Towards Optimal and Expressive Kernelization for \(d\)-Hitting Set

René van Bevern

Institut für Softwaretechnik und Theoretische Informatik, TU Berlin, Germany,
rene.vanbevern@tu-berlin.de

Abstract. A sunflower in a hypergraph is a set of hyperedges pairwise intersecting in exactly the same vertex set. Sunflowers are a useful tool in polynomial-time data reduction for problems formalizable as \(d\)-HITTING SET, the problem of covering all hyperedges (of cardinality at most \(d\)) of a hypergraph by at most \(k\) vertices. Additionally, in fault diagnosis, sunflowers yield concise explanations for “highly defective structures”. We provide a linear-time algorithm that, by finding sunflowers, transforms an instance of \(d\)-HITTING SET into an equivalent instance comprising at most \(O(k^d)\) hyperedges and vertices. In terms of parameterized complexity theory, we show a problem kernel with asymptotically optimal size (unless \(\text{coNP} \subseteq \text{NP/poly}\)). We show that the number of vertices can be reduced to \(O(k^{d-1})\) with additional processing in \(O(k^{1.5d})\) time—nontrivially combining the sunflower technique with known problem kernels due to Abu-Khzam and Moser.

1 Introduction

Many practically relevant problems like the examples given below boil down to solving the following NP-hard problem:

\(d\)-HITTING SET

\textbf{Input:} A hypergraph \(H = (V, E)\) with hyperedges of cardinality at most \(d\) and a natural number \(k\).

\textbf{Question:} Is there a hitting set \(S \subseteq V\) with \(|S| \leq k\) and \(\forall e \in E: e \cap S \neq \emptyset\) ?

Examples for NP-hard problems encodeable into \(d\)-HITTING SET arise in the following fields.

1. Fault diagnosis: The task is to detect faulty components of a malfunctioning system. To this end, those sets of components are mapped to hyperedges of a hypergraph that are assumed to contain at least one broken component [1, 14, 21]. By the principle of Occam’s Razor, a small hitting set is then a likely explanation of the malfunction.

2. Data clustering: all optimization problems in the complexity classes \(\text{MIN } F^+ \Pi_1\) and \(\text{MAX } \text{NP}\), including (\(s\)-PLEX) CLUSTER VERTEX DELETION [5, 13] and

\* Supported by the DFG, project DA PA, NI 369/12. The main part of the work was done under DFG project AREG, NI 369/9.
all problems of establishing by means of vertex deletion a graph property characterized by forbidden induced subgraphs with at most \(d\) vertices, can be formalized as \(d\)-HITTING SET \([15]\).

All problems have in common that a large number of \textquote{conflicts} (the possibly \(O(|V|^d)\) hyperedges in a \(d\)-HITTING SET instance) is caused by a small number of components (the hitting set \(S\)), whose removal or repair could fix a broken system or establish a useful property. However, often, not only a solution to a problem is sought, but also a concise explanation of why a solution should be of the given form. In this work, we contribute to this by combining concise explanations with data reduction, wherein our data reduction preserves the possibility of finding optimal solutions and gives a performance guarantee. This kind of data reduction or, more specifically, problem kernels, are a powerful tool in attacking NP-hard problems like \(d\)-HITTING SET \([6, 11]\). Our main ingredient and contribution are efficient algorithms to find sunflowers, a structure first observed by Erdős and Rado \([9]\):

\textbf{Definition 1.} In a hypergraph \((V,E)\), a \textit{sunflower} is a set of petals \(P \subseteq E\) such that every pair of sets in \(P\) intersects in exactly the same set \(C \subseteq V\), called the core (possibly, \(C = \emptyset\)). The size of the sunflower is \(|P|\).

A sunflower with \(k + 1\) petals in a \(d\)-HITTING SET instance yields a concise explanation of why one of the elements in its core should be removed or repaired: For every sunflower \(P\) with at least \(k + 1\) petals, every hitting set of size at most \(k\) contains one of \(P\)'s core-elements, since it cannot contain an element of each of the \(k + 1\) petals. Analyzing the petals of this sunflower could guide the \textquote{causal analysis} of a problem. We illustrate this using an example.

\textbf{Example 1.} Figure 1(a) shows a boolean circuit. It gets as input a 4-bit string \(x = x_1 \ldots x_4\) and outputs a 5-bit string \(f(x) = f_1(x) \ldots f_5(x)\). The nodes drawn as circles represent boolean gates, which output some bit depending on their two input bits. They might, for example, represent the logical operations \textquote{\&} or \textquote{\lor}.

