first for each $v \in S$, for instance). Note that we have

$|S'| \leq 2k^2$. Consider the graph $G' = G[V \setminus S']$. Since S was the set of high-degree vertices, G' has maximum-degree $d \leq 2k$. Our circuit now removes all isolated vertices from G' , resulting in G'' , and then checks if we have $|V(G'')| \geq k \cdot 2d$. If so, we can output a trivial yes-instance since a graph with maximum degree d and minimum degree 1 always contains a matching of size $|V(G'')|/2d \geq k$. If, on the other hand, we have $|V(G'')| \leq k \cdot 2d \leq 4k^2$, the circuit outputs $G[S' \cup V(G'')]$ together with the unchanged number k .

The output clearly always has size at most $O(k^2)$. To see that $G[S' \cup V(G'')]$ is a kernel, we clearly only have to show that if G has a size- k matching M , so does $G[S' \cup V(G'')]$ (the other direction is trivial). To see this, first note that any edge in M that does not have an endpoint in S must lie in G'' and, hence, is also present in $G[S' \cup V(G'')]$. Next, all other edges in M must have an endpoint in S and, thus, there can be at most $|S|$ many such edges. While not all of these edges need to be present in $G[S']$, we can greedily construct a matching of size $|S|$ in $G[S']$ (by the same argument as the one of the beginning of this proof for $|S| \geq k$). This means that we find a matching of size $|M|$ also in $G[S' \cup V(G'')]$. \square

The circuits of Lemma 3.5 need their threshold gates “only” to count up to k . We can thus deduce the following corollary (the proof argument is the same as for Lemma 3.1):

Corollary 3.6. *For every $\delta \in \mathbb{N}$ there is a DLOGTIME-uniform family of AC^0 -circuits that, on input of a tuple (G, k) , outputs a p -MATCHING kernel with at most $O(2^{\delta\sqrt{k}})$ vertices.*

4 Parallel Kernels for the Feedback Vertex Set Problem

The input for p -FEEDBACK-VERTEX-SET = p -FVS is an undirected multigraph $G = (V, E)$ and an integer k , the question is whether it is possible to delete k vertices such that the remaining graph is a forest. The problem is well-known to be fixed-parameter tractable. Concerning the parallel complexity, it is known that membership in FPT can be witnessed by a machine that uses “FPT time and XL space” [16] and the problem was recently shown to lie in $\text{para-NC}^{2+\epsilon} \subseteq \text{para-NC}^3$ [3].

A lot of effort has been put into the design of sequential kernels for this problem, ultimately resulting in a kernel with $O(k^2)$ vertices [8, 7, 36, 24]. Much less is known concerning parallel kernels. Since the $k = 0$ slice of p -FVS is exactly the L-complete [13] problem whether a given graph is a forest, we get as a lower bound that no kernel of any size can be computed for p -FVS by any circuit class C unless $L \subseteq C$ and the smallest AC-class for which this is known is AC^1 . On the other hand, the mentioned membership in $\text{para-NC}^{2+\epsilon}$ together with Theorem 2.3 yield an $\text{NC}^{2+\epsilon}$ kernel. In summary:

Lemma 4.1. *There is a DLOGTIME-uniform family of $\text{NC}^{2+\epsilon}$ -circuits that, on input of a tuple (G, k) , outputs a p -FVS kernel with at most $f(k)$ vertices. There is no such family of $\text{AC}^{1-\epsilon}$ -circuits, unless $L \subseteq \text{AC}^{1-\epsilon}$.*

A natural first question arising from this lemma is: Can we improve the bounds? It turns out that we can lower the upper bound from $\text{NC}^{2+\epsilon}$ to $\text{AC}^{1+\epsilon}$ by observing the reduction rules used in sequential kernels for p -FVS can, in certain cases, be applied in parallel. In detail, the known sequential kernels for p -FVS all repeatedly apply (at least) the below rules, whose correctness is very easily seen. We will show that each of the first three rules can individually be applied exhaustively in AC^1 . Based on this, we show p -FVS \in $\text{para-AC}^{1\uparrow}$.

Leaf Rule Delete a vertex v of degree 1.

Chain Rule Contract a vertex v of degree 2 to one of its neighbors.

Loop Rule Delete a vertex v with $v \in N(v)$, reduce k by 1.

Flower Rule Delete a vertex v that appears in more than k cycles that only share the vertex v , reduce k by 1.

Lemma 4.2. *There is a DLOGTIME-uniform family of AC^1 -circuits that, on input of a tuple (G, k) , outputs a tuple (G', k') that results from repeatedly applying (only) the Leaf Rule as long as possible. The same holds for the Chain Rule and for the Loop Rule.*

Proof. The claim follows immediately for the Loop Rule as we may delete all such vertices in parallel and since the deletion of a vertex cannot create new vertices with a self-loop. For the other two rules observe that an “exhaustive application” equals either the deletion of attached trees (for the Leaf Rule), or the contraction of induced paths (for the Chain Rule). For the first case, the circuit must be able to detect if a vertex v becomes a leaf at some point of the computation (of course, the circuit cannot sequentially delete degree one vertices). The following observation provides a locally testable property that allows precisely such a detection: A vertex v is contained in an attached tree if, and only if, it is possible to delete a single edge such that (a) the graph decomposes into two components and such that (b) the component of v is a tree [16]. Both properties can be tested in logspace (and hence in AC^1), and an AC^1 -circuit can test them for all vertices and all edges in parallel. Finally, for the Chain Rule, observe that an AC^1 -circuit can mark all degree two vertices in parallel and that such a circuit, afterwards, only has to connect the two endpoints of highlighted paths – which is again a logspace task. \square

Theorem 4.3. $p\text{-FVS} \in \text{para-}AC^{1\uparrow}$.

