Serial and parallel kernelization of Multiple Hitting Set parameterized by the Dilworth number, implemented on the GPU[⋆]

René van Bevern[a,*], Artem M. Kirilin[b], Daniel A. Skachkov[a], Pavel V. Smirnov[c], Oxana Yu. Tsidulko[a]

[a]*Huawei Technologies Co., Ltd.*
[b]*Siberian Federal University, Krasnoyarsk, Russian Federation*
[c]*Recraft*

Abstract

The NP-hard Multiple Hitting Set problem is the problem of finding a minimum-cardinality set intersecting each of the sets in a given input collection a given number of times. Generalizing a well-known data reduction algorithm due to Weihe, we show a problem kernel for Multiple Hitting Set parameterized by the Dilworth number, a graph parameter introduced by Foldes and Hammer in 1978 yet seemingly so far unexplored in the context of parameterized complexity theory. Using matrix multiplication, we speed up the algorithm to quadratic sequential time and logarithmic parallel time. We experimentally evaluate our algorithms. By implementing our algorithm on GPUs, we show the feasibility of realizing kernelization algorithms on SIMD (Single Instruction, Multiple Data) architectures.

1. Introduction

The following fundamental combinatorial optimization problem arises in bioinformatics [44], medicine [42, 52], clustering [13, 37], automatic reasoning [17, 26, 49], feature selection [16, 31], radio frequency allocation [50], software engineering [48], and public transport optimization [15, 53].

Problem 1.1 (Multiple Hitting Set)**.**
Input: A hypergraph $H = (V, E)$ with *vertices* $V = \{v_1, \dots, v_n\}$ and *hyperedges* $E = \{e_1, \dots, e_m\} \subseteq 2^V$, a *demand* function $f : E \to \{1, \dots, \alpha\}$, and $k, \alpha \in \mathbb{N}$.
Question: Is there a *hitting set* $S \subseteq V$ of cardinality at most k such that $|e \cap S| \geq f(e)$ for each $e \in E$?

Already the special case with $f \equiv 1$, known as simply *Hitting Set*, is NP-complete [38]. Exact algorithms for NP-complete problems usually take time exponential in the input size. Thus, an important preprocessing step is data reduction, which has proven to significantly shrink real-world instances of NP-hard problems [2, 3, 11, 12, 14, 15, 42, 53]. In the context of public transport optimization, Weihe [53] introduced a simple but very effective in experiments [14, 15, 53] data reduction algorithm for Hitting Set. It exhaustively applies two data reduction rules that do not alter the answer to the input instance:

(W1) If there is a pair of distinct vertices v_i and v_j such that every hyperedge containing v_j also contains v_i, delete v_j.

(W2) If there is a pair of distinct hyperedges e_i and e_j such that $e_i \subseteq e_j$, then delete e_j.

Using *kernelization* from *parameterized complexity theory*, which is formally defined in Section 2, our work contributes to the understanding, generalizes and speeds up Weihe's data reduction algorithm in the following ways.

[⋆]The results presented in this work are unrelated to the authors' work at Huawei or Recraft, respectively.
[*]Corresponding author.
Email addresses: `rene.van.bevern@huawei.com` (René van Bevern), `kirilinartem@gmail.com` (Artem M. Kirilin), `skachkov.daniel@huawei-partners.com` (Daniel A. Skachkov), `pavel.vl.smirnov@recraft.ai` (Pavel V. Smirnov), `tsidulko.oksana@huawei.com` (Oxana Yu. Tsidulko)

Understanding. While the data reduction effect of Weihe's algorithm is experimentally proven [14, 15, 53], finding theoretical explanations for its effectivity is challenging [15, 27]. For example, Fellows [27] asks "Weihe's example looks like an FPT kernelization, but what is the parameter?" We show that Weihe's algorithm actually computes problem kernels for Hitting Set whose number of vertices and hyperedges is linear in the *Dilworth number* of the incidence graph of the input hypergraph (see Section 2.3 for a definition).

The Dilworth number has been introduced by Foldes and Hammer [30] in 1978, which led to a series of studies of the structure of graphs with small Dilworth number [18, 28, 36, among others]. However, the Dilworth number until now seems unexplored in a parameterized complexity context. For example, neither Gera et al. [35] nor Sorge and Weller [51] list it. This is surprising, since the Dilworth number is bounded from above by the neighborhood diversity (see Section 2.3), which is a well-established parameter in parameterized complexity studies [4, 32–34, 40], and it seems a logical step to analyze which parameterized algorithms for the neighborhood diversity can be strengthened to use the Dilworth number instead.

We note here that the Dilworth number of the incidence graph of a hypergraph $H = (V, E)$ can be exponential in the number $|V|$ of vertices; however, one *in principle* cannot expect problem kernels of size polynomial in $|V|$ for Hitting Set unless the polynomial-time hierarchy collapses [22, 24]. Indeed, for problem kernels of size polynomial in $|V|$ to exist, the cardinality of each input hyperedge must be bounded from above by a constant d [22]. In this case, problem kernels of size $k^{O(d)}$ have been shown [1, 5, 7, 8, 14, 21, 25, 29, 39, 46, 47].

Generalization. Motivated by problems in feature selection, optimal cancer medication, and genome-wide association studies, attempts have been undertaken to generalize Weihe's data reduction algorithm to Multiple Hitting Set [20, 41, 42]. However, Cotta et al. [20] only generalize the hyperedge deletion rule (W2). The generalization of the vertex deletion rule (W1) of Moscato et al. [45] is wrong: as shown in Appendix A, it *does* alter the answer to the input instance.

We provide a full generalization of Weihe's algorithm in the sense that we obtain a problem kernel for Multiple Hitting Set whose number of vertices and hyperedges is linear in the Dilworth number of the incidence graph of the input hypergraph and in the maximum hyperedge demand α.

We provide additional data reduction rules that, in the case of Multiple Hitting Set and in contrast to (W2), allow for safe deletion even of hyperedges that are *not* contained within each other. While not provably lowering the size of problem kernels, we show their significant additional data reduction effect in experiments.

Speed-up. A comparison between Weihe's algorithm and linear-time data reduction algorithms for Hitting Set has shown that the data reduction effect of Weihe's algorithm is clearly superior when there are large hyperedges, yet the algorithm is significantly slower, even so much so as to make it inapplicable to large hypergraphs [14]. By looking at the algorithm through the lens of multiplying incidence matrices of the input hypergraph, we provide quadratic sequential-time and logarithmic parallel-time variants of our kernelization algorithms, thus contributing to parallel kernelization studies, which have recently gained increased interest [8, 10]. In contrast to previous, purely theoretic and proof-of-concept work on parallel kernelization, we implement our algorithm on GPUs and thus prove the feasibility of realizing kernelization algorithms on SIMD (Single Instruction, Multiple Data) architectures, in which all cores of a multiprocessor execute the same operation, yet on different (parts of) the data. Kernelization algorithms, which are often a set of data reduction rules applied on different parts of the input, seem to lend themselves to implementation on SIMD architectures.

Organization of this work. Section 2 introduces basic graph-theoretic, parameterized complexity, and kernelization terminology. Section 3 presents known and new data reduction rules for Multiple Hitting Set, and shows that they yield a problem kernel whose number of vertices and hyperedges is linear in the Dilworth number. Section 4 shows how to obtain fast serial and parallel implementations of the data reduction rules via matrix multiplication. Section 5 experimentally evaluates the effect and speed of our data reduction rules.

2. Preliminaries

In this section, we introduce basic graph-theoretic, parameterized complexity, and kernelization terminology.

2.1. Graphs and hypergraphs

Hypergraphs. A *hypergraph* is a pair $H = (V, E)$, where V is a set of *vertices* and $E \subseteq 2^V$ is a set of *(hyper)edges*. We will often denote the vertices and hyperedges as $V = \{v_1, \dots, v_n\}$ and $E = \{e_1, \dots, e_m\}$, respectively.

For a vertex $v \in V$, by $E(v) := \{e \in E \mid v \in e\}$, we denote the set of hyperedges containing v. By $|H| = |V| + \sum_{e \in E} |e|$, we denote the *size* of the hypergraph. This notion of size is motivated by the fact that each incidence between a vertex and a hyperedge has to be encoded in some form. The *incidence matrix* $A(H)$ of a hypergraph $H = (V, E)$ is an $(m \times n)$-matrix such that

$$A_{ij} = \begin{cases} 1, & \text{if } v_j \in e_i \text{ and} \\ 0, & \text{otherwise.} \end{cases}$$

Undirected graphs. A hypergraph whose hyperedges have cardinality two is an *(undirected) graph*. For a vertex $v \in V$ of a graph $G = (V, E)$, $N(v) := \{u \in V \mid \{u, v\} \in E\}$ denotes the *open neighborhood of* v and $N[v] := N(v) \cup \{v\}$ denotes the *closed neighborhood*. To avoid confusion, throughout this work, the notation $N(v)$ is exclusively applied to graphs, whereas $E(v)$ is exclusively applied to hypergraphs (that are not graphs).

A *matching* in a graph is a set of pairwise disjoint edges. The *matching number* $\nu(G)$ of a graph G is the maximum cardinality of any matching in G. The *incidence graph* $\mathcal{I}(H)$ of a hypergraph $H = (V, E)$ is a bipartite graph with the vertex set $V \cup E$ and an edge $\{v, e\}$ for each $v \in V$ and $e \in E$ such that $v \in e$. That is, for any hyperedge e of H, we have $N(e) = e$ in the incidence graph $\mathcal{I}(H)$.

Directed graphs. A *directed graph* $G = (V, A)$ consists of vertices $V = \{v_1, \dots, v_n\}$ and *arcs* $A = \{a_1, \dots, a_m\} \subseteq V^2$. If $(v_i, v_j) \in A$, we call v_i a *parent* of v_j. We call v_i an *ancestor* of v_j if $v_i = v_j$, if v_i is a parent of v_j, or if v_i is a parent of an ancestor of v_j. A vertex without parents is called a *source*, a vertex that is not a parent of any other vertex is a *sink*.

A *cycle* in a directed graph is a sequence of vertices $v_1, v_2, \dots, v_\ell$ such that $(v_i, v_{i+1}) \in A$ for $i \in \{1, \dots, \ell - 1\}$ and $(v_\ell, v_1) \in A$. A directed graph is *acyclic* if it contains no cycles.

Observation 2.1. *Let $G = (V, A)$ be a directed acyclic graph and $X \subseteq V$. Every $v \in X$ has an ancestor u in X, all of whose parents are not in X (possibly, $u = v$ and that set of parents may be empty).*

2.2. Complexity theory and kernelization

Formally, we study decision problems $\Pi \subseteq \{0, 1\}^*$, where the task is to decide whether a given $x \in \{0, 1\}^*$ belongs to Π.

Parameterized complexity. A *parameterized problem* is a pair (Π, κ) where $\Pi \subseteq \{0, 1\}^*$ is a decision problem and $\kappa \colon \{0, 1\}^* \to \mathbb{N}$ is a polynomial-time computable function called a *parameterization*.

A *kernelization* for a parameterized problem (Π, κ) is a polynomial-time algorithm that maps any instance $x \in \{0, 1\}^*$ to an instance $x' \in \{0, 1\}^*$ such that $x \in \Pi \iff x' \in \Pi$ and such that $|x'| \le g(\kappa(x))$ for some computable function g. We call x' the *problem kernel* and g its *size*.

Circuit families. A *Boolean circuit* with n inputs and m outputs is a directed acyclic graph with n sources and m sinks; each of its its non-source vertices v has one of the following three types:

— v is labeled "$\neg$" and has exactly one parent,

— v is labeled "$\vee$" and has exactly two parents,

— v is labeled "$\wedge$" and has exactly two parents.