Assume that all output bits of \(f(x)\) are observed to be the opposite of what would have been expected by the designer of the circuit. We want to identify broken
gates. For each wrong output bit $f_i(x)$, we obtain a set $S_i$ of gates for which we know that at least one is broken, because $f_i(x)$ is wrong. That is, $S_i$ contains precisely those gates that have a directed path to $f_i(x)$ in the graph shown in Figure 1(a). We obtain the sets illustrated in Figure 1(b):

$$
S_1 = \{v_1, v_2, w_1\}, \quad S_2 = \{v_2, v_3, w_2\}, \quad S_3 = \{v_3, v_4, w_3\},
$$

$$
S_4 = \{v_3, v_4, w_4\}, \quad S_5 = \{v_3, v_4, w_5\}.
$$

The sets $S_1$ and $S_5$ are disjoint. Therefore, the wrong output is not explainable by only one broken gate. Therefore, we now assume that there are two broken gates and search for a hitting set of size $k = 2$ in the hypergraph with the vertices $v_1, \ldots, v_4, w_1, \ldots, w_5$ and hyperedges $S_1, \ldots, S_5$. The set $(S_3, S_4, S_5)$ is a sunflower of size $k + 1 = 3$ with core $\{v_3, v_4\}$. Therefore, the functionality of gate $v_3$ and $v_4$ must be checked. If, in contrast to our expectation, both gates $v_3$ and $v_4$ turn out to be working correctly, the usefulness of the sunflower becomes even more apparent: it is immediately clear not only that at least three gates are broken, but it is also clear which gates have to be checked for malfunctions next: $w_3$, $w_4$, and $w_5$. □

In addition to fault diagnosis, sunflowers also yield a good tool for data reduction preserving optimal solutions, so that we can remove hyperedges and vertices from the input hypergraph, until it is small enough to be analyzed as a whole. This can be seen as follows. For every sunflower $P$ with at least $k + 1$ petals, every hitting set of size at most $k$ contains one of $P$’s core-elements. Therefore, we can repeatedly discard a petal of a sunflower of size $k + 2$ from the hypergraph, yielding a decision-equivalent $d$-HITTING SET instance whose largest sunflower has $k + 1$ petals [15]. This, by the sunflower lemma of Erdős and Rado [9], implies that the resulting hypergraph has $O(4^d)$ hyperedges [10, Theorem 9.8], therefore showing that this form of data reduction yields a problem kernel [6, 11].

**Previous work.** Downey and Fellows [8] showed that HITTING SET is W[2]-complete with respect to the parameter $k$ when the cardinality of the hyperedges is unbounded. Hence, unless $\text{FPT} = \text{W}[2]$, it has no problem kernel. Various problem kernels for $d$-HITTING SET have been developed [2, 10, 15, 16, 18, 19]. However, the problem kernels aiming for efficiency faced some problems: Niedermeier and Rossmanith [18] showed a problem kernel for 3-HITTING SET of size $O(k^3)$. They implicitly claimed that a polynomial-size problem kernel for $d$-HITTING SET is computable in linear time, not giving a proof for the running time. Nishimura et al. [19] claimed that a problem kernel with $O(k^{d-1})$ vertices is computable in $O(k(|V| + |E|) + k^d)$ time, which, however, does not always yield correct problem kernels [2]. The problem kernels of Flum and Grohe [10] and Kratsch [15] exploit the sunflower lemma by Erdős and Rado [9] and therefore yield concise explanations of why certain vertices should be part of optimal solutions. However, their running times are only analyzed to be polynomial in the input size. Abu-Khzam [2] showed a problem kernel with $O(k^{d-1})$ vertices for $d$-HITTING SET, thus proving the previously claimed result of Nishimura et al. [19] on the number of vertices in the problem kernel. The problem kernel of Abu-Khzam [2] may still comprise
Ω(k^{2d-2}) hyperedges.¹ Dell and van Melkebeek [7] showed that the existence of a problem kernel with O(k^{d-\varepsilon}) hyperedges for any ε would imply coNP \subseteq NP/poly and a collapse of the polynomial-time hierarchy to the third level. Therefore, a problem kernel with O(k^{d-\varepsilon}) hyperedges is presumed not to exist.

Our results. We show that a problem kernel for \textit{d-Hitting Set} with O(k^d) hyperedges and vertices is computable in linear time. Thereby, we prove the previously claimed result by Niedermeier and Rossmanith [18] and complement recent results in improving the efficiency of kernelization algorithms [4, 12, 20]. In contrast to many other problem kernels [2, 16, 18, 19], our algorithm outputs sunflowers to guide fault diagnosis. Additionally, using ideas from Abu-Khzam [2] and Moser [16], we show that the number of vertices can be further reduced to O(k^d - 1) with an additional amount of O(k^{1.5d}) time. Summarizing, by merging these techniques, we can compute in O(|V| + |E| + k^{1.5d}) time a problem kernel comprising O(k^d) hyperedges and O(k^d - 1) vertices.

Preliminaries. A hypergraph H = (V, E) consists of a set of vertices V and a set of (hyper)edges E, where each hyperedge in E is a subset of V. In a d-uniform hypergraph every edge has cardinality exactly d. A 2-uniform hypergraph is a graph. A hypergraph G = (V', E') is a subgraph of its supergraph H if V' \subseteq V and E' \subseteq E. A set S \subseteq V intersecting every set in E is a hitting set. A parameterized problem is a subset L \subseteq \Sigma^* \times \mathbb{N} [8, 10, 17]. A problem kernel for a parameterized problem L is a polynomial-time algorithm that, given an instance (I, k), computes an instance (I', k') such that |I'| + k' \leq f(k) and (I', k') \in L \iff (I, k) \in L. Herein, the function f is called the size of the problem kernel and depends only on k.