Proof. We have to construct a family of AC -circuits of depth $f(k) \cdot \log n$ and size $f(k) \cdot n^c$. The circuits will consist of k layers such that every layer finds a set of at most $3k$ vertices to branch on (which will be done for the next layer). Note that layer i contains at most $3k$ as many subcircuits as layer $i - 1$.

Each layer consists of multiple AC^1 -circuits that work independently of each other on different possible graphs (depending on the branches of the previous layer). Each of these circuits first checks if the input is a yes-instance (input is a tree and $k \geq 0$), or a no-instance ($k < 0$) – in the first case it just globally signals this circumstance and in the second case it truncates this path of the computation. If the subcircuit has not decided yet, it applies first the Leaf Rule exhaustively, and then Chain Rule exhaustively – both are possible due to Lemma 4.2. The circuit now applies the Loop Rule (again, using Lemma 4.2), if the rule has an effect (that is, k was reduced by at least one) the circuit is done and just pipes the result to the next layer. If not, the circuit tests in parallel if there are two vertices v and u that are connected by a multi-edge (that is, by at least two edges). If this is the case, any feedback vertex set must contain either v or u and, hence, the circuit branches on these two vertices and pipes the two resulting graphs to the next layer. Otherwise, we know that we have no vertex with a self-loop, no vertices with multi-edges, and a minimum vertex-degree of at least three. The circuit then uses the simple fact that any size k feedback vertex set in such graph must contain at least one vertex of the $3k$ vertices of highest degree, and hence, may simply branch over these [14].

Since each layer reduced k in each branch by at least one, after at most k layers every branch has decided if it deals with a yes- or a no-instance. Since each layer is implemented by an AC^1 -circuit, the claim follows. \square

Corollary 4.4. *There is a DLOGTIME-uniform family of $AC^{1+\epsilon}$ -circuits that, on input of a tuple (G, k) , outputs a $p\text{-FVS}$ kernel with at most $f(k)$ vertices.*

Proof. Follows by Theorem 2.3 and by the fact that $\text{para-}AC^{i\uparrow} \subseteq \text{para-}AC^{i+\epsilon}$ [2]. \square

We now have rather tight bounds (an upper bound of $AC^{1+\epsilon}$ and a conditional lower bound of $AC^{1-\epsilon}$) on how quickly we can compute *some* kernel for $p\text{-FVS}$ in parallel. However, there is a natural second question arising from Lemma 4.1: Can we also compute a polynomial kernel in parallel?

We claim that progress towards such a kernel cannot solely be based on the presented reduction rules. In the proof of Theorem 4.3 we may need to branch after the exhaustive application of one of the rules Leaf Rule, Chain Rule, or Loop Rule. If we seek to implement a polynomial kernel for $p\text{-FVS}$ in NC , we have to implement these rules without branching and have to apply the rules exhaustively *together* while they may influence each other. Figure 1 provides an intuition why this interplay is “very sequential,” and Theorem 4.5 provides evidence that it is in fact very unlikely that there exists a parallel algorithm that computes the result of jointly applying all rules exhaustively.



Figure 1: A graph that is fully reduced by the Chain Rule and the Loop Rule in $k = 6$ rounds. In every round, the Chain Rule can only be applied after the Loop Rule was used exhaustively.

Theorem 4.5. *The problem of deciding whether a specific vertex of a given graph will be removed by an exhaustive application of the Leaf Rule, the Chain Rule, and the Loop Rule is P-hard under NC^1 -reduction.*

Proof. We will reduce from the monotone circuit value problem (MCVP), which is known to be P-complete under NC^1 -reduction [21]. The input to this problem is a monotone circuit (it consists only of AND-gates and OR-gates of indegree 2, and it has a single gate marked as output) and an assignment of the input gates, the question is whether or not the output gate evaluates to true. We will transform the input circuit into a multi-graph by replacing any gate with a small gadget. Every gadget will have two vertices marked as “input” and one marked as “output”. The “input” vertices are incident to exactly one edge outside of the gadget (which connects them to the “output” vertex of another gadget), the “output” vertex of the gadget may have edges to an arbitrary number of other “input” vertices. The semantic then is as follows: The edge of an “input” vertex that leaves the gadget will be removed by the reduction rules when the corresponding wire of the circuit would have the value true for the given assignment of the input gates; similarly the “output” vertex of the gadget will be removed if, and only if, the corresponding gate would evaluate to true under the given assignment (this in turn removes the edges to other “input” vertices and propagates the computation of the circuit). By induction, the “output” vertex of the gadget corresponding to the output gate will then be removed if, and only if, the circuit evaluates to true – which completes the proof.

We start with a description of the transformation. For clarity, we stipulate that a self-loop contributes two to the degree of a vertex, similarly multi-edges increase the degree by their multiplicity. Therefore, the Chain Rule may not be applied to a leaf with a self-loop. We further stipulate that the Chain Rule may not be applied to a self-loop, i. e., it has to contract two distinct vertices (and hence, self-loops may only be handled by the Loop Rule). For the input gates, we use the gadgets \downarrow and \boxtimes for gates with assignment true or false, respectively. For AND-gates, we use the gadget , and for OR-gates . In these figures, the two top circled vertices are the ones we call “input”, while the bottom vertex is the “output” vertex. The dotted lines indicate edges that leave the gadget. For every “input” vertex there will be exactly one outgoing edge, as any gate has exactly two incoming wires. The “output” vertex may have edges to an arbitrary number of successor gates; to ensure that there is at least some edge, we fully connect such vertices to cliques of size three (that is, the “output” vertex is part of a clique of size four) – this ensures that the degree of “output” vertices is always greater than two.

