

Kernelizing the Hitting Set Problem in Linear Sequential and Constant Parallel Time

Max Bannach

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

Malte Skambath

Department of Computer Science, Kiel University, Germany
malte.skambath@email.uni-kiel.de

Till Tantau

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

Abstract

We analyze a reduction rule for computing kernels for the hitting set problem: In a hypergraph, the *link* of a set c of vertices consists of all edges that are supersets of c . We call such a set *critical* if its link has certain easy-to-check size properties. The rule states that the link of a critical c can be replaced by c . It is known that a simple linear-time algorithm for computing hitting set kernels (number of edges) at most k^d (k is the hitting set size, d is the maximum edge size) can be derived from this rule. We parallelize this algorithm and obtain the first AC^0 kernel algorithm that outputs polynomial-size kernels. Previously, such algorithms were not even known for artificial problems. An interesting application of our methods lies in traditional, non-parameterized approximation theory: Our results imply that uniform AC^0 -circuits can compute a hitting set whose size is polynomial in the size of an optimal hitting set.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases Kernelization, Approximation, Hitting Set, Constant-Depth Circuits

Digital Object Identifier 10.4230/LIPIcs.SWAT.2020.9

1 Introduction

In the theory of fixed-parameter tractability, *kernelization algorithms* play an important role. They shrink input instances to membership-equivalent instances (called *kernels*) whose size depend only on the input's parameters. A rich theory has been developed around this idea, with results ranging from highly efficient kernel algorithms to lower bounds showing that some problems do not have polynomial-size kernels unless complexity class collapses occur [8]. Another recent direction is the question of how quickly kernels can be computed in parallel, and which trade-offs regarding kernel size are incurred by parallelisation [5, 6, 11].

In the present paper we are interested in computing kernels for the *hitting set problem* both, sequentially and in parallel. We study the following version of the problem on d -*hypergraphs*, which are pairs (V, E) consisting of a set V of *vertices* and a set E of *hyperedges* (which we will call just *edges*) such that for all edges $e \in E$ we have $e \subseteq V$ and $|e| \leq d$:

► **Problem 1.1** (p_k - d -HITTING-SET for fixed $d \in \mathbb{N}$).

Instances: A d -hypergraph $H = (V, E)$ and a *parameter* $k \in \mathbb{N}$.

Question: Is there a *size- k hitting set* $X \subseteq V$, that is, $|X| \leq k$ and $X \cap e \neq \emptyset$ for all $e \in E$?



© Max Bannach, Malte Skambath, and Till Tantau;
licensed under Creative Commons License CC-BY

17th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2020).

Editor: Susanne Albers; Article No. 9; pp. 9:1–9:16



Leibniz International Proceedings in Informatics

LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

When d is not fixed, but part of the input, the resulting problem $\text{p}_k\text{-HITTING-SET}$ is complete for $\text{W}[2]$ and, thus, no kernelization is possible unless $\text{W}[2] = \text{FPT}$ [13]. In contrast, $\text{p}_k\text{-}d\text{-HITTING-SET}$ can be kernelized in polynomial time for every d . A standard way of doing so is based on the following reduction rule, where a *sunflower* is a set $S \subseteq E$ such that any two elements of S have the same intersection c_S , called the *core* of the sunflower:

► **Rule 1.2.** Find a sunflower S of size $|S| \geq k + 1$ and replace E by $(E \setminus S) \cup \{c_S\}$.

It is easy to see that this rule is safe and not-so-easy to see that it results in a kernel with at most $d!k^d$ edges (this follows from the Sunflower Lemma [15], by which a sunflower of size $k + 1$ always exists as long as $|E| > d!k^d$). However, finding sunflowers is a difficult problem in itself, requiring either expensive methods like color coding [3] or approximate solutions [18]. Therefore, the current state-of-the-art algorithm due to Fafianie and Kratsch uses the simpler and yet more powerful *critical core rule* [16]:

► **Rule 1.3.** Find a *critical core* $c \subseteq e \in E$ and replace E by $(E \setminus L_c) \cup \{c\}$.

The rule hinges on how we setup the definition of “critical” cores. This will depend on the computational model; in the simplest setup (see Definition 3.2 for the general case), a core c is critical if (1) for all supersets $c' \supsetneq c$ we have $|L_{c'}| \leq k^{d-|c'|}$ and (2) we have $|L_c| > k^{d-|c|}$. Checking this for a given core c is relatively easy, making the rule fairly easy to implement. While a bit of effort is needed to show that the rule is safe, it clearly yields a kernel with k^d edges¹: When it is no longer applicable, the empty set is no longer critical and by (2) we have $|E| = |L_\emptyset| \leq k^{d-0}$.

Less is known in the parallel setting. Only for $d = 2$ (the *vertex cover problem*) do we know anything concerning the parallel computation of polynomial-size kernels: TC^0 -circuits can compute quadratic kernels [14]. A complex argument shows that AC^0 -circuits can compute exponential-size kernels for the hitting set problem [6]. However, AC^0 -circuits were not known to be able to compute polynomial-size kernels for the vertex cover problem (nor, for that matter, for any other problem). For $d > 2$, the best parallel kernelization is the logspace algorithm due to Fafiane and Kratsch [16].

Our Contributions. We rephrase the kernelization algorithm from Fafianie and Kratsch [16] in a terminology that is more suited for changing the computational model on which the algorithm is implemented. As by-product, we obtain a slightly tighter bound on the kernel size: While the original paper achieves a kernel size of $(k + 1)^d$ edges, we obtain a kernel of size k^d . Furthermore, we can construct a kernel of size $\sum_{i=0}^d k^i$ that has the desirable property that it is a subsets of the original edge set. However, the main objective of this paper is a parallel constant-time implementation of the critical core rule:

Contribution I: First Polynomial-Size Constant-Time Kernels. By adjusting the setup when a core is considered critical, we are able to derive the first polynomial-size kernel that can be computed by AC^0 -circuits. It was previously known that *exponential-size* kernels for the hitting set problem can be computed by AC^0 -circuits, but *no* problem (not even an artificial one) was known for which AC^0 -circuits can compute a polynomial-size kernel.

There is more to the construction than just adjusting some thresholds: Any natural implementation of the critical core rule internally needs threshold gates, leading to TC^0 -circuits. To get down to AC^0 , we use *fuzzy* threshold gates, which behave like a normal threshold gate only when the number of input 1-bits is below the threshold or “well above” it. We show that the “fuzziness” of the gates is not a problem when computing kernels.

¹ In this paper, we are only interested in minimizing the number of edges and call it the kernel size. Reducing the number of vertices is also of interest, but not addressed by us. The set V is immutable.

Contribution II: Constant-Time Approximations. There are known connections between kernelization and approximation algorithms [1, 17, 21]. These connections carry over to AC^0 and we can derive an *approximation algorithm* for the hitting set problem that works in constant time: There is a family of AC^0 -circuits that on input of any d -hypergraph $H = (V, E)$ outputs a hitting set X of H such that if X^* is a minimum hitting set of H , then $|X| \in O(|X^*|^d)$. While clearly worse than the best approximation ratio (namely d) achieved sequentially, observe that AC^0 -circuits can only reliably “count up to polylogarithmic numbers,” but we must still compute an approximate hitting set when $|X^*| = n^\epsilon$.

Related Work. The “classical” way of computing kernels for p_k - d -HITTING-SET, due to Flum and Grohe [18, Section 9.1], is based on Erdős and Rado’s [15] Sunflower Lemma. Variations of this algorithm were proposed by van Bevern [24] and by Damaschke [12].

Attempts to parallelise parameterized algorithms date back to the late 1990s [9, 10]. A structural study of parameterized circuit complexity was started around 2015 by Elberfeld et al. [14]. The parameterized circuit model we use within this paper was introduced in [4]. It is known that a decidable problem is in $\text{para-}AC^i$ if, and only if, it has a kernel function in AC^i [6]. A first AC^0 kernelization for p_k - d -HITTING-SET was presented by Chen, Flum, and Huang [11], which was later improved to a AC^0 kernelization for $p_{k,d}$ -HITTING-SET (here, d is parameter and not a fixed constant) [5]. All of these kernels have exponential size.