2 A Linear-Time Problem Kernel for \textit{d-Hitting Set}

This section shows a linear-time problem kernel for \textit{d-Hitting Set} comprising O(k^d) edges. That is, we show that a hypergraph H can be transformed in linear time to a hypergraph G such that G has O(k^d) edges and allows for a hitting set of size k if and only H does. In Section 3, we show how to shrink the number of vertices to O(k^{d-1}).

Theorem 1. \textit{d-Hitting Set} admits a linear-time computable problem kernel comprising O(k^d) hyperedges and vertices.

Until now, problem kernels based on the sunflower lemma by Erdős and Rado [9] proceed in the following fashion [10, 15]: repeatedly (i) find a sunflower of size k + 1 in the input graph and (ii) delete redundant petals until no more sunflowers of size k + 1 exist. This approach has the drawback of finding only one sunflower at a time and restarting the process from the beginning.

¹ Although not directly given in the work of Abu-Khzam [2] itself, this can be seen as follows: the kernel comprises vertices of each hyperedge in a set W of pairwise "weakly related" hyperedges and an independent set I. In the worst case, |W| = k^{d-1} and |I| = dk^{d-1} and each hyperedge in W has d subsets of size d - 1. Then, each subset can constitute a hyperedge with each vertex in I and the kernel has \Omega(k^{2d-2}) hyperedges.
Algorithm 1: Linear-Time Kernelization for \(d\)-Hitting Set

| Line | Code |
|------|------|
| 1    | \(E' \leftarrow \emptyset; \) // Initialization for each edge |
| 2    | \foreach e \in E do // Initialization for all possible cores of sunflowers |
| 3    | \quad \foreach C \subseteq e do // No petals found for sunflower with core \(C\) yet |
| 4    | \quad \quad petals[C] \leftarrow 0; |
| 5    | \quad \foreach v \in e do // No vertex in a petal of a sunflower with core \(C\) yet |
| 6    | \quad \quad used[C][v] \leftarrow false |
| 7    | \foreach e \in E do delete all e' \supseteq e from \(E\); // Every vertex that hits \(e\) also hits \(e'\) |
| 8    | \foreach e \in E do |
| 9    | \quad if \(\forall C \subseteq e: \text{petals}[C] \leq k\) then |
| 10   | \quad \quad E' \leftarrow E' \cup \{e\}; |
| 11   | \quad \foreach C \subseteq e do // Consider all possible cores for the petal \(e\) |
| 12   | \quad \quad if \(\forall v \in e \setminus C: \text{used}[C][v] = false\) then |
| 13   | \quad \quad \quad petals[C] \leftarrow petals[C] + 1; |
| 14   | \quad \quad \foreach v \in e \setminus C do used[C][v] \leftarrow true; |
| 15   | \quad V' := \bigcup_{e \in E} e; |
| 16   | \quad return \((V', E')\); |

In contrast, to prove Theorem 1, we construct a subgraph \(G = (V', E')\) of a given hypergraph \(H = (V, E)\) not by edge deletion; instead, we follow a bottom-up approach that allows us to “grow” many sunflowers in \(G\) simultaneously, stopping “growing sunflowers” if they become too large. Algorithm 1 repeatedly (after some initialization work in lines 1–7) in line 10 copies a hyperedge \(e\) from \(H\) to the initially empty \(G\) unless we find in line 9 that \(e\) contains the core \(C\) of a sunflower of size \(k + 1\) in \(G\). To check this, the number of petals found for a core \(C\) is maintained in a data structure petals[C]. If we find that an edge \(e\) is suitable as petal of a sunflower with core \(C\) in line 12, then we mark the vertices in \(e \setminus C\) as “used” for the core \(C\) in line 14. This information is maintained by setting “used[C][v] \leftarrow true”. In this way, vertices in \(e \setminus C\) are not considered again for finding petals for the core \(C\) in line 12, therefore ensuring that additionally found petals intersect \(e\) only in \(C\).

We now prove the correctness and running time of Algorithm 1, which will, together with the result that the output graph contains no large sunflowers, provide a proof of Theorem 1. Note that, by storing in petals[C] a list of found petals, they can serve as concise explanations of why a small hitting set contains vertices of \(C\).

Lemma 1. The hypergraph \(G\) returned by Algorithm 1 on input \(H\) admits a hitting set of size \(k\) if and only if \(H\) does.

Proof. We first show that, if \(H\) admits a hitting set of size \(k\), then so does \(G\). For every hitting set \(S\) for \(H = (V, E)\), the set \(S' := S \cap V'\) is a hitting set for \(G = (V', E')\) with \(|S'| \leq |S|\): the set \(S\) contains an element of every edge in \(E\) and, since \(E' \subseteq E\) and \(V' = \bigcup_{e \in E'} e\), the set \(S'\) contains an element of every edge in \(E'\). It remains to
show that if $G$ admits a hitting set of size $k$, then so does $H$. Assume that $S$ is a hitting set of size $k$ for $G$. Obviously, all edges that $H$ and $G$ have in common are hit in $H$ by $S$. It remains to show that every edge $e$ in $H$ that is not in $G$ is also hit.