We first prove that these gadgets work locally as intended, that is, that they perform as an input gate that is set to true or false, or as an AND- or OR-gate. Observe that the gadget for input gates that are true contains a self-loop, that is, it gets removed by the Loop Rule; and observe that the gadget for a false input gate is a clique to which no rule can be applied. Now for the internal gates, observe that all vertices have degree at least three and no self-loop, that is, no rule can be applied unless one “incoming” edge gets removed (recall that each “output” vertex is fully connected to a clique of size three). In other words, the gadgets simulate the corresponding gates correctly for the assignment “(false, false)”, a case distinction shows that this is also the case for the other assignments:

Assignment	Behaviour of the	
	AND-gadget	OR-gadget
(true, false)	 \mapsto 	 \mapsto 
(false, true)	 \mapsto 	 \mapsto 
(true, true)	 \mapsto 	 \mapsto 

Observe that the “output” vertex obtains a self-loop (and hence gets removed by the Loop Rule) if, and only if, the corresponding gate evaluates to true; note that the degree of a “output” vertex is always greater than two and, hence, it is never affected by the Leaf Rule or the Chain Rule.

We now show the correctness of the construction by an induction over the gates of the input circuit in topological order. The induction hypothesis is that the gadget corresponding to the current gate gets modified by the Leaf Rule, the Loop Rule and the Chain Rule in the same way as the gate gets evaluated. The base case is given as this is true for the input gates by construction. For the inductive step consider the gadget corresponding to any gate g , and let it have the vertices x , y , and z where z is the “output” vertex. By the induction hypothesis the vertices marked as “input” (i. e., x and y) lose an incident edge for input wires that evaluate to true (as the gates corresponding to these gadgets precede g in the topological order), the above table then states that the gadget works correctly. The only pitfall we need to address is that the simulation does not “work backwards”, that is, that reduction rules in g trigger a reduction rule for the “output” vertex v of a gadget that corresponds to a gate that precedes g in the topological order. The only way in which the described scenario appears is when v obtains a self-loop (as v is an “output” vertex it is connected to a clique of size three and, hence, the only rule that can delete v is the Loop Rule). The only way to generate a self-loop is to contract a cycle of degree-2 vertices to v . Without loss of generality, we may assume that v is connected to exactly one of x and y and we may assume that it is x . Therefore, to generate a self-loop on v the reduction rules have to modify the gadget such that x has exactly two incident edges which both are connected to the vertex v – the case distinction in the table shows that this can not happen.

The induction completes the proof. A full example of the construction is provided in Figure 2. It is worth mentioning that the construction almost never generates vertices to which the Leaf Rule can be applied, the sole exception are OR-gates in which only one input is set to true (after the deletion of the “output” vertex, the second input vertex becomes a leaf). However, in this case the application of the Leaf Rule has no effect on the behaviour of the gadgets. Also note that the simulation still works if we alternately apply the Chain Rule and the Loop Rule exhaustively (i. e., exhaustively apply the Chain Rule, exhaustively apply the Loop Rule, exhaustively apply the Chain Rule, and so forth). These observations yield Remark 4.6. \square

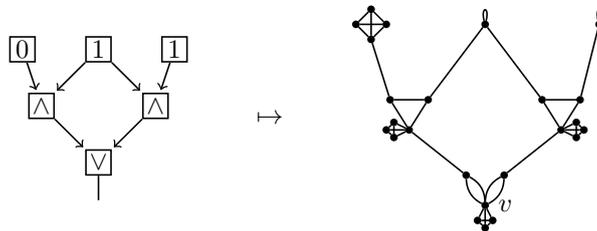


Figure 2: An example of the construction from the proof of Theorem 4.5. The vertex labeled v is the “output” vertex of the gadget corresponding to the output gate of the circuit, that is, v gets removed by the reduction rules if, and only if, the circuit evaluates to true. This is the case in this example, but if we change, for instance, the third input gate to false (replace the self-loop with a clique), v would not be removed.

Remark 4.6. *The proof of Theorem 4.5 shows that the problem remains P-hard restricted to the Chain Rule and the Loop Rule, even if they are alternately executed exhaustively.*

We close this section with the observation that also the last rule, the Flower Rule, is unlikely to yield a parallel algorithm.

Theorem 4.7. *Unless $\text{MATCHING} \in \text{NC}$, there is no DLOGTIME-uniform family of NC^i circuits for any i that determines, given a graph $G = (V, E)$, an integer k , and a vertex v , whether the Flower Rule can be applied to v .*

Proof. Assume we have access to an NC-circuit that determines if the Flower Rule can be applied to some vertex v . We construct an NC-circuit for MATCHING. On input $G = (V, E)$ and $k \in \mathbb{N}$, the circuit constructs a new graph G' by adding a vertex s to G , which is connected to all vertices of V . It then uses

the NC-circuit for the Flower Rule on s and the value $k - 1$. We claim G has a matching of size k if, and only if, the Flower Rule can be applied to s . For the first direction observe that, if s is in k disjoint cycles, we may take one edge of every cycle and, hence, have a matching of size k . Now assume G contains a matching of size k , say $M = \{(m_1, m_2), (m_3, m_4), \dots, (m_{2k-1}, m_{2k})\}$. Then s is contained in k disjoint cycles, namely $s - m_i - m_{i+1} - s$ for $i \in \{1, 3, 5, \dots, 2k - 1\}$. \square

5 Parallel Kernels for Structural Parameterizations

It is known that NP-hard graph parameters that are closed under taking disjoint union do not allow a polynomial kernel unless $\text{NP} \subseteq \text{coNP}/\text{poly}$ [14]. Famous problems that suffer from this result are the decision versions of tree width, path width, and tree depth, which has led to a growing body of research that considers structural parameters for these problems [6, 5, 27]. A commonly used parameter in this line of research is the vertex cover number of the input graph and in this section we extend the cited results by proving that the corresponding kernels can be computed in small circuit classes.