Approximation algorithms based on kernelizations were studied by Abu-Khzam et al. [1], by Fellow et al. [17] as “fidelity kernels,” and by Lokshtanov et al. [21] as “lossy kernels.”

Structure of this Article. This article has four main sections: In Section 2 we explore the properties of “fuzzy” threshold gates. In Section 3 we review the critical core reduction rule and use it to compute polynomial-size kernels in linear or constant parallel time. In Section 4, we show how our results allow us to develop constant-depth approximation algorithms. We close the article with a look at the set packing problem in Section 5.

2 Technical Tools

On a technical level, we need a few standard notions, briefly reviewed in the following. The technical tool of *fuzzy threshold gates* is presented in more detail.

Circuits. We use standard notions of Boolean circuits: AC -circuits have n input gates and m output gates as well as AND-, OR-, and NOT-gates. The AND-gates and OR-gates may have unbounded fan-in. A TC-circuit may additionally have unbounded fan-in $\text{THRESHOLD}_{\leq t}^{\geq t}$ -gates, which output 0 if at most t of its inputs are set to 1 and which outputs 1 otherwise. For a circuit C and a bitstring $x \in \{0, 1\}^n$ we write $C(x)$ for the length- m bitstring that is output by the circuit on input x .

A problem $L \subseteq \{0, 1\}^*$ is in AC^0 if there is a family $(C_n)_{n \in \mathbb{N}}$ of AC -circuits, where each C_n has n inputs and only one output, such that (1) we have $x \in L$ if, and only if, $C_{|x|}(x) = 1$ and (2) $\text{depth}(C_n) \leq c$ and $\text{size}(C_n) \leq n^c$ for some constant c . (Here, the size and depth functions for circuits can be defined in any sensible way.) We also require that the families are $DLOGTIME$ -uniform, which is the strongest form of uniformity commonly required [7]. The definition of the class TC^0 is analogue. In slight abuse of notation, we also use AC^0 and TC^0 to denote the functional classes (more correctly known as FAC^0 and FTC^0) containing functions $f: \{0, 1\}^* \rightarrow \{0, 1\}^*$ by allowing that the C_n have more than one output gate.

Parameterized Problems. A *parameterized problem* is a set $Q \subseteq \Sigma^* \times \mathbb{N}$. It lies in the class FPT (also known as para-P), if on input (x, k) we can decide whether $(x, k) \in Q$ holds in time $f(k) \cdot |x|^c$ for some computable function f and some constant c . The problem lies in the class para-AC⁰ if there is a family $(C_{n,k})_{n,k \in \mathbb{N}}$ such that (1) we have $(x, k) \in Q$ if, and only if, $C_{|x|,k}(x) = 1$ and (2) we have $\text{depth}(C_{n,k}) \leq c$ and $\text{size}(C_{n,k}) \leq f(k) \cdot n^c$. We also require a DLOGTIME-uniformity condition, which in this context means that the i th bit of a suitable encoding of $C_{n,k}$ can be computed by a deterministic Turing machine that obtains i , n , and k encoded as binary numbers as input and that runs in time $f(k) + O(\log n + \log i)$. The class para-TC⁰ is defined in the same way by additionally allowing threshold gates.

Kernels. A *kernel function for a parameterized problem* Q is a mapping $K: \Sigma^* \times \mathbb{N} \rightarrow \Sigma^* \times \mathbb{N}$ such that for $K(x, k) =: (x', k')$ we have $(x, k) \in Q \Leftrightarrow (x', k') \in Q$, $|x'| \leq s(k)$, and $|k'| \leq \rho(k)$ for two computable functions s and ρ . The function s is called the *size (function)* of the kernel and we are particularly interested in the case that s is a polynomial.² The function ρ is less important and will always be the identity in this paper. It is well-known that for decidable problems Q we have $Q \in \text{para-P}$ if, and only if, Q has a kernel function $K \in \text{FP}$ (that is, computable in polynomial time) [18]; we have $Q \in \text{para-AC}^0$ if, and only if, Q has a kernel function K in the function class AC⁰ (where $(x, k) \in \Sigma^* \times \mathbb{N}$ is suitably encoded as a bitstring) [6]; and that an analogous result holds for para-TC⁰.

Most kernel algorithms are based on *reduction rules*: They take an input $(x, k) \in \Sigma^* \times \mathbb{N}$ and are either not applicable or output some (x', k') with $|x'| < |x|$ and $k' \leq k$. A rule is called *safe* for a problem Q if we always have $(x, k) \in Q \iff (x', k') \in Q$. A set of such rules *yields a kernel* if for any input (x, k) at least one rule is still applicable as long as $|x'| > s(k)$, that is, as long as the input has not yet shrunk to a kernel.

Kernel functions can map inputs (x, k) to some (x', k') that have “very little to do with the original (x, k) ” except for being membership-equivalent, while it is often desirable that kernels should preserve some of the properties of the input. When x is a hypergraph $H = (V, E)$ and the objective is to find a set $X \subseteq V$ with certain properties, we say that a kernel function K *preserves solutions* if (x, k) and $K(x, k)$ always have the same solutions, and it *preserves edges* if the edges in $K(x, k)$ constitute a subset of E . The *full kernels* of Damaschke [12] preserve minimal solutions in our sense and the *explanatory kernels* of van Bevern [24] are edge-preserving kernels in our sense. An edge-preserving kernel does not need to be solution-preserving, but this will always be the case in the present paper.

Fuzzy Thresholds. It is well-known [23] that $\text{THRESHOLD}_{\leq t}^{\geq t}$ -gates can be simulated by AC⁰-circuits for polylogarithmic t , that is, for each exponent c we have $\{1^t 0^b \mid t \in \mathbb{N}, b \in \{0, 1\}^*, \sum_{i=1}^{|b|} b[i] \leq t \leq \log_2^c |b|\} \in \text{AC}^0$ and TC⁰-families using only polylogarithmic thresholds can be replaced by equivalent AC⁰-families. However, $\text{MAJORITY} \in \text{TC}^0 \setminus \text{AC}^0$ shows that for $t = \lfloor n/2 \rfloor$ an AC⁰-circuit simulating a $\text{THRESHOLD}_{\leq t}^{\geq t}$ -gate would need superpolynomial size [19]. To step beyond the “polylogarithmic boundary” we use *fuzzy threshold gates*: They behave the same way as ordinary threshold gates when the number of 1-bits in the input is below a threshold t_1 or above a larger threshold $t_2 > t_1$, but no guarantee is made about their behavior in between. Clearly, such gates are less useful than normal threshold gates – the MAJORITY problem demonstrates the importance of precisely distinguishing between $\lfloor n/2 \rfloor$ and $\lfloor n/2 \rfloor + 1$ many 1-bits – but they turn out to be sufficient for the computation of certain kernels. Crucially, *fuzzy threshold gates can be simulated by AC⁰-circuits for superlogarithmic thresholds*, allowing us to turn such “fuzzy TC⁰ algorithms” into AC⁰-circuits.

² As remarked earlier, in this paper “only the edge sets” of hypergraphs will be kernelized since we insisted that the vertex set is immutable; $|x'|$ should in this case be read as $|E'|$.

In detail, just as for the standard class TC^0 , we consider DLOGTIME -uniform circuit families $(C_n)_{n \in \mathbb{N}}$ of constant depth and polynomial size. However, *instead of* $\text{THRESHOLD}_{\leq t}^t$ -gates, the circuits now contain the following gates (in addition to AND-, OR-, and NOT-gates):

► **Definition 2.1.** *On input $b \in \{0, 1\}^l$, a (“fuzzy”) $\text{THRESHOLD}_{\leq t_1}^{> t_2}$ gate g with l inputs outputs one bit $g(b) \in \{0, 1\}$ such that for $s = \sum_{i=1}^l b[i]$ we have:*

1. *If $s \leq t_1$, then $g(b) = 0$.*
2. *If $s > t_2$, then $g(b) = 1$.*

In all other cases, when s exceeds the threshold only “slightly” ($t_1 < s \leq t_2$), no guarantees are made about $g(b)$: it can be 0 or 1.