First, consider the case where $e$ was not deleted in line 7. Then, adding $e$ to $G$ in line 10 of Algorithm 1 has been skipped, because the condition in line 9 is false. That is, petals$[C] \supseteq k + 1$ for some $C \subseteq e$. Consequently, for this particular $C$, a sunflower $P$ with $k + 1$ petals and core $C$ exists in $G$, since we only increment petals$[C]$ in line 13 if we find $e$ to be suitable as additional petal for a sunflower with core $C$ in line 12. Note that $C \neq \emptyset$, because otherwise $k + 1$ pairwise disjoint edges would exist in $G$, contradicting our assumption that $S$ is a hitting set of size $k$ for $G$. Since $|S| \leq k$, we have $S \cap C \neq \emptyset$ as discussed in the introduction. Therefore, since $C \subseteq e$ and $C \subseteq V$, the edge $e$ is hit by $S$ also in $H$.

Second, in the case where $e$ was removed in line 7, $e$ is also hit by $S$, because either $G$ contains a sub-edge $e' \subsetneq e$ or $e'$ is hit since its addition to $G$ was skipped in application of the previous case. We conclude that $S$ is a hitting set of size $k$ also for $H$.

\textbf{Lemma 2.} Given a hypergraph $H = (V, E)$ and a natural number $k$, Algorithm 1 can be implemented to run in $O(d|V| + 2^d |E|)$ time.

\textbf{Proof.} We first describe the data structure that is used to maintain petals$[C]$ and used$[C][v]$, then its initialization in lines 1–6, then the implementation of lines 8–15, and finally that of line 7.

We assume that every vertex is represented as an integer in $\{1, \ldots, |V|\}$ and that every edge is represented as a sorted array. We can initially sort all edges of $H$ in $O(E(d \log d))$ total time. Then, the set subtraction operation needed in line 12 can be executed in $O(d)$ time such that the resulting set is again sorted. Moreover, we can generate all subsets of a sorted set such that the resulting subsets are sorted. Hence, we can assume to always deal with sorted edges and thus obtain a canonical representation of an edge as a length-$d$ character string over the alphabet $V$. This enables us to maintain petals$[C]$ and used$[C][v]$ in a trie: a trie is a tree-like data structure, in which, when each of its inner nodes is implemented as an array, a value associated with a character string $X$ can be looked up and stored in $O(|X|)$ time [3, Section 5.3]. Hence, we can look up and store values associated with a set $C$ in $O(d)$ time. We use such a trie to associate with some sets $C \subseteq V$, $|C| \leq d$, an integer petals$[C]$, and a pointer to a vector used$[C][]$ of length $|V|$.

For initial creation of the trie in lines 1–6, we do not initialize every cell of the array that implements an inner node of the trie, as this would take $O(|V|)$ time for each non-empty node. However, we have to initialize all cells that will be accessed: otherwise, it will be unknown if a cell contains a pointer to another node or random data. We achieve this as follows: in lines 1–6, we obtain a length-$2^d |E|$ list $L$ of all possible sets $C \subseteq e$ for all $e \in E$. We will only associate values with sets in $L$, and therefore initialize the inner nodes of the trie to only hold values associated with sets in $L$. This works in $O(d|V| + 2^d d \cdot |E|)$ time, since the representation of sets in $L$ as length-$d$ strings over the alphabet $V$ enables us to sort $L$ in $O(d(|V| + |L|)) = O(d|V| + 2^d d \cdot |E|)$ time using Radix Sort [3, Section 8.3].
We build the trie by iterating over $L$ once: in each iteration, we check in $O(d)$ time in which positions the character string for a set $C$ differs from the character string of its predecessor set in $L$. This tells us which array entries of the inner nodes of the trie have to be newly initialized, and which nodes in the trie on the path to the leaf corresponding to $C$ have been previously initialized and may not be overwritten. Hence, we can implement lines 1–6 to run in $O(d|V| + 2^d d|E|)$ time, observing that line 5 can be implemented to run in $O(d)$-time, as only one look-up to $\text{used}[C][]$ is needed to obtain an array, in which then $O(d)$ necessary values are initialized.

The for-loop in line 8 iterates $|E|$ times. Its body works in $O(2^d d)$ time: obviously, this time bound holds for lines 9 and 10; it remains to show that the body of the for-loop in line 11 works in $O(d)$ time. This is easy to see if one considers that, in lines 12 and 14, one only has to do one look-up to $\text{used}[C][]$ to find an array that holds the values for the at most $d$ vertices $v' \in e$. Also line 15 works in linear time by first initializing all entries of an array vertices[] of size $|V|$ to “false”. Then, for each edge $e \in E'$ and each vertex $v \in e$, set “vertices[\text{\text{e}}] ← true” in $O(d)$ time. Afterward, let $V'$ be the set of vertices $v$ for which vertices[\text{\text{e}}] = true. This takes $O(|V| + d|E|)$ time.

