Computing Hitting Set Kernels by AC^0-Circuits

Max Bannach
Institute of Theoretical Computer Science, Universität zu Lübeck, Lübeck, Germany
bannach@tcs.uni-luebeck.de

Till Tantau
Institute of Theoretical Computer Science, Universität zu Lübeck, Lübeck, Germany
tantau@tcs.uni-luebeck.de

Abstract

Given a hypergraph $H = (V, E)$, what is the smallest subset $X \subseteq V$ such that $e \cap X \neq \emptyset$ holds for all $e \in E$? This problem, known as the *hitting set problem,* is a basic problem in parameterized complexity theory. There are well-known kernelization algorithms for it, which get a hypergraph H and a number k as input and output a hypergraph H' such that (1) H has a hitting set of size k if, and only if, H' has such a hitting set and (2) the size of H' depends only on k and on the maximum cardinality d of edges in H. The algorithms run in polynomial time, but are highly sequential. Recently, it has been shown that one of them can be parallelized to a certain degree: one can compute hitting set kernels in parallel time $O(d)$ – but it was conjectured that this is the best parallel algorithm possible. We refute this conjecture and show how hitting set kernels can be computed in *constant* parallel time. For our proof, we introduce a new, generalized notion of hypergraph sunflowers and show how iterated applications of the color coding technique can sometimes be collapsed into a single application.

2012 ACM Subject Classification Mathematics of computing → Hypergraphs, Theory of computation → Fixed parameter tractability, Theory of computation → Circuit complexity

Keywords and phrases Parallel Computation, Fixed-parameter Tractability, Kernelization

Digital Object Identifier 10.4230/LIPIcs.STACS.2018.9

Related Version Full proofs can be found in the full version of the paper [5], `http://arxiv.org/abs/1801.00716`.

1 Introduction

The hitting set problem is the following combinatorial problem: Given a hypergraph $H = (V, E)$ as input, consisting of a set V of vertices and a set E of *hyperedges* with $e \subseteq V$ for all $e \in E$, find a set $X \subseteq V$ of minimum size that “hits” all hyperedges $e \in E$, that is, $e \cap X \neq \emptyset$. Many problems reduce to the hitting set problem, including the vertex cover problem (it is exactly the special case where all edges have size $|e| = 2$) and the dominating set problem (a dominating set of a graph is exactly a hitting set of the hypergraph whose hyperedges are the closed neighborhoods of the graph's vertices). The computational complexity of the hitting set problem is thus of interest both in classical complexity theory and in parameterized complexity theory.

The first result on the parameterized complexity of the hitting set problem was an efficient *kernelization algorithm* for this problem restricted to edges of cardinality three [16]. This was later improved to a kernelization for the d-uniform version (all hyperedges have size exactly d) [15], which is based on the so-called Sunflower Lemma [13]. We will later have a closer look at this algorithm; at this point let us just summarize its main idea by “repeatedly find sunflowers and replace them by their cores until there are no more sunflowers.” The

Sunflower Lemma tells us that this algorithm will stop only when the input graph has been reduced to a kernel. The just-sketched kernelization algorithm is highly sequential, but Chen et al. [11] have recently shown that it can be parallelized: Instead of reducing sunflowers one-at-a-time, one can replace all sunflowers in a hypergraph by their cores simultaneously in constant parallel time. This process only needs to be repeated $d(H) = \max_{e \in E} |e|$ times, leading to a parallel algorithm running in time $O(d(H))$. However, there were good reasons to believe that this algorithm is essentially the best possible (we will later discuss them) and Chen et al. conjectured that the hitting set problem does not admit a kernelization algorithm running in constant parallel time (that is, in time completely independent of the input graph).

Our Contributions. In the present paper we refute the conjecture of Chen et al. and show that there is a constant parallel time kernelization algorithm for the hitting set problem:

▶ **Problem 1.1.** $p_{k,d}$-HITTING-SET
Instance: *A hypergraph $H = (V, E)$ and a number $k \in \mathbb{N}$.*
Parameter: *$k + d(H)$*
Question: *Does H have a hitting set X with $|X| \leq k$?*

▶ **Theorem 1.2** (Main Theorem). *There is a* DLOGTIME*-uniform* AC^0*-circuit family that maps every hypergraph $H = (V, E)$ and number k to a new hypergraph $H' = (V, E')$ that has the same size-k hitting sets as H, has $d(H') \leq d(H)$, and has $|E'| \leq f(k, d(H))$ for some fixed computable function f.*

Let us stress at this point that the AC^0-family from the theorem really has a size that is polynomial in the input length (no exponential or even worse dependency on the parameters) and has a depth that is completely independent of the input. The hypergraph H' has the same vertex set V as H – a feature shared by all hypergraphs considered in this paper that simplifies the presentation. However, since V is still "large," the circuit is not quite a kernelization algorithm. Fortunately, this is easy to fix by replacing the vertex set of H' by $V' = \bigcup_{e \in E'} e$, yielding the following corollary:

▶ **Corollary 1.3** (Constant-Time Kernelization). *There is a* DLOGTIME*-uniform* AC^0*-circuit family that computes a kernel for every instance for $p_{k,d}$-*HITTING-SET.

The theorem and corollary imply that all problems that can be reduced to $p_{k,d}$-HITTING-SET via a parameter-preserving AC^0-reduction admit a kernelization computable by an AC^0-circuit family. This includes p_k-VERTEX-COVER, which is just $p_{k,d}$-HITTING-SET with d fixed at 2; p_k-TRIANGLE-REMOVAL, where the objective is to remove at most k vertices from an undirected graph so that no triangles remain; and also $p_{k,\text{deg}}$-DOMINATING-SET, where we must find a dominating set of size at most k in an undirected graph and we parameterized by k and the maximum degree of the vertices.

Our proof of the main theorem requires the development of two new ideas, which we believe may also be useful in other situations. The above-mentioned parallel kernelization algorithm for the hitting set problem with runtime $O(d(H))$ essentially does the following: "Repeat $d(H)$ times: replace all sunflowers of size $k + 1$ by their cores" and the difficult task in each of the $d(H)$ iterations is to find the sunflowers. It turns out that this can be done in constant parallel time using the *color coding* technique [2] and it has been shown in [3] and again in [11] that this technique can be implemented in constant time. Our first idea for turning the circuits depth from $O(d)$ into $O(1)$ is to *collapse the color codings from the d rounds into a single application of the color coding technique:* Instead of applying color coding in each round to filter and describe "objects," we would like to apply one global application of color coding that already contains the internal colorings and does away with the intermediate objects.