Identifying 1 with the Boolean value *true* and 0 with the Boolean value *false*, we can inductively define the *value* $\mathrm{val}(v)$ of each vertex v of the Boolean circuit on *input* $x \in \{0, 1\}^n$ as follows.

— The value of the i-th source is x_i,

— For any vertex v labeled "$\neg$" with parent u, $\mathrm{val}(v) := \neg\mathrm{val}(u)$.

— For any vertex v labeled "$\vee$" with parents u_1 and u_2, $\mathrm{val}(v) := \mathrm{val}(u_1) \vee \mathrm{val}(u_2)$.

— For any vertex v labeled "$\wedge$" with parents u_1 and u_2, $\mathrm{val}(v) := \mathrm{val}(u_1) \wedge \mathrm{val}(u_2)$.

Denoting the sinks as $u_1, \ldots, u_m$, we call $\mathrm{val}(u_1)\mathrm{val}(u_2) \ldots \mathrm{val}(u_m) \in \{0,1\}^m$ the *output* of the Boolean circuit on input x. The *size* of a Boolean circuit is its number of vertices. The *depth* of a Boolean circuit is the length of the longest path from any source to any sink.

An *NC^1 circuit family* is a sequence $(C_n)_{n\in\mathbb{N}}$ of Boolean circuits, each with n inputs, $O(\log n)$ depth, and poly(n) size. We say that an NC^1 circuit family $(C_n)_{n\in\mathbb{N}}$ decides a problem $\Pi \subseteq \{0,1\}^*$ if $x \in \Pi$ if and only if the circuit $C_{|x|}$ outputs 1 when given x on the input. More generally, we say that an NC^1 circuit family $(C_n)_{n\in\mathbb{N}}$ computes a function $f\colon \{0,1\}^* \to \{0,1\}^*$ if, for any $x \in \{0,1\}^*$, the circuit $C_{|x|}$ outputs $f(x)$ on input x.

The outputs of NC^1 circuits can be computed in logarithmic time on a parallel computer with a polynomial number of processors, where each processor computes the value of one vertex, taking as input the values of its parents [6, Section 6.7].

Input encoding. For our parallel hypergraph algorithms (or NC^1 circuits), we will generally assume incidence matrices as input, so that each parallel processor gets one matrix entry as input. For sequential hypergraph algorithms, we assume a list of vertices and a list of hyperedges on the input.

2.3. *Dilworth number and neighborhood diversity*

Consider the relation $\sqsubseteq$ on the vertices of a graph $G = (V, E)$ such that

$$u \sqsubseteq v \iff N(u) \subseteq N[v].$$

This relation is reflexive and transitive and is called the *vinical order* of G [30]. We call two vertices u and v *incomparable* if $u \not\sqsubseteq v$ and $v \not\sqsubseteq u$. A *chain* is a subset of pairwise comparable vertices, whereas an *antichain* is a subset of pairwise incomparable vertices. The *Dilworth number* $\nabla(G)$ of G is the size of a largest antichain in $\sqsubseteq$ [30], which, by Dilworth's theorem [23], is equivalent to the minimum number of chains whose union is V.

A related frequently studied graph parameter is the neighborhood diversity [4, 32–34, 40]. To introduce it, consider the relation $\sim$ on the vertices of a graph $G = (V, E)$ such that

$$u \sim v \iff N(u) \setminus \{v\} = N(v) \setminus \{u\}.$$

This is an equivalence relation [40] and the *neighborhood diversity* $\delta(G)$ of G is the number of equivalence classes of $\sim$.

Foldes and Hammer [30] relate the Dilworth number to many other graph parameters, yet not to the neighborhood diversity, which is frequently used for parameterized complexity analysis but was introduced much later [40]. It is not hard to show that the Dilworth number is upper-bounded by the neighborhood diversity, but the gap between the two can be arbitrarily large:

Lemma 2.2.
(i) For any graph G, one has $\nabla(G) \leq \delta(G)$.
(ii) For any $n \in \mathbb{N}$, there is a graph G on $2n$ vertices with $1 = \nabla(G) \leq \delta(G) = 2n - 1$.

Proof. (i) Consider any pair of vertices u and v of G. Then,

$$\begin{aligned} u \sim v &\implies & N(u) \setminus \{v\} \subseteq N(v) \setminus \{u\} \\ &\implies & N(u) \setminus \{v\} \subseteq N(v) \\ &\implies & N(u) \subseteq N(v) \cup \{v\} \\ &\implies & u \sqsubseteq v. \end{aligned}$$

By definition, the vinical order $\sqsubseteq$ of G has an antichain C of cardinality $\nabla(G)$. For any pair of distinct vertices u and v in C, we have $u \not\sqsubseteq v$ and thus $u \nsim v$. It follows that $\sim$ has at least $|C|$ equivalence classes, and thus $\nabla(G) = |C| \leq \delta(G)$.

(ii) Construct a graph G on $2n$ vertices as follows. Start with an empty graph. Then, for each $i \in \{1, \ldots, n\}$, add first an isolated vertex u_i and then add a vertex v_i that is adjacent to all previously added vertices. We get $\delta(G) = 2n - 1$ since each pair of vertices in G, except for u_1 and v_1, are pairwise nonequivalent under $\sim$:

— For any $i, j \in \{1, \dots, n\}$ such that $i \neq j$, one has $u_i \not\sim v_j$ since v_j is adjacent to u_j but u_i is not

— For any $i \in \{2, \dots, n\}$, one has $u_i \not\sim v_i$, since v_i is adjacent to v_1 but u_i is not.

— For $1 \leq i < j \leq n$, one has $v_i \not\sim v_j$ since v_j is adjacent to u_j, but v_i is nonadjacent to u_j, and one has $u_i \not\sim u_j$ since v_i is adjacent to u_i but u_j is nonadjacent to v_i.

We also get $\nabla(G) = 1$ since each pair of vertices is comparable in the vinical order of G: for $1 \leq i \leq j \leq n$, one has

$N(v_i) \subseteq N[v_j]$ since any vertex added to G before v_j is obviously in $N[v_j]$ and any vertex added to G after v_j is adjacent either to both of v_i and v_j or to none of them,

$N(u_j) \subseteq N[u_i]$ since any vertex added to G before u_j is nonadjacent to u_j and any vertex added to G after u_j is adjacent either to all or none of u_i and u_j.

Finally, one has $N(u_1) \subseteq N[v_1]$, that is, one can cover all vertices by the single chain $u_n \sqsubseteq u_{n-1} \sqsubseteq \dots \sqsubseteq u_1 \sqsubseteq v_1 \sqsubseteq v_2 \sqsubseteq \dots \sqsubseteq v_n$. Thus, $\nabla(G) = 1$. □

3. A problem kernel for Multiple Hitting Set

In Section 3.1, we present several data reduction rules for Multiple Hitting Set that generalize Weihe's rules (W1) and (W2). In Section 3.2, we show how problem kernels can be obtained by applying (subsets of) these data reduction rules. Later, in Section 5, we experimentally evaluate several combinations of these data reduction rules.

3.1. Data reduction rules

The following two data reduction rules were suggested by Cotta et al. [20]. The "full edge" rule (FE) exploits that all vertices in a hyperedge e_j with demand $f(e_j) = |e_j|$ must belong to any feasible solution:

(FE) If there is a hyperedge $e_j \in E$ such that $|e_j| = f(e_j)$, then delete e_j, delete each $v \in e_j$, decrement k by $|e_j|$, and decrement $f(e_i)$ by one for each hyperedge e_i containing v, deleting hyperedges e_i for which $f(e_i)$ reaches 0.

The "superedge" rule (SE) is a straightforward generalization of (W2) and exploits that the deleted hyperedge e_j will be hit whenever e_i is hit:

(SE) If there is a pair of distinct hyperedges $e_i, e_j \in E$ such that $e_i \subseteq e_j$ and $f(e_i) \geq f(e_j)$, then delete e_j.

Interestingly, one can further generalize (SE) so that it may delete even hyperedges that are *not* contained one in another. Assume, for example, two distinct yet intersecting hyperedges $e_i, e_j \in E$. Any hitting set S has to contain $f(e_i)$ vertices of e_i, yet there are only $|e_i \setminus e_j|$ elements in e_i that are not also in e_j. Thus, the remaining $f(e_i) - |e_i \setminus e_j|$ elements of S must be in $e_i \cap e_j$, we say that e_i *pushes* this amount of demand to e_j. If e_i pushes at least $f(e_j)$ units of demand to e_j, then we know that e_j will be hit whenever e_i is, and can thus delete e_j.

Definition 3.1. *A hyperedge e_i* supersedes *a hyperedge e_j if $f(e_i) - |e_i \setminus e_j| \geq f(e_j)$.*

This gives rise to the following "demand pushing" rule (DP), which subsumes rule (SE) of Cotta et al. [20].

(DP) If there is a pair of distinct hyperedges $e_i, e_j \in E$ such that e_i supersedes e_j, then delete e_j.

To further generalize the data reduction rule, one can exploit that, if a hyperedge $e_j \in E$ intersects several hyperedges, then every hyperedge e_i intersecting e_j pushes some demand to $e_i \cap e_j$. If satisfying these demands requires at least $f(e_j)$ elements from e_j, then one can delete e_j. This leads to the following data reduction rule.

(LP) For a hyperedge $e_j \in E$, consider the hypergraph H_j on the same vertex set as H yet containing, for each hyperedge $e_i \in E$, a hyperedge $e_i \cap e_j$ with demand $f(e_i) - |e_i \setminus e_j|$ whenever this value is positive. Now, consider a lower bound L_j on the cardinality of any multiple hitting set for H_j.[1] If $L_j \geq f(e_j)$, then delete e_j.

[1]The lower bound L_j can be obtained, for example, via an integer linear programming relaxation.

To verify the correctness of (LP), observe that, even if the multiple hitting set in H contains $e_i \setminus e_j$ for each $e_i \in E$, it still needs to contain at least $f(e_i) - |e_i \setminus e_j|$ from $e_i \cap e_j$. If meeting these demands for all $e_i \cap e_j$ requires at least $f(e_j)$ vertices, that is, if the rule's condition is satisfied, then e_j will be hit $f(e_j)$ times anyway, since $e_j \supseteq e_i \cap e_j$ for each $e_i \in E$.

Finally, to prove a problem kernel, we will also provide a generalization of Weihe's data reduction rule (W1) to Multiple Hitting Set. Unfortunately, a previous generalization attempt of Moscato et al. [45] turned out to be wrong (see Appendix A). To state the data reduction rule, we need the following definition.

Definition 3.2. *For a pair of distinct vertices $v_i, v_j \in V$, we say that v_i* dominates (or is a dominator for) *v_j if $E(v_j) \subseteq E(v_i)$. By $\mathrm{Dom}(v_j)$, we denote all dominators for v_j.*

Note that, if v_i dominates v_j, then v_j can be replaced by v_i in any hitting set. Thus, if $|\mathrm{Dom}(v_j)| \geq \max_{e \in E(v_j)} f(e)$, then we can safely delete v_j from the hypergraph, since $\mathrm{Dom}(v_j)$ contains a sufficient amount of vertices to satisfy the demand of any hyperedge containing v_j. This gives rise to the following "multiple domination" rule.

(MD) If there is a vertex v_j such that $|\mathrm{Dom}(v_j)| \geq \max_{e \in E(v_j)} f(e)$, then delete v_j.

3.2. Problem kernel size

In this section, we show how to use the data reduction rules presented in Section 3.1 to obtain a problem kernel for Multiple Hitting Set.

For the kernel size analysis, we will use the following lemma, which relates vertex dominance and hyperedge inclusion within a hypergraph to comparability in the vinical order of the incidence graph.