A fuzzy threshold circuit C_n with m output gates may produce as output any string from a set $C_n^{\text{possible}}(b) \subseteq \{0, 1\}^m$ of possible outputs, depending on how the fuzzy gates happen to behave. The objective is, of course, that no matter how the fuzzy gates actually behave and no matter which $z \in C_n^{\text{possible}}(b)$ we actually get, it will be a valid “solution” for the given input b . A bit more formally, we define a *solution relation* as a relation $S \subseteq \{0, 1\}^* \times \{0, 1\}^*$ such that for each possible input $b \in \{0, 1\}^*$ the set $S(b) := \{s \mid (b, s) \in S\}$ of (“allowed” or “permissible”) *solutions* is a subset of $\{0, 1\}^{p(|b|)}$ for some fixed polynomial p . As an example, consider the task of finding quadratic kernels for the vertex cover problem. We model this by an S containing all pairs (b, s) where b is (the encoding of) a graph G and a number k and s is (the encoding of) a graph K of size at most k^2 such that G has a size- k vertex cover if, and only if, K does.

► **Definition 2.2.** *Let S be a solution relation. We say that a family $(C_n)_{n \in \mathbb{N}}$ of fuzzy threshold circuits computes solutions for S if $C_n^{\text{possible}}(b) \subseteq S(b)$ holds for all $b \in \{0, 1\}^*$.*

We derive rather tight fuzzy threshold gates from an result of Ajtai about approximate counting with first-order formulas over arithmetic structures [2]:

► **Fact 2.3** (Theorem 2.1 in [2]). *For each positive integer i there exists an $\text{FO}[+, \times]$ formula $\varphi(x, X)$ with free variable x and free unary set variable X such that we have for each relational structure \mathcal{S} over size- n universe U , each threshold $t \in U$, and each set $A \subseteq U$:*

$$\begin{aligned} |A| \leq (1 - (\log n)^{-i}) \cdot t &\implies \mathcal{S} \models \neg \varphi(t, A), \\ |A| \geq (1 + (\log n)^{-i}) \cdot t &\implies \mathcal{S} \models \varphi(t, A). \end{aligned}$$

► **Corollary 2.4.** *Let t be a threshold and $\epsilon > 0$ be a small error term. There is a constant c such that for each n -input circuit C with $\text{THRESHOLD}_{\leq t}^{>(1+\epsilon)t}$ gates there is a circuit C' without such gates (a normal AC-circuit) with*

1. *$\text{depth}(C') = \text{depth}(C) \cdot c$ and $\text{size}(C') = \text{size}(C)^c$ such that*
2. *for all $b \in \{0, 1\}^n$ we have $C'(b) \in C^{\text{possible}}(b)$.*

Proof. Let C be an n -input circuit with an \tilde{n} -input $\text{THRESHOLD}_{\leq t}^{>(1+\epsilon)t}$ gate g . We construct a circuit C' by replacing g with an AC^0 -subcircuit. By Fact 2.3 and the well-known relation that the set of decision problems definable in $\text{FO}[+, \times]$ is exactly DLOGTIME -uniform AC^0 [20], we know that for every $i \in \mathbb{N}$ and every $a \in \mathbb{N}$ there is an AC^0 -circuit \tilde{C} such that for all $w \in \{0, 1\}^{\tilde{n}}$ we have:

$$\begin{aligned} \sum_{i=1}^{|w|} w_i \leq (1 - (\log \tilde{n})^{-i}) \cdot a &\implies \tilde{C}_{|w|}(w) = 0, \\ \sum_{i=1}^{|w|} w_i \geq (1 + (\log \tilde{n})^{-i}) \cdot a &\implies \tilde{C}_{|w|}(w) = 1. \end{aligned}$$

In order to replace the gate g with an AC^0 -circuit, observe that there is a constant $n_0 > 0$ such that $1 - (\log n)^{-1} \geq (1 + \epsilon/2)^{-1}$ and $(1 + \epsilon) \cdot (1 + \epsilon/2)^{-1} \geq 1 + (\log n)^{-1}$ for all $n \geq n_0$. If the number of inputs \tilde{n} of g is smaller than n_0 , we can replace g by a hard-wired constant-size AC^0 -circuit. For $\tilde{n} \geq n_0$, we set $a = (1 + \epsilon/2) \cdot t$ and replace g by the circuit from above. Let $x := \sum_{i=1}^{|w|} w_i$ and observe that $x \leq t$ implies $x \leq t = (1 + \epsilon/2)^{-1} \cdot a \leq (1 - (\log n)^{-1}) \cdot a$ and $x > (1 + \epsilon)t = (1 + \epsilon) \cdot (1 + \epsilon/2)^{-1} \cdot a$ implies $x \geq (1 + (\log n)^{-1}) \cdot a$. ◀

► **Corollary 2.5.** *Let S be a solution relation and let $(C_n)_{n \in \mathbb{N}}$ be a family of fuzzy threshold circuits that compute solutions for S . Let $\text{depth}(C_n) \in O(1)$ and $\text{size}(C_n) \in n^{O(1)}$. Then there is a function $f \in AC^0$ that maps every $b \in \{0, 1\}^*$ to a solution $f(b) \in S(b)$.*

3 Three Ways of Implementing the Critical Core Rule

Recall the “classical” sunflower reduction rule, Rule 1.2, which asks us to find and then replace a sunflower S of size $k + 1$ by its core. The reason this rule is *safe* is that there is no way of hitting all $p \in S$ with k vertices without hitting the core, as all $p \in S$ are disjoint outside the core. In other words, the hypergraph with edge set $S \ominus c := \{p \setminus c \mid p \in S\}$ has no hitting set of size k . The key insight behind the critical core rule is that *there are other hypergraphs that also do not have hitting sets of size k , but are easier to find*: A hypergraph with $|E| > k \cdot \Delta$, where Δ is the maximum vertex degree, cannot have a hitting set of size k . Naturally, the maximum degree Δ of an arbitrary hypergraph is unbounded *a priori*, but we can still turn this observation into a safe reduction rule, namely Rule 1.3.

In the introduction, we left open the details of the central definition of a *critical core*, which we remedy presently. First, it will be useful to call the number $i(c) := d - |c|$ the *index i* of a core c . Our algorithms will typically process cores in increasing order of index, which means in decreasing order of size. Next, a *setup* is a system of bounds – tailored to a specific computational model – that determines which cores are critical. In detail:

► **Definition 3.1 (Setup, Factor).** *A setup (w, u, l) consists of three monotone sequences of positive integers: the weight sequence $w = (w_i)_{i \in \mathbb{N}}$, prescribing a weight for cores of index i , the uncritical bound sequence $u = (u_i)_{i \in \mathbb{N}}$, and the light bound sequence $l = (l_i)_{i \in \mathbb{N}}$. These sequences must satisfy $w_0 = u_0 = l_0 = 1$ and $w_i, u_i \in \{1, \dots, l_i\}$. The factor of a setup is the largest number f such that $u_{i+1} \geq f \cdot l_i$ holds for all $i \in \mathbb{N}$.*

For a core c and a setup (w, u, l) , the three numbers we will be particularly interested in are $w_{i(c)}$, $u_{i(c)}$, and $l_{i(c)}$; and we will write $w(c)$, $u(c)$, and $l(c)$ for them, respectively.

► **Definition 3.2 (Critical Cores).** *Let (w, u, l) be a setup and let c be a core. The weight of L_c is $w(L_c) = \sum_{e \in L_c} w(e)$. The core is light if $w(L_c) \leq l(c)$, otherwise it is heavy. The core is critical if $L_c \setminus \{c\}$ is not empty, all $c' \supseteq c$ are light, and $w(L_c) > u(c)$.*

We spell out what these definitions mean for an exemplary setup: $w_i = 1$ and $u_i = l_i = k^i$. The factor is k since $u_{i+1} = k^{i+1} = k \cdot k^i = kl_i$ holds. The weight $w(L_c)$ of a link L_c is simply $|L_c|$ since all weights are 1. A core c is light if $|L_c| \leq k^{i(c)} = k^{d-|c|}$, and it is critical if for all $c' \supseteq c$ we have $|L_{c'}| \leq k^{d-|c'|}$, but $|L_c| > k^{d-|c|}$ ($L_c \setminus \{c\} \neq \emptyset$ then holds automatically).