Unfortunately, there does not appear to be a simple (or any) way of actually collapsing the colorings used when we "replace all sunflowers by their cores": The coloring coding technique is good at imposing requirements of the form "these objects must be disjoint," but cannot impose requirements of the form "these objects must be the same." For this reason, as our second new idea, we develop a generalization of the notion of a sunflower (which we dub "pseudo-sunflowers") that is tailored to the collapsing of color coding.

Related Work. The sequential kernelization algorithm for the hitting set problem based on the Sunflower Lemma has been known for a longer time [15], but there have been recent improvements that bring down the runtime to linear time [17]. A parallel version has recently been studied by Chen et al. [11] and they show how kernels for $p_{k,d}$-HITTING-SET can be computed by circuits of depth $O(d(H))$. Chen et al. also conjecture that the circuit depth of $O(d(H))$ is unavoidable (which we refute).

The results of this paper fit into the larger, fledgling field of parallel parameterized complexity theory, which has already been studied both from a practical [1] and a theoretical point of view [8]. First results go back to research on *parameterized logarithmic space* [7, 10, 14], since it is known from classical complexity theory that problems that are solvable with such a resource bound can also be parallelized. A more structured analysis of parameterized space and circuit classes was later made by Elberfeld et al. [12], which addresses parallelization more directly. Current research on parameterized parallelization – including this paper – focuses on constant-time computations, that is, on a parameterized analogue of AC^0 [9, 11, 3, 4]. We remark that many previous results (including several of the authors) boil down to showing that instead of using a known reduction rule many times sequentially, one can simply apply it in parallel "everywhere," but "only once." In contrast, the kernelization algorithm developed in the present paper had no previous counterpart in the sequential setting.

Organization of This Paper. After a short section on preliminaries, in Section 3 we review known kernelization algorithms for the hitting set problem – both the sequential ones and the parallel one. In Section 4 we discuss the obstacles that must be surmounted to turn the known parallel algorithm into one that needs only constant time. Towards this aim, we introduce the notions of pseudo-cores and pseudo-sunflowers as replacements for the cores and sunflowers used in the known algorithms. In Section 5 we then argue that these pseudo-sunflowers can be computed in constant time by "collapsing" multiple rounds of color coding into a single round. Full proofs can be found in the full version of the paper [5].

2 Preliminaries

A *hypergraph* is a pair $H = (V, E)$ such that for all *hyperedges* $e \in E$ we have $e \subseteq V$. We write $V(H) = V$ and $E(H) = E$ for the vertex and hyperedge sets of H. Let $d(H) = \max_{e \in E} |e|$. Throughout this paper, all hypergraphs will always have the same vertex set V, which is the input vertex set. For this reason, in slight abuse of notation, for two hypergraphs $H_1 = (V, E_1)$ and $H_2 = (V, E_2)$ we also write $H_1 \subseteq H_2$ for $E(H_1) \subseteq E(H_2)$ and $H_1 \cup H_2$ for $(V, E(H_1) \cup E(H_2))$.

Concerning circuit classes and parallel computations, we will only need the notion of AC-circuit families, which are sequences $C = (C_0, C_1, C_2, \dots)$ of Boolean circuits where each

C_i is a directed acyclic graph whose vertices are gates such that there are i input gates, the inner gates are $\wedge$-gates or $\vee$-gates with unbounded fan-in, or $\neg$-gates; and the number of output gates is either 1 (for decision problems) or depends on the number of input gates (for circuits computing a function). The *size function* S maps circuits to their size (number of gates) and the *depth function* D maps them to their depth (longest path from input gates to output gates). When $D(C_n) \in O(1)$ and $S(C_n) \in n^{O(1)}$ hold, we call C an AC^0-circuit family. Concerning circuit uniformity, all circuit families in this paper will be DLOGTIME uniform, which is the strongest notion of uniformity commonly considered [6] and defined as follows: there is a DTM that on input of $\text{bin}(i)\#\,\text{bin}(n)$, where $\text{bin}(x)$ is the binary encoding of x, outputs the ith bit of a suitable encoding of C_n in at most $O(\log n)$ steps.

Even though this paper is about a parallel kernelization algorithm, we will need only little from the machinery of parallel parameterized complexity theory. We do need the following notions: A *parameterized problem* is a pair (Q, κ) where $Q \subseteq \Sigma^*$ is a language and κ is a function $\kappa\colon \Sigma^* \to \mathbb{N}$ that is computable by a DLOGTIME-uniform AC^0-circuit family. When we write down a parameterized problem such as $p_{k,d}$-HITTING-SET, the indices of "p" (for "parameterized") indicate which parameter function κ we mean. A *kernelization* for a parameterized problem (Q, κ) is a function K that maps every instance $x \in \Sigma^*$ to a new instance $K(x) \in \Sigma^*$ such that for all $x \in \Sigma^*$ we have (1) $x \in Q \iff K(x) \in Q$ and (2) $|K(x)| \le f(\kappa(x))$ for some fixed computable function f.

A parameterized problem (Q, κ) lies in FPT if $x \in Q$ can be decided by a sequential algorithm running in time $f(\kappa(x)) \cdot |x|^{O(1)}$ for a computable function f. The AC^0-analogue of FPT is the class para-AC^0. It contains all problems (Q, κ) for which there is a circuit family $(C_{n,k})_{n,k\in\mathbb{N}}$ such that for all inputs x we have $C_{|x|,\kappa(x)}(x) = 1$ if, and only if, $x \in Q$, and $D(C_{n,k}) \in O(1)$ and $S(C_{n,k}) \in f(k) \cdot n^{O(1)}$. It is well-known that $(Q, \kappa) \in$ FPT holds if, and only if, Q is decidable and there is a kernelization for (Q, κ) that is computable in polynomial time. The same proof as for the polynomial-time case also shows that we have $(Q, \kappa) \in$ para-AC^0 if, and only if, Q is decidable and (Q, κ) has a kernelization that can be computed by an AC^0-circuit family. (We stress once more that this means that the kernelization is a normal AC^0-circuit family, having size $S(C_n) \in n^{O(1)}$.)