We use the following standard definitions: A *tree decomposition* of a graph $G = (V, E)$ is a tuple (T, ι) where T is a tree and ι a mapping from the nodes of T to subsets of V (which we call *bags*) such that for every $u \in V$ and every $\{v, w\} \in E$ there is (1) a node n with $u \in \iota(n)$, (2) a node m with $\{v, w\} \subseteq \iota(m)$, and (3) the set $\{n \mid u \in \iota(n)\}$ is connected in T . The *width* of a tree decomposition is the maximum size of the bags minus one. For a graph G , its *tree width* $\text{tw}(G)$ is the minimum width of all tree decompositions of G , its *path width* $\text{pw}(G)$ is the minimum width of all tree decompositions of G that are paths, and its *tree depth* $\text{td}(G)$ is the minimum width of all tree decompositions (T, ι) of G that can be rooted in such a way that for all $n, m \in V(T)$ we have $\iota(n) \subsetneq \iota(m)$ if m is a descendant of n . The following two facts will be useful, where $N(v) = \{u \mid \{u, v\} \in E\}$ is the neighborhood of v , $N[v] = N(v) \cup \{v\}$, and where we call a vertex v *simplicial* if $N(v)$ is a clique:

Fact 5.1 ([6, 5, 27]). *Let $G = (V, E)$ be a graph with tree width, path width, or tree depth at most k and with $u, v \in V$, $\{u, v\} \notin E$, and $|N(u) \cap N(v)| > k$. Then adding the edge $\{u, v\}$ to G will not increase the tree width, path width, or tree depth of G , respectively.*

Fact 5.2 ([4]). *Let $G = (V, E)$ be a graph and $v \in V$ be a simplicial vertex, then we have $\text{tw}(G) \geq |N(v)|$.*

Computing a Kernel for Tree Width. For the problem $p_{\text{vc}}\text{-TREE-WIDTH}$ we are given a graph $G = (V, E)$, an integer k , and a vertex cover $S \subseteq V$ of G ; the parameter is $|S|$ and the question is whether $\text{tw}(G) \leq k$ holds.

Theorem 5.3. *There is a DLOGTIME-uniform family of TC^0 -circuits that, on inputs of a triple (G, k, S) , outputs a $p_{\text{vc}}\text{-TREE-WIDTH}$ kernel with at most $O(|S|^3)$ vertices.*

Proof. On input (G, k, S) the circuit can easily check if S is actually a vertex cover and if we have $k < |S|$. If not, it outputs a trivial no-instance in the first case and a yes-instance in the second case (a tree decomposition of width $|S|$ can easily be obtained from S).

The circuit now checks in parallel for every pair $u, v \in S$ with $\{u, v\} \notin E$ if we have $|N(u) \cap N(v) \cap (V \setminus S)| > k$, that is, if the two vertices have more than k common neighbors in $V \setminus S$. If this is the case, the circuit adds the edge $\{u, v\}$. Note that this operation is safe by Fact 5.1 and can be applied in parallel as we consider only neighbors in $V \setminus S$ while we only add edges in S . Finally, the circuit considers all simplicial vertices $v \in V \setminus S$ in parallel: if $|N(v)| > k$, the circuit safely outputs a trivial no-instance by Fact 5.2, otherwise the circuit safely removes v from the graph by standard arguments [5].

We now argue that, if the circuit has not decided yet, the remaining graph has at most $O(|S|^3)$ vertices: the remaining graph consist of the vertices in S , and the nonsimplicial vertices $I \subseteq (V \setminus S)$. We have $|I| \leq |S|^3$ as any vertex $u \in I$ must have at least two neighbors v, w in S with $\{v, w\} \notin E$ (as otherwise u would be simplicial), however, every pair of nonadjacent vertices in S can have at most k common neighbors (as otherwise the circuit would have added the edge). Since we have at most $|S|^2$ such pairs, the claim follows by $k \leq |S|$. \square

Corollary 5.4. *For every $\delta \in \mathbb{N}$ there is a DLOGTIME-uniform family of AC^0 -circuits that, on input of a triple (G, k, S) , outputs a $p_{\text{vc}}\text{-TREE-WIDTH}$ kernel with at most $2^{\delta \sqrt{|S|}}$ vertices.*

Corollary 5.5. $p_{\text{vc}}\text{-TREE-WIDTH} \in \text{para-AC}^0$.

Computing a Kernel for Path Width. We define the problem p_{vc} -PATH-WIDTH analogously to p_{vc} -TREE-WIDTH and the aim of this section is to reformulate Theorem 5.3 in terms of path width. The main difference is that we cannot simply delete simplicial vertices as this would, for instance, eliminate trees completely. We can, however, use the following weaker result:

Fact 5.6 ([5]). *Let $G = (V, E)$ be a graph, $k \in \mathbb{N}$, and $v \in V$ be a simplicial vertex. If the degree $|N(v)|$ of v is 1 and the neighbor of v has another degree-1 neighbor, or if we have $2 \leq |N(v)| \leq k$ and for each pair $x, y \in N(v)$ there is a simplicial vertex $w \in N(x) \cap N(y)$ with $w \notin N[v]$, then $\text{pw}(G) \leq k$ if, and only if, $\text{pw}(G[V \setminus \{v\}]) \leq k$.*

Theorem 5.7. *There is a DLOGTIME-uniform family of TC^0 -circuits that, on input of a triple (G, k, S) , outputs a p_{vc} -PATH-WIDTH kernel with at most $O(|S|^3)$ vertices.*

Proof. The circuit works as in Theorem 5.3 and differs only in the last step, that is, the handling of simplicial vertices. We have to identify the vertices for which Fact 5.6 applies in constant parallel time, which is not trivial since we have dependencies between these vertices. The circuit *marks* simplicial vertices to which Fact 5.6 does not apply or which we will use as conditions when applying the fact to other vertices as follows: The circuit first marks for every $v \in S$ the lexicographically smallest degree-1 neighbor of v . Then for every simplicial vertex $v \in V \setminus S$ of degree at least 2, the circuit marks for every pair of neighbors x, y of v the lexicographically smallest simplicial vertex $w \in (N(x) \cap N(y)) \setminus N[v]$. If for any pair such a vertex does not exist, v marks itself. Note that all simplicial vertices that are not marked can safely be removed by Fact 5.6 and since, furthermore, the safeness is witnessed by marked vertices, the circuit can remove them all in parallel.