► **Lemma 3.3.** *For a factor- k setup, let c be critical. Then $L_c \ominus c$ has no size- k hitting set.*

Proof. Let i be the index of c . Suppose $L_c \ominus c$ had a hitting set $X \subseteq V \setminus c$ of size k . Then $L_c = \bigcup_{v \in X} L_{c \cup \{v\}}$ and, therefore, also $w(L_c) \leq \sum_{v \in X} w(L_{c \cup \{v\}})$. As c is critical by assumption, we have $w(L_c) > u_i$ and $w(L_{c \cup \{v\}}) \leq l_{i-1}$ since $c \cup \{v\} \supseteq c$. However, this yields $u_i < w(L_c) \leq \sum_{v \in X} w(L_{c \cup \{v\}}) \leq \sum_{v \in X} l_{i-1} = kl_{i-1}$, contradicting $u_i \geq kl_{i-1}$, which we assumed (the setup has factor k). ◀

► **Corollary 3.4.** *For any factor- k setup, any size- k hitting set must hit all critical cores. In particular, Rule 1.3 is a safe kernel rule for the hitting set problem.*

A simple, but crucial property of the critical core rule is that it removes all heavy links:

► **Lemma 3.5.** *Exhaustively applying the critical core reduction rule to a d -hypergraph H yields an edge set K without heavy cores. In particular, the empty set will be a light core and $|K| = |L_\emptyset| \leq l_d$.*

Proof. If there is a core that is heavy for K , then there is also a heavy core c of minimal index i – meaning that all $c' \supseteq c$ are light. By definition of c being heavy, $w(L_c) > l_i$. But $l_i \geq u_i$ and all $c' \supseteq c$ are light. Thus, c is critical contrary to the assumption. ◀

The following algorithm and theorem summarizes the above findings.

■ **Algorithm 1** The critical core reduction algorithm, which we run on an hypergraph $H = (V, E)$ for some fixed setup.

```

1  while there is a critical core  $c \subseteq e \in E$  do
2     $E \leftarrow (E \setminus L_c) \cup \{c\}$ 
3  return  $E$ 

```

► **Theorem 3.6.** *For every factor- k setup, Algorithm 1 outputs an edge set K of size $|K| \leq l_d$ that is solution-preserving for the hitting set problem.*

Clearly, the smaller l_d , the smaller our kernels. Since by Definition 3.1 we need to ensure $l_{i+1} \geq u_{i+1} \geq kl_i$, the smallest l_d is obtained when we set $u_i = l_i = k^i$. Interestingly, the weights w_i are not relevant yet (we will need them later on) and can be chosen arbitrarily between 1 and $l_i = k^i$.

3.1 Computing Kernels in Linear Time

In this section we discuss a linear time implementation of Algorithm 1, similar to the one provided by Fafianie and Kratsch [16]. However, we differ in two regards: First, the cited implementation directly computes an edge-preserving kernel of size $(k+1)^d$, while we compute a solution-preserving kernel of size k^d . Secondly, we introduce another rule – *the critical core expansion rule* – that allows us to transform the previous computed kernel to an edge-preserving kernel of size $(k+1)^d$. This approach turns out to be slightly more complicated than the implementation of Fafianie and Kratsch, but it allows a straight forward parallelization, which we discuss in the following sections.

It is not too hard to implement a *single application* of the critical core rule in time $|E| \cdot 2^d \text{poly}(d)$: Iterate over all $e \in E$ and for each $c \subseteq e$ increase a counter $n[c]$ by the edge's weight. Determine a maximal c with $w(L_c) = n[c] > u_i$ in a second loop (i is the index of c) and then apply the rule. Since each application of the rule reduces the number of edges by at least 1, we get a total runtime of $|E|^2 \cdot 2^d \text{poly}(d)$ to compute the kernel.

To improve the runtime to $|E| \cdot 2^d \text{poly}(d)$, we need a new definition: For a d -hypergraph $H = (V, E)$, let us call a set K of subsets of V *light* if in the hypergraph (V, K) all edges are light, and we say that K *can be obtained from E* if there is a sequence E_0, E_1, \dots, E_q with $E_0 = E$ and $E_q = K$ such that each E_{j+1} is obtained from E_j through one of two actions:

1. We can set $E_{j+1} = E_j \cup \{c\}$ if c is critical in (V, E_j) .
2. We can set $E_{j+1} = E_j \setminus \{e\}$ if there is a $c \subsetneq e$ with $c \in E_j$.

The critical core rule is now the special case where we always do the first action for some critical c , immediately followed by doing the second action for all $e \supseteq c$, that is, of all of L_c (except for c itself, of course).

► **Theorem 3.7.** *For any factor- k setup, there is a $|E| \cdot 2^d \text{poly}(d)$ time algorithm that, on input of a d -hypergraph $H = (V, E)$, obtains a size- l_d solution-preserving kernel K for the hitting set problem. In particular, for $l_i = k^i$, a kernel of size k^d can be computed in time $|E| \cdot 2^d \text{poly}(d)$.*

Proof. The algorithm iterates over all indices $i \leftarrow 1, \dots, d$ in d rounds. At the start of round i , all cores c of index $i' < i$ will be light and the objective is to ensure that at the end of the round there is no heavy core c of index i . Towards this aim, we wish to modify E (compute E_{j+1} from the current E_j , but let us just write that we “modify E ”) by

1. *removing* all edges in $R := \{e \in E \mid e \supseteq c \text{ for some critical set } c \text{ of index } i\}$,
2. *adding* all edges in a set $A_0 \subseteq A := \{c \mid c \text{ is critical and has index } i\}$ such that
3. A_0 has the properties $|A_0| \leq |R|$ and $R = \{e \in E \mid \exists c \in A_0 : e \supseteq c\}$.

Suppose we could find A_0 and R efficiently. Then we can, indeed, add all of A_0 and then remove all of R in accordance with the two rules: All $c \in A_0$ are initially critical and stay this way when we add other $c' \in A_0$ of the same size to E ; and we can then safely remove all of R as all $e \in R$ are proper supersets of some $c \in A_0$ that we have just added. Finally note that after we have added A_0 and removed R , there are no heavy c of size $|c| = r$ for the resulting set E : If c with $|c| = r$ were still heavy in j , then c would also have been heavy in the original E prior to the modifications and, thus, also critical. But, then, $e \in R$ would have been true for all e in c 's link in E and, thus, all these e would have been removed. Putting it all together, we see that we can, indeed, use the updated E for the next round.

It remains to show how R and A_0 can be found. First, we iterate over all $e \in E$ and all $c \subseteq e$ with $|c| = i$ and for each such c increment a counter $n[c]$ by $w_{d-|e|}$. Note that at the end of the first loop we have $n[c] = w(L_c)$ for all such c . Second, we once more iterate over all $e \in E$ and for each of them we now check whether there is a $c \subsetneq e$ of index i such that $n[c] > u_{i-1}$. If so, we add e to R and we mark *one* such c as to belonging to A_0 . Clearly, at the end of the second loop we will have correctly computed R and a set A_0 of critical edges with $|A_0| \leq |R|$ and $R = \{e \in E \mid \exists c \in A_0 : e \supseteq c\}$.

The runtime of the algorithm is $|E| \cdot 2^d \text{poly}(d)$, assuming an efficient implementation of the array of counters $n[c]$: In the round for core size r , we iterate twice over at most $|E|$ edges e and each time consider at most $\binom{d}{i}$ subsets $c \subsetneq e$. Removing R and adding A_0 takes time linear in their sizes. Finally, $|A_0| \leq |R|$ ensures that $|E|$ can only shrink in each round, yielding a total runtime of at most $\sum_{i=0}^{d-1} \binom{d}{i} |E| \cdot \text{poly}(d) = |E| \cdot 2^d \text{poly}(d)$ as claimed. ◀

While we can now compute solution-preserving kernels K in time $|E| \cdot 2^d \text{poly}(d)$, the kernels are *not yet* edge-preserving: many $e \in K$ will not be elements of the original edge set E . In the following we present an *expansion* rule that can be used to turn K into an edge-preserving kernel. Recall that the critical core *reduction* rule replaces E by $(E \setminus L_c) \cup \{c\}$ for a critical c . The “reverse” version replaces K by $(K \setminus \{c\}) \cup L'_c$, where $L'_c \subseteq L_c$ is a subset large enough to ensure that c becomes critical.