We will use the *color coding technique* a lot. First introduced in [2], it has recently been shown to work in the context of constant time computations [3, 11]. The key observation underlying this technique is the following: Suppose we are given a set of n elements and suppose you have k special elements $x_1, \dots, x_k$ together with some specific colors $c_1, \dots, c_k$ for them "in mind". Then we can compute a set Λ of "candidate colorings" of all elements of the set such that at least one $\lambda \in \Lambda$ colors each "in mind" vertex x_i with the "desired" color c_i, that is $\lambda(x_i) = c_i$. Formally, the following holds (the original version of this lemma due to Alon et al [2] is equivalent to the statement below – only without any depth guarantees):

► **Fact 2.1** (Color Coding Lemma, [3]). *There is a* DLOGTIME*-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 a set Λ of functions $\lambda\colon \{1, \dots, n\} \to \{1, \dots, c\}$ (coded as a sequence of function tables) with the property that for any k mutually distinct $x_1, \dots, x_k \in \{1, \dots, n\}$ and any $c_1, \dots, c_k \in \{1, \dots, c\}$ there is a function $\lambda \in \Lambda$ with $\lambda(x_i) = c_i$ for all $i \in \{1, \dots, k\}$,*
2. *has constant depth (independent of n, k, or c), and*
3. *has size at most $O(\log c \cdot c^{k^2} \cdot k^4 \cdot n \log^2 n)$.*

3 Known Kernelization Algorithms for the Hitting Set Problem

Known Sequential Kernelization Algorithms. Known algorithms for computing kernels for $p_{k,d}$-HITTING-SET are based on the so-called *Sunflower Lemma.* The perhaps simplest application of this lemma is to repeatedly collapses sufficiently large sunflowers to their cores until there are no longer any large sunflowers in the graph and, then, the Sunflower Lemma tells us that the graph "cannot be very large." In detail, the definitions and algorithm are as follows:

▶ **Definition 3.1** (Sunflower). A *sunflower S with core C* is a set of proper supersets of C such that for any two distinct $p, q \in S$ we have $p \cap q = C$. The elements of a sunflower are called *petals.* A *sunflower in a hypergraph* is a sunflower whose petals are hyperedges of the hypergraph.

▶ **Fact 3.2** (Sunflower Lemma [13]). *Every hypergraph H with more than $k^{d(H)} \cdot d(H)!$ hyperedges contains a sunflower of size $k+1$.*

The importance of the Sunflower Lemma for the hitting set problem lies in the following observation: Suppose a hypergraph H contains a sunflower S of size at least $k+1$. Then H has a size-k hitting set if, and only if, the hypergraph obtained from H by removing all petals of the sunflower and adding its core has such a hitting set (we cannot hit the $k+1$ petals in the sunflower using only k vertices without using at least one vertex of the core; thus, we hit all petals if, and only if, we hit the core). In other words, replacing a sunflower of size $k+1$ by its core is a reduction rule for the hitting set problem; and if we can no longer apply this rule, the Sunflower Lemma tells us that the hypergraph's size is bounded by a function that depends only on k and $d(H)$ – in other words, it is a kernel.

The just-described kernelization algorithm is simple, but "very sequential." It is, however, not too difficult to turn it into a more parallel algorithm – at least, as long as $d(H)$ is fixed. This was first noted by Chen et al. [11] and we explain the ideas behind their proof below, rephrased for the purposes of the present paper.

A better sequential kernelization algorithm has recently [17] been proposed (it runs in time $O(2^{d(H)}|E|)$, which is linear from a parameterized point of view) – but the algorithm is arguably "even more sequential" and does not lend itself to easy parallelization.

Known Parallel Kernelization Algorithm. The first step towards a parallel kernelization is the observation that we can compute many cores in parallel. Given a hypergraph $H = (V, E)$ and a number k, let a *k-core in H* be a core C of a sunflower in H with more than k petals. Let $k\text{-cores}(H) = \bigl(V, \{C \mid C \text{ is a } k\text{-core in } H\}\bigr)$. While in the sequential algorithm we always replace one sunflower by its core, we now replace *all* sunflowers by their cores. This leaves behind some hyperedges, but the Sunflower Lemma will show that their number is "small." Unfortunately, the set of cores itself may still be large and we need to apply the replace-all-sunflowers-by-cores operation repeatedly. This process *does* stop after at most $d(H)$ rounds since the *size* of the cores decreases by 1 in each round and, hence, after $d(H)$ rounds it has shrunk to 0.

Let us now formalize these ideas a bit: Let $H_0 = H$ and let $H_{i+1} = k\text{-cores}(H_i)$. Then H_0 is the original hypergraph; H_1 is the set of its k-cores; H_2 is the set of H_1's k-cores and thus the set of "cores of cores" of H; next H_3 is the set of "cores of cores of cores" of H; and so on, see Figure 1 for an example. In a sense, each H_i is nested into the previous hypergraph, leading to a whole sequence resembling a matryoshka doll. Below, we define a

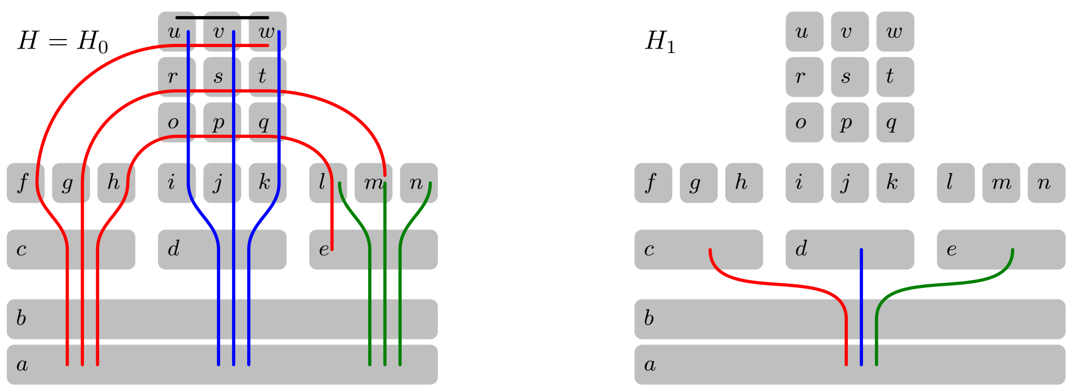


Figure 1 Visualization of a hypergraph H_0 and of its 2-cores $H_1 = \text{2-cores}(H_0)$. Vertices are drawn as rectangles, while the ten hyperedges of H_0 are drawn as lines: they contain all vertices that they touch. For instance, the leftmost line starting in the vertex a in H_0 visualizes the hyperedge $\{a, b, c, f, u, v, w\}$ and the rightmost line visualizes the hyperedge $\{a, b, e, n\}$. The hypergraph H_0 contains three sunflowers of size 3, visualized by the red, blue, and green lines, respectively. Their cores are the hyperedges shown in H_1. These cores, in turn, form a sunflower in H_1 with core $\{a, b\}$, but note that $\{a, b\}$ is *not* a 2-core of H_0. It is the only hyperedge of H_2.