We are left with the task to show that there are at most $O(|S|^3)$ marked vertices left (the other vertices can be counted as in Theorem 5.3). We have at most $|S|$ marked vertices of degree 1 (one for each vertex in S), and at most $|S|^2$ marked vertices of degree greater than 1: each such vertex v has a pair of neighbors in S that has v as sole simplicial neighbor. \square

Corollary 5.8. *For every $\delta \in \mathbb{N}$ there is a DLOGTIME-uniform family of AC^0 -circuits that, on input of a triple (G, k, S) , outputs a p_{vc} -PATH-WIDTH kernel with at most $2^{\delta \sqrt{|S|}}$ vertices.*

Corollary 5.9. p_{vc} -PATH-WIDTH \in para- AC^0

Computing a Kernel for Tree Depth. The last problem we consider is tree depth, and, as for path width, we prove a version of Theorem 5.3 for it. The main problem is once more that we cannot simply remove simplicial vertices. However, by the following fact of Kobayashi and Tamaki there are still enough simplicial vertices that are safe to remove:

Fact 5.10 ([27]). *Let $G = (V, E)$ be a graph, $k \in \mathbb{N}$, and let $v \in V$ be a simplicial vertex with $1 \leq |N(v)| \leq k$. If every neighbor of v has degree at least $k + 1$, then we have $\text{td}(G) \leq k$ if, and only if, $\text{td}(G[V \setminus \{v\}]) \leq k$.*

Theorem 5.11. *There is a DLOGTIME-uniform family of TC^0 -circuits that, on input of a triple (G, k, S) , outputs a p_{vc} -TREE-DEPTH kernel with at most $O(|S|^3)$ vertices.*

Proof. We proceed again as in Theorem 5.3 and only differ in the way we handle simplicial vertices. In particular, we argue how we can apply Fact 5.10 in parallel constant time. The circuit starts by marking for every vertex $v \in S$ with $|N(v)| > k$ the $k + 1$ lexicographically smallest neighbors of v , then the circuit marks every simplicial vertex $v \in V \setminus S$ that has at least one neighbor of degree less than k . Note that every simplicial vertex that is not marked can safely be removed by Fact 5.10 and, since this safeness is witnessed by marked vertices, these vertices can be removed in parallel.

The amount of remaining vertices can be computed as in Theorem 5.3, we will end the proof by counting the number of marked vertices. There are at most $|S|^2 + |S|$ marked vertices that were marked in the first step, as every vertex in S marks only $k + 1$ neighbors. Additionally we may have some simplicial vertices that are marked because they have a neighbor of degree at most k . Since every degree k vertex in S can produce at most k such vertices, the number of these vertices can be bounded by $|S|^2$ as well. \square

Corollary 5.12. *For every $\delta \in \mathbb{N}$ there is a DLOGTIME-uniform family of AC^0 -circuits that, on inputs of a triple (G, k, S) , outputs a p_{vc} -TREE-DEPTH kernel with at most $2^{\delta \sqrt{|S|}}$ vertices.*

Corollary 5.13. p_{vc} -TREE-DEPTH \in para- AC^0

6 A Parallel Kernel for Point Line Cover

In this section we study a natural, well-known problem for which we can prove (unconditionally) that we *cannot* compute a kernel using AC^0 -circuits while we *can* compute polynomially-sized kernels in TC^0 . In the p -POINT-LINE-COVER problem we are given distinct points $p_1, \dots, p_n \in \mathbb{Z}^d$ for some dimension $d \geq 2$ and a natural number $k \in \mathbb{N}$, the question is whether we can cover all points by at most k lines. This problem is NP-hard in general (even for $d = 2$) and in FPT parameterized by k [29]. There is a simple k^2 kernel, which is essentially optimal [29]: If any line covers at least $k + 1$ points, remove all points on this line and reduce k by one. This is safe since we would require at least $k + 1$ different lines if we would not use this line. Because no set of $k + 1$ points lies on the same line after the reduction, we have at most k^2 points left or we deal with a no-instance.

Lemma 6.1. *There is a DLOGTIME-uniform family of TC^0 -circuits that, on input of a dimension d , a set of distinct points $p_1, \dots, p_n \in \mathbb{Z}^d$, and an integer k , outputs a p -POINT-LINE-COVER kernel with at most k^2 points.*

Proof. First observe that the reduction rule “for a line covering at least $k + 1$ points, remove all points on this line and reduce k by 1” can be applied in parallel, as removing all points from a line removes at most one point from any other line. To complete the proof, note that it is sufficient to check all n^2 line segments defined by pairs of points in parallel; and that a TC^0 -circuit can check if another point lies on such a line segment as it can multiply and divide binary numbers [23]. \square

The lemma shows that the optimal kernel for p -POINT-LINE-COVER can be computed in TC^0 and it is natural to ask if we can do the same using a AC^0 -circuit or, failing that, to at least compute *some* kernel using a AC^0 -circuit (as we could for the problems in the previous sections). We answer this question in the negative, settling the complexity of the problem to para- TC^0 :