► **Rule 3.8** (Critical Core Expansion Rule). In a d -hypergraph $H = (V, E)$, let $c \notin E$ be critical and let $E' = (E \setminus L_c) \cup \{c\}$. Determine an $L' \subseteq L_c$ with $l(c) \geq w(L') > u(c)$ and replace E' by $E'' := (E' \setminus \{c\}) \cup L'$.

(If $c \in E$ is critical, the critical core reduction rule just removes all supersets of c from E . In this case we also consider the expansion rule to be applicable and set $E'' = E'$.)

► **Lemma 3.9.** *For every setup with $w_i = l_i$, if we apply the critical core expansion rule to E' for some c , then $w(E'') \leq w(E')$ will hold.*

Proof. $w(E'') = w(E') - w(c) + w(L')$ and $w(c) = l(c) \geq w(L')$. ◀

► **Lemma 3.10.** *For every factor- k setup with $w_{i-1} + u_i \leq l_i$, if $c \notin E$ is critical for E , then there is a set L' such that the critical core expansion rule is applicable to the set $E' = (E \setminus L_c) \cup \{c\}$ and E'' will have the same size- k hitting sets as E (and E').*

Proof. Let c be critical for E and have index i . The set L' can be obtained from L_c by iteratively adding elements of L_c to L' until we have $w(L') > u(c)$ for the first time (at the latest when all of L_c has been added). Let $e \supseteq c$ be the last element added to L' . Then $w(L' \setminus \{e\}) \leq u(c) = u_i$ and $w(L') = w(L' \setminus \{e\}) + w(e) \leq u_i + w_{i-1} \leq l_i$. To see that E' and E'' have the same size- k hitting sets, observe that each hitting set of E' must hit $c \in E'$ and is thus also a hitting set of E'' . For the other direction note that c is critical in E'' : No $c' \supseteq c$ can be heavy in E'' since no such c' was heavy in $E \supseteq E''$. With c being critical in E'' , any size- k hitting set of E'' must hit c and, thus, all of E' and thus also all of E . ◀

► **Theorem 3.11.** *For every factor- k setup with $w_i = l_i$ and $w_{i-1} + u_i \leq l_i$, there is an algorithm running in time $|E| \cdot 2^d \text{poly}(d)$ that on input of a d -hypergraph $H = (V, E)$ outputs a kernel K of size l_d that is edge- and solution-preserving for the hitting set problem.*

Proof of Theorem 3.11. By the conditions we impose on the weights, the two lemmas tell us that any application of the critical core reduction rule can be reversed by expansion (Rule 3.8). In particular, an algorithm can use the rule to “undo” the replacement of E by $E' := (E \setminus R) \cup A_0$ by finding a set L' for each $c \in A_0$. Observe that we can remove all of A_0 from E' (except for those $c \in A_0$ that were already present in E) and add an appropriate L' for each such $c \in A_0$ to, still, ensure that all $c \in A_0$ are still critical. ◀

The smallest setup that satisfies the conditions of the theorem is $w_i = l_i = (k+1)^i$ and $u_i = k(k+1)^{i-1}$ for $i \geq 1$. Thus, the theorem tells us that we can compute edge-preserving kernels of size $(k+1)^d$ for the hitting set problem in linear time.

We point out that there is a much simpler way of implementing the critical core rule sequentially in order to compute an edge-preserving kernel. Algorithm 2 is essentially the algorithm of Fafianie and Kratsch [16] in our terminology.

■ **Algorithm 2** The critical core filter algorithm. When started on $H = (V, E)$, it will output a kernel $K \subseteq E$ of size at most l_d – provided that the setup satisfies $u_i + w_i \leq l_i$, see Theorem 3.12.

```

1  $K \leftarrow \emptyset$ 
2 for  $e \in E$  do
3   if there is no  $c \subseteq e$  that is critical with respect to  $K$  then
4      $K \leftarrow K \cup \{e\}$ 
5 return  $K$ 

```

It is easy to implement the algorithm so that it runs in time $|E| \cdot 2^d \text{poly}(d)$ by keeping track of the weights of all links in K . Unfortunately, the algorithm does not seem to be suitable for parallelisation. However, analyzing the algorithm with our proposed setups still allows us to bound the kernel size slightly better than it was done by Fafianie and Kratsch [16]:

► **Theorem 3.12.** *For every factor- k setup with $u_i + w_i \leq l_i$, the output of Algorithm 2 is an edge-preserving hitting set kernel for H of size at most l_d .*

Proof. By construction, we have $K \subseteq E$. We have $|K| \leq l_d$, since $c = \emptyset$ is light in K . To see that K is a kernel, consider an $e \in E$ that we do not add to K . Then there must be a core $c \subseteq e$ such that for the link L_c of c in K we have $w(L_c^K) + w(e) > l(c)$ (otherwise we would have added e to K). For $i = i(c)$, we then have $l_i < w(L_c^K) + w_i$ and, by assumption, $u_i \leq l_i - w_i < w(L_c^K)$. This means that c was critical in K and, hence, every size- k hitting set of K also hits c and, thus, in particular also $e \supseteq c$ and there is no need to add e to K . ◀

The slowest growing setup with factor k satisfying $u_i + w_i \leq l_i$ is given by $w_i = 1$ and $u_i = \sum_{j=1}^i k^j = k + k^2 + \dots + k^i$ and $l_i = u_i + 1 = \sum_{j=0}^i k^j = 1 + k + k^2 + \dots + k^i$. Thus, a kernel with $\sum_{j=0}^i k^j$ edges can be achieved, which is slightly better than the previously known bound of $(k + 1)^d$.

3.2 Computing Kernels by Constant-Depth Threshold Circuits

The algorithm from Theorem 3.7 allows an efficient parallel implementation:

► **Theorem 3.13.** *For each d , the hitting set kernels from Theorems 3.7 and 3.11 can be computed by TC^0 -circuits. In particular, TC^0 -circuits can compute solution-preserving kernels of size k^d and edge-preserving kernels of size $(k + 1)^d$ for the hitting set problem for d -hypergraphs.*

Proof. Just observe that a TC^0 -circuit can determine whether $n[c] > u(c)$ holds for a given c without iterating over all $e \in E$ sequentially, but by using a single threshold gate. In particular, we can compute R in parallel and thus also A_0 . This yields a circuit whose depth is linear in d (a constant) and whose size is polynomial in $|E| \cdot 2^d$ and thus polynomial in the input length for constant d . We can also implement the reverse critical link rule using threshold gates since to identify the sets L' from the rule, we just need threshold gates to find the first edge $e \in L_c$ for which the sum of the weights of all edges in L_c prior to this edge together with $w(e)$ exceeds u_i . ◀

We point out that for solution-preserving kernels, the above theorem was already known for $d = 2$ [6], while only the logspace kernelization from Fafiane and Kratsch [16] were known for $d > 2$.

3.3 Computing Kernels by Constant-Depth Fuzzy Threshold Circuits

Our final goal is an implementation of our kernelization using AC^0 -circuits. As pointed out earlier, previously it was not known how kernels of polynomial size can be computed by AC^0 -circuits for *any* problem. The difficulty in computing polynomial-size kernels using AC^0 -circuits lies in the inability of such circuits to count precisely when the parameter k is no longer polylogarithmic. We overcome this problem by using fuzzy threshold gates.

Critical Core Reduction by Fuzzy Thresholds. We wish to replace the TC^0 -circuit family from Theorem 3.13 by a fuzzy one, and then we wish to apply Corollary 2.5 to turn it into an AC^0 -circuit family. The threshold circuits from Theorem 3.13 really need to be precise: If we naively replace all threshold gates by fuzzy ones, it can happen that a critical c is not detected, but a smaller one is – resulting in an incorrect application of the rule. The other way round, it may also happen that the rule is not applied when it actually should.