matryoshka sequence as a sequence that has this "nested in some sense" property and then show in Lemma 3.4 that $(H_0, H_1, \dots)$ is, indeed, such a matryoshka sequence:

▶ **Definition 3.3** (Matryoshka Sequence). A *matryoshka sequence for a hypergraph $H = (V, E)$ and a number k* is a sequence $(M_0, M_1, \dots, M_{d(H)})$ of hypergraphs, all of which have the same vertex set V, with the following properties for all $i \in \{0, \dots, d(H)\}$:

1. $M_0 = H$,
2. $d(M_i) \le d(H) - i$,
3. $k\text{-cores}(M_i) \subseteq M_{i+1}$, and
4. every size-k hitting set of H is also a hitting set of M_i.

▶ **Lemma 3.4** (Cores of Cores Form a Matryoshka Sequence). *For every hypergraph H and number k, the sequence $(H_0, \dots, H_{d(H)})$ is a matryoshka sequence for H and k.*

Sketch of Proof. The first three items follow directly from the definition. The fourth item is proven by induction over i, where the inductive step hinges on the observation that the only way to hit a sunflower of size $k + 1$ with a size k set X is to hit its core. ◀

Recall that the idea behind the parallel computation of a kernel for the hitting set problem is to repeatedly remove all sunflowers from H, each time perhaps leaving a manageable number of hyperedges – and after d rounds, no hyperedges will remain. We use the following notation for the "removal" operation: For two hypergraphs $H = (V, E)$ and $H' = (V, E')$ let $H \ominus H' = \big(V, \{\, e \in E \mid \forall e' \in E' \colon e' \not\subseteq e \,\}\big)$, that is, we remove all hyperedges from H that contain a hyperedge of H'. Thus, $H \ominus H_1$ is the set of all hyperedges in H that are not involved in any sunflower of size at least $k + 1$ since we remove all edges that contain a core.

The following theorem shows that the repeated removing operation only leaves behind a "small" number of hyperedges. We formulate the theorem for arbitrary matryoshka sequences (we will need this later on), but it is best to think of the M_i as the sets H_i.

▶ **Theorem 3.5** (Kernel Theorem). *Let $(M_0, \dots, M_{d(H)})$ be a matryoshka sequence for H and k. Let $K = (M_0 \ominus M_1) \cup (M_1 \ominus M_2) \cup (M_2 \ominus M_3) \cup \dots \cup (M_{d(H)-1} \ominus M_{d(H)}) \cup M_{d(H)}$.*

1. *Then K has at most $\sum_{i=0}^{d(H)} k^i i!$ hyperedges and*
2. *H and K have the same size-k hitting sets.*

Sketch of Proof. For the first item we observe that $M_i \ominus M_{i+1}$ does not contain a sunflower and apply the Sunflower Lemma. The second item is proven by induction over i, where the base case is given by the first property of a matryoshka sequence, and where the inductive step can be derived from the fourth property of a matryoshka sequence. ◀

Instantiating the theorem with $(H_0, \dots, H_{d(H)})$ tells us that, if we can compute the elements of $K = (H_0 \ominus H_1) \cup \dots \cup (H_{d(H)-1} \ominus H_{d(H)}) \cup H_{d(H)}$ in parallel, we can compute a kernel for the hitting set problem in parallel. Clearly, "computing K" essentially boils down to "computing the H_i" in parallel. Thus, the real question, which we address next, is how quickly and easily we can compute the hypergraphs H_i.

At this point, we briefly need to address some technical issues concerning the coding of hypergraphs. For our purposes, it is largely a matter of taste how the input hypergraph H_0 is encoded, but the encoding of the later graphs H_i becomes important in the context of parallel constant-time computations. We consider $H = (V, E)$ fixed and encoded using, for instance, an incidence matrix (having $|V|$ columns and $|E|$ rows). We encode a *refinement of H*, that is, a hypergraph $H' = (V, E')$ with the property that each $e' \in E'$ is a subset of some $e \in E$, using a matrix of $2^{d(H)}$ columns and $|E|$ rows. There is a column for each of the at most $2^{d(H)}$ possible subsets of an edge $e \in E$ and the entry at the column for a given row is 1 if this subset is an element of E'; otherwise it is 0. Let us call this the *refinement matrix encoding* of hypergraph H' (with respect to the fixed input hypergraph H).

▶ **Lemma 3.6** (Computing Cores in Constant Depth)**.** *For each d and i there is a* DLOGTIME-*uniform family of* AC-*circuits that*

1. *on input of the incidence matrix of a hypergraph H with $d(H) \leq d$, a number k, and the refinement matrix encoding of the hypergraph H_i,*
2. *outputs the refinement matrix encoding of H_{i+1},*
3. *has constant depth, and*
4. *has size $f(k, d) \cdot |V|^{O(1)}|E|^{O(1)}$ where f is some computable function.*

Sketch of Proof: We can test all $|E| \cdot 2^{d(H)}$ possible cores C in parallel and, for each of them, we can search a corresponding sunflower via color coding: for each petal p_i the vertices in $p_i - C$ should receive color i. ◀

The lemma tells us that once we have computed some H_i, we can compute the next H_{i+1} using only constant additional depth and using $f(k, d) \cdot |V|^{O(1)}|E|^{O(1)}$ additional size. Since $H_i \ominus H_{i+1}$ can easily be computed from H_i and H_{i+1} in constant depth, we get:

▶ **Theorem 3.7** (Depth-$O(d)$ Kernelization Algorithm, [11])**.** *For each d there is a* DLOGTIME-*uniform family of* AC-*circuits that*

1. *on input of a hypergraph H with $d(H) \leq d$ and a number k*
2. *outputs a hypergraph K having the same size-k hitting sets as H and having at most $\sum_{i=0}^{d(H)} k^i i!$ hyperedges,*
3. *has depth $O(d)$,*
4. *and has size $f(k, d) \cdot |V|^{O(1)}|E|^{O(1)}$ where f is some computable function.*

4 Pseudo-Cores and Pseudo-Sunflowers

The parallel kernelization algorithm described in the previous section has a depth that is linear in the parameter d, the maximum size of any hyperedge in the input hypergraph. The reason for this linear dependency was that, while we managed to reduce not just one but all sunflowers in the hypergraph to their cores in parallel, we had to repeat this "reduce to core" procedure d times – and each round adds a constant number of layers to the circuit.