Lemma 3.3. *Let H be a hypergraph and $\sqsubseteq$ be the vinical order of its incidence graph $\mathcal{I}(H)$. Then*

(i) if, for two hyperedges e_i and e_j, one has $e_i \nsubseteq e_j$ and $e_j \nsubseteq e_i$, then e_i and e_j are incomparable with respect to $\sqsubseteq$,

(ii) if, out of two vertices v_i and v_j, neither dominates the other, then v_i and v_j are incomparable with respect to $\sqsubseteq$,

Proof. (i) Note that, in the incidence graph $\mathcal{I}(H)$, one has $N(e_i) = e_i$ and $N(e_j) = e_j$. Towards a contradiction, assume $e_i \sqsubseteq e_j$. Then, by the definition of the vinical order $\sqsubseteq$, one has $e_i = N(e_i) \subseteq N[e_j] = e_j \cup \{e_j\}$. Since hyperedges do not contain other hyperedges as elements, we definitively have $e_j \notin e_i$, and thus, in fact, $e_i \subseteq e_j$, contradicting the assumption that $e_i \nsubseteq e_j$. Analogously, from $e_j \sqsubseteq e_i$ follows the contradiction $e_j \subseteq e_i$.

(ii). Note that, in the incidence graph $\mathcal{I}(H)$, one has $N(v_i) = E(v_i)$ and $N(v_j) = E(v_j)$. Towards a contradiction, assume that $v_j \sqsubseteq v_i$, that is, $E(v_j) = N(v_j) \subseteq N[v_i] = E(v_i) \cup \{v_i\}$. Since definitively $v_i \notin E(v_j)$, we in fact have $E(v_j) \subseteq E(v_i)$, contradicting the assumption that v_i does not dominate v_j. Analogously, the assumption that $v_i \sqsubseteq v_j$ contradicts the assumption that v_j does not dominate v_i. □

Theorem 3.4. *Given an instance of Multiple Hitting Set $H = (V, E)$ with $f : E \to \{1, \ldots, \alpha\}$, a problem kernel $H' = (V', E')$ with $|V'| + |E'| \leq 2\alpha\nabla(\mathcal{I}(H))$ and hyperedge demands $f' = f$ can be computed as follows:*

Step 1. Apply (SE) as long as possible,

Step 2. Apply (MD) as long as possible.

Proof. Let $H^* = (V, E^*)$ be the hypergraph obtained from H by exhaustive application of (SE), that is, H^* is H after Step 1. Consider $E_i^* := \{e \in E^* : f(e) = i\}$ for $i \in \{1, \ldots, \alpha\}$. Then, $\bigcup_{i=1}^{\alpha} E_i^* = E^*$. By the pigeonhole principle, $|E_{i^*}^*| \geq |E^*|/\alpha$ for some $i^* \in \{1, \ldots, \alpha\}$. Since the hyperedges in $e \in E_{i^*}^*$ have equal demand and survived (SE), they are not contained in each other, neither in H^* nor in H. Thus, by Lemma 3.3(i), $\nabla(\mathcal{I}(H)) \geq |E_{i^*}^*| \geq |E^*|/\alpha \geq |E'|/\alpha$, since the second step does not increase the number of hyperedges.

We have shown $|E'| \leq \alpha\nabla(\mathcal{I}(H))$. It remains to show $|V'| \leq \alpha\nabla(\mathcal{I}(H))$. To this end, let $D \subseteq V'$ be of maximal cardinality such that no vertex in D dominates any other vertex in V'. Then, we can write

$$V' = D \cup \bigcup_{u \in D} \mathrm{Dom}(u).$$

Moreover, since (MD) is inapplicable to H', we have $|\mathrm{Dom}(u)| \leq \alpha - 1$ for any $u \in D$. Also, by Lemma 3.3(ii), the vertices in D are incomparable in the vinical order of $\mathcal{I}(H')$. Therefore, $|V'| \leq \alpha|D| \leq \alpha\nabla(\mathcal{I}(H')) \leq \alpha\nabla(\mathcal{I}(H))$. □

We point out that there are graphs for which the analysis provided in Theorem 3.4 is tight: consider the hypergraph $H = (V, E)$ with n vertices $\{v_1, \dots, v_n\}$ and n hyperedges of the form $\{v_i\}$ for $i \in \{1, \dots, n\}$. Its incidence graph is a disjoint union of n copies of K_2. Its Dilworth number is n, none of (SE) and (MD) is applicable, and one has $|V| + |E| = 2n = 2\alpha\nabla(\mathcal{I}(H))$.

4. Efficient implementation using matrix multiplication

In this section, we show efficient parallel and sequential implementations of (MD) and (DP), the latter of which supersedes (SE). The algorithms presented in this section thus yield problem kernels for Multiple Hitting Set via Theorem 3.4.

Actually, they go even further: Theorem 3.4 tells us a two-step recipe for computing problem kernels for Multiple Hitting Set. However, after Step 2, Step 1 may become applicable again and shrink the input hypergraph further. The algorithms presented in this section repeat the two steps until full exhaustion. Interestingly, the asymptotic running time of our sequential algorithm will be the same regardless of whether we apply the two steps in Theorem 3.4 once or until exhaustion.

The key observation to both algorithms is the following. Consider the $(m \times n)$-incidence matrix A of a hypergraph H with vertices $v_1, \dots, v_n$ and hyperedges $e_1, \dots, e_m$, and the $(m \times m)$-matrix $I^E := AA^T$. Then,

$$I^E_{ij} = \sum_{k=1}^{n} A_{ik}A_{jk} = |e_i \cap e_j|, \qquad \text{in particular,} \qquad I^E_{ii} = |e_i|. \tag{1}$$

Thus, if $I^E_{ij} = I^E_{ii}$, then we know $e_i \subseteq e_j$ and that (SE) may be applicable. More generally, if $f(e_i) - (I^E_{ii} - I^E_{ij}) \geq f(e_j)$, we know that e_i supersedes e_j and that (DP) is applicable. Similarly, for the $(n \times n)$-matrix $I^V := A^T A$,

$$I^V_{ij} = \sum_{k=1}^{m} A_{ki}A_{kj} = |E(v_i) \cap E(v_j)|, \qquad \text{in particular,} \qquad I^V_{ii} = |E(v_i)|. \tag{2}$$

Thus, if $I^V_{ij} = I^V_{jj}$, then $E(v_j) \subseteq E(v_i)$ and we know that v_i dominates v_j in the sense of (MD). The trick for the efficient parallel algorithm is now that matrix multiplication is efficiently parallelizable. The trick for the sequential algorithm is that the matrices I^E and I^V, once computed, can be efficiently updated on vertex and hyperedge deletion.

4.1. Parallel algorithm

In this section, we prove the following theorem.

Theorem 4.1. *Given an instance of Multiple Hitting Set $H = (V, E)$ with hyperedge demands $f : E \to \{1, \dots, \alpha\}$,*

(i) a problem kernel $H' = (V', E')$ with $|V'| + |E'| \leq 2\alpha\nabla(\mathcal{I}(H))$ can be computed by an NC^1 circuit family and

(ii) H can be exhaustively reduced with respect to both (DP) and (MD) in $O(\nu(\mathcal{I}(H)) \log |H|)$ time on poly$(|H|)$ processors, where $\nu(\mathcal{I}(H))$ is the matching number of $\mathcal{I}(H)$.

The proof of Theorem 4.1 works as follows. Algorithm 1 exhaustively applies (DP), thus realizes Step 1 of Theorem 3.4. Algorithm 2 exhaustively applies (MD), thus realizes Step 2 of Theorem 3.4. Thus, implementing Algorithm 1 and Algorithm 2 as NC^1 circuit families, we will prove Theorem 4.1(i). Then, we will prove Theorem 4.1(ii) using Algorithm 3, which applies Algorithms 1 and 2 until the hypergraph does not change.

The main challenge with the proof of Theorem 4.1 is that, although the data reduction rules in Section 3.1 are correct when applied sequentially, their independent parallel application may be wrong: for example, the algorithm may find that a hyperedge e_j supersedes a hyperedge e_i and vice versa, and delete both.

Algorithm 1 applies (DP) using matrix multiplication and (1) to compute which hyperedges are superseded. Herein, the algorithm contains a tie breaker: if the algorithm finds two hyperedges superseding each other, it deletes only the hyperedge with higher index. We still have to show that an application of (DP) in this form is correct and exhaustive.

Lemma 4.2. *Algorithm 1 is correct.*

Algorithm 1: Parallel algorithm for exhaustive application of (DP).

```
Input  : Incidence matrix $A$ of a hypergraph $H = (V, E)$ and hyperedge demands $f\colon E \to \mathbb{N}$.
Result : Incidence matrix of the hypergraph obtained via exhaustive application of (DP) to $H$.
1 $I^E \leftarrow AA^T$.                                        // $I^E_{ij} = |e_i \cap e_j|$, as in (1)
2 $D \leftarrow (m \times m)$-matrix of all zeroes.
3 $R \leftarrow$ column vector of $m$ ones.                     // $R_j = 1 \iff e_j$ will be in the output
4 foreach $1 \leq i, j \leq m$ in parallel                      // $D_{ij} = 1 \iff e_i$ supersedes $e_j$
5     if $f(e_i) - (I^E_{ii} - I^E_{ij}) \geq f(e_j)$ then $D_{ij} \leftarrow 1$.
6 foreach $1 \leq i, j \leq m$ in parallel                      // Exhaustive application of (DP)
7     if $(D_{ij} = 1) \wedge ((D_{ji} = 0) \vee (i < j))$ then $R_j \leftarrow 0$.
8 return rows $j$ of $A$ for which $R_j = 1$.
```

Proof. Let H be the input hypergraph and H' the output hypergraph. We prove that

(i) every multiple hitting set for H is one for H',

(ii) every multiple hitting set for H' is one for H,

(iii) H' is exhaustively reduced with respect to (DP).

Claim (i) is trivial since H' is a sub-hypergraph of H and the algorithm does not change edge demands. For (ii), we first prove that hyperedge supersedence is transitive. Let $e_i, e_j, e_k \in E$ be hyperedges. Observe that $e_k \setminus e_i \subseteq (e_k \setminus e_j) \cup (e_j \setminus e_i)$ and, thus,

$$|e_k \setminus e_i| \leq |e_k \setminus e_j| + |e_j \setminus e_i|. \tag{3}$$

Assume that e_i supersedes e_j and that e_j supersedes e_k, that is, $f(e_j) - |e_j \setminus e_i| \geq f(e_i)$ and $f(e_k) - |e_k \setminus e_j| \geq f(e_j)$. Adding up the two inequalities, we get $f(e_k) - (|e_k \setminus e_j| + |e_j \setminus e_i|) \geq f(e_i)$, and, using (3), $f(e_k) - |e_k \setminus e_i| \geq f(e_i)$. Thus, e_i supersedes e_k. In other words, in Line 5 of Algorithm 1, we have that

$$D_{ij} = 1 \text{ and } D_{jk} = 1 \text{ implies } D_{ik} = 1. \tag{4}$$

Now, consider the directed graph G whose vertices are the hyperedges of H and that contains an arc (e_i, e_j) whenever, in Line 7,

$$D_{ij} = 1 \wedge ((D_{ji} = 0) \vee (i < j)).$$

That is, if e_i may cause the deletion of e_j in Line 7, then (e_i, e_j) is an arc. Due to (4), this directed graph is acyclic.

Finally, let S be a multiple hitting set for H' and let e_j be a hyperedge in H that is not in H'. Then e_j has some source e_i as ancestor in G. Nothing causes the deletion of e_i, so it exists in H' and, moreover, e_i supersedes e_j. Thus, since S satisfies the demand of e_i, it also satisfies the demand of e_j.