It remains to discuss the running time of line 7. Similarly as in lines 8–14, we iterate over all edges $e \in E$, and for all proper subsets $e' \subset e$ add a pointer to the position of $e$ in $E$ to the list supersets[e'][\text{\text{e}}] (associated with $e'$ using a trie). It then remains to remove the edges in supersets[e'][\text{\text{e}}] from $E$ for each edge $e \in E$.

We now show that there is an upper bound on the size of the sunflowers in the graph output by Algorithm 1. This enables us to upper-bound the size of the output graph similarly to how the sunflower lemma of Erdős and Rado [9] is used in the $d$-Hitting SET kernel of Flum and Grohe [10, Theorem 9.8].

**Lemma 3.** Given a hypergraph $H = (V,E)$ and a natural number $k$, Algorithm 1 outputs a hypergraph $G$ whose largest sunflower has $d(k + 1)$ petals.

**Proof.** Let $P$ be a sunflower with core $C$ in $G$. If $C \in P$, then $|P| = 1$ because of line 7 of Algorithm 1. If $C \notin P$, the following two observations yield $|P| \leq d(k + 1)$:

(i) Every petal $e \in P$ present in $G$ is copied from $H$ in line 10 of Algorithm 1. Consequently, every petal $e \in P$ contains a vertex $v$ satisfying used[\text{\text{e}}][v] = true: if this condition would be violated in line 12, then line 14 applies “used[\text{\text{e}}][v] ← true” to all vertices $v \in e \setminus C$.

(ii) Whenever petals[\text{\text{e}}] is incremented by one in line 13, then, in line 14, “used[\text{\text{e}}][v] ← true” is applied to the at most $d$ vertices $v \in e$. Thus, since always petals[\text{\text{e}}] \leq k + 1, at most $d(k + 1)$ vertices $v$ satisfy used[\text{\text{e}}][v] = true. Moreover, since, by line 14, no $v \in C$ satisfies used[\text{\text{e}}][v] = true and the petals in $P$ pairwise intersect only in $C$, it follows that at most $d(k + 1)$ petals in $P$ contain vertices satisfying used[\text{\text{e}}][v] = true.

The last ingredient in the proof of Theorem 1 is the sunflower lemma by Erdős and Rado [9]. In a similar way as Flum and Grohe [10, Lemma 9.7], we can show the following refined version, which we need for Section 3. Note that, for $b = 1$, this is exactly the sunflower lemma [10].
Lemma 4. Let $H = (V,E)$ be an $\ell$-uniform hypergraph, $b,c \in \mathbb{N}$, and $b \leq \ell$ such that every pair of edges in $H$ intersects in at most $\ell - b$ vertices. If $H$ contains more than $\ell!c^{\ell+1-b}$ edges, then $H$ contains a sunflower with more than $c$ petals.

We finally have all ingredients to show that $d$-HITTING SET admits a linear-time computable $O(k^d)$-size problem kernel, thus proving Theorem 1.

**Proof (Theorem 1).** Lemma 1 and Lemma 2 show that Algorithm 1 executes linear-time data reduction such that the input and output graph are equivalent with respect to $d$-HITTING SET. It remains to show that the graph $G$ output by Algorithm 1 comprises at most $d! \cdot d^{d+1} \cdot (k+1)^k \in O(2^k)$. This then also implies that $G$ has $O(k^d)$ vertices, as the vertex set of $G$ is constructed as the union of its edges in line 15 of Algorithm 1.

To bound the number of edges, consider for $1 \leq \ell \leq d$ the $\ell$-uniform hypergraph $G_\ell = (V_\ell,E_\ell)$ comprising only the edges of size $\ell$ of $G$. If $G$ had more than $d! \cdot d^{d+1} \cdot (k+1)^k$ edges, then, for some $\ell \leq d$, $G_\ell$ would have more than $d! \cdot d^d \cdot (k+1)^{d}$ edges. This, however, leads to a contradiction with Lemma 3: Lemma 4 with $b = 1$ and $c = d(k+1)$ states that if $G_\ell$ had more than $\ell! \cdot d^\ell \cdot (k+1)^\ell$ edges, then $G_\ell$ would contain a sunflower with more than $d(k+1)$ petals. This sunflower would also exist in the supergraph of $G_\ell$. □

### 3 Reducing the Number of Vertices to $O(k^{d-1})$

In this section, we combine our problem kernel shown in Section 2 with techniques from Abu-Khzam [2] and Moser [16, Section 7.3]. Thus, we obtain a problem kernel for $d$-HITTING SET comprising $O(k^d)$ edges and $O(k^{d-1})$ vertices in $O(|V| + |E| + k^{1.5d})$ time. To this end, we first briefly sketch the running-time bottleneck of the kernelization idea of Abu-Khzam [2], which is also a bottleneck in the algorithm of Moser [16].