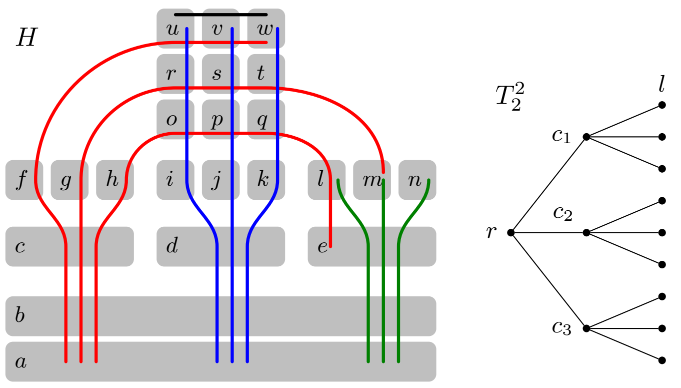
It is not obvious how this build-up of layers can be avoided. In the following, we first explain why there are good reasons to believe that the computation of the hypergraphs H_i necessitates deeper and deeper circuits. Following this discussion, we explain our proposal for side-stepping these difficulties: we replace the hypergraphs H_i by new hypergraphs H'_i that are easier to compute but still form a matryoshka sequence and – hence – can serve as a replacement for the H_i in the Kernel Theorem, Theorem 3.5.

The Difficulty: Cores of Cores Are Hard to Compute. There are several reasons to believe that one cannot compute kernels for the hitting set problem in constant depth using the repeated sunflower-reduction-procedure. A first idea for reaching a constant depth is to apply the reduction procedure only a constant number of times (instead of d times). Indeed, it is not immediately clear that a "core of cores" is not already a core in the first round – so do we actually need more than *one* round? Unfortunately, the answer is "yes, we do": Figure 1 shows an example where $\{a, b\}$ is a 2-core of the 2-cores, but it is not a 2-core of the original hypergraph. For a more complex example, where $d - 1$ rounds are needed to arrive at a constant size kernel, consider the trees T_d^ℓ (defined in detail later on) that are perfectly balanced trees of depth d with $\ell + 1$ children per node for a number $\ell \geq k$ – and now consider the hypergraph H^d that has one hyperedge for each node of T_d^ℓ and this hyperedge contains all the nodes on the path from the node to the root r. Now, for $i > 0$ we have k-cores$(H^i) = H^{i-1}$ and the latter hypergraphs all have a size of at least the arbitrarily large ℓ for $i > 1$. Thus, we need to apply the "core of cores" procedure at least $d - 1$ times before arriving at a hypergraph whose size depends only on the parameter.

A second, more promising idea is the observation that it might be possible to somehow "collapse" two (and then, hopefully, all) applications of the sunflower-reduction-procedure "into a single application." Unfortunately, we also run into a problem here, namely in the "collapsed color coding process." In essence, color coding is great at ensuring that certain vertex sets are disjoint (namely those vertex sets that receive different colors), but fails at enforcing that the same vertices are used in different hyperedges – which is exactly what is needed when the definition of some H_i refers to H_{i-1}, which in turn refers to some H_{i-2}.

These problems with avoiding the build-up of additional layers with rising d have led Chen et al. [11] to the conjecture that the build-up is unavoidable and that all parallel kernelization algorithms for $p_{k,d}$-HITTING-SET have a runtime that is linear in d. We agree with Chen et al. in their assessment that the computation of the H_i presumably necessitates a linear circuit depth – but, nevertheless, we will refute their conjecture in the following.

The Solution: Pseudo-Cores As a Replacement For Cores. Our idea is *not* to compute the sets H_i (we do not see how this can be done in constant time), but to compute hypergraphs H'_i with rather similar properties (formally, they will form matryoshka sequences as well) that we *can* compute in constant time for all d and i. We introduce a new notion of *k-pseudo-cores of level i* and H'_i will be the hypergraph whose edges are the k-pseudo-cores of level i. Crucially, the definition of H'_i (only) refers directly to the original input graph H and its



$S(l,0)$	$S(l,1)$	$S(l,2)$
$\{a,b\}$	$\{c,f\}$	$\{u,v,w\}$
$\{a,b\}$	$\{c,g\}$	$\{r,s,t,m\}$
$\{a,b\}$	$\{c,h,o\}$	$\{p,q,l,e\}$
$\{a,b\}$	$\{d,i\}$	$\{o,r,u\}$
$\{a,b\}$	$\{d,j\}$	$\{p,s,v\}$
$\{a,b\}$	$\{d\}$	$\{k,q,t,w\}$
$\{a,b\}$	$\{e\}$	$\{l\}$
$\{a,b\}$	$\{e\}$	$\{m\}$
$\{a,b\}$	$\{e\}$	$\{n\}$

Figure 2 A T_2^2-pseudo-sunflower S for the level 2 pseudo-core $\{a,b\}$ in the hypergraph H. The four properties of pseudo-sunflowers hold: In "column $S(l,0)$" we always have the pseudo-core, the union of each row is a hyperedge, the sets in a row form a partition of this hyperedge, and – most importantly – we have the disjointness property at each "branch" of the tree. This property requires that for column $S(l,1)$ the sets of all red vertices, of all blue vertices, and of all green vertices are pairwise disjoint; whereas for column $S(l,2)$ it requires that the three red sets are pairwise disjoint, likewise for the three blue sets, and the three green sets. However, it is permissible (and the case) that a red vertex in the third column is the same as green vertex in the third or the second column.

hyperedges can be obtained from H directly using color coding. At the same time, the H_i' will form a matryoshka sequence and, hence, just as for the H_i, the core of any sunflower of H_{i-1}' must already be present in H_i'.