(iii) Assume that H' contains two hyperedges e_i and e_j such that e_i supersedes e_j. Then there is an arc between e_i and e_j in the directed acyclic graph G, contradicting the fact that H' contains only hyperedges that are sources in G. ☐

We have shown a parallel algorithm for exhaustively applying (DP). To prove a parallel kernelization algorithm using Theorem 3.4, we still need to exhaustively apply (MD), for which we apply Algorithm 2. It checks vertex dominance using matrix multiplication and (2). Again, a tie breaker is applied: if it finds that a vertex v_i dominates a vertex v_j and vice versa, then the vertex with lower index is considered to dominate the vertex of higher index.

Lemma 4.3. *Algorithm 2 is correct.*

Proof. Let H be the input hypergraph and H' the output hypergraph. We prove that

(i) for any multiple hitting set for H, there is a multiple hitting set of at most the same size for H',

Algorithm 2: Parallel algorithm for exhaustive application of (MD).

```
    Input  : Incidence matrix $A$ of a hypergraph $H = (V, E)$ and hyperedge demands $f\colon E \to \mathbb{N}$.
    Result : Incidence matrix of the hypergraph obtained from $H$ via exhaustive application of (MD).
1   $I^V \leftarrow A^T A$.                                              // $I_{ij} = |E(v_i) \cap E(v_j)|$, as by (2)
2   $R \leftarrow$ column vector of $n$ ones.                              // $R_j = 1 \iff v_j$ will be in the output
3   $D \leftarrow (n \times n)$-matrix of all zeroes.
4   foreach $1 \le i, j \le n$ in parallel                                // $D_{ij} = 1 \implies v_i$ dominates $v_j$
5       if $(I_{ij} = I_{jj}) \wedge ((I_{ij} \neq I_{ii}) \vee (i < j))$ then $D_{ij} \leftarrow 1$.
6   $C \leftarrow (1, \dots, 1) \cdot D$.                                  // $C_j$ = number of $i$'s such that $D_{ij} = 1$
7   foreach $1 \le j \le n$ in parallel                                   // Delete $v_j$ if $C_j \ge \max_{e \in E(v_i)} f(e)$
8       if $\bigwedge_{i=1}^{m} (A_{ij} = 0 \vee C_j \ge f(e_i))$ then $R_j \leftarrow 0$.
9   return columns $j$ of $A$ for which $R_j = 1$.
```

(ii) every multiple hitting set for H' is one for H,

(iii) H' is exhaustively reduced with respect to (DP).

Claim (ii) follows since every vertex of H' is also in H and both hypergraphs have the same hyperedges and demands.

Towards (i), consider a directed graph $G = (V, A)$ and having an arc $(v_i, v_j) \in A$ whenever $D_{ij} = 1$ after Line 5. Since a vertex v_i dominates a vertex v_j if and only if $I_{jj} = I_{ij}$, we get that $(v_i, v_j) \in A$ if and only if v_i dominates v_j and, if v_j also dominates v_i, then $i < j$. The graph G is acyclic: if there was a cycle L, then any two vertices on L would dominate each other due to the transitivity of vertex dominance. Since the vertex index cannot always increase along the cycle, there is an edge (v_i, v_j) on L such that $i > j$ and the vertices dominate each other. This contradicts the rules by which we built G.

Now, assume that a multiple hitting set S for H contains some vertex v_j that is not in H'. By Observation 2.1, v_j has some ancestor v_i such that v_i is not in H' but all its ancestors in G are in H' (possibly, $i = j$). Since any ancestor of v_i is also one of v_j, and v_i dominates v_j, that is, $E(v_j) \subseteq E(v_i)$, it follows that there are at least

$$\max_{e \in E(v_i)} f(e) \ge \max_{e \in E(v_j)} f(e)$$

ancestors of v_j left in H'. We can replace v_j by one of them in S to get a multiple hitting set for H'; if all of these ancestors are already in S, then we do not need v_j in the multiple hitting set at all.

(iii) Assume that H' contains a vertex v_j to which (MD) is applicable. That is, H' contains a set X of at least $\max_{e \in E(v_j)} f(e)$ dominators of v_j. If $D_{ij} = 1$ for each $v_i \in X$ in Line 5, then $C_j \ge |X| \ge \max_{e \in E(v_j)} f(e)$ in Line 6 and v_j would have been deleted in Line 8. Thus, there is some maximum i such that $v_i \in X$ and $D_{ij} = 0$. Since v_i dominates v_j but $D_{ij} = 0$, we know $D_{ji} = 1$ and $j < i$. Thus, v_j also dominates v_i, that is, $E(v_j) = E(v_i)$.

We now show that, in contradiction to the assumption that all vertices in X are in H', Algorithm 2 would have deleted v_i from H' in Line 8. To this end, we show that $D_{ki} = 1$ for any $v_k \in X' = (X \setminus \{v_i\}) \cup \{v_j\})$ and, thus, in Line 6,

$$C_i \ge |X'| = |X| \ge \max_{e \in E(v_j)} f(e) = \max_{e \in E(v_i)} f(e).$$

For $k = j$, since v_i dominates v_j and $D_{ij} = 0$, we have $D_{ki} = D_{ji} = 1$. For $k \neq j$, v_k dominates v_j, since $v_k \in X$. Since v_j dominates v_i, we also know that v_k dominates v_i. If $D_{ki} = 1$, then we are done. Otherwise, if $D_{ki} = 0$, then we know $D_{ik} = 1$, $j < i < k$, and $E(v_i) = E(v_j) = E(v_k)$. It follows that v_k and v_j dominate each other, that $j < k$, and thus $D_{kj} = 0$ and $D_{jk} = 1$. This contradicts the choice of i. □

We have proved the correctness of Algorithms 1 and 2. Together with a complexity analysis of these algorithms, this will yield Theorem 4.1(i). We will apply them repeatedly as long as possible to prove Theorem 4.1(ii).

Proof of Theorem 4.1. (i) By Theorem 3.4 and Lemmas 4.2 and 4.3, one application of Algorithm 1 followed by one application of Algorithm 2 is enough to produce a problem kernel of the desired size. We argue that both algorithms

Algorithm 3: Algorithm for the proof of Theorem 4.1(ii).

```
Input  : Incidence matrix $A$ of a hypergraph $H = (V, E)$ and hyperedge demands $f\colon E \to \mathbb{N}$.
Result : Incidence matrix of the hypergraph obtained from $H$ via exhaustive application of (DP) and (MD).
1 do
2     $A' \leftarrow$ apply Algorithm 1 to $A$.
3     $A \leftarrow$ apply Algorithm 2 to $A'$.
4 while $A$ changes.
5 return $A$.
```

can be realized by an NC^1-circuit family. The key point is that integer multiplication, addition, subtraction, comparison, and integer matrix multiplication can be realized as NC^1-circuits [19]. The parallel for loops can also be realized by NC^1-circuits, using one subcircuit for each pair i, j. Herein, the only thing noteworthy is that the "$\wedge$"-operator in Line 8 of Algorithm 2 can be realized by a tree of binary "$\wedge$"-operators of depth $\log m$.

(ii) It remains to analyze the number of iterations of Algorithm 3. Each iteration exhaustively applies first (DP), then exhaustively applies (MD). If, during the ℓ-th iteration, some hyperedge $e_j \in E$ is removed, then this is due to some hyperedge e_i superseding e_j at iteration ℓ but not at iteration $\ell - 1$. Thus, at iteration $\ell - 1$, some vertex v contained in $e_i \setminus e_j$ was removed, so as to satisfy the condition $f(e_i) - |e_i \setminus e_j| \geq f(e_j)$ at iteration ℓ.

Consequently, for any iteration ℓ, except the first one, there is a pair $p_\ell = \{v^{(\ell)}, e^{(\ell)}\}$ of a vertex $v^{(\ell)}$ and a hyperedge $e^{(\ell)} \in E$ such that $v^{(\ell)} \in e^{(\ell)}$, both of which are deleted by the end of iteration ℓ. Observe that p_ℓ is an edge in the incidence graph $\mathcal{I}(H)$ and that any two such edges p_i and p_j for $i < j$ are disjoint. Consequently, the pairs p_i form a matching in the incidence graph $\mathcal{I}(H)$ and the number of iterations cannot exceed $\nu(\mathcal{I}(H)) + 1$. □

4.2. Sequential algorithm

In this section, we present a sequential algorithm that exhaustively reduces hypergraphs with respect to both (DP) and (MD). Its running time is quadratic at worst and matches the running time that one would expect from first applying (DP) exhaustively and then applying (MD) exhaustively, which may be required to be repeated $\nu(\mathcal{I}(H))$ times for an exhaustive application of both, as we have seen in the proof of Theorem 4.1. Also note that the running time is subquadratic for sparse hypergraphs.

Theorem 4.4. *(DP) and (MD) can be exhaustively applied in $O(|H| \cdot (|V| + |E|))$ time.*

To prove the theorem, we first show that Algorithms 4 and 5 exhaustively apply (DP) and (MD), respectively. These algorithms are applied in a loop in Algorithm 6. We show that running time of Algorithm 6 satisfies the requirements of Theorem 4.4, which proves the theorem.

Each of the Algorithms 4 and 5 implements the exhaustive application of one of the reduction rules (DP) and (MD) in the same way as its parallel counterpart, Algorithm 1 or Algorithm 2, respectively, sequentalizing the parallel loops. There are two important differences, however. Firstly, Algorithms 4 and 5 do not compute the matrices I^E and I^V from (1) and (2), but get them as input along with two Boolean arrays R^E and R^V such that

$$R^E_i = 0 \iff e_i \in E \text{ is removed, and} \tag{5}$$

$$R^V_i = 0 \iff v_i \in V \text{ is removed.} \tag{6}$$

It is the responsibility of Algorithms 4 and 5 to update I^E, I^V, R^E and R^V in place after any hyperedge or vertex removal.

Secondly, while the parallel algorithms check all pairs of hyperedges for supersedence and all pairs of vertices for dominance, the sequential algorithms save time by narrowing their search. When Algorithm 4 removes a hyperedge e, it stores the indices of e's vertices in a list P^V. Then, Algorithm 5 searches dominators only for the vertices in P^V. Similarly, Algorithm 5 stores indices of hyperedges from $E(v)$ for any removed vertex v in a list P^E. Then, Algorithm 4 only searches for hyperedges that are superseded by those in P^E. More formally, Algorithms 4 and 5 ensure the following invariant:

Invariant 4.5. *I^E, I^V satisfy (1) and (2); R^E, R^V satisfy (5) and (6); P^E contains the indices of a superset of hyperedges that supersede others in the sense of (DP); P^V contains the indices of a superset of vertices that can be removed by (MD).*

Algorithm 4: Sequential algorithm for exhaustive application of (DP).

```
Input  : A demand function $f$; and $I^E, I^V, R^E, R^V, P^E, P^V$ satisfying Invariant 4.5 for some hypergraph $H$.
Result : All inputs are updated in-place so as to satisfy Invariant 4.5 for the hypergraph obtained from $H$ by
         exhaustive application of (DP).
1  $Q \leftarrow$ empty list                                          // $Q$ stores indices of removed hyperedges
2  for $j \in \{1, \dots, m\}$ such that $R^E_j = 1$ do                 // iterate through all possibly superseded hyperedges
3      for $i \in P^E$ such that $i \neq j$ and $R^E_i = 1$ do          // iterate through all possibly superseding hyperedges
4          $D_{ij} \leftarrow$ if $f(e_i) - (I^E_{ii} - I^E_{ij}) \geq f(e_j)$ then 1 else 0
5          $D_{ji} \leftarrow$ if $f(e_j) - (I^E_{jj} - I^E_{ji}) \geq f(e_i)$ then 1 else 0
6          if $(D_{ij} = 1) \wedge ((D_{ji} = 0) \vee (i < j))$ then     // apply rule (DP) to $e_i$ and $e_j$
7              $Q \leftarrow Q \cup \{j\}$
8              break
9  for $j \in Q$ do
10     $R^E_j \leftarrow 0$
11     for $(v_i, v_k) \in e_j \times e_j$ do
12         $I^V_{ik} \leftarrow I^V_{ik} - 1$                             // update $I^V$ after $e_j$'s removal
13     $P^V \leftarrow P^V \cup \{i \in \{1, \dots, n\} \mid v_i \in e_j\}$   // Removal of $e_j$ may make (MD) applicable to vertices in $e_j$
14 $P^E \leftarrow$ empty list
```

Lemma 4.6. *Algorithm 4 is correct.*

Proof. We prove the algorithm correctness by showing that it removes exactly those hyperedges that Algorithm 1 removes. In order to simulate the parallel run of Algorithm 1, the sequential algorithm uses nested loops in Lines 2 and 3 applying the rule (DP) in Line 6 exactly as in Algorithm 1. Since Invariant 4.5 guarantees that P^E contains the indices of all hyperedges that can supersede others, these loops iterate through all pairs of hyperedges to check for supersedence. Therefore, in Line 7, the list Q contains the indices of all hyperedges that would be removed by Algorithm 1.