The problem kernels of Abu-Khzam [2] and Moser [16, Section 7.3]. Given a hypergraph $H = (V,E)$ and a natural number $k$, Abu-Khzam [2] first computes a maximal weakly related set $W$, where data reduction ensures $|W| \leq k^{d-1}$.

**Definition 2 ([2]).** A set $W \subseteq E$ is weakly related if every pair of edges in $W$ intersects in at most $d-2$ vertices.

Whether a given edge $e$ can be added to a set $W$ of weakly related edges can be checked in $O(d|W|)$ time. After adding $e$, data reduction on $W$ is executed in $O(2^d|W|\log|W|)$ time. Hence, since always $|W| \leq k^{d-1}$, Abu-Khzam [2] can compute $W$ in $O(2^dk^{d-1}\log k \cdot |E|)$ time.

Since $|W| \leq k^{d-1}$, it remains to bound the size of the set $I$ of vertices not contained in edges of $W$. The set $I$ is an independent set, that is, $I$ contains no pair of vertices occurring in the same edge [2]. A bipartite graph $B = (I \cup S, E')$ is constructed, where $S := \{e \subseteq V \mid \exists w \in I : \exists w \in W : e \subseteq w, \{v\} \cup e \in E\}$ and $E' := \{(v,e) \mid v \in I, e \in S, \{v\} \cup e \in E\}$. Whereas Abu-Khzam [2] shrinks the size of $I$ using so-called
Algorithm 2: Linear-time computation of a maximal weakly related set

Input: Hypergraph $H = (V, E)$, natural number $k$.
Output: Maximal weakly related set $W$.

1. $W \leftarrow \emptyset$;
2. foreach $e \in E$ do // Initialization for each edge
   3. foreach $C \subseteq e, |C| = d - 1$ do
      4. intersection[C] ← false; // No edges in $W$ contain $C$ yet.
      5. intersection[e \setminus C] ← false; // The vertex in $e \setminus C$ is not in $W$ yet. We use
         // this later to compute an independent set.
   6. foreach $e \in E$ do
      7. if $\forall C \subseteq e, |C| = d - 1$: intersection[C] = false then
      8. $W \leftarrow W \cup \{e\}$;
      9. foreach $C \subseteq e, |C| = d - 1$ do
         10. intersection[C] ← true;
         11. intersection[e \setminus C] ← true;
      12. return $W$;

crown reductions, Moser [16, Lemma 7.16] shows that it is sufficient to compute a maximum matching in $B$ and to remove unmatched vertices in $I$ from $G$ together with the edges containing them. The bound of the number of vertices in the problem kernel is thus $O(k^{d-1})$, since $|W| \leq k^{d-1}$, and therefore $|I| \leq |S| \leq dk^{d-1}$.

Our improvements. Given a hypergraph $H = (V, E)$ and a natural number $k$, we can first compute our problem kernel in $O(|V| + |E|)$ time, leaving $O(k^{d-1})$ edges in $H$. Afterward applying the problem kernel of Abu-Khzam [2] would reduce the number of vertices to $O(k^{d-1})$. However, the computation of the maximal weakly related set on our reduced instance already takes $O(2^d k^{d-1} \log k \cdot |E|) = O(k^{2d-1} \log k)$ additional time, as discussed above. We improve the running time of this step in order to show:

Theorem 2. $d$-Hitting Set admits a problem kernel comprising $O(k^d)$ hyperedges and $O(k^{d-1})$ vertices computable in $O(|V| + |E| + k^{1.5d})$ time.

To prove Theorem 2, we compute a maximal weakly related set $W$ in linear time. Then, we show that our problem kernel already ensures $|W| \in O(k^{d-1})$ and that further data reduction on $W$ is therefore unnecessary. This makes finding a maximum matching the new bottleneck of the kernelization described by Moser [16, Section 7.3].

Lemma 5. Given a hypergraph $H = (V, E)$, a maximal weakly related set is computable in $O(d^2 |V| + d^2 \cdot |E|)$ time.

To prove Lemma 5, we employ Algorithm 2.

Proof. First, observe that the set $W$ returned in line 12 of Algorithm 2 is indeed weakly related: let $w_1 \neq w_2 \in E$ intersect in more than $d - 2$ vertices and assume
that \( w_1 \) is added to \( W \) in line 8. Let \( C := w_1 \cap w_2 \). Obviously, \(|C| = d - 1\). Hence, when \( w_1 \) is added to \( W \), then we apply “intersection[\( C \) ← true]” in line 10. Therefore, when \( e = w_2 \) is considered in line 6, the condition in line 7 does not hold, which implies that \( w_2 \) is not added to \( W \) in line 8. In the same way it follows that every edge is added to \( W \) that does not intersect any edge of \( W \) in more than \( d - 2 \) vertices. Therefore, \( W \) is maximal.