The definition of pseudo-cores is somewhat technical. We will, however, show that all cores are pseudo-cores of level 1, cores of cores are pseudo-cores of level 2, and so on. The reverse implication does not hold (for instance, pseudo-cores of level 2 need not be cores of cores). For a "level" L and a number k, let T_L^k denote the rooted tree in which all leafs are at the same depth L and all inner nodes have exactly $k+1$ children. The root of T_L^k will always be called r in the following. Thus, T_1^k is just a star consisting of r and its $k+1$ children, while in T_2^k each of the $k+1$ children of r has $k+1$ new children, leading to $(k+1)^2$ leafs in total. For each $l \in \text{leafs}(T_L^k) = \{\, l \mid l \text{ is a leaf of } T_L^k \,\}$ there is a unique path $(l^0, l^1, \dots, l^L)$ from $l^0 = r$ to $l^L = l$. An example for the following definition is shown in Figure 2.

▶ **Definition 4.1** (Pseudo-Sunflowers and Pseudo-Cores). Let $H = (V,E)$ be a hypergraph and let L and k be fixed. A set $C \subseteq V$ is called a *k-pseudo-core of level L in H* if there exists a mapping $S\colon \text{leafs}(T_L^k) \times \{0,1,\dots,L\} \to 2^V$, called a *$T_L^k$-pseudo-sunflower for H with pseudo-core C*, such that for all $l, m \in \text{leafs}(T_L^k)$ with $l \neq m$ we have:

1. $S(l,0) = C$.
2. $S(l,0) \cup S(l,1) \cup \dots \cup S(l,L) \in E$ and let us write $S(l)$ for this hyperedge.
3. $S(l,i) \cap S(l,j) = \emptyset$ for $0 \le i < j \le L$, but $S(l,i) \neq \emptyset$ for $i \in \{1,\dots,L\}$.
4. Let $z \in \{1,\dots,L\}$ be the smallest number such that $l^z \neq m^z$, that is, z is the depth where the path from r to l and the path from r to m diverge for the first time. Then $S(l,z) \cap S(m,z) = \emptyset$ must hold.

▶ **Definition 4.2.** For a hypergraph $H = (V,E)$ and numbers k and $i \ge 1$ let $H_i' = \bigl(V, \{C \mid C \text{ is a } k\text{-pseudo-core of level } i \text{ of } H\}\bigr)$ and let $H_0' = H$.

To get some intuition, let us have a closer look at H_1'. As the following lemma shows, pseudo-cores and cores are still *very* closely related at this first level – while for larger levels, we no longer have $H_i = H_i'$, but only $H_i \subseteq H_i'$.

▶ **Lemma 4.3.** *Let H be a hypergraph and k a number. Then $H_1 = H'_1$.*

Sketch of Proof: For $L = 1$, the properties of a pseudo-sunflower enforce exactly that the unions $S(l,0) \cup S(l,1)$ form petals of a sunflower with core $S(l,0)$. ◀

5 The Constant-Depth Kernelization

We show that hitting set kernels can be computed in constant depth in two steps:

1. We show that $(H'_0, \dots, H'_{d(H)})$ is a matryoshka sequence.
2. We show that all H'_i can be computed by a constant depth circuit whose depth is independent of both k and $d(H)$.

By the Kernel Theorem, Theorem 3.5, taken together, these two items yield the desired kernelization algorithm.

Step 1: Pseudo-Cores Form Matryoshka Sequences. Our first aim is to show the following theorem, which is an analogue of Lemma 3.4 for pseudo-cores:

▶ **Theorem 5.1.** *For every hypergraph H and number k, the sequence $(H'_0, \dots, H'_{d(H)})$ from Definition 4.2 is a matryoshka sequence for H and k.*

The proof consists of four lemmas, one for each of four properties of a matryoshka sequence:

▶ **Lemma 5.2.** *$H'_0 = H$.*

Proof. By definition. ◀

▶ **Lemma 5.3.** *$d(H'_L) \leq d(H) - L$ holds for all $L \in \{0, \dots, d(H)\}$.*

Proof. For every leaf l we have $S(l) = S(l,0) \mathbin{\dot\cup} S(l,1) \mathbin{\dot\cup} \cdots \mathbin{\dot\cup} S(l,L)$ and all $S(l,i)$ for $i \in \{1, \dots, L\}$ are non-empty sets. This implies that $|S(l,0)| \leq |S(l)| - L \leq d(H) - L$. ◀

▶ **Lemma 5.4.** *k-cores$(H'_L) \subseteq H'_{L+1}$ holds for all $L \in \{0, \dots, d(H)\}$.*

Sketch of Proof. Proof by induction over L, where the base case is given by Lemma 4.3. For the inductive step, consider a k-core $C \in H'_L$, which is witnessed by a sunflower that consists of $k+1$ different T^k_L-pseudo-sunflowers. From these we construct a T^k_{L+1}-pseudo-sunflower with pseudo-core C and conclude $C \in H'_{L+1}$. ◀

▶ **Lemma 5.5.** *Every size-k hitting set of H is also a size-k hitting set of H'_L for all $L \in \{0, \dots, d(H)\}$.*

Sketch of Proof. Given a size-k hitting set X, say that it *hits a node n of T^k_L* if there is a leaf $l \in \text{leafs}(T^k_L)$ and a depth i with $n = l^i$ such that $X \cap (S(l,0) \cup \dots \cup S(l,i)) \neq \emptyset$. With this definition, X trivially hits all leafs of T^k_L. By the fourth property of a pseudo-sunflower, if X hits all children of a node, it also hits the node. By structural induction we get that the root $r = l^0$ gets hit and, thus, $\emptyset \neq X \cap S(L,0) = X \cap C$. ◀

Step 2: Pseudo-Cores Can Be Computed in Constant Depth. Theorem 5.1 states that the hypergraphs H'_i form a matryoshka sequence and, thus, the Kernel Theorem tells us that the following hypergraph is a kernel for the hitting set problem:

$$K = (H'_0 \ominus H'_1) \cup (H'_1 \ominus H'_2) \cup \cdots \cup (H'_{d(H)-1} \ominus H'_{d(H)}) \cup H'_{d(H)}.$$

Of course, the whole effort that went into the definition of the H'_i and the proof of the matryoshka properties would be for nothing, if the H'_i were not easier to compute than the H_i.

This is exactly what we claim in the following theorem and prove in the rest of this paper: It is an analogue of Lemma 3.6 for pseudo-cores. The crucial difference in the formulation is that, now, we no longer get H'_{i-1} as input when we compute H'_i, but rather we compute H'_i "directly" from the original graph H.