We now show that, after execution of the algorithm, the data structures satisfy Invariant 4.5 with respect to the hypergraph obtained by removing the hyperedges in Q. The update happens in Line 9. First, each removed hyperedge e_j is marked as removed in R^E in Line 10. Then, in Line 11, the matrix I^V is updated using the fact that, by removing e_j, we delete it from any intersection $E(v_i) \cap E(v_k)$ for each $\{v_i, v_k\} \subseteq e_j$. Although, the matrix I^E should also be updated in the j-th row and the j-th column, these row and column will never be accessed since $R^E_j = 0$, so we do not do this update. To see why the update of P^V in Line 13 is correct, consider a vertex $v \in V$ that becomes removable by (MD) after hyperedges from Q were removed. Then $|\mathrm{Dom}(v)| \geq \max_{e \in E(v)} f(e)$ is true after the removal, but not before it. Thus, $E(v)$ must contain at least one edge from Q. Therefore, we populate P^V with all vertices contained in hyperedges from Q. Finally, P^E is cleared in Line 14, since no hyperedge can supersede another after (DP) is applied exhaustively. □

Lemma 4.7. *Algorithm 5 is correct.*

Proof. We prove the lemma by showing that the algorithm removes exactly the same vertices as Algorithm 2. To this end, the algorithm counts the dominators for a vertex in Line 6 and applies the rule (MD) in Line 7 exactly as in Algorithm 2. Since Invariant 4.5 guarantees that P^V contains the indices of all vertices that can be removed by (MD), the loops iterate through all removable vertices and for each of them — through all dominators. Therefore, all vertices that would be removed by Algorithm 2 are stored in the list Q due to Line 8.

Algorithm 5: Sequential algorithm for exhaustive application of (MD).

```
Input  : a demand function $f$; $I^E, I^V, R^E, R^V, P^E, P^V$ satisfying Invariant 4.5.
Result : All inputs are updated in-place so as to satisfy Invariant 4.5 for the hypergraph obtained from $H$ by
         exhaustive application of (MD).
1  $Q \leftarrow$ empty list                                         // $Q$ stores indices of removed vertices
2  for $j \in P^V$ such that $R^V_j = 1$ do                          // iterate through all potentially removable vertices
3      $c \leftarrow \max_{e \in E(v_j)} f(e)$                       // how many dominators does $v_j$ need to apply (MD)
4      for $i \in \{1, \dots, n\}$ such that $i \neq j$ and $R^V_i = 1$ do   // iterate through all potential dominators of $v_j$
5          if $(I^V_{ij} = I^V_{jj}) \wedge ((I^V_{ij} \neq I^V_{ii}) \vee (i < j))$ then   // check for domination, with tie breaker
6              $c \leftarrow c - 1$
7              if $c = 0$ then                                       // remove $v_j$ via (MD)
8                  $Q \leftarrow Q \cup \{j\}$
9                  break
10 for $j \in Q$ do
11     $R^V_j \leftarrow 0$
12     for $(e_i, e_k) \in E(v_j) \times E(v_j)$ do
13         $I^E_{ik} \leftarrow I^E_{ik} - 1$                          // update $I^E$ after $v_j$'s removal
14     $P^E \leftarrow P^E \cup \{i \in \{1, \dots, m\} \mid e_i \in E(v_j)\}$  // removal of $v_j$ may make (DP) applicable to hyperedges in $E(v_j)$
15 $P^V \leftarrow$ empty list
```

We now show that after the algorithm, all data structures satisfy Invariant 4.5 with respect to the hypergraph obtained by removing the vertices in Q. The data structures are updated in the loop in Line 10. First, for each $j \in Q$, vertex v_j is marked as removed in R^V. The matrix I^E is updated using the fact that, by removing v_j, we extract it from any intersection $e_i \cap e_k$ for each $\{e_i, e_k\} \subseteq E(v_k)$. Although the matrix I^V should also be updated in the j-th row and the j-th column, this matrix entry will never be accessed since $R^V_j = 0$, so we do not update it. To see why P^E is updated correctly, consider two hyperedges $e_i, e_j \in E$, such that e_i supersedes e_j after removal of vertices in Q but not before the removal. Then, $f(e_i) - |e_i \setminus e_j| \geq f(e_j)$ holds after removal, but not before removal. Thus, $|e_i \setminus e_j|$ has decreased, and since hyperedge sizes cannot increase, at least one vertex in e_i must have been removed. The index of this vertex is contained in Q. Thus, we populate P^E with all hyperedges containing any vertex whose index is in Q in Line 14. Finally, P^V is cleared in Line 15, since no vertex can be removed by (MD) after it is applied exhaustively. □

Proof of Theorem 4.4. Algorithm 6 applies (DP) and (MD) as long as possible. We first prove correctness of the algorithm. To this end, we first show that Invariant 4.5 holds with respect to the input hypergraph H at the start of the loop in Line 11. The matrix I^E is computed in Line 2 using the following observation: each vertex $v \in V$ is counted exactly once in each intersection of hyperedges in $E(v)$. It thus satisfies Invariant 4.5.

Similarly, each hyperedge $e \in E$ is counted exactly once in each intersection of vertices in e, thus I^V satisfies Invariant 4.5 in Line 5. In Lines 9 and 10 we put the indices of all hyperedges into P^E and the indices of all vertices into P^V, so that Invariant 4.5 is trivially satisfied. Lemmas 4.6 and 4.7 ensure that Invariant 4.5 holds after each iteration of the loop, in particular after the last iteration, after which none of the two algorithms reduces any more vertices or hyperedges.

Thus, according to Invariant 4.5, after the loop in Line 11, in Lines 15 and 16, from R^E and R^V we extract the hyperedges and vertices of a hypergraph H' that was obtained from H by exhaustive application of both (DP) and (MD).

We now analyze the running time of Algorithm 6. Before Line 11, the algorithm spends $O(|H| \cdot (|V| + |E|))$ time for establishing Invariant 4.5, in particular for filling the matrices I^E and I^V. Indeed, to fill matrix I^E the algorithm spends $O(|E(v_k)|^2)$ time for each vertex $v_k \in V$, which sums up to $O(\sum_{v_k \in V} |E(v_k)|^2) \in O(|E| \sum_{v_k \in V} |E(v_k)|) \in O(|E| \cdot |H|)$. Similarly, for filling I^V the running time is $O(\sum_{e_k \in E} |e_k|^2) \in O(|V| \sum_{e_k \in E} |e_k|) \in O(|V| \cdot |H|)$. Next, we analyze the accumulated running time of Algorithms 4 and 5 over all iterations.

Algorithm 6: Sequential algorithm for exhaustive application of (DP) and (MD)

```
Input  : A hypergraph $H = (V, E)$ and a demand function $f$.
Output : Hypergraph $H' = (V', E')$ obtained from $H$ by exhaustive application of (DP) and (MD).
1  $I^E \leftarrow (m \times m)$-matrix of zeroes
2  for $k \in \{1, \ldots, n\}$ and $(e_i, e_j) \in E(v_k) \times E(v_k)$ do
3      $I^E_{ij} \leftarrow I^E_{ij} + 1$                // $I^E_{ij} = |e_i \cap e_j|$, as in (1)
4  $I^V \leftarrow (n \times n)$-matrix of zeros
5  for $k \in \{1, \ldots, m\}$ and $(v_i, v_j) \in e_k \times e_k$ do
6      $I^V_{ij} \leftarrow I^V_{ij} + 1$                // $I^V_{ij} = |E(v_i) \cap E(v_j)|$, as in (2)
7  $R^E \leftarrow$ length-$m$ array of ones          // $R^E_j = 0 \iff e_j$ is removed
8  $R^V \leftarrow$ length-$m$ array of ones          // $R^V_j = 0 \iff v_j$ is removed
9  $P^E \leftarrow [1, \ldots, m]$                    // list of indices of potentially superseding hyperedges
10 $P^V \leftarrow [1, \ldots, n]$                    // list of indices of potentially removable vertices
11 do                                                 // At this point, Invariant 4.5 holds for the input hypergraph $H$
12     Apply Algorithm 4 to $(f, I^E, I^V, R^E, R^V, P^E, P^V)$
13     Apply Algorithm 5 to $(f, I^E, I^V, R^E, R^V, P^E, P^V)$
14 while $R^E$ or $R^V$ changes
15 $E' \leftarrow \{e_j \in E \mid R^E_j = 1\}$
16 $V' \leftarrow \{v_j \in V \mid R^V_j = 1\}$
17 return $H' = (V', E')$
```

In Algorithm 4, the two nested loops in Lines 2 and 3 are executed in $O(|E| \cdot |P^E|)$ time. Any hyperedge e_i is placed into P^E once during the initialization of Algorithm 6 and when Algorithm 5 removes one of its vertices, which happens at most once per vertex in e_i. Thus, the accumulated running time of Algorithm 4 is $O(|E| \sum_{e \in E}(1 + |e_i|)) \subseteq O(|E| \cdot |H|)$. The next loop in Algorithm 4 is in Line 9. It runs in time $O(|Q| + \sum_{i \in Q} |e_i|^2)$. Since each hyperedge is deleted (that is, in Q) at most once, this is $O(|E| + \sum_{i \in Q} |e_i|^2) \subseteq O(|E| + |V| \sum_{i \in Q} |e_i|) \subseteq O(|E| \cdot |H|)$.

It remains to analyze the time spent in Algorithm 5 during all iterations. Algorithm 5 has two nested loops in Lines 2 and 4, which run in $O(|V| \cdot |P^V|)$ time. A vertex v_i is placed in P^V once during the initialization of Algorithm 6 and when Algorithm 4 removes a hyperedge containing v_i. Thus, the amortized running time of Algorithm 5 is $O(|V| \sum_{v \in V}(1 + |E(v_i)|)) \subseteq O(|V| \cdot |H|)$. The loop in Line 10 runs in $O(|Q| + \sum_{i \in Q} |E(v_i)|^2)$ time. Over all executions of Algorithm 5, this sums up to $O(|V| + \sum_{i \in Q} |E(v_i)^2|) \subseteq O(|V| + |E| \sum_{i \in Q} |E(v_i)|) \subseteq O(|V| \cdot |H|)$, since no vertex is removed twice. The overall running time of Algorithm 6 is therefore $O(|H| \cdot (|V| + |E|))$. □

5. Experiments

In Section 3.1, we presented several data reduction rules for Multiple Hitting Set. Of these, Theorem 3.4 shows that an exhaustive application of (SE) followed by an exhaustive application of (MD) is enough to yield a problem kernel. It is easy to come up with examples where the additional data reduction rules have no additional data reduction effect, for example, in the case of unit demands. Also, we provided efficient implementations only of (DP) (which supersedes (SE)) and (MD), whereas application of the other data reduction rules, namely (FE) and (LP), is more time-consuming.