Lemma 6.2. *For every fixed k , the k th slice of p -POINT-LINE-COVER is TC^0 -complete under AC^0 -reduction.*

Proof. We start with the case $k = 1$ and $d = 2$, which is clearly in TC^0 , as an instance is a yes-instance if, and only if, the input points are colinear. To see that the problem is TC^0 -hard we reduce from DIVISION defined as: Given three numbers x , y , and z , is it true that $x/y = z$? This is a classical TC^0 -complete problem [23]. For the reduction let x , y , z be the DIVISION-instance, we construct the instance $a = (0, 0)$, $b = (x, z)$, $c = (y, 1)$ of 1-POINT-LINE-COVER. This is a yes-instance if the points are colinear, that is, if we have $(b - a) \cdot (c - a) = 0$ or, equivalently: $\frac{x-0}{y-0} = \frac{z-0}{1-0} \iff x/y = z$. Since the cases $k > 1$ and / or $d > 2$ are generalizations, they remain TC^0 -hard. To see that these cases are also in TC^0 , observe that we have to consider at most n^2 line segments from which we have to pick k , that is, there are at most $\binom{n^2}{k} \leq n^{2k}$ solution candidates. For fixed k , these candidates can be checked in parallel by a TC^0 -circuit and can be evaluated as in the case of $k = 1$. \square

Corollary 6.3. *p -POINT-LINE-COVER is para- TC^0 -complete under AC^0 -reduction.*

Now assume there would be a uniform family of AC^0 -circuits computing a kernel of arbitrary size for p -POINT-LINE-COVER. Then by Theorem 2.3 the problem is in para- AC^0 , which on the other hand implies that for every fixed k the problem must be in AC^0 . This contradicts Lemma 6.2 as it is known that $AC^0 \subsetneq TC^0$ [19]. Therefore, no family of AC^0 -circuits can compute such a kernel.

Corollary 6.4. *para- $AC^0 \not\equiv p$ -POINT-LINE-COVER \in para- TC^0*

7 Problems for Which Computing a Kernelization is Inherently Sequential

As surprisingly many problems have NC-computable, in fact often even AC^0 -computable, kernelizations, we may ask which problems do not have this property. We would like to find problems for which the computation of any kernel is P-complete or, equivalently, which are FPT-complete under AC^0 - or NC^1 -reductions. While it is easy to find artificial problems with this property – such as any P-complete problem (like CVP) with the trivial parametrization ($\kappa(x) \equiv 1$) –, no *natural* problems that are FPT-complete for sensible parametrizations can be found in the literature. We remedy this situation in the following; but

must caution the reader that in all our results the hardness of the parameterized problem for FPT stems from the fact that some slice of the problem is (essentially) a known P-complete problem. Unfortunately, it is known [18] that this “cannot be helped” since all FPT-complete problems have this property. Our main contribution here lies, thus, in the assembly of a diverse body of relevant, non-trivial FPT-problems that will serve as starting points for further studies of the limits of parameterized parallelization.

Strong Backdoors to Satisfiability. A *strong backdoor set* of a propositional formula ϕ is a set of variables such that under any assignment of these variables the resulting formula ϕ' belongs to a certain class of formulas [20]. In the p -STRONG-BACKDOOR- $\{\text{HORN}, 2\text{CNF}\}$ -SAT problems, we are given a formula ϕ and an integer k , the question is whether ϕ is satisfiable and has a strong backdoor set of size k to Horn- or 2CNF-formulas, respectively. Solving such problems is usually done in two phases: first *detect* the backdoor set and, second, *solve* the satisfiability problem of the formula for every assignment of the backdoor set. While the first part might seem harder in general, it is not from a parameterized point of view: (1) A strong backdoor set to Horn formulas is exactly a vertex cover of size k in the positive primal graph of ϕ , that is, the graph that has a vertex for each variable and an edge between any two variables appearing together positively in a clause; (2) strong backdoor sets to 2CNF-formulas are exactly the hitting sets of the hypergraph that has the variables of ϕ as vertices and that connects three vertices by a hyperedge if they appear together in a clause. Since p -VERTEX-COVER \in para-AC⁰ and also p -3-HITTING-SET \in para-AC⁰ [2, 12], we can conclude:

Corollary 7.1. *There is a DLOGTIME-uniform family of para-AC⁰-circuits that, on input of a propositional formula ϕ and an integer k , either outputs a size- k strong backdoor set to $\{\text{Horn}, 2\text{CNF}\}$ -formulas, or concludes that no such set exists.*

The second step of solving p -STRONG-BACKDOOR- $\{\text{HORN}, 2\text{CNF}\}$ -SAT is to solve the satisfiability problem for ϕ on every assignment to the variables of the backdoor set. While we can nicely handle all assignments in parallel, checking if the formulas are satisfiable in parallel is difficult. Indeed, it is known that, under AC⁰-reductions, the satisfiability problem is NL-complete for 2CNF-formulas, and is even P-complete for Horn formulas [1].

Corollary 7.2. *p -STRONG-BACKDOOR-2CNF-SAT is para-NL-complete under AC⁰-reduction.*

Corollary 7.3. *p -STRONG-BACKDOOR-HORN-SAT is FPT-complete under AC⁰-reduction.*

The last corollary implies that there is no parallel kernelization running in polylogarithmic time for p -STRONG-BACKDOOR-HORN-SAT that produces a kernel of any size, unless NC = P.