We first sort all edges of \( H \) in \( O(|E|d \log d) \) time. Using the trie data structure and initialization method as used in Lemma 2, we can do each look-up of a value \( \text{intersection}[C] \) in \( O(d) \) time if \( C \) is the result of a set subtraction operation of two sorted sets. We initialize the trie to associate values with at most 2\( d \cdot |E| \) sets. Hence, as discussed in Lemma 2, the initialization in lines 1–5 can be done in \( O(d \cdot |V| + d^2 \cdot |E|) \) time. Finally, for every edge, the body of the for loop in line 6 can be executed in \( O(d^2) \) time doing \( O(d) \)-time look-ups for each of the \( 2d \cdot |E| \) sets. □

We can now prove Theorem 2 by showing how to compute a problem kernel with \( O(k^{d-1}) \) vertices in \( O(|V| + |E| + k^{1.5d}) \) time.

**Proof (of Theorem 2).** We may assume that the hypergraph \( H = (V,E) \) in an instance of \( d \)-Hitting Set satisfies \(|V| + |E| \in O(k^d)\) and contains sunflowers with at most \( d(k + 1) \) petals, since otherwise using Algorithm 1, we can transform \( H \) accordingly in linear time, as stated by Lemma 3 and Theorem 1. To reduce the number of vertices in \( H \) to \( O(k^{d-1}) \), we follow the approach of Moser [16, Lemma 7.16] as discussed in the beginning of this section.

First, compute a maximal weakly related set \( W \) in \( H \) in \( O(|V| + |E|) = O(k^d) \) time using Algorithm 2. We show that \(|W| \in O(k^{d-1})\). Because every pair of edges in \( W \) intersects in at most \( d - 2 \) vertices, by Lemma 3 and Lemma 4 for \( b = 2 \) and \( c = d(k + 1) \), we know that the hypergraph \((V,W_\ell)\) for \( \ell \geq 2 \), where \( W_\ell \) is the set of cardinality-\( \ell \) edges in \( W \), has at most \( O(k^{d-1}) \) edges. Moreover, \( W_1 \) contains at most \( O(k) \) edges, as they form a sunflower with empty core. Therefore, \(|W| \in O(k^{d-1})\).

Next, we construct a bipartite graph \( B = (I \cup S, E') \), where

1. \( I \) is the set of vertices in \( V \) not contained in any edge in \( W \), forming an independent set [2, 16],
2. \( S := \{ e \subseteq V \mid \exists w \in I : \exists w \in W : e \subseteq w, \{v\} \cup e \in E \} \), and
3. \( E' := \{ (v,e) \mid v \in I, e \in S, \{v\} \cup e \in E \} \).

This can be done in \( O(|E'|) = O(k^d) \) time as follows: for each \( e \in E \) with \(|e| = d\) and each \( v \in e \), add \( (v, e \setminus \{v\}) \) to \( E' \) if and only if \( \text{intersection}[e \setminus \{v\}] = \text{true} \) and \( \text{intersection}[\{v\}] = \text{false} \). In this case, it follows that \( e \) is separable into

1. a subset \( e \setminus \{v\} \) of an edge of \( W \), since \( \text{intersection}[e \setminus \{v\}] = \text{true} \), and
2. the vertex \( v \) that is not contained in any edge in \( W \) and, hence, contained in \( I \), since \( \text{intersection}[\{v\}] = \text{false} \).

Thus, \( e \) clearly satisfies the definition of \( E' \). Finally, for each edge \( \{v,C\} \) added to \( E' \), add \( v \) to \( I \) and \( C \) to \( S \). Herein, checking that an element is not added to \( I \) or \( S \) multiple times can be done in \( O(d) \) time per element: to this end, we use a trie data structure similarly as “petals[]” in Lemma 2 or “intersection[]” in Algorithm 2. Similarly to Lemma 2, the trie can be initialized in linear time, since we know the elements to be added to \( I \) and \( S \) in advance.
It remains to shrink $I$ to $O(k^{d-1})$ vertices by computing a maximum matching in $B$ and deleting from $H$ the unmatched vertices in $I$ and the edges containing them. However, note that by construction of $B$, for each edge in $H$, we add at most one edge and two vertices to $B$. Therefore, $B$ has $O(kd)$ edges and vertices. Hence, a maximum matching on $B$ can be computed in $O(|I| + |H| + |E'|) = O(k^{1.5d})$ time using the algorithm of Hopcroft and Karp [22, Theorem 16.4].

\[\square\]

4 Conclusion

We have improved the running times of the $O(k^{d-1})$-vertex problem kernels for $d$-Hitting Set by Abu-Khzam [2] and Moser [16]. To this end, we showed, as claimed earlier by Niedermeier and Rossmanith [18], that a polynomial-size problem kernel for $d$-Hitting Set can be computed in linear time—more specifically, a problem kernel comprising $O(kd)$ hyperedges and vertices. In contrast to these problem kernels, our algorithm maintains expressiveness by finding, in forms of sunflowers, concise explanations of potential problem solutions. However, the constant hidden in our $O(k^{d-1})$-bound on the number of vertices is $d^{l_0d_0}$ and therefore higher than the constant $2d - 1$ obtained by Abu-Khzam [2]. This is due to the fact that our upper bound on the size of the weakly related set $W$ comes from the sunflower lemma in Lemma 3, whereas Abu-Khzam [2] executes more effective data reduction on $W$. Regarding these constants, first experiments with an implementation of our algorithm show that the data reduction is indeed effective. It is interesting whether a problem kernel with $O(k^{d-1})$ vertices and $O(kd)$ edges for $d$-Hitting Set can be computed in linear time. This would merge the best known results for problem kernels for $d$-Hitting Set. However, all known $O(k^{d-1})$-vertex problem kernels for $d$-Hitting Set, that is, the problem kernels by Abu-Khzam [2] and Moser [16, Section 7.3], involve the computation of maximum matchings. This seems to be a difficult to avoid bottleneck.