Thus, in this section, we experimentally evaluate the effect of various combinations of the data reduction rules presented in Section 3.1 and also analyze which combinations reach real speed-ups compared to the data reduction algorithms built into general optimization tools such as CBC.

First, in Section 5.1, we describe our experimental setup. Then, in Section 5.2, we describe experiments on real-world data arising in cancer drug design. Finally, in Section 5.3, we describe experiments on random hypergraphs modeling transportation network optimization tasks.

5.1. *Experimental setup*

Experiments were run on an AMD Ryzen 9 5950X 16-Core CPU, GeForce RTX 3090 GPU, on Ubuntu 18.04.6. We implemented our algorithms in C++ and compiled the code with GCC 9.2.1. The parallel algorithm was implemented on the GPU using OpenCL 2.2.[2] All implemented algorithms are wrapped into a Python package.[3]

ILP solving. We measure not only the data reduction effect, but also the effect that the data reduction has on the total time of solving Multiple Hitting Set instances. To this end, after data reduction, we solve Multiple Hitting Set instances using the CBC ILP solver[4] with the following ILP model:

$$\begin{aligned} \min \sum_{v\in V} x_v &\quad \text{s.t.} \\ \sum_{v\in e} x_v \geq f(e) &\quad \text{for all } e \in E \\ x_v \in \{0,1\} &\quad \text{for all } v \in V \end{aligned} \tag{7}$$

Here, x_v indicates whether v is taken into the hitting set or not.

Implementation details. The data reduction rule (FE) is implemented in a straightforward way. To implement (LP), we compute the required lower bounds L_i for each edge e_i using CBC on an LP relaxation of (7). The data reduction rules (DP) and (MD) have been implemented in the following three variants.

KernelGPU is an implementation of the parallel algorithm Algorithm 3 on the GPU using OpenCL.

KernelCPU is an implementation of the sequential Algorithm 6 on the CPU.

KernelGPU-FE alternatingly applies KernelGPU and (FE) until exhaustion.

The data reduction result of KernelGPU and KernelCPU is the same, only their running times may differ. In the OpenCL implementation of KernelGPU, as in the description of Algorithms 1 and 2, each pair of hyperedges (or vertices) is tested for supersedence (or dominance) independently. However, supersedence (or dominance) is checked by iterating over possibly common vertices (or hyperedges) sequentially. This does not pose a problem, since the GPU does not have enough processors for full parallelization anyway.

Before applying any of Algorithms 1 and 2 in Algorithm 3, the incidence matrix of the hypergraph is updated on the CPU, so as to keep only vertices and hyperedges that are not yet deleted. The $(m \times n)$-incidence matrix is stored in a space-efficient manner: if $n \gg m$, then the column for each vertex consists of $\lceil m/32 \rceil$ integers of 32 bit, so that the bits in each integer indicate the incidence relation of the vertex with 32 consecutive hyperedges. This allows us not only to allocate the optimal amount of memory for the incidence matrix, but also apply efficient bitwise operations provided by the GPU. This speeds up the linear run through the hyperedges for each vertex pair, which is the more time-consuming step when $n \gg m$. If $m \gg n$, we would want to condense rows instead of columns.

5.2. *Real-world data set*

In this section, we present experimental results on a more recent version of the data set used by Vazquez [52] and Mellor et al. [42]: in these hypergraphs, each hyperedge corresponds to a line of cancer cells, each vertex corresponds to a chemical compound. A vertex is contained in a hyperedge if the corresponding chemical compound is observed to inherit the growth of the corresponding cancer cell line. Then, Hitting Set solves the problem of selecting a minimum set of compounds that inherit the growth of all cancer cell lines in the data set. Mellor et al. [42] motivate the use of *Multiple* Hitting Set by noise in the data: to be on the safe side, each cell line is hit not only once, but several times.

In more detail, we downloaded the 2020 version of the NCI60 human anti-cancer drug screen set.[5] This data set contains response data of over 40,000 drugs against several cell lines. In the same way as Vazquez [52] and Mellor

[2]`https://www.khronos.org/opencl/`
[3]The source code is freely available at `https://gitlab.com/PavelSmirnov/hs-dilworth-kernel`.
[4]`https://github.com/coin-or/Cbc`
[5]`http://dtp.nci.nih.gov/`

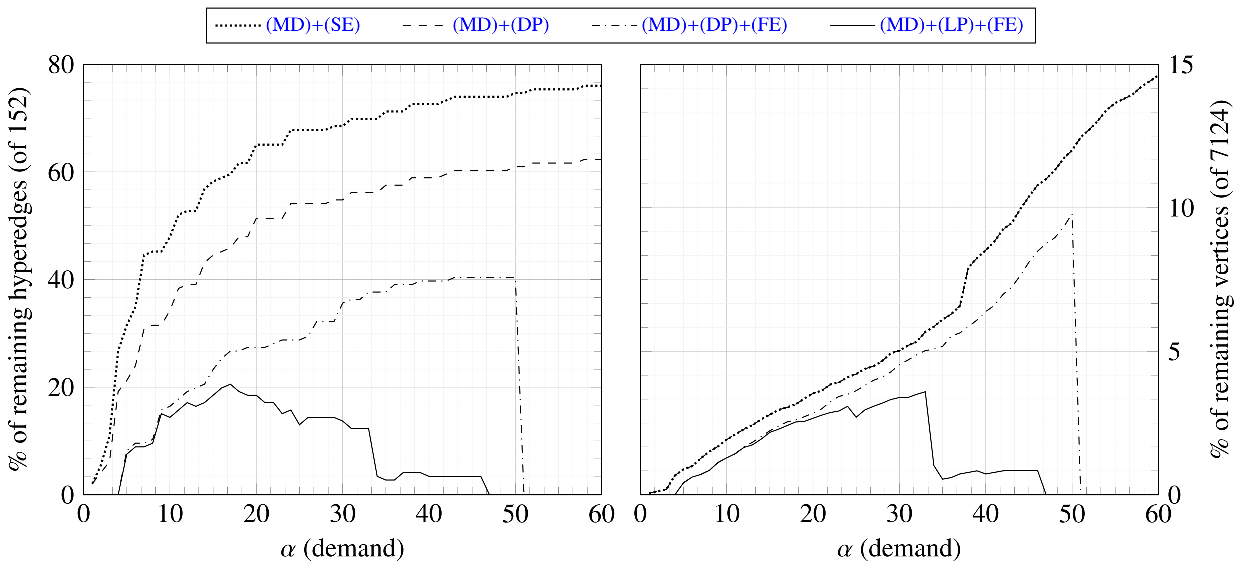

Figure 1: Data reduction effect of various data reduction rules from Section 3.1 on the real-world data set. The data reduction rules specified in the legend are applied exhaustively. On the right, the graph for (MD)+(DP) coincides with the graph for (MD)+(SE).

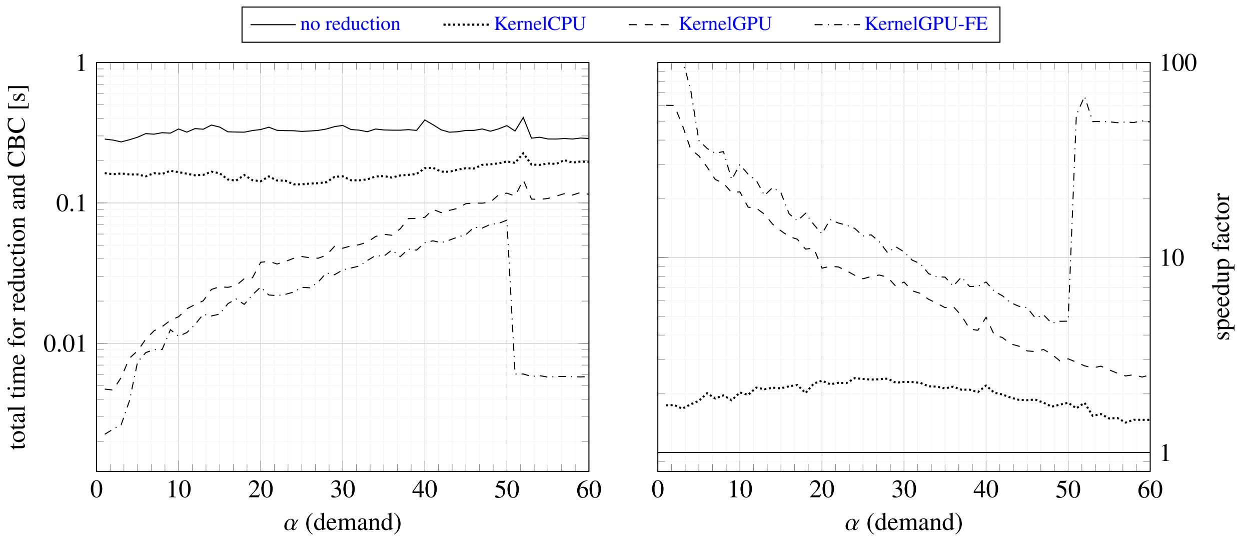

Figure 2: Total running time for applying our data reduction algorithms to our Multiple Hitting Set instances from the real-world data set and solving them with CBC afterwards.

et al. [42], we add a chemical compound to a hyperedge, corresponding to a cell line, whenever its effect on the cell line is stronger than the mean by more than two standard deviations. As a result, we obtain a hypergraph with 152 hyperedges and 7124 vertices.

The hyperedge demands are governed by a parameter α and are $f(e) = \min\{\alpha, |e|\}$ for each $e \in E$. We run experiments for all $\alpha \in \{1, \ldots, 60\}$, since for larger α the instances turned out to be trivial.

We chose this data set since the resulting Multiple Hitting Set instances are easily solved by CBC and we can thus see the trade-offs between power and speed of our data reduction rules.

Results. Figure 1 shows the data reduction effect of various combinations of the data reduction rules in Section 3.1. We see that (DP) has a significantly stronger data reduction effect than its weaker version (SE). We also see that (FE) has a strong additional data reduction effect. This is because it causes many cascading effects: it can make (DP) and (MD) applicable again. In particular, the combinations with (FE) solve the problem instance optimally for $\alpha \geq 51$ and for $\alpha \in \{1, 2, 3, 4\}$, whereas without (FE) the instances are solved optimally only for $\alpha \in \{1, 2, 3\}$.

Figure 2 shows how our data reduction algorithms influence the speed for solving our Multiple Hitting Set instances using CBC. We see that the parallel implementations of Algorithm 3 on GPUs are able to speed up CBC by a factor of more than 10. The speedup caused by KernelGPU-FE is comparable to that caused by KernelGPU, although it makes two to four (most commonly three) data reduction iterations. For $\alpha \geq 51$ the speed-up is exceptionally high because KernelGPU-FE solves the instance completely. The sequential algorithm KernelCPU speeds up CBC only by an observed factor of two. The data reduction rule (LP) slows CBC down on the considered data set. Thus, in practise, we can recommend (LP) only for hard Multiple Hitting Set instances, since solving an LP relaxation for each hyperedge in the input hypergraph is rather expensive (but takes polynomial time).

The fact that KernelGPU speeds up CBC by an order of magnitude, whereas KernelCPU gives yields speed-ups of only a factor of two, shows that, in practice, parallelization in kernelization can indeed make a significant difference. In cases where the GPU has too little memory to execute KernelGPU, it may still make sense to fall back to KernelCPU.

5.3. *Generated data set*

In this section, we present experimental results on a data set generated using a random hypergraph model proposed by Bläsius et al. [15]. According to their results, choosing the following values for the model parameters yields Hitting Set problem instances close to those arising in real-world public transportation optimization problems:

— $a = 4 \cdot 10^{-5}$. This value affects the average vertex degree, which will be close to 2 on average.