▶ **Theorem 5.6** (Computing Pseudo-Cores in Constant Depth). *There is a* DLOGTIME*-uniform family of* AC*-circuits that*

1. *on input of the incidence matrix of a hypergraph $H = (V, E)$ and numbers k and L,*
2. *outputs the refinement matrix encoding of H'_L,*
3. *has constant depth (in particular, it is independent of $|V|$, $|E|$, $d(H)$, k, and L), and*
4. *has size $f(k, d(H)) \cdot |V|^{O(1)}|E|^{O(1)}$ where f is some computable function.*

To compute the encoding of H'_L, we can consider all candidate pseudo-cores in parallel. Thus, proving the theorem boils down to deciding for a subset $C \subseteq V$ whether there exists a T^k_L-pseudo-sunflower S of H whose pseudo-core is C. Of course, we wish to use color coding for this and our definition of pseudo-cores and pseudo-sunflowers was carefully crafted so that it includes only requirements of the form "these parts of these hyperedges must be disjoint" (and not – as is necessary for describing cores of cores – statements like "these hyperedges must *share* the vertices that form petals"). Unfortunately, while we no longer need to *ensure* that certain parts of different hyperedges are identical, we must be careful that we do not inadvertently *forbid* vertices to be the same across hyperedges when we "do not care whether they are the same":

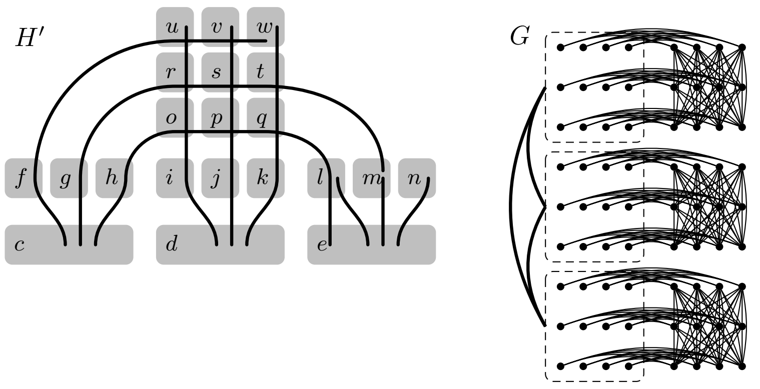
▶ **Example 5.7.** Suppose we wish to find two disjoint hyperedges $e_1 = \{v_1, v_2, v_3\}$ and $e_2 = \{v_4, v_5, v_6\}$ in a hypergraph H plus another hyperedge $e_3 = \{x, y\}$ such that $x \notin e_1 \cup e_2$, but do not care whether $y \in e_1 \cup e_2$ holds or not. We can easily enforce the disjointness properties by coloring v_1 to v_6 using colors 1 to 6 and x using color 7. However, how should we color y for which *we do not care about disjointness* (at least with respect to e_1 and e_2)? Fixing any of the colors 1 to 3 for y or any of the colors 4 to 6 (or, for that matter, any other color) would be wrong, since this would enforce either $y \notin e_2$ or $y \notin e_1$ (or both).

Fortunately, there is a way out of the dilemma: we consider all feasible colors y could get in parallel. To formalize this "trick", we define a technical problem in which an undirected graph G is used to specify which vertices in hyperedges of a hypergraph H should be different. As is customary, a *proper coloring* of an undirected graph $G = (U, F)$ is a mapping $c \colon U \to C$ to some set C of colors with $c(u) \neq c(v)$ for all $\{u, v\} \in F$. Let us write $f[X] = \{f(x) \mid x \in X\}$ for the image of a set X under a function f. For an example instance see Figure 3.

▶ **Problem 5.8.** p_G-RESTRICTED-COLORING

Instance: *A hypergraph $H = (V, E)$ and an undirected graph $G = (U, F)$ together with a partition $U = U_1 \dot\cup \cdots \dot\cup U_m$ of U.*

Parameter: $|G|$



Proper coloring

c	f	f	f	u	v	w	w
c	g	g	g	r	s	t	m
c	h	o	o	p	q	l	e
d	i	i	i	o	r	u	u
d	j	j	j	p	s	v	v
d	d	d	d	k	q	t	w
e	e	e	e	l	l	l	l
e	e	e	e	m	m	m	m
e	e	e	e	n	n	n	n

Figure 3 An instance of p_G-RESTRICTED-COLORING consisting of a hypergraph H' and a graph G (a thick edge connecting two areas with dashed borders indicates that there is an edge between each vertex of the first area and each vertex of the second area; thus, in the example, each thick edge corresponds to $12 \cdot 12 = 144$ edges). This instance is the one resulting from the reduction described in the proof of Theorem 5.6 for $L = 2$, the hypergraph H from Figure 1, and the core $\{a, b\}$ (except that we use only four vertices in G per set $S(l, i)$ instead of $d = 9$). A proper coloring is shown right (the table indicates the values $c(u) \in V(H')$ for the corresponding vertices u of G).

Question: *Is there a proper coloring $c\colon U \to V$ of G such that $c[U_i] \in E$ holds for all $i \in \{1, \dots, m\}$?*

▶ **Lemma 5.9.** *The problem p_G-*RESTRICTED-COLORING *can be solved by a* DLOGTIME-*uniform family of* AC-*circuits of constant depth and size $f(|G|)|V|^{O(1)}|E|^{O(1)}$ for some computable function f.*

Sketch of Proof. We show that c exists if, and only if, there is a mapping $d\colon V \to \{1, \dots, |U|\}$ with the following two properties (considering all possible proper colorings c' in parallel is the formal implementation of the above-mentioned "trick"):

1. There is a proper coloring $c'\colon U \to \{1, \dots, |U|\}$ of G such that
2. for each $i \in \{1, \dots, m\}$ there is a hyperedge $e_i \in E$ with $|d[e_i]| = |e_i|$ and $d[e_i] = c'[U_i]$.

Having proved this equivalence, we observe that we can determine c' in constant depth via "brute force", as the number of candidates depends only on the parameter. The function d can be found via color coding, since (a) the cardinality of its image depends only on the parameter and since (b) we are only interested in the values of d on the subset of V that is used by c as a color (the size of this subset depends only on the parameter). ◀

Sketch of Proof of Theorem 5.6. For each of the $|E| \cdot 2^{d(H)}$ possible pseudo-cores C we reduce the question of whether there is a T^k_L-pseudo-sunflower S with pseudo-core C to an instance for p_G-RESTRICTED-COLORING. The constructed graph G has the vertex set $\text{leafs}(T^L_k) \times \{1, \dots, L\} \times \{1, \dots, d\}$, which means that for each set $S(l, i)$ in the pseudo-sunflower's "table" there are d vertices available (which can then be mapped surjectively to the elements $S(l, i)$). We use the partition of U to ensure that $S(l)$ is always a hyperedge and we insert edges to ensure the disjointness properties of pseudo-sunflowers. ◀

Theorem 5.6 now implies Theorem 1.2 by simple standard arguments.