We solve these problems by using a setup with worse (but still polynomial) bounds. The jump from one bound to the next is so big that the fuzziness is no longer a problem:

► **Theorem 3.14.** *For any $\delta > 1$, factor- k setup with $l_i \geq \delta \cdot u_i$ and $d \in \mathbb{N}$, there is a constant-depth, polynomial-size family $(C_n)_{n \in \mathbb{N}}$ of fuzzy threshold circuits that, on input of a hypergraph $H = (V, E)$ and a number k , outputs a solution-preserving size- l_d hitting set kernel K . In particular, a kernel of size $\delta^{d-1}k^d$ can be computed using $u_i = \delta^{i-1}k^i$ and $l_i = \delta^i k^i$.*

Proof. Let us first reiterate the steps from the proof of Theorem 3.13, but now with a closer look at where and how threshold gates are needed (and with which thresholds).

The idea behind the kernelization was that at the beginning of a round for some index $i \in \{1, \dots, d\}$, all cores of smaller index are light and the objective is to ensure that at the end of the round this is also true for all cores of index i . To ensure this, we identified all c of index i that were critical: For each possible $c \subseteq e \in E$ (of which there can be at most $\binom{d}{i}|E|$ many), a normal TC^0 -circuit would use a single threshold gate at this point to determine whether $w(L_c^E)$ exceeds the threshold u_i . If so, c gets marked and then included in the process by which A_0 and R are determined, but this process does not include any use of threshold gates – it is only the test whether $w(L_c^E) > u_i$ where we need threshold gates.

In the fuzzy setting, we will need to use a fuzzy $\text{THRESHOLD}_{\leq t}^{\delta t}$ -gate to implement the test $w(L_c) > u_i$ using, of course, $t = u_i$. This has the following effects:

1. If $w(L_c^E) \leq u_i = t$ holds, then we safely and correctly identify c as noncritical as the test will always yield 0.
2. If $w(L_c^E) > \delta t = \delta u_i$, then c is safely and correctly identified as critical as the test will always yield 1.
3. If $w(L_c^E)$ is in the fuzzy range, then c is guaranteed to be *critical, but not heavy* (since $\delta u_i \leq l_i$).

The important observation is that in the third item, it is *correct* to apply the critical link rule to c (which we do if the fuzzy threshold outputs 1), but it is not *necessary* to do so since c is light. In particular, after the round for i is done, there will be *no heavy c of index i in E* , which was exactly our objective. ◀

► **Corollary 3.15.** *For every d and any $\epsilon > 0$, there is a function in AC^0 that maps every d -hypergraph and number k to a solution-preserving hitting set kernel of size at most $(1 + \epsilon)k^d$.*

Critical Core Expansion by Fuzzy Thresholds. Like the critical core reduction rule, the critical core expansion rule can be implemented by fuzzy threshold circuits. Recall that Rule 3.8 allowed us to safely replace a set E' of edges, which had been obtained by setting $E' = (E \setminus L_c) \cup \{c\}$, by the set $E'' = (E' \setminus \{c\}) \cup L'$ for any set $L' \subseteq L_c$ of appropriate size – namely for $l(c) \geq w(L') > u(c)$. As we did earlier, using a setup with more “slack” will allow us to replace threshold gates by fuzzy threshold gates here.

► **Theorem 3.16.** *For any $\delta > 1$, factor- k setup with $w_i = l_i$ and $w_{i-1} + \delta u_i \leq l_i$ and $d \in \mathbb{N}$, there is a constant-depth, polynomial-size family $(C_n)_{n \in \mathbb{N}}$ of fuzzy threshold circuits that, on input of a d -hypergraph $H = (V, E)$ and a number k , outputs a size- l_d hitting set kernel K that is edge- and solution-preserving. In particular, a kernel of size $\delta^d(k+1)^d$ can be computed using the setup with $w_i = l_i = \delta^i(k+1)^i$ and $u_i = \delta^{i-1}k(k+1)^{i-1}$.*

Proof. We once more need to look more closely at how threshold gates are used in Theorem 3.13 to obtain $L' \subseteq L_c$: For $L_c = \{e_1, e_2, \dots, e_{|L_c|}\}$ we use $|L_c|$ threshold gates in parallel, each of which tests for some number $j \in \{1, \dots, |L_c|\}$ whether $w(\{e_1, \dots, e_j\}) > u(c)$ holds – and the smallest j for which is the case determines $L' = \{e_1, \dots, e_j\}$. Then,

clearly, $w(L') > u(c)$ and also $w(L') \leq u_i + w_{i-1}$, which will be less than l_i by assumption. Now, in the fuzzy setting, we use fuzzy gates for tests, which will still ensure that $w(L') = w(\{e_1, \dots, e_j\}) > u(c)$ holds; but for the upper bound we can only ensure $w(L') \leq \delta u_i + w_{i-1}$ since as long as $w(\{e_1, \dots, e_j\}) \leq \delta u_i$ holds, the fuzzy threshold gates might still output 0, making us (incorrectly) believe that $\{e_1, \dots, e_j\}$ is not yet heavy enough. Thus, in order for the critical core expansion rule to work, we need $\delta u_i + w_{i-1} \leq l_i$, which is exactly our assumption.

For the setup $w_i = l_i = \delta^i(k+1)^i$ and $u_i = kl_{i-1} = \delta^{i-1}k(k+1)^{i-1}$, observe that it does, indeed, satisfy all properties needed by the different lemmas on which the correctness of the rule is based:

1. It satisfies $w_i \leq l_i$ and $u_i = \delta^{i-1}k(k+1)^{i-1} \leq \delta^i(k+1)^i = l_i$ (needed by Definition 3.1).
2. It has factor k since $u_{i+1} = \delta^i k(k+1)^i = kl_i$ (needed by Lemma 3.3).
3. It has $\delta u_i + w_{i-1} = \delta^i k(k+1)^{i-1} + \delta^{i-1}(k+1)^{i-1} = \delta^i(k+1)^{i-1}(k+1/\delta) < \delta^i(k+1)^i = l_i$ (needed by this theorem). ◀

4 Constant-Time Approximation Algorithms for Hitting Set

Approximation algorithms compute solutions for optimization problems that, while perhaps not optimal, have a size that is at least “close” to the optimum. For a minimization problem like VERTEX-COVER, the ultimate objective is to output a vertex cover X whose size is at most $(1 + \epsilon)|X^*|$, where X^* denotes some optimal solution. It turns out that unless some complexity classes collapse, such near-optimal approximations cannot be computed for the vertex cover problem. The best approximation to date is to compute a maximal matching and to then take all vertices involved in it. This algorithm delivers solutions of size at most $2|X^*|$ and can be implemented using NC²-circuits. However, no approximation algorithm that produces a solution that is at least polynomially bounded in $|X^*|$ was known to be implementable with circuits below NC². In this section we present such an algorithm.

Our strategy is based on a simple observation: The set of vertices in any solution-preserving kernel is already a solution for the original graph and, thus, also an approximation. However, we still have the problem that we do not know the size of the optimal solution and, thus, do not know which number k we should use with our kernel algorithms.

► **Theorem 4.1.** *For each d and $\epsilon > 0$, there are functions f and g that map (the encodings of) d -hypergraphs H to hitting sets of H , such that (let X^* be a minimum hitting set of H):*

1. $f \in \text{TC}^0$ and $|f(H)| \leq d|X^*|^d$;
2. $g \in \text{AC}^0$ and $|g(H)| \leq (1 + \epsilon) \cdot d|X^*|^d$.

Proof. The idea is identical for both claims. On input $H = (V, E)$, we run the following algorithm in parallel for each $k \in \{1, \dots, |V|\}$: Compute a solution-preserving kernel K_k for H using the circuits from Theorem 3.13 for TC⁰ or using those from Corollary 3.15 for AC⁰. Then test whether K_k is actually a hitting set of H (this test can easily be done using AC⁰-circuits). Output the set $\bigcup K_k$ (the set of all vertices mentioned in any edges $e \in K_k$) for the smallest k that passes the test, that is, for which K_k is still a hitting set of K .