— $\beta = 3.5$. This value controls the *heterogeneity* of vertex degrees.

— $T = 0.5$. The *temperature* influences the *locality* of the network. In short, the geographic positions of a transportation network's vertices seem to be the cause of similarities in the network's connections (that is, hyperedges). For definitions and an in-depth analysis, we refer the reader to the original article of Bläsius et al. [15].

For each demand $\alpha \in \{1, \ldots, 30\}$, we generated 10 random instances with the above parameters and an upper bound of 20 000 vertices and 2 000 hyperedges. We removed possible empty hyperedges and isolated vertices from the generated hypergraphs and thus obtained 10 instances with 15 014.8 vertices and 1 903.2 hyperedges on average. The average size of a hyperedge was about 8, so we chose the maximum α to be 30. On the obtained 10 instances, we ran the implementations described in Section 5.1 for all values of α. For each particular value of α, we report the total numbers of vertices, the total number of hyperedges, the total data reduction time, and the total CBC solution time, over the 10 instances. The speed-up factor is reported with respect to the total solution time.

Results. Figure 3 shows the data reduction effect of various sets of data reduction rules. This figure is similar to Figure 1, Compared to, Figure 1, the gap between (MD)+(DP) and (MD)+(DP)+(FE) is much larger. Whereas (MD)+(DP) reduces the number of hyperedges to 40 % for $\alpha = 1$, and is almost useless for $\alpha \geq 10$, (MD)+(DP)+(FE) reduces the number of hyperedges to at most 10 % and leaves at most 1 % of the hyperedges for larger α. A similar behaviour can be seen for the vertex reduction.

Concerning the running time, Figure 4 shows that, generally speaking, among our data reduction algorithms, the KernelGPU-FE variant yields the highest speed-up, again showing the feasibility of implementing kernelization on the

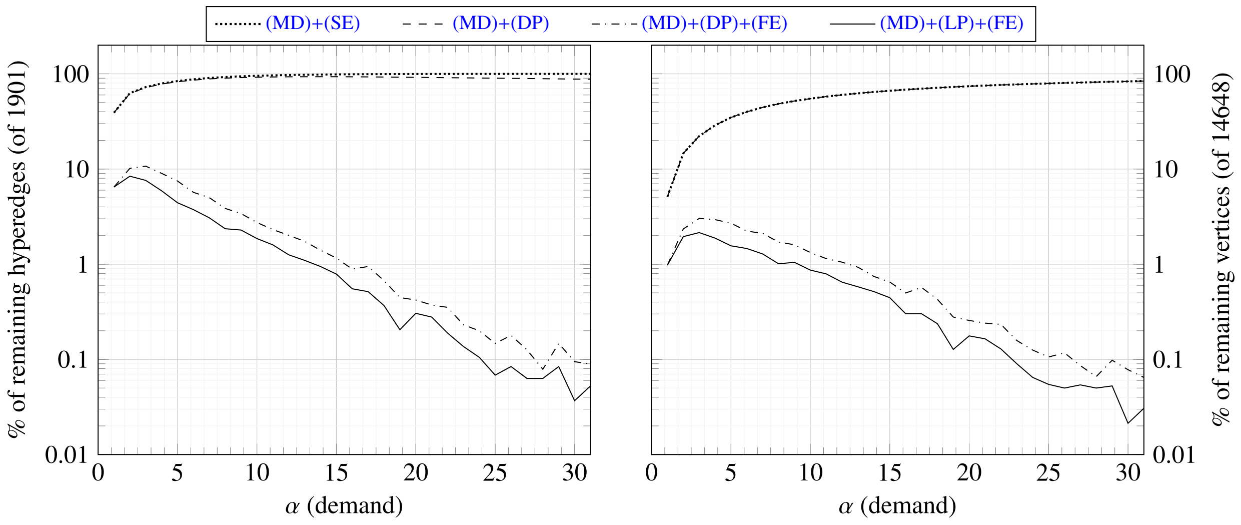

Figure 3: Data reduction effect of various data reduction rules from Section 3.1 on the generated data set. The data reduction rules specified in the legend are applied exhaustively; the graphs for (MD)+(SE) and (MD)+(DP) are nearly identical.

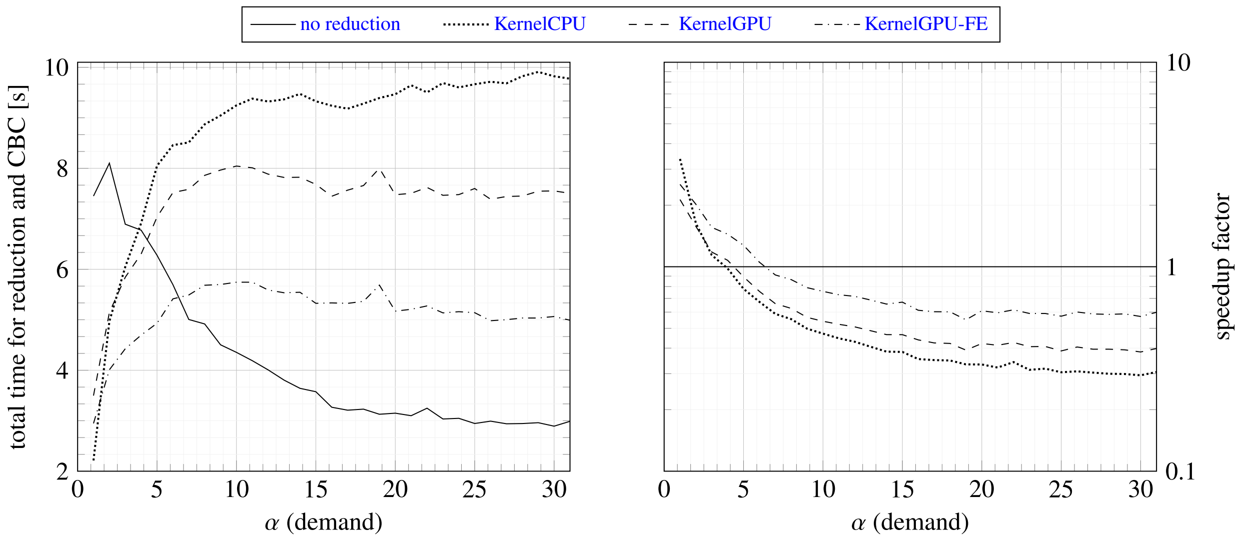

Figure 4: Total running time for applying our data reduction algorithms to our Multiple Hitting Set instances from the generated data set and solving them with CBC afterwards.

GPU. However, our data reduction algorithms yield speed-ups only for small values of α. For $\alpha \geq 7$, it is better to apply CBC without data reduction. Regarding rule (LP), it did not yield any speed-up on this data set either and we hence omitted it from the plot.

6. Conclusion

We contributed to Weihe's data reduction algorithm for Hitting Set in three ways: we proved that it yields a problem kernel with a number of vertices and hyperedges linear in the Dilworth number of the incidence graph; we presented efficient parallel and sequential realizations and experimentally evaluated them; and we generalized it to Multiple Hitting Set. However, the problem kernel with respect to the Dilworth number is just a first step to understanding the structure of hypergraphs that are well reducible by Weihe's algorithm: the problem kernel is obtained already after one exhaustive application of (DP) and one exhaustive application of (MD), yet the data reduction rules can be applied repeatedly to shrink the hypergraph, even more so in combination with (FE). This naturally raises two questions: Which natural hypergraph parameter bounds the size of the reduced instance after exhaustive applications of all of (DP), (MD), and (FE)? Can the exhaustive application of all three of them be realized effectively, say, on NC^1 circuits, or in quadratic sequential time?

Besides this, we can draw three more general conclusions from our work. The first is that it seems worthwhile to study the parameterized complexity of problems with respect to the Dilworth number, rather than with respect to the neighborhood diversity, which is already well-studied.

The second is that, both, theoretical bounds on problem kernel sizes, as well as empirical kernel size measurements, are insufficient to make conclusions on the effect of data reduction on the running time of solving real problem instances. In the case where problem instances are already solved quite well, we have seen that quite some additional effort (in form of parallelization on GPUs) might be required to speed up the solution process.

The third is what made it feasible to implement our kernelization algorithms on GPUs: note that we have shown a parallel kernelization algorithm in terms of NC^1 circuits, where each gate executes its own instruction on its own data, whereas GPUs operate in terms of the SIMD (Single Instruction, Multiple Data) model: all cores of one multiprocessor perform the same operations, yet on different data. Data reduction algorithms seem to lend themselves well to realization on GPUs since, often, they consist of a fixed set of data reduction rules, applied to different parts of the data. We thus expect that, although previously more often studied in the context of attacking NP-hard problems [8–10], parallel kernelization will have a stronger real-world impact in the context of speeding up polynomial-time algorithms on large data sets, such as linear-time data reduction was applied to speed up matching algorithms [43], or in the context of designing parallel fixed-parameter algorithms [9] for P-complete problems, which do not give in to massive parallelization.

Funding. This work was initiated during the First Workshop of Mathematical Center in Akademgorodok, which was held during July 13–17 and August 10–14, 2020, under support of the Ministry of Science and Higher Education of the Russian Federation, agreement No. 075-15-2019-167. Until the end of 2020, R. van Bevern, P. V. Smirnov, and O. Yu. Tsidulko were supported by Russian Foundation for Basic Research, project 18-501-12031 NNIO_a.

References

[1] F. N. Abu-Khzam, *A kernelization algorithm for d-Hitting Set*, Journal of Computer and System Sciences **76** (2010), 524–531, doi:10.1016/j.jcss.2009.09.002.

[2] F. N. Abu-Khzam, S. Lamm, M. Mnich, A. Noe, C. Schulz, and D. Strash, *Recent advances in practical data reduction*, H. Bast, C. Korzen, U. Meyer, and M. Penschuck (eds.), *Algorithms for Big Data*, Springer, *LNCS*, vol. 13201, 2022, pp. 97–133, doi:10.1007/978-3-031-21534-6_6.

[3] T. Achterberg, R. E. Bixby, Z. Gu, E. Rothberg, and D. Weninger, *Presolve reductions in mixed integer programming*, INFORMS Journal on Computing **32** (2020), 473–506, doi:10.1287/ijoc.2018.0857.

[4] A. Agrawal, N. Aravind, S. Kalyanasundaram, A. S. Kare, J. Lauri, N. Misra, and I. V. Reddy, *Parameterized complexity of happy coloring problems*, Theoretical Computer Science (2020), doi:10.1016/j.tcs.2020.06.002.

[5] J. Alman, M. Mnich, and V. Vassilevska Williams, *Dynamic parameterized problems and algorithms*, ACM Transactions on Algorithms **16** (2020), doi:10.1145/3395037.

[6] S. Arora and B. Barak, *Computational Complexity: A modern approach*, Cambridge University Press, 2009.

[7] M. Bannach, Z. Heinrich, R. Reischuk, and T. Tantau, *Dynamic kernels for hitting sets and set packing*, Algorithmica **84** (2022), 3459–3488, doi:10.1007/s00453-022-00986-0.

[8] M. Bannach, M. Skambath, and T. Tantau, *Kernelizing the hitting set problem in linear sequential and constant parallel time*, S. Albers (ed.), *SWAT 2020*, Schloss Dagstuhl–Leibniz-Zentrum für Informatik, *LIPIcs*, vol. 162, 2020, pp. 9:1–9:16, doi:10.4230/LIPIcs.SWAT.2020.9.

[9] M. Bannach, C. Stockhusen, and T. Tantau, *Fast parallel fixed-parameter algorithms via color coding*, T. Husfeldt and I. Kanj (eds.), *IPEC 2015*, Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2015, *LIPIcs*, vol. 43, doi:10.4230/LIPICS.IPEC.2015.224.

[10] M. Bannach and T. Tantau, *Computing hitting set kernels by AC_0-circuits*, Theory of Computing Systems **62** (2020), 374–399, doi:10.1007/s00224-019-09941-z.

[11] R. van Bevern, T. Fluschnik, and O. Yu. Tsidulko, *On approximate data reduction for the Rural Postman Problem: Theory and experiments*, Networks **76** (2020), 485–508, doi:10.1002/net.21985.

[12] R. van Bevern, A. Melnikov, P. Smirnov, and O. Tsidulko, *On data reduction for dynamic vector bin packing*, Operations Research Letters (in press), doi:10.1016/j.orl.2023.06.005.

[13] R. van Bevern, H. Moser, and R. Niedermeier, *Approximation and tidying—a problem kernel for s-Plex Cluster Vertex Deletion*, Algorithmica **62** (2012), 930–950, doi:10.1007/s00453-011-9492-7.

[14] R. van Bevern and P. V. Smirnov, *Optimal-size problem kernels for d-hitting set in linear time and space*, Information Processing Letters **163** (2020), 105 998, doi:10.1016/j.ipl.2020.105998.

[15] T. Bläsius, P. Fischbeck, T. Friedrich, and M. Schirneck, *Understanding the effectiveness of data reduction in public transportation networks*, K. Avrachenkov, P. Prałat, and N. Ye (eds.), *WAW 2019*, Springer, *LNCS*, vol. 11631, 2019, pp. 87–101, doi:10.1007/978-3-030-25070-6_7.

[16] T. Bläsius, T. Friedrich, J. Lischeid, K. Meeks, and M. Schirneck, *Efficiently enumerating hitting sets of hypergraphs arising in data profiling*, S. Kobourov and H. Meyerhenke (eds.), *ALENEX 2019*, SIAM, 2019, pp. 130–143, doi:10.1137/1.9781611975499.11.

[17] G. Brewka, M. Thimm, and M. Ulbricht, *Strong inconsistency*, Artificial Intelligence **267** (2019), 78–117, doi:10.1016/j.artint.2018.11.002.

[18] T. Calamoneri and R. Petreschi, *On pairwise compatibility graphs having Dilworth number k*, Theoretical Computer Science **547** (2014), 82–89, doi:10.1016/j.tcs.2014.06.024.

[19] S. A. Cook, *A taxonomy of problems with fast parallel algorithms*, Information and Control **64** (1985), 2–22, doi:10.1016/S0019-9958(85)80041-3.

[20] C. Cotta, C. Sloper, and P. Moscato, *Evolutionary search of thresholds for robust feature set selection: Application to the analysis of microarray data*, G. R. Raidl, S. Cagnoni, J. Branke, D. W. Corne, R. Drechsler, Y. Jin, C. G. Johnson, P. Machado, E. Marchiori, F. Rothlauf, G. D. Smith, and G. Squillero (eds.), *Applications of Evolutionary Computing*, Springer, 2004, pp. 21–30, doi:10.1007/978-3-540-24653-4_3.

[21] P. Damaschke, *Parameterized enumeration, transversals, and imperfect phylogeny reconstruction*, Theoretical Computer Science **351** (2006), 337–350, doi:10.1016/j.tcs.2005.10.004.

[22] H. Dell and D. van Melkebeek, *Satisfiability allows no nontrivial sparsification unless the polynomial-time hierarchy collapses*, Journal of the ACM **61** (2014), 23:1–23:27, doi:10.1145/2629620.

[23] R. P. Dilworth, *A decomposition theorem for partially ordered sets*, The Annals of Mathematics **51** (1950), 161, doi:10.2307/1969503.

[24] M. Dom, D. Lokshtanov, and S. Saurabh, *Kernelization lower bounds through colors and IDs*, ACM Transactions on Algorithms **11** (2014), 13, doi:10.1145/2650261.

[25] S. Fafianie and S. Kratsch, *Streaming kernelization*, E. Csuhaj-Varjú, M. Dietzfelbinger, and Z. Ésik (eds.), *MFCS 2014*, Springer, *LNCS*, vol. 8635, 2014, pp. 275–286, doi:10.1007/978-3-662-44465-8_24.

[26] K. Fazekas, F. Bacchus, and A. Biere, *Implicit hitting set algorithms for maximum satisfiability modulo theories*, D. Galmiche, S. Schulz, and R. Sebastiani (eds.), *IJCAR 2018*, Springer, *LNCS*, vol. 10900, 2018, pp. 134–151, doi:10.1007/978-3-319-94205-6_10.

[27] M. R. Fellows, *Parameterized complexity: The main ideas and connections to practical computing*, R. Fleischer, B. Moret, and E. M. Schmidt (eds.), *Experimental Algorithmics: From Algorithm Design to Robust and Efficient Software*, Springer, 2002, pp. 51–77, doi:10.1007/3-540-36383-1_3.

[28] S. Felsner, V. Raghavan, and J. Spinrad, *Recognition algorithms for orders of small width and graphs of small Dilworth number*, Order **20** (2003), 351–364, doi:10.1023/B:ORDE.0000034609.99940.fb.

[29] J. Flum and M. Grohe, *Parameterized Complexity Theory*, Texts in Theoretical Computer Science, An EATCS Series, Springer, 2006, doi:10.1007/3-540-29953-X.

[30] S. Foldes and P. L. Hammer, *The Dilworth number of a graph*, B. Alspach, P. Hell, and D. Miller (eds.), *Algorithmic Aspects of Combinatorics*, Elsevier, *Annals of Discrete Mathematics*, vol. 2, 1978, pp. 211–219, doi:https://doi.org/10.1016/S0167-5060(08)70334-0.

[31] V. Froese, R. van Bevern, R. Niedermeier, and M. Sorge, *Exploiting hidden structure in selecting dimensions that distinguish vectors*, Journal of Computer and System Sciences **82** (2016), 521–535, doi:10.1016/j.jcss.2015.11.011.

[32] R. Ganian, *Using neighborhood diversity to solve hard problems*, Available on arXiv, 2012, URL `http://arxig.org/abs/1201.3091`.

[33] L. Gargano and A. A. Rescigno, *Complexity of conflict-free colorings of graphs*, Theoretical Computer Science **566** (2015), 39–49, doi:10.1016/j.tcs.2014.11.029.

[34] T. Gavenčiak, M. Koutecký, and D. Knop, *Integer programming in parameterized complexity: Five miniatures*, Discrete Optimization (2020), 100 596, doi:10.1016/j.disopt.2020.100596.

[35] R. Gera, T. W. Haynes, S. T. Hedetniemi, and M. A. Henning, *An annotated glossary of graph theory parameters, with conjectures*, *Graph Theory*, Springer, 2018, pp. 177–281, doi:10.1007/978-3-319-97686-0_14.

[36] C. Hoáng and N. Mahadev, *A note on perfect orders*, Discrete Mathematics **74** (1989), 77–84, doi:10.1016/0012-365X(89)90200-8.

[37] F. Hüffner, C. Komusiewicz, H. Moser, and R. Niedermeier, *Fixed-parameter algorithms for cluster vertex deletion*, Theory of Computing Systems **47** (2010), 196–217, doi:10.1007/s00224-008-9150-x.

[38] R. M. Karp, *Reducibility among combinatorial problems*, R. E. Miller, J. W. Thatcher, and J. D. Bohlinger (eds.), *Complexity of Computer Computations*, The IBM Research Symposia Series, Springer, 1972, pp. 85–103, doi:10.1007/978-1-4684-2001-2_9.

[39] S. Kratsch, *Polynomial kernelizations for MIN $F^+\Pi_1$ and MAX NP*, Algorithmica **63** (2012), 532–550, doi:10.1007/s00453-011-9559-5.

[40] M. Lampis, *Algorithmic meta-theorems for restrictions of treewidth*, Algorithmica (2012), 19–37, doi:10.1007/s00453-011-9554-x.

[41] L. Mathieson, A. Mendes, J. Marsden, J. Pond, and P. Moscato, *Computer-aided breast cancer diagnosis with optimal feature sets: Reduction rules and optimization techniques*, J. M. Keith (ed.), *Bioinformatics: Volume II: Structure, Function, and Applications*, Springer, New York, NY, 2017, pp. 299–325, doi:10.1007/978-1-4939-6613-4_17.

[42] D. Mellor, E. Prieto, L. Mathieson, and P. Moscato, *A kernelisation approach for multiple d-hitting set and its application in optimal multi-drug therapeutic combinations*, PLOS ONE **5** (2010), doi:10.1371/journal.pone.0013055.

[43] G. B. Mertzios, A. Nichterlein, and R. Niedermeier, *The power of linear-time data reduction for maximum matching*, Algorithmica **82** (2020), 3521–3565, doi:10.1007/s00453-020-00736-0.

[44] E. Moreno-Centeno and R. M. Karp, *The implicit hitting set approach to solve combinatorial optimization problems with an application to multigenome alignment*, Operations Research **61** (2013), 453–468, doi:10.1287/opre.1120.1139.

[45] P. Moscato, L. Mathieson, A. Mendes, and R. Berretta, *The electronic primaries: Predicting the U. S. presidency using feature selection with safe data reduction*, V. Estivill-Castro (ed.), *ACSC 2005*, ACS, *CRPIT*, vol. 38, 2005, pp. 371–380.

[46] H. Moser, *Finding Optimal Solutions for Covering and Matching Problems*, Cuvillier, Göttingen, Germany, 2010.

[47] R. Niedermeier and P. Rossmanith, *An efficient fixed-parameter algorithm for 3-Hitting Set*, Journal of Discrete Algorithms **1** (2003), 89–102, doi:10.1016/S1570-8667(03)00009-1.

[48] R. O'Callahan and J.-D. Choi, *Hybrid dynamic data race detection*, R. Eigenmann and M. Rinard (eds.), *PPoPP'03*, ACM, 2003, pp. 167–178, doi:10.1145/781498.781528.

[49] R. Reiter, *A theory of diagnosis from first principles*, Artificial Intelligence **32** (1987), 57–95, doi:10.1016/0004-3702(87)90062-2.

[50] M. Sorge, H. Moser, R. Niedermeier, and M. Weller, *Exploiting a hypergraph model for finding Golomb rulers*, Acta Informatica **51** (2014), 449–471, doi:10.1007/s00236-014-0202-1.

[51] M. Sorge and M. Weller, *The graph parameter hierarchy*, 2019, URL `https://manyu.pro/assets/parameter-hierarchy.pdf`.

[52] A. Vazquez, *Optimal drug combinations and minimal hitting sets*, BMC Systems Biology **3** (2009), 81, doi:10.1186/1752-0509-3-81.

[53] K. Weihe, *Covering trains by stations or the power of data reduction*, R. Battiti and A. A. Bertossi (eds.), *Proceedings of Algorithms and Experiments (ALEX 1998)*, 1998, pp. 1–8.

Appendix A. Error in the data reduction of Moscato et al. [45]

Moscato et al. [45, Section 2.2, Rule 2] attempt to generalize (W2) to Multiple Hitting Set using the following data reduction rule: if there are two vertices v_i, v_j such that $E(v_j) \subseteq E(v_i)$, and for every $e \in E(v_j)$ one has $|e| > f(e)$, then delete v_j can be deleted. We now show that this rule, in fact, can change the cardinality of an optimal solution.

Consider the following hypergraph with three hyperedges $e_1 = \{v_1, v_2\}$, $e_2 = \{v_2, v_3, v_4\}$, and $e_3 = \{v_2, v_3, v_5\}$ with equal demands $f(e_1) = f(e_2) = f(e_3) = 2$. It has a multiple hitting set $\{v_1, v_2, v_3\}$, yet the rule suggested by Moscato et al. [45] could delete v_3 due to v_2. After deletion of v_2, however, the minimum multiple hitting set is $\{v_1, v_2, v_4, v_5\}$ and has cardinality four.