Mixed Integer Linear Programming. The FPT-complete problem above is an intermediate problem between a P-complete problem (HORN-SAT) and a NP-complete problem (SAT); the transition between the problems is caused by the backdoor variables. A similar intermediate problem is known for LINEAR-PROGRAMMING (another classical P-complete problem) and its integer variant (which is NP-complete). The intermediate version of these problems is called p -MIXED-INTEGER-PROGRAMMING, which asks, given a matrix $A \in \mathbb{Z}^{n \times n}$, vectors $b \in \mathbb{Z}^n$, $c \in \mathbb{Z}^n$, and integers k and w , if there is a vector $x \in \mathbb{R}^n$ such that $Ax \leq b$, $c^T x \geq w$, and such that $x[i] \in \mathbb{Z}$ for $0 \leq i < k$. A celebrated result by Lenstra states that an instance I of this problem can be solved in time $2^{O(k^3)} \cdot |I|^c$ for a suitable constant c , that is, the problem is in FPT. Therefore, every slice of the problem is in P and, as LINEAR-PROGRAMMING trivially reduced to it, we get that k -MIXED-INTEGER-PROGRAMMING is P-complete for every k (under NC¹-reductions [37]).

Corollary 7.4. *p -MIXED-INTEGER-PROGRAMMING is FPT-complete under NC¹-reductions.*

Maximum Flow with Minimum Quantities. The last problem we review in this section is the maximum flow problem with minimum quantities: Inputs are directed graphs $G = (V, E)$ with $s, t \in V$, two weight functions $u, l: E \rightarrow \mathbb{N}$, an integer $w \in \mathbb{N}$, and a set of edges $B \subseteq E$; the question is whether there is a set $A \subseteq B$ such that in $G' = (V, E \setminus A)$ there is a valid s - t -flow f of value at least w that fulfills the flow conservation constraints and $l(e) \leq f(e) \leq u(e)$ for all $e \in E \setminus A$. For $B = \emptyset$ the problem boils down to classical maximum flow with lower bounds on the edges, which can be solved in polynomial time [26] and which is known to be P-hard under NC¹-reduction [37]. On the other hand, for $B = E$ the problem becomes NP-complete even on serial-parallel graphs [22] and it is also NP-hard to approximate

the problem within any positive factor [35]. The intermediate problem between this two cases is the parameterized problem p -MAX-FLOW-QUANTITIES where the cardinality of B is the parameter.

Lemma 7.5. p -MAX-FLOW-QUANTITIES is FPT-complete under NC^1 -reduction.

Proof. Containment in FPT follows by the simple algorithm that iterates over all $2^{|B|}$ possible sets $A \subseteq B$ and which computes a maximum flow in $G' = (V, E \setminus A)$ with, for instance, a variant of Ford-Fulkerson [26]. The algorithm also implies that for every fixed k the slice k -MAX-FLOW-QUANTITIES is in P and, since it is a generalization of classical MAX-FLOW, it is also P-complete. \square

8 Conclusion and Outlook

Kernelization is a fundamental concept of parameterized complexity and we have studied its parallelization. Since traditional descriptions of kernelization algorithms are inherently sequential, we found it surprising how many parameterized problems lie in para-AC^0 – the smallest robust class in parallel parameterized complexity theory. We found, furthermore, that for many problems the equation “smaller circuit class = larger kernel” holds, see Table 1 for a summary of our results.

Apart from classifying more parameterized problems in the spirit of this paper, namely according to how well small kernels can be computed by small circuits, an interesting open problem is to improve any of the AC^0 -kernelizations presented in the paper so that they produce a *polynomially sized* kernel (which we, at best, can currently do only in TC^0). Perhaps even more challenging seems to be the design of a framework for proving that polynomially sized kernels for these problems *cannot* be computed in AC^0 .

References

- [1] E. Allender, M. Bauland, N. Immerman, H. Schnoor, and H. Vollmer. The complexity of satisfiability problems: Refining Schaefer’s theorem. *J. Comput. Syst. Sci.*, 75(4):245–254, 2009. doi:10.1016/j.jcss.2008.11.001.
- [2] M. Bannach, C. Stockhusen, and T. Tantau. Fast parallel fixed-parameter algorithms via color coding. In *Proceedings of IPEC 2015*, pages 224–235, 2015. doi:10.4230/LIPIcs.IPEC.2015.224.
- [3] M. Bannach and T. Tantau. Parallel multivariate meta-theorems. In *Proceedings of IPEC 2016*, pages 4:1–4:17, 2016. doi:10.4230/LIPIcs.IPEC.2016.4.
- [4] H. L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.*, 25(6):1305–1317, 1996. doi:10.1137/S0097539793251219.
- [5] H. L. Bodlaender, B. M. P. Jansen, and S. Kratsch. Kernel bounds for structural parameterizations of pathwidth. In *Proceedings of SWAT 2012*, pages 352–363, 2012. doi:10.1007/978-3-642-31155-0_31.
- [6] H. L. Bodlaender, B. M. P. Jansen, and S. Kratsch. Preprocessing for treewidth: A combinatorial analysis through kernelization. *SIAM J. Discrete Math.*, 27(4):2108–2142, 2013. doi:10.1137/120903518.
- [7] H. L. Bodlaender and T. C. van Dijk. A cubic kernel for feedback vertex set and loop cutset. *Theory Comput. Syst.*, 46(3):566–597, 2010. doi:10.1007/s00224-009-9234-2.
- [8] Kevin Burrage, Vladimir Estivill-Castro, Michael R. Fellows, Michael A. Langston, Shev Mac, and Frances A. Rosamond. The undirected feedback vertex set problem has a $\text{poly}(k)$ kernel. In *Proceedings of IWPEC 2006*, pages 192–202, 2006. doi:10.1007/11847250_18.
- [9] L. Cai, J. Chen, R. G. Downey, and M. R. Fellows. Advice classes of parameterized tractability. *Ann. Pure Appl. Logic*, 84(1):119–138, 1997. doi:10.1016/S0168-0072(95)00020-8.
- [10] J. Chen, I. A. Kanj, and W. Jia. Vertex cover: Further observations and further improvements. *J. Algorithms*, 41(2):280–301, 2001. doi:10.1006/jagm.2001.1186.
- [11] Y. Chen and J. Flum. Some lower bounds in parameterized AC^0 . In *Proceedings of MFCS 2016*, pages 27:1–27:14, 2016. doi:10.4230/LIPIcs.MFCS.2016.27.