Trivially, the outputs of the described circuits will be hitting sets. For the size bounds, observe that all K_k are solution-preserving: They have the same size- k hitting sets as the original hypergraph H . In particular, for $k = |X^*|$, the kernel K_k will have X^* as a hitting set and $\bigcup K_k$ will contain X^* and will thus hit all of H . The size bounds now follow from the size bounds on K_k in Theorem 3.13 and Corollary 3.15. ◀

5 Adapting the Approach for the Set Packing Problem

The dual problem of p_k - d -HITTING-SET is the *set packing problem*, in which we are asked to find k disjoint edges in a hypergraph:

► **Problem 5.1** (p_k - d -SET-PACKING for fixed $d \in \mathbb{N}$).

Instances: A d -hypergraph $H = (V, E)$ and a *parameter* $k \in \mathbb{N}$.

Question: Is there a set $X \subseteq E$ with $|X| \geq k$ such that any different $e, f \in X$ are disjoint?

Kernelizations based on the critical core rule tend to carry over to the set packing problem [22]. It is thus not too surprising that our approach also works for set packings, though there are also some subtleties.

5.1 Computing Set Packing Kernels

Recall that the safety of the critical core rule for the hitting set problem hinged on Lemma 3.3: For critical c , the set $L_c \ominus c$ has no size- k hitting set and, thus, for the question of whether H has a size- k hitting set all the edges in L_c can be represented by c . We show that a similar situation arises for the set packing problem: For the question of whether H has a size- k set packing, all edges in L_c can be represented by c , which gives us an analogue of Corollary 3.4:

► **Lemma 5.2.** *For a setup with factor $d(k-1)$, let c be critical in a d -hypergraph H . Then for every $U \subseteq V$ of size $|U| \leq d(k-1)$ there is a set $x \in L_c \ominus c$ such that $x \cap U = \emptyset$.*

Proof. Let i be the index of c . We may assume that $U \cap c = \emptyset$ holds (otherwise we can just replace U by $U \setminus c$). Now consider the set $I = \bigcup_{v \in U} L_{c \cup \{v\}} \subseteq L_c$. As c is critical by assumption, we have $w(L_c) > u_i$ and also $w(L_{c \cup \{v\}}) \leq l_{i-1}$. This gives us $w(I) \leq \sum_{v \in U} w(L_{c \cup \{v\}}) \leq |U|l_{i-1} \leq d(k-1)l_{i-1} \leq u_i < w(L_c)$. Hence, there must be an edge $e \in L_c \setminus I$, which means $e \cap U = \emptyset$. Then $x = e \setminus c$ is the desired element of $L_c \ominus c$. ◀

► **Lemma 5.3.** *For any setup with factor $d(k-1)$, let c be critical for $H = (V, E)$.*

1. *If $c \neq \emptyset$, then $(E \setminus L_c) \cup \{c\}$ has a set packing of size k if, and only if, E does.*
2. *If $c = \emptyset$, then E has a set packing of size k .*

Proof. For the first item, first note that if $X \subseteq E$ is a set packing (of any size), then $(X \setminus L_c) \cup \{c\} \subseteq (E \setminus L_c) \cup \{c\}$ is a set packing of the same size as X . For the other direction, $X \subseteq (E \setminus L_c) \cup \{c\}$ is a size- k set packing. Clearly, if $c \notin X$, we have $X \subseteq E$ and we are done, so suppose $c \in X$. Consider the set $U := \bigcup_{x \in X \setminus \{c\}} x$ of all vertices mentioned in any edge of X , except for those in c . Then $|U| \leq d(k-1)$ since H is a d -hypergraph and each of the $k-1$ many elements of $X \setminus \{c\}$ contributes at most d vertices to U . By Lemma 5.2, there is a edge $x \in L_c \ominus c$ that is disjoint from U and trivially also from c . This means that $X' = (X \setminus \{c\}) \cup \{x \cup c\}$ is also a set packing of size k and $X' \subseteq E$.

For the second item, we repeat the following instructions k times, starting with $U = \emptyset$ and $X = \emptyset$: Invoke Lemma 5.2 to obtain $x \in L_c \ominus c = L_c = E$ with $x \cap U = \emptyset$ and update $X \leftarrow X \cup \{x\}$ and $U \leftarrow U \cup x$. Note that in invocations of the lemma we have $|U| \leq d(k-1)$, so we always get a fresh $x \in E$ that is disjoint from all previous elements of X . ◀

The lemma states that just as for the hitting set problem, the critical core rule is *safe* – as long as c is nonempty; and when c is empty and critical, we actually *know* that there is a set packing of size k and can output a trivial kernel. The observations prove the following:

► **Theorem 5.4.** *For any setup with factor $d(k-1)$ and input $H = (V, E)$, let K be a kernel computed by (1) applying the critical core rule for nonempty cores as long as possible and (2) possibly setting K to a trivial yes-instance if the empty set becomes critical at some point. Then K has a size- k set packing if, and only if, H has one; and $|K| \leq l_d$.*

The smallest possible setup with factor $d(k-1)$ is of course $u_i = l_i = (d(k-1))^i$, leading to a kernel size of $l_d = (d(k-1))^d \leq d^d k^d$. Since this kernelization works with the *exact same* reduction rule we used for the hitting set problem – and since the special case of a critical empty core is easy to take care of – we deduce the following results:

► **Corollary 5.5.** *For each $d \in \mathbb{N}$ and $\epsilon > 0$, one can compute on input of a d -hypergraph $H = (V, E)$ a set packing kernel*

- *in time $|E| \cdot 2^d \text{poly}(d)$ of size $(d(k-1))^d$ and an edge-preserving one of size $(dk)^d$,*
- *by TC^0 -circuits of size $(d(k-1))^d$ and an edge-preserving one of size $(dk)^d$, and*
- *by AC^0 -circuits of size $(1+\epsilon)(d(k-1))^d$ and an edge-preserving one of size $(1+\epsilon)(dk)^d$.*

5.2 Approximation Algorithms for the Set Packing Problem

When one compares our kernelization results on the hitting set and the set packing problems, it may seem that these problems behave in identical ways and only the sizes of the outputs differ slightly. However, in the approximation setting, the situation is quite different: For the set packing problem, we do not know how we can extract an approximation from a kernel. To appreciate the underlying difficulties, observe that it is not even clear how one can compute in AC^0 a matching for a graph that is known to be a complete bipartite graph with two given shores U and W of identical size k .

We do not know whether it is possible to compute approximate matchings, let alone set packings, using AC^0 -circuits. It is thus a bit surprising that we can, nevertheless, approximate the *size* of optimal matchings and set packings:

► **Theorem 5.6.** *For each d and $\epsilon > 0$, there are functions f and g that map (the encoding of) each d -hypergraphs H to a number such that (let X^* denote a largest set packing of H):*

1. $f \in \text{TC}^0$ and $\frac{1}{d} |X^*|^{1/d} < f(H) \leq |X^*|$.
2. $g \in \text{AC}^0$ and $\frac{1}{(1+\epsilon)d} |X^*|^{1/d} < g(H) \leq |X^*|$.

Proof. On input $H = (V, E)$, we compute in parallel for each $k \in \{1, \dots, |V|\}$ a set packing kernel K_k for H using the circuits from Corollary 5.5. For each kernel we perform a simple test: Is K_k the trivial kernel? (Recall that this is the case when $c = \emptyset$ because critical during the computation of the kernel and, then, we have a “witness” that the original hypergraph has a set packing of size at least k .) We output the largest k that passes this test.

For the upper bounds, note that whatever k we output, we know that there is a set packing of this size in the output. For the lower bounds, we show that whenever $|X^*| > l_d$, then the trivial kernel will be output. Assume that this not the case for K_k . We observe that $X := X^*$ is a set packing of size $|X^*|$ and for any nonempty c , the set $(X \setminus L_c) \cup \{c\}$ is also a set packing of the same size (see the argument at the beginning of the proof of Lemma 5.3). Thus, the final K_k , which arose through a series of applications of $E \leftarrow (E \setminus L_c) \cup \{c\}$ for different, nonempty c , contains some set packing of size $|X^*|$. In particular, $|K_k| \geq |X^*| > l_d$. But, then, $c = \emptyset$ is critical in K_k , contrary to our assumption. We know that all k pass the test “Is K_k the trivial kernel?” for which $|X^*| > l_d$ holds. For the function f we used $l_d = (d(k-1))^d$, so all k with $|X^*|^{1/d}/d > k-1$ pass the test. For g we used $l_d = ((1+\epsilon)d(k-1))^d$, so now all k with $|X^*|^{1/d}/((1+\epsilon)d) > k-1$ pass the test. ◀

6 Conclusion

We analyzed a simple reduction rule for the hitting set problem in a parallel setting: The *critical core rule* states that for any *critical* set c , we can safely replace the link of c by c . Whether or not a set is critical depended only on the weights of links and by varying the thresholds, we got different kernelization algorithms with different properties, see Table 1.