Acknowledgment. The author is very thankful to Rolf Niedermeier for many valuable comments.

Bibliography

[1] R. Abreu, P. Zoeteweij, and A. J. C. van Gemund. A dynamic modeling approach to software multiple-fault localization. In Proc. 19th DX, pages 7–14, Blue Mountains, NSW, Australia, 2008. [p. 1].
[2] F. N. Abu-Khzam. A kernelization algorithm for $d$-hitting set. J. Comput. Syst. Sci., 76(7):524–531, 2010. [pp. 3, 4, 8, 9, 10, and 11].
[3] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. Data Structures and Algorithms. Addison-Wesley, 1983. [p. 6].
[4] R. van Bevern, S. Hartung, F. Kammer, R. Niedermeier, and M. Weller. Linear-time computation of a linear problem kernel for dominating set on planar graphs. In Proc. 6th IPEC, volume 7112 of LNCS, pages 194–206. Springer, 2012. [p. 4].
[5] R. van Bevern, H. Moser, and R. Niedermeier. Approximation and tidying—a problem kernel for s-plex cluster vertex deletion. Algorithmica, 62(3):930–950, 2012. [p. 1].

[6] H. L. Bodlaender. Kernelization: New upper and lower bound techniques. In Proc. 4th IWPEC, volume 5917 of LNCS, pages 17–37. Springer, 2009. [pp. 2 and 3].

[7] H. Dell and D. van Melkebeek. Satisfiability allows no nontrivial sparsification unless the polynomial-time hierarchy collapses. In Proc. 42nd STOC, pages 251–260. ACM, 2010. [p. 4].

[8] R. G. Downey and M. R. Fellows. Parameterized Complexity. Springer, 1999. [pp. 3 and 4].

[9] P. Erdős and R. Rado. Intersection theorems for systems of sets. J. London Math. Soc., 35:85–90, 1960. [pp. 2, 3, 4, and 7].

[10] J. Flum and M. Grohe. Parameterized Complexity Theory. Springer, 2006. [pp. 3, 4, and 7].

[11] J. Guo and R. Niedermeier. Invitation to data reduction and problem kernelization. SIGACT News, 38(1):31–45, 2007. [pp. 2 and 3].

[12] T. Hagerup. Linear-time kernelization for planar dominating set. In Proc. 6th IPEC, volume 7112 of LNCS, pages 181–193. Springer, 2011. [p. 4].

[13] F. Hüffner, C. Komusiewicz, H. Moser, and R. Niedermeier. Fixed-parameter algorithms for cluster vertex deletion. Theory Comput. Syst., 47(1):196–217, 2010. [p. 1].

[14] J. de Kleer and B. C. Williams. Diagnosing multiple faults. Artif. Intell., 32(1):97–130, 1987. [p. 1].

[15] S. Kratsch. Polynomial kernelizations for MIN F^+Π_1 and MAX NP. Algorithmica, 63(1):532–550, 2012. ISSN 0178-4617. [pp. 3, 4, and 7].

[16] H. Moser. Finding Optimal Solutions for Covering and Matching Problems. PhD thesis, Institut für Informatik, Friedrich-Schiller-Universität Jena, 2010. [pp. 3, 4, 8, 9, 10, and 11].

[17] R. Niedermeier. Invitation to Fixed Parameter Algorithms. Oxford University Press, USA, 2006. [p. 4].

[18] R. Niedermeier and P. Rossmanith. An efficient fixed-parameter algorithm for 3-hitting set. J. Discrete Algorithms, 1(1):89–102, 2003. [pp. 3, 4, and 11].

[19] N. Nishimura, P. Ragde, and D. M. Thilikos. Smaller kernels for hitting set problems of constant arity. In Proc. 1st IWPEC, volume 3162 of LNCS, pages 121–126. Springer, 2004. [pp. 3 and 4].

[20] F. Protti, M. Dantas da Silva, and J. Szwarcfiter. Applying modular decomposition to parameterized cluster editing problems. Theory Comput. Syst., 44:91–104, 2009. [p. 4].

[21] R. Reiter. A theory of diagnosis from first principles. Artif. Intell., 32(1):57–95, 1987. [p. 1].

[22] A. Schrijver. Combinatorial Optimization: Polyhedra and Efficiency, volume A. Springer, 2003. [p. 11].