- [12] Yijia Chen, Jörg Flum, and Xuanguai Huang. Slicewise Definability in First-Order Logic with Bounded Quantifier Rank. In Valentin Goranko and Mads Dam, editors, *Proceedings of CSL 2017*, volume 82, pages 19:1–19:16, 2017. doi:10.4230/LIPIcs.CSL.2017.19.
- [13] Stephen A. Cook and Pierre McKenzie. Problems complete for deterministic logarithmic space. *J. Algorithms*, 8(3):385–394, 1987. doi:10.1016/0196-6774(87)90018-6.
- [14] M. Cygan, F. V. Fomin, L. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, and S. Saurabh. *Parameterized Algorithms*. Springer Berlin Heidelberg, 2015.
- [15] R. G. Downey and M. R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013. doi:10.1007/978-1-4471-5559-1.
- [16] M. Elberfeld, C. Stockhusen, and T. Tantau. On the space and circuit complexity of parameterized problems: Classes and completeness. *Algorithmica*, 71(3):661–701, 2015. doi:10.1007/s00453-014-9944-y.
- [17] S. A. Fenner, R. Gurjar, and T. Thierauf. Bipartite perfect matching is in quasi-NC. In *Proceedings of STOC 2016*, pages 754–763, 2016. doi:10.1145/2897518.2897564.
- [18] J. Flum and M. Grohe. *Parameterized Complexity Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2006. doi:10.1007/3-540-29953-x.
- [19] M. L. Furst, J. B. Saxe, and M. Sipser. Parity, circuits, and the polynomial-time hierarchy. *Mathematical Systems Theory*, 17(1):13–27, 1984. doi:10.1007/BF01744431.
- [20] S. Gaspers and S. Szeider. Backdoors to satisfaction. In H.L. Bodlaender, R. Downey, F. V. Fomin, and D. Marx, editors, *The Multivariate Algorithmic Revolution and Beyond*, pages 287–317. Springer, 2012. doi:10.1007/978-3-642-30891-8_15.
- [21] Raymond Greenlaw, H. James Hoover, and Walter L. Ruzzo. *Limits to Parallel Computation: P-completeness Theory*. Oxford University Press, 1995.
- [22] D. Haugland, M. Eleyat, and M. Lie Hetland. The maximum flow problem with minimum lot sizes. In *Proceedings of ICCL 2011*, pages 170–182, 2011. doi:10.1007/978-3-642-24264-9_13.
- [23] W. Hesse. Division is in uniform TC⁰. In *Proceedings of ICALP 2001*, pages 104–114, 2001. doi:10.1007/3-540-48224-5_9.
- [24] Yoichi Iwata. Linear-time kernelization for feedback vertex set. In *Proceedings of ICALP 2017*, pages 68:1–68:14, 2017. doi:10.4230/LIPIcs.ICALP.2017.68.
- [25] Joseph JáJá. *An Introduction to Parallel Algorithms*. Addison-Wesley, 1992.
- [26] Jon M. Kleinberg and Éva Tardos. *Algorithm design*. Addison-Wesley, 2006.
- [27] Y. Kobayashi and H. Tamaki. Treedepth parameterized by vertex cover number. In *Proceedings of IPEC 2016*, pages 18:1–18:11, 2016. doi:10.4230/LIPIcs.IPEC.2016.18.
- [28] D. König. Über Graphen und ihre Anwendung auf Determinantentheorie und Mengenlehre. *Mathematische Annalen*, 77(4):453–465, 1916. doi:10.1007/BF01456961.
- [29] S. Kratsch, G. Philip, and S. Ray. Point line cover: The easy kernel is essentially tight. *ACM Trans. Algorithms*, 12(3):40:1–40:16, 2016. doi:10.1145/2832912.
- [30] M. Lampis. A kernel of order $2k - c \log k$ for vertex cover. *Inf. Process. Lett.*, 111(23-24):1089–1091, 2011. doi:10.1016/j.ipl.2011.09.003.
- [31] K. Mulmuley, U. V. Vazirani, and V. V. Vazirani. Matching is as easy as matrix inversion. *Combinatorica*, 7(1):105–113, 1987. doi:10.1007/BF02579206.
- [32] G. L. Nemhauser and L. E. Trotter Jr. Properties of vertex packing and independence system polyhedra. *Math. Program.*, 6(1):48–61, 1974. doi:10.1007/BF01580222.

- [33] I. Newman, P. Ragde, and A. Wigderson. Perfect hashing, graph entropy, and circuit complexity. In *Proceedings of STC 1990*, pages 91–99. IEEE Computer Society, Los Alamitos, California, 1990. doi:10.1109/SCT.1990.113958.
- [34] A. Soleimanfallah and A. Yeo. A kernel of order $2k - c$ for vertex cover. *Discrete Mathematics*, 311(10-11):892–895, 2011. doi:10.1016/j.disc.2011.02.014.
- [35] Clemens Thielen and Stephan Westphal. Complexity and approximability of the maximum flow problem with minimum quantities. *Networks*, 62(2):125–131, 2013. doi:10.1002/net.21502.
- [36] Stéphane Thomassé. A $4k^2$ kernel for feedback vertex set. *ACM Trans. Algorithms*, 6(2):32:1–32:8, 2010. doi:10.1145/1721837.1721848.
- [37] Jacobo Toran. P-completeness. In Alan Gibbons and Paul Spirakis, editors, *Lectures on parallel computation*, pages 177–196. Cambridge University Press, 1993.