From the perspective of circuit complexity, this paper gives two new insights: First, it is possible to compute polynomial-size kernels for difficult problems using AC^0 -circuits; and second, it is possible to find polynomial-factor approximations for the hitting set problem using AC^0 -circuits.

■ **Table 1** Summary of our results concerning kernels and approximations for the hitting set problem (above) and the set packing problem (below) in d -hypergraphs. The number opt is the size $|X^*|$ of a smallest hitting set or a largest set packing of the input, respectively.

Hitting Set Kernelization Results

<i>Runtime</i>	<i>What is Computed?</i>	<i>Size</i>	<i>Reference</i>
$ E \cdot 2^d \text{ poly}(d)$	solution-preserving hitting set kernel	k^d edges	Theorem 3.7
TC^0	solution-preserving hitting set kernel	k^d edges	Theorem 3.13
AC^0	solution-preserving hitting set kernel	$(1 + \epsilon)k^d$ edges	Corollary 3.15
$ E \cdot 2^d \text{ poly}(d)$	edge-preserving hitting set kernel	$\sum_{j=0}^d k^j$ edges	Theorem 3.12
$ E \cdot 2^d \text{ poly}(d)$	edge-preserving hitting set kernel	$(k + 1)^d$ edges	Theorem 3.11
TC^0	edge-preserving hitting set kernel	$(k + 1)^d$ edges	Theorem 3.13
AC^0	edge-preserving hitting set kernel	$(1 + \epsilon)(k + 1)^d$ edges	Corollary 3.15
NC^2	a hitting set	$d \cdot opt$ vertices	folklore
TC^0	a hitting set	opt^d vertices	Theorem 4.1
AC^0	a hitting set	$(1 + \epsilon)opt^d$ vertices	Theorem 4.1

Set Packing Kernelization Results

<i>Runtime</i>	<i>What is Computed?</i>	<i>Size</i>	<i>Reference</i>
$ E \cdot 2^d \text{ poly}(d)$	set packing kernel	$(d(k - 1))^d$ edges	Corollary 5.5
TC^0	set packing kernel	$(d(k - 1))^d$ edges	Corollary 5.5
AC^0	set packing kernel	$(1 + \epsilon)(d(k - 1))^d$ edges	Corollary 5.5
$ E \cdot 2^d \text{ poly}(d)$	edge-preserving set packing kernel	$(dk)^d$ edges	Corollary 5.5
TC^0	edge-preserving set packing kernel	$(dk)^d$ edges	Corollary 5.5
AC^0	edge-preserving set packing kernel	$(1 + \epsilon)(dk)^d$ edges	Corollary 5.5
NC^2	number z	$\frac{1}{d} opt \leq z \leq opt$	folklore
TC^0	number z	$\frac{1}{d} opt^{1/d} \leq z \leq opt$	Theorem 5.6
AC^0	number z	$\frac{1}{(1+\epsilon)d} opt^{1/d} \leq z \leq opt$	Theorem 5.6

References

- 1 F. Abu-Khazam, C. Bazgan, M. Chopin, and H. Fernau. Approximation algorithms inspired by kernelization methods. In *ISAAC*, 2014. doi:10.1007/978-3-319-13075-0_38.
- 2 M. Ajtai. Approximate counting with uniform constant-depth circuits. In *Advances In Computational Complexity Theory*, 1990.
- 3 N. Alon, R. Yuster, and U. Zwick. Color-coding. *Journal of the ACM*, 42(4), 1995. doi:10.1145/210332.210337.
- 4 M. Bannach, C. Stockhusen, and T. Tantau. Fast Parallel Fixed-parameter Algorithms via Color Coding. In *IPEC*, 2015. doi:10.4230/LIPIcs.IPEC.2015.224.
- 5 M. Bannach and T. Tantau. Computing Hitting Set Kernels By AC^0 -Circuits. In *STACS*, 2018. doi:10.4230/LIPIcs.STACS.2018.9.
- 6 M. Bannach and T. Tantau. Computing Kernels in Parallel: Lower and Upper Bounds. In *IPEC*, 2018.
- 7 D. Barrington, N. Immerman, and H. Straubing. On Uniformity within NC^1 . In *Structure in Complexity Theory*, 1988. doi:10.1109/SCT.1988.5262.
- 8 H. Bodlaender, R. Downey, M. Fellows, and D. Hermelin. On problems without polynomial kernels. *J. Comput. Syst. Sci.*, 75(8), 2009. doi:10.1016/j.jcss.2009.04.001.
- 9 L. Cai, J. Chen, R. Downey, and M. R. Fellows. Advice Classes of Parameterized Tractability. *Annals of Pure and Applied Logic*, 84(1), 1997. doi:10.1016/S0168-0072(95)00020-8.
- 10 M. Cesati and M. Di Ianni. Parameterized parallel complexity. In *Euro-Par*, 1998. doi:10.1007/BFb0057945.
- 11 Y. Chen, J. Flum, and X. Huang. Slicewise Definability in First-Order Logic with Bounded Quantifier Rank. In *CSL*, 2017. doi:10.4230/LIPIcs.CSL.2017.19.
- 12 P. Damaschke. Parameterized Enumeration, Transversals, and Imperfect Phylogeny Reconstruction. *Theo. Comp. Science*, 351(3), 2006. doi:10.1016/j.tcs.2005.10.004.
- 13 R. Downey and M. Fellows. Fixed-parameter tractability and completeness I: basic results. *SIAM J. Comput.*, 24(4), 1995. doi:10.1137/S0097539792228228.
- 14 M. Elberfeld, C. Stockhusen, and T. Tantau. On the space and circuit complexity of parameterized problems: Classes and completeness. *Algorithmica*, 71(3), 2015. doi:10.1007/s00453-014-9944-y.
- 15 P. Erdős and R. Rado. Intersection Theorems for Systems of Sets. *Journal of the London Mathematical Society*, 1(1), 1960.
- 16 S. Fafanie and S. Kratsch. A shortcut to (sun)flowers: Kernels in logarithmic space or linear time. In *MFCS*, 2015. doi:10.1007/978-3-662-48054-0_25.
- 17 M. Fellows, A. Kulik, F. Rosamond, and H. Shachnai. Parameterized approximation via fidelity preserving transformations. *J. Comput. Syst. Sci.*, 93, 2018. doi:10.1016/j.jcss.2017.11.001.
- 18 J. Flum and M. Grohe. *Parameterized Complexity Theory*. Springer, 2006. doi:10.1007/3-540-29953-X.
- 19 M. Furst, J. Saxe, and M. Sipser. Parity, circuits, and the polynomial-time hierarchy. *Mathematical Systems Theory*, 17(1), 1984. doi:10.1007/BF01744431.
- 20 N. Immerman. *Descriptive complexity*. Graduate texts in computer science. Springer, 1999.
- 21 D. Lokshantov, F. Panolan, M. Ramanujan, and S. Saurabh. Lossy kernelization. In *STOC*, 2017. doi:10.1145/3055399.3055456.
- 22 H. Moser. A problem kernelization for graph packing. In *SOFSEM*, volume 5404 of *Lecture Notes in Computer Science*, pages 401–412. Springer, 2009.
- 23 I. Newman, P. Ragde, and A. Wigderson. Perfect hashing, graph entropy, and circuit complexity. In *Structure in Complexity Theory*, 1990. doi:10.1109/SCT.1990.113958.
- 24 R. van Bevern. Towards Optimal and Expressive Kernelization for d -Hitting Set. *Algorithmica*, 70(1), September 2014. doi:10.1007/s00453-013-9774-3.