6 Conclusion

The results of this paper can be summarized as $p_{k,d}$-HITTING-SET $\in$ para-AC^0 or, equivalently, that kernels for the hitting set problem parameterized by k and d can be computed by a single AC^0-circuit family. This result refutes a conjecture of Chen et al. [11]. The proof introduced a new technique: Iterated applications of color coding can sometimes be "collapsed" into a single application. This collapsing is not always straightforward (as the present paper showed) and additional technical machinery may be needed to make it work.

The proof of our main result would be *much* simpler if the number of k-cores of a hypergraph depended only on the parameters k and d (since, then, only one round would be needed in the parallel algorithm). While we gave examples that refute this hope, it might be possible to tweak the idea a bit: We can compute in constant parallel time the set of all *inclusion-minimal* k*-cores* of a hypergraph. We believe that we can prove that the number of these inclusion-minimal k-cores depends only on k and d (unfortunately, we need rather involved and technical combinatorics and the dependence on k and d seems to be "quite bad"). Nevertheless, if this is the case, we get a different proof that $p_{k,d}$-HITTING-SET has an AC^0-kernelization, where the complexity of proving correctness is shifted away from the algorithm (which gets much simpler) towards the underlying graph theory and combinatorics (which get more complex).

References

1 Faisal N. Abu-Khzam, Michael A. Langston, Pushkar Shanbhag, and Christopher T. Symons. Scalable parallel algorithms for FPT problems. *Algorithmica*, 45(3):269–284, 2006. doi:10.1007/s00453-006-1214-1.

2 Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *J. ACM*, 42(4):844–856, 1995. doi:10.1145/210332.210337.

3 Max Bannach, Christoph Stockhusen, and Till Tantau. Fast parallel fixed-parameter algorithms via color coding. In Thore Husfeldt and Iyad A. Kanj, editors, *10th International Symposium on Parameterized and Exact Computation, IPEC 2015, September 16-18, 2015, Patras, Greece*, volume 43 of *LIPIcs*, pages 224–235. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015. doi:10.4230/LIPIcs.IPEC.2015.224.

4 Max Bannach and Till Tantau. Parallel multivariate meta-theorems. In Jiong Guo and Danny Hermelin, editors, *11th International Symposium on Parameterized and Exact Computation, IPEC 2016, August 24-26, 2016, Aarhus, Denmark*, volume 63 of *LIPIcs*, pages 4:1–4:17. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016. doi:10.4230/LIPIcs.IPEC.2016.4.

5 Max Bannach and Till Tantau. Computing hitting set kernels by AC^0-circuits. Technical Report arxiv:1801.00716 [cs.CC], ArXiv e-prints, 2018. URL: http://arxiv.org/abs/1801.00716.

6 David A. Mix Barrington, Neil Immerman, and Howard Straubing. On uniformity within nc^1. In *Proceedings: Third Annual Structure in Complexity Theory Conference, Georgetown University, Washington, D. C., USA, June 14-17, 1988*, pages 47–59. IEEE Computer Society, 1988. doi:10.1109/SCT.1988.5262.

7 Liming Cai, Jianer Chen, Rodney G. Downey, and Michael R. Fellows. Advice classes of parameterized tractability. *Ann. Pure Appl. Logic*, 84(1):119–138, 1997. doi:10.1016/S0168-0072(95)00020-8.

8 Marco Cesati and Miriam Di Ianni. Parameterized parallel complexity. In David J. Pritchard and Jeff Reeve, editors, *Euro-Par '98 Parallel Processing, 4th International Euro-*

Par Conference, Southampton, UK, September 1-4, 1998, Proceedings, volume 1470 of *Lecture Notes in Computer Science*, pages 892–896. Springer, 1998. `doi:10.1007/BFb0057945`.

9 Yijia Chen and Jörg Flum. Some lower bounds in parameterized acˆ0. In Piotr Faliszewski, Anca Muscholl, and Rolf Niedermeier, editors, *41st International Symposium on Mathematical Foundations of Computer Science, MFCS 2016, August 22-26, 2016 - Kraków, Poland*, volume 58 of *LIPIcs*, pages 27:1–27:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016. `doi:10.4230/LIPIcs.MFCS.2016.27`.

10 Yijia Chen, Jörg Flum, and Martin Grohe. Bounded nondeterminism and alternation in parameterized complexity theory. In *18th Annual IEEE Conference on Computational Complexity (Complexity 2003), 7-10 July 2003, Aarhus, Denmark*, pages 13–29. IEEE Computer Society, 2003. `doi:10.1109/CCC.2003.1214407`.

11 Yijia Chen, Jörg Flum, and Xuangui Huang. Slicewise definability in first-order logic with bounded quantifier rank. In Valentin Goranko and Mads Dam, editors, *26th EACSL Annual Conference on Computer Science Logic, CSL 2017, August 20-24, 2017, Stockholm, Sweden*, volume 82 of *LIPIcs*, pages 19:1–19:16. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017. `doi:10.4230/LIPIcs.CSL.2017.19`.

12 Michael Elberfeld, Christoph Stockhusen, and Till 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`.

13 P. Erdős and R. Rado. Intersection theorems for systems of sets. *Journal of the London Mathematical Society*, 1(1):85–90, 1960.

14 Jörg Flum and Martin Grohe. Describing parameterized complexity classes. In Helmut Alt and Afonso Ferreira, editors, *STACS 2002, 19th Annual Symposium on Theoretical Aspects of Computer Science, Antibes - Juan les Pins, France, March 14-16, 2002, Proceedings*, volume 2285 of *Lecture Notes in Computer Science*, pages 359–371. Springer, 2002. `doi:10.1007/3-540-45841-7_29`.

15 Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2006. `doi:10.1007/3-540-29953-X`.

16 Rolf Niedermeier and Peter Rossmanith. An efficient fixed-parameter algorithm for 3-hitting set. *J. Discrete Algorithms*, 1(1):89–102, 2003. `doi:10.1016/S1570-8667(03)00009-1`.

17 René van Bevern. Towards optimal and expressive kernelization for d-hitting set. *Algorithmica*, 70(1):129–147, 2014. `doi:10.1007/s00453-013-9774-3`.