A Branch-&-Bound Algorithm for Fractional Hypertree Decomposition

Zongyan He
The Chinese University of Hong Kong
Hong Kong, China
zyhe@se.cuhk.edu.hk

Jeffrey Xu Yu
The Chinese University of Hong Kong
Hong Kong, China
yu@se.cuhk.edu.hk

ABSTRACT

Conjunctive queries (*CQ*s) have been widely used in database systems in which acyclic *CQ*s can be computed efficiently, whereas cyclic *CQ*s may not. Here, a *CQ* is acyclic if its hypergraph representation $\mathcal{H}$ is acyclic. In order to find a class of *CQ*s that are "mildly cyclic", hypertree decompositions (HDs) have been studied. The quality of such HDs is by the so-called hypertree width. The class of acyclic queries is the queries whose hypertree width is 1, and a mildly cyclic *CQ* can be processed efficiently if its hypertree width is bounded. There are several HDs, such as tree decomposition (TD), generalized hypertree decomposition (GHD), fractional hypertree decomposition (FHD), as well as hypertree decomposition (HD). The minimum hypertree width by FHD is the smallest among all, and it is NP-complete to check if the minimum hypertree width by FHD exists for a given hypertree width at most k. In the literature, there is no dynamic programming (*DP*) algorithm or branch-&-bound algorithm reported to compute FHD. In this paper, we show that there is a *DP* algorithm for FHD, and we give a branch-&-bound algorithm based on our *DP* algorithm to compute FHD with upper/lower bounds. We confirm the effectiveness and efficiency of our algorithm by testing all 3,648 hypergraphs given in a benchmark for HDs, and we also confirm our approach in query evaluation in real database systems.

PVLDB Reference Format:
Zongyan He and Jeffrey Xu Yu. A Branch-&-Bound Algorithm for Fractional Hypertree Decomposition. PVLDB, 17(13): 4655 - 4667, 2024.
doi:10.14778/3704965.3704973

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/hzy9819/BB4fhd.

1 INTRODUCTION

Conjunctive queries (*CQ*s) are the queries specified by first-order logic using conjunctions, and have been widely used in database systems over decades. In *CQ*s, there are acyclic queries and cyclic queries. A query is acyclic iff its hypergraph representation is acyclic, that is, the hypergraph has a join-tree representation [7]. The join-tree based definition coincides with α-acyclic as defined in [12]. And an acyclic query can be efficiently processed by Yannakakis's algorithm [38]. Whether a query is acyclic can be recognized in linear time as the acyclicity of its hypergraph can be recognized in linear time [36]. However, a significant proportion of *CQ*s that have been used in the real world are not acyclic. In *Hyperbench* [14], it includes 70 cyclic queries in SPARQL [9] and 354 cyclic queries in WIKIDATA [28]. There is a very large cyclic query (`tpcds64_0`) in the TPC-DS benchmark, which contains 633 vertices and 38 hyperedges. Also in [30], it studies cycle queries and grid queries over more than 50 relations. There have been great efforts to find a class of *CQ*s that is not acyclic but are "mildly cyclic" [17] in *CQ*s so that we can process such mildly cyclic queries efficiently. In other words, it is a question of how to find a cyclic query that cannot be represented by a join-tree, but can be represented by a tree-like structure. To this goal, hypertree decompositions (HDs) have been studied to decompose a hypergraph for a given *CQ* into a hypertree.

Hypertree decompositions have been widely used as a basic component in query processing [24, 26, 27, 35, 37, 39]. In brief, for a cyclic *CQ* query q, HD decomposes its hypergraph into a hypertree, where a node in the hypertree represents a subset of vertices in q, called a "bag", in the hypergraph. Here, a node in the hypertree represents a subquery of q. The query processing of such q is to process every node (a subquery) in the hypertree, and then process q as an acyclic query over the intermediate relations for all nodes. The quality of HDs depends on the size of the bag, called a "width", and the size bound of the join result is $|D|^{width}$ where $|D|$ is the database size. The key behind HD is to find a hypertree in which the largest width is as small as possible. Intuitively, with a smaller width, each subquery represented by a node in the hypertree can be processed more efficiently. There have been several HDs being studied with a different width measurement method. Such as tree decomposition (TD), generalized hypertree decomposition (GHD) [20], fractional hypertree decomposition (FHD) [23], as well as hypertree decomposition (HD) [20]. The widths are known as $\mathsf{tw}(\mathcal{H})$, $\mathsf{ghtw}(\mathcal{H})$, $\mathsf{fhtw}(\mathcal{H})$, and $\mathsf{htw}(\mathcal{H})$, respectively. It is proved that $\mathsf{fhtw}(\mathcal{H}) \leq \mathsf{ghtw}(\mathcal{H}) \leq \mathsf{tw}(\mathcal{H})$, where $\mathsf{tw}(\mathcal{H})$ is too loose to be used, and $\mathsf{fhtw}(\mathcal{H})$ is the most effective width to be used which is recognized by the AGM bound [6] and can be implemented by Worst Case Optimal Join (WCOJ) [31]. However, finding such width (either $\mathsf{fhtw}(\mathcal{H})$ or $\mathsf{ghtw}(\mathcal{H})$) is hard, as it is NP-complete to check if its width is at most up to a user-given k. To address the NP-completeness, HD is proposed in [20] as one whose width (e.g., $\mathsf{htw}(\mathcal{H})$) is greater than $\mathsf{ghtw}(\mathcal{H})$ but can be checked in polynomial time.

Proceedings of the VLDB Endowment, Vol. 17, No. 13 ISSN 2150-8097.
doi:10.14778/3704965.3704973

The advantages of HDs in selecting a query plan to process are: ❶ stronger theoretically guaranteed, ❷ bounded size of the intermediate results, and ❸ highly parallelizable execution [18]. And HDs are used in query optimization in database systems. ① EmptyHeaded [3] and LevelHeaded [2] use FHD as the basis of query optimizer and code generator for asymptotically stronger runtime guarantees and bounded intermediate results in query processing. ② Secco [40] uses FHD in query optimization, and processes an FHD-based plan in parallel. ③ FBench [32] uses a similar notion of FHD, called d-tree, in query processing. ④ SparkSQL$^+$ [11] combines FHD and the worst-case optimal join as a query plan to process. ⑤ In [16], it uses HD in query optimization, and designs a hybrid optimizer with theoretical guarantees to process queries in PostgreSQL. ⑥ HDs are also used as a data structure in join algorithms [24–26, 35, 37] and graph algorithms [39]. In addition, HDs can also be used in solving constraint satisfaction problems (CSPs) [5, 10, 41].

In this paper, we study computing the fractional hypertree decomposition (FHD), which is mainly studied in theory. In the reported studies, a database system either needs to use a brute force algorithm to find FHD, or requests a user-given FHD to be used for the given conjunctive query. For instance, a brute force algorithm is implemented in EmptyHeaded [3], which can only find the optimal FHD (or the minimum $\mathsf{fhtw}(\mathcal{H})$) within a reasonable time if the hypergraph representation $\mathcal{H}$ for a conjunctive query has 6 vertices at most. It is important to note that *CQ*s in the real world can have dozens or hundreds of vertices in their hypergraph representations. The only existing practical approach to computing FHD is using an *SMT* (Satisfiability Modulo Theories) solver [13]. In brief, it is to check if the minimum $\mathsf{fhtw}(\mathcal{H})$ is at most k by encoding such check as an *SMT* problem, and solve the *SMT* problem using an *SMT* solver. There are several problems. First, it is not to find the minimum $\mathsf{fhtw}(\mathcal{H})$ but to check whether fhtw(H) is less than or equal to a given width k. As FHD is fractional hypertree decomposition, the minimum $\mathsf{fhtw}(\mathcal{H})$ is a real number that is hard to be bound and found still. Second, an *SMT* solver is a black-box, and will give an answer if it can terminate. But, it may not terminate in a reasonable time (e.g., hours).

Contributions: First, we give an approach that is not based on checking the existence of $\mathsf{fhtw}(\mathcal{H})$ with a user-given real number k at most. We show that FHD can be computed using a dynamic programming (*DP*) algorithm. Second, we give a branch-&-bound algorithm with several upper/lower bounds, which makes our *DP* algorithm much more efficient. Our branch-&-bound algorithm is an anytime algorithm that can terminate at any time. That is, we can give a feasible solution within the time limit, and we can give a better solution if we have more time. To the best of our knowledge, there is no *DP* algorithm and no branch-&-bound algorithm reported for FHD yet. Third, we discuss how to reduce the cost of computing some fundamental operations used in FHD computing. Finally, we conduct extensive experimental studies to test all 3,648 hypergraphs given in *Hyperbench* [14], and confirm the effectiveness in query evaluation in real database systems.

2 PRELIMINARIES

A hypergraph is defined as $\mathcal{H} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is a set of vertices and $\mathcal{E}$ is a set of hyperedges, where a hyperedge $e \in \mathcal{E}$ is a subset of $\mathcal{V}$. Two vertices, u and v in $\mathcal{V}$ are adjacent, if there exists a hyperedge $e \in \mathcal{E}$ that contains both u and v. Two adjacent vertices are neighbors to each other. The neighborhood of a vertex v in $\mathcal{V}$, denoted by $N_{\mathcal{H}}(v)$, is all the vertices adjacent to v such that $N_{\mathcal{H}}(v) = \{u \in \mathcal{V} | u \neq v \wedge \exists e \in \mathcal{E} \text{ s.t. } \{u, v\} \subseteq e\}$, and the closed neighborhood of a vertex v is $N_{\mathcal{H}}[v] = N_{\mathcal{H}}(v) \cup \{v\}$. The degree of a vertex v in a hypergraph $\mathcal{H}$ is the number of hyperedges that contain v. A simple hypergraph is a hypergraph without loops and repeated edges, where a loop is a hyperedge with a single vertex and repeated edges are the hyperedges that contain the same set of vertices. A path between two vertices, u and v in a hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ is a sequence of vertices, $u = v_1, v_2, \cdots, v_k = v$ such that every consecutive vertices, v_i, v_{i+1}, for $1 \leq i \leq k-1$, appear in a hyperedge in $\mathcal{E}$. Given a subset W ($\subseteq \mathcal{V}$) in a hypergraph $\mathcal{H}$, a subset $\mathcal{V}'$ ($\subseteq \mathcal{V}$) is called a $[W]$-component if $\mathcal{V}' \subseteq \mathcal{V} \setminus W$ is a maximal connected nonempty set, where every pair of vertices, u and v, in $\mathcal{V}'$ are connected via paths that do not pass any vertices in W. In this regard, W is called a separator that separates $\mathcal{H}$ into several $[W]$-components. Given a subset $\mathcal{V}'$ ($\subseteq \mathcal{V}$) in a hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$, an edge cover of $\mathcal{V}'$ is a subset of $\mathcal{E}'$ ($\subseteq \mathcal{E}$) such that for every $v \in \mathcal{V}'$, there is a hyperedge $e \in \mathcal{E}'$ that contains v. Such an edge cover of $\mathcal{V}'$ can be defined using a λ function, which assigns a hyperedge 1 if it is used to cover V', 0 otherwise, as $\lambda : \mathcal{E} \rightarrow \{0, 1\}$. It is obvious that, for every $v \in \mathcal{V}'$, the sum of the weight of the edges $e \in \mathcal{E}$ that contains v, denoted as $w(v)$, should be greater than or equal to 1.

$$w(v) = \sum_{e \in \mathcal{E} \wedge v \in e} \lambda(e) \geq 1 \tag{1}$$

Let the weight of λ, denoted as $\mathsf{weight}(\lambda)$, be the sum of weights as follows.

$$\mathsf{weight}(\lambda) = \sum_{e \in \mathcal{E}} \lambda(e) \tag{2}$$

The minimum edge cover of $\mathcal{V}'$ is the edge cover that has a minimum weight, denoted as $EC(\mathcal{V}')$, and the minimum edge cover weight is the weight of the minimum edge cover, denoted as $\rho(\mathcal{V}')$.

$$EC(\mathcal{V}') = \arg\min_{\lambda} \mathsf{weight}(\lambda) \tag{3}$$

$$\rho(\mathcal{V}') = \min_{\lambda} \mathsf{weight}(\lambda) \tag{4}$$

By combining the Eq. (1)-(4), we have the minimum edge cover of $\mathcal{V}'$ regarding $\mathcal{E}$ as follows.

$$\rho(\mathcal{V}', \mathcal{E}) = \min_{\lambda} \sum_{e \in \mathcal{E}} \lambda(e) \quad \text{s.t.} \quad w(v) \geq 1, \text{ for each } v \in \mathcal{V}' \tag{5}$$

The definition of fractional edge cover (*FEC*) can be defined in a similar manner using a weight function, $\gamma : \mathcal{E} \rightarrow [0, 1]$, which gives a hyperedge a fractional weight between 0 and 1 if it is used to cover, 0 otherwise. In the following, we use $\rho^*(\mathcal{V}', \mathcal{E})$ for *FEC* by replacing λ with γ in Eq. (5).

$$\rho^*(\mathcal{V}', \mathcal{E}) = \min_{\gamma} \sum_{e \in \mathcal{E}} \gamma(e) \quad \text{s.t.} \quad w(v) \geq 1, \text{ for each } v \in \mathcal{V}' \tag{6}$$

Example 2.1: The hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ in Fig. 1 represents a conjunctive query over 4 relations, $R_1(A, B, D, E) \wedge R_2(C, D, E, F) \wedge R_3(A, C, H) \wedge R_4(F, G)$. Here, $\mathcal{V} = \{A, B, C, D, E, F, G, H\}$ and $\mathcal{E} = \{e_1, e_2, e_3, e_4\}$ for $e_1 = \{A, B, D, E\}$, $e_2 = \{C, D, E, F\}$, $e_3 = \{A, C, H\}$,

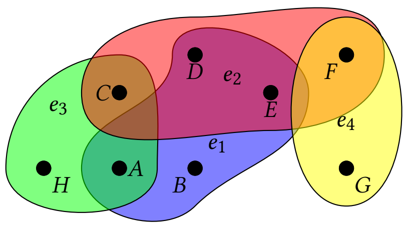


Figure 1: A hypergraph for a conjunctive query

and $e_4 = \{F, G\}$. Let $W = \{A, C, D\}$ be a separator. The hypergraph $\mathcal{H}$ is separated into two $[W]$-components, $\{H\}$ and $\{B, E, F, G\}$.

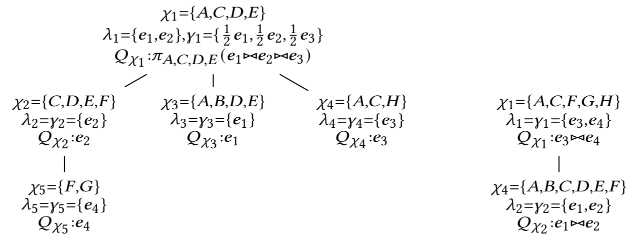
Example 2.2: Consider $\mathcal{V}' = \{A, C, D, E\}$ for the hypergraph $\mathcal{H}$ in Example 2.1. An edge cover of $\mathcal{V}'$ is $EC(\mathcal{V}') = \{e_1, e_2\}$, with weight$(\lambda) = 2$, for $\lambda(e_1) = \lambda(e_2) = 1$. $EC(\mathcal{V}')$ is the minimum edge cover of $\mathcal{V}'$. The minimum fractional edge cover is $FEC(\mathcal{V}') = \{e_1, e_2, e_3\}$ with weight$(\gamma) = \frac{3}{2}$ for $\gamma(e_1) = \gamma(e_2) = \gamma(e_3) = \frac{1}{2}$.

A primal graph of a hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ is a graph denoted as $G(\mathcal{H}) = (\mathcal{V}, E_G)$, where $\mathcal{V}$ is the set of vertices in $\mathcal{H}$, and E_G is a set of edges, (u, v), if u and v appear in a hyperedge in $\mathcal{E}$. Here, the primal graph $G(\mathcal{H})$ is a simple graph that represents the adjacency of vertices $\mathcal{V}$ in the corresponding hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$.

There are several ways to decompose a hypergraph. A tree decomposition (TD) of a hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ is a pair (T, χ). Here, $T = (V_T, E_T)$ is a tree, and χ is a function that maps each node t_i in T to a subset of vertices in $\mathcal{V}$ such as $\chi(t_i) \subseteq \mathcal{V}$. Note that $\chi(t_i)$ is called the bag at a node t_i. The conditions for TD to be held are as follows. ❶ Each vertex $v \in \mathcal{V}$ is covered such that there is a node t_i in T for $v \in \chi(t_i)$. ❷ Each hyperedge $e \in \mathcal{E}$ is covered such that there is a node t_i in T for $e \subseteq \chi(t_i)$. ❸ The bags that contain the same vertex $v \in \mathcal{V}$ are connected in T such that for any three nodes, t_i, t_j, and t_k, in T, we have $\chi(t_j) \subseteq \chi(t_i) \cap \chi(t_k)$ if t_j is on the path between t_i and t_k in T.

The width of tree decomposition (T, χ) is $\max_{t_i \in T} |\chi(t_i)| - 1$, and the treewidth of a hypergraph $\mathcal{H}$, denoted as $\mathsf{tw}(\mathcal{H})$, is defined as the minimum width of all tree decompositions of $\mathcal{H}$. A generalized hypertree decomposition (GHD) of a hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ is a triple (T, χ, λ) [20]. Here, (T, χ) is a tree decomposition of $\mathcal{H}$, and λ is a function that assigns each node $t_i \in T$ an edge cover $\lambda(t_i)$ to cover $\chi(t_i)$. The width of a generalized hypertree decomposition (T, χ, λ) is $\max_{t_i \in T}\{\mathsf{weight}(\lambda(t_i))\}$, and the generalized hypertree width of a hypergraph $\mathcal{H}$, denoted as $\mathsf{ghtw}(H)$, is the minimum width of a GHD of $\mathcal{H}$. A fractional hypertree decomposition (FHD) of a hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ is a triple (T, χ, γ) [23]. Here, (T, χ) is a tree decomposition of $\mathcal{H}$, and γ is a function that assigns each node $t_i \in T$ a fractional edge cover $\gamma(t_i)$ to cover $\chi(t_i)$. The width of a fractional hypertree decomposition (T, χ, γ) is $\max_{t_i \in T}\{\mathsf{weight}(\gamma(t_i))\}$, and the fractional hypertree width of a hypergraph $\mathcal{H}$, denoted as $\mathsf{fhtw}(\mathcal{H})$, is the minimum width of a FHD of $\mathcal{H}$.

Example 2.3: We illustrate two hypertree decompositions T_1 and T_2 for the hypergraph $\mathcal{H}$ in Fig. 1. All the tree decompositions share the same pair of (T, χ) following the 3 conditions given for tree decomposition (TD). For simplicity, we use χ_i for $\chi(t_i)$ where t_i is a node in T. For GHD (FHD), we use λ_i (γ_i) to represent its edge cover (fractional edge cover) for its bag χ_i together with its weight assignment by $\lambda(\cdot)$ ($\gamma(\cdot)$). We use Q_{χ_i} to represent the join query within the bag χ_i. Consider the node t_1 in T_1 with $\chi_1 = \chi(t_1) = \{A, C, D, E\}$, we have $\lambda_1 = \{e_1, e_2\}$ with $\lambda(e_1) = \lambda(e_2) = 1$, and we have $\gamma_1 = \{\frac{1}{2}e_1, \frac{1}{2}e_2, \frac{1}{2}e_3\}$ $\gamma(e_1) = \gamma(e_2) = \gamma(e_3) = \frac{1}{2}$. For T_1, the width of TD, GHD and FHD are $\mathsf{tw}(T_1) = 3$, $\mathsf{ghtw}(T_1) = 2$ and $\mathsf{fhtw}(T_1) = \frac{3}{2}$, respectively. For T_2, the widths are $\mathsf{tw}(T_2) = 6$, $\mathsf{ghtw}(T_2) = \mathsf{fhtw}(T_2) = 2$. Regarding $\mathsf{fhtw}(\cdot)$, by T_1 it is to join a triangle of e_1, e_2, and e_3, and possibly result in a smaller intermediate result; by T_2 it is to join e_1 and e_2 and join e_3 and e_4 separately, which possibly result in larger intermediate results. When finally evaluating all nodes as an acyclic query, the complexity is $O(IN + OUT)$ following Yannakakis's algorithm [38], where $IN = |D|^{\frac{3}{2}}$ for T_1 and $IN = |D|^2$ for T_2, for $|D|$ to be the database size.



(a) T_1 **(b)** T_2

Figure 2: Two Hypertree Decompositions for $\mathcal{H}$ in Fig. 1

Problem Statement: It is known that the inequality holds on $\mathsf{tw}(\mathcal{H})$, $\mathsf{ghtw}(\mathcal{H})$, and $\mathsf{fhtw}(\mathcal{H})$: $\mathsf{fhtw}(\mathcal{H}) \leq \mathsf{ghtw}(\mathcal{H}) \leq \mathsf{tw}(\mathcal{H}) + 1$. In this work, we focus on FHD, and study algorithms to compute $\mathsf{fhtw}(\mathcal{H})$.

We discuss the hardness of FHD computing which is given in [14, 15] together with the hardness of GHD, as finding $\mathsf{fhtw}(\mathcal{H})$ by FHD shares some similarities with finding $\mathsf{ghtw}(\mathcal{H})$ by GHD. The key difference between GHD and FHD is the functions used, namely, $\lambda(\cdot)$ and $\gamma(\cdot)$, where the former is a function that assigns a hyperedge in EC a value in $\{0, 1\}$ and the latter is a function that assigns a hyperedge in FEC a value in the range of $[0, 1]$.

For finding GHD [14, 15], as shown below, it is to check if GHD exists for a given width k (e.g., $\mathsf{ghtw}(\mathcal{H}) \leq k$) for a hypergraph $\mathcal{H}$ by searching all hypertrees $\leq k$ of width using separators.

CHECK(GHD, k) **input** a hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ **output** GHD of $\mathcal{H}$ of width $\leq k$ if it exists and answer 'no' otherwise

As discussed in [14, 15], CHECK(GHD, k) is NP-complete even for $k = 2$. Two search algorithms to search hypertree width $\leq k$, GlobalBIP and LocalBIP, are given in [14]. These algorithms are designed based on some properties, namely, Bounded Intersection Property (BIP) and Bounded Multi-Intersection Property (BMIP), which are held for certain classes of conjunctive queries. Additionally, a search algorithm called BalSep is also given in [14] using balanced separators, and its parallel version is given in [21].

As a main result in [14, 15], deciding $\mathsf{fhtw}(\mathcal{H}) \leq 2$ for a hypergraph $\mathcal{H}$ is NP-complete, and CHECK(FHD, k) is intractable even for $k = 2$. Intuitively, finding FHD is even harder as it is difficult to make use of an upper bound like k used in CHECK(GHD, k) for GHD, due to the nature of FHD, which is fractional. In [29], it approximates $\mathsf{fhtw}(\mathcal{H})$ in polynomial time with a cubic error factor, which is too large to be used. Some improvements in the errors in the experimental studies can be found [14].

An *SMT* Approach: We discuss the only approach to compute the exact FHD which is done using an *SMT* solver [13]. The Boolean Satisfiability Problem (*SAT*) is the first problem proven to be NP-complete. It has been extensively studied to develop *SAT* solvers with various techniques. An *SAT* encoding method is to encode an NP-complete problem into an *SAT* problem, and solve it using an *SAT* solver. In [13], it follows the ideas used in [33] which encodes TD at most k width (e.g, CHECK(TD, k)) into an *SAT* problem, and solves it using an *SAT* solver. Furthermore, in [13], it encodes FHD at most k width (e.g, CHECK(FHD, k)) into an *SMT* problem to deal with real numbers. The *SMT* approach has several shortcomings. (1) There is an issue of precision loss in an *SMT* solver caused by its internal representation of the real numbers. Such errors are reported in *Hyperbench* [14], and we confirm that some results by *SMT* differs from the exact answers more than 1e-3. That is, there is no guarantee that the answer of FHD by *SMT* is the optimum, and it is possible that the answer by *SMT* is a suboptimal solution. It is worth mentioning that this error is not caused by the encoding method [13], but due to the limitations of the SMT solver in processing real numbers. (2) It is still based on the approaches taken to check (i.e., CHECK(FHD, k)). The *SMT* solver needs to be called a large number of times to find a real number k as the minimum width, and it is difficult to upper/lower bound such a real number k. (3) The *SMT* encoding treats the original problem as a black-box, which makes the efficiency of the algorithm highly dependent on encoding quality and the *SMT* solver efficiency. In other words, an *SMT* solver will give an answer if it terminates. But, it may not terminate in a reasonable time (e.g., hours).

3 A NEW *DP* ALGORITHM

In this section, we give a dynamic programming (*DP*) algorithm. First, we discuss the elimination order, and show how we deal with partial elimination orders with which partial eliminatied hypergraphs are constructed in tree decomposition. Second, we discuss *DP* algorithm design for FHD which is independent of (partial) elimination orders over a hypergraph. Third, we present the *DP* algorithm in detail.

3.1 Elimination Order

Elimination order has been used in tree decomposition over a graph G [8], and in hypertree decomposition [34] and fractional hypertree decomposition (FHD) [13] over a hypergraph $\mathcal{H}$, because such decompositions can be formulated as a linear order problem to solve. We discuss the elimination order over a hypergraph below.

An elimination order is a linear order of vertices in a hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$, defined as a bijection $\pi : \mathcal{V} \rightarrow \{1, 2, \cdots, |\mathcal{V}|\}$. For simplicity, we use π_i to represent a vertex $v_j \in \mathcal{V}$ as the i-th vertex in the linear order ($\pi(v_j) = i$). With a given linear order, a sequence of $|\mathcal{V}|$ hypergraphs, $(\mathcal{H}_\pi^0, \mathcal{H}_\pi^1, \cdots, \mathcal{H}_\pi^{|\pi|})$, can be constructed for $\mathcal{H}_\pi^i = (\mathcal{V}_\pi^i, \mathcal{E}_\pi^i)$. Here, $\mathcal{H}_\pi^0 = \mathcal{H}$. $\mathcal{H}_\pi^i = \mathsf{elim}(\mathcal{H}_\pi^{i-1}, \pi_i)$ where $\mathsf{elim}(\mathcal{H}, v)$ is an elimination operation that removes a vertex v from the input hypergraph $\mathcal{H}$, adds a hyperedge into the input hypergraph $\mathcal{H}$, and results in a new hypergraph $\mathcal{H}' = (\mathcal{V} \setminus \{v\}, \mathcal{E} \cup \{N_{\mathcal{H}}(v)\})$. The last $\mathcal{H}_\pi^{|\pi|}$ in the sequence is an empty hypergraph. With the sequence of hypergraphs constructed, it can obtain a set of bags, $\chi = \{\chi_1, \chi_2, \cdots, \chi_{|\pi|}\}$ for $\chi_i = N_{\mathcal{H}_\pi^{i-1}}[\pi_i]$.

In [13], it shows that a tree decomposition T of (T, χ) exists as it can be constructed backwards from $\chi_{|\pi|}$ to χ_1, and proves that, for a hypergraph $\mathcal{H}$, there exists a linear order π of $\mathcal{H}$, such that the FHD width built by π equals to $\mathsf{fhtw}(\mathcal{H})$. Therefore, the problem of finding a minimum FHD width is equivalent to finding a minimum width by a linear order.

In order to design a *DP* algorithm for FHD, we define a partial elimination order, and a way to construct a tree decomposition in a forward manner to be used together with the partial elimination order. In other words, in our *DP* algorithm, we need to partially construct a tree and enlarge the tree constructed step-by-step.

Partial elimination order: Let π be an elimination order of $\mathcal{H} = (\mathcal{V}, \mathcal{E})$, a partial elimination order π' is a prefix of π over $\mathcal{V}'$ $(\subseteq \mathcal{V})$. Given a partial elimination order π', in a similar manner, a sequence of $|\mathcal{V}'|$ hypergraphs can be constructed using $\mathsf{elim}(,)$.

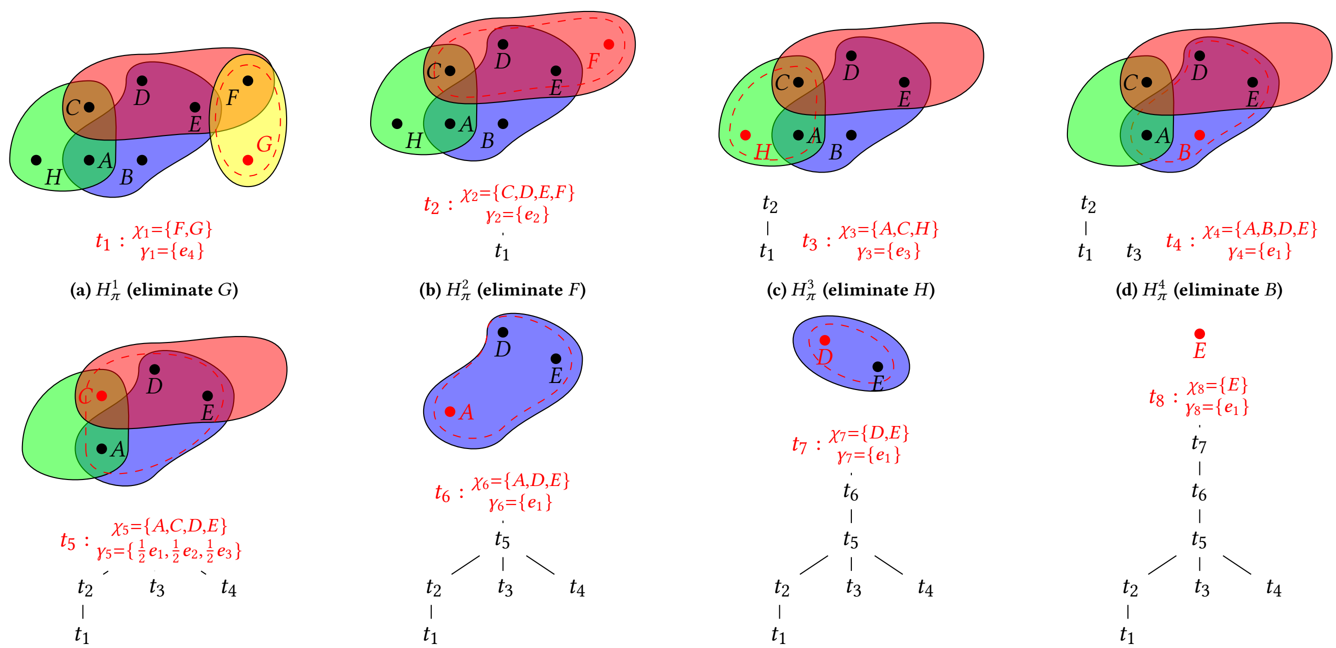
The forward construction of a tree decomposition from χ_1 to $\chi_{|\pi|}$ is given below.

- Create a new node t_i with the bag of χ_i, and mark t_i unlinked;
- For every unlinked node t_j, $j < i$, in T, add a new edge (t_j, t_i), and mark node t_j linked, if $\chi_j \setminus \{\pi_j\} \subseteq \chi_i$.

The correctness of the forward construction can be proved in a similar way used to prove the backward construction in [13], which we omit it here.

In the following, we call $\mathcal{H}_\pi^i$ in a sequence of $|\mathcal{V}|$ hypergraphs, $(\mathcal{H}_\pi^1, \cdots, \mathcal{H}_\pi^{|\pi|})$, a **partial eliminated hypergraph**, which eliminates the vertices from π_1 up to π_i, and we call the current π up to $\pi_1 \pi_2 \cdots \pi_i$ a partial eliminated order if $i < |\mathcal{V}|$.

Example 3.1: Consider an elimination order, $\pi = (G, F, H, B, C, A, D, E)$, for the hypergraph $\mathcal{H}$ in Fig. 1. There is a sequence of hypergraphs, $(\mathcal{H}_\pi^0, \mathcal{H}_\pi^1, \cdots, \mathcal{H}_\pi^{|\pi|})$, constructed following the elimination order. Here, $\mathcal{H}_\pi^0 = \mathcal{H}$. We show how to construct a tree decomposition T with the elimination order for $\mathcal{H}$ (Fig. 1) in a forward fashion from χ_1 to $\chi_{|\pi|}$, in Fig. 3. Note that with π, we have $\pi_1 = G$, $\pi_2 = F$, $\cdots$, $\pi_{|\pi|} = E$. In Fig. 3, $\mathcal{H}_\pi^i$ is shown in a subfigure in which a red vertex is the i-th vertex (π_i) to be removed from $\mathcal{H}_\pi^{i-1}$, a hyperedge to be added into $\mathcal{H}_\pi^i$ is indicated by the dashed line excluding the red vertex if it does not exist, χ_i is the closed neighborhood of π_i, and γ_i is the fractional edge cover (*FEC*) of χ_i. Fig. 3(a) shows $\mathcal{H}_\pi^1$ by eliminating $\pi_1 = G$ from $\mathcal{H}_\pi^0$ (Fig. 1). We have $\chi_1 = \{F, G\}$, a node t_1 is created in T with the bag of χ_1. Its *FEC* of χ_1 is $\gamma = \{e_4\}$. Fig. 3(b) shows $\mathcal{H}_\pi^2$ by eliminating $\pi_2 = F$ from $\mathcal{H}_\pi^1$. We have $\chi_2 = \{C, D, E, F\}$, and a node t_2 is created in T with the bag of χ_2. There is an edge (t_1, t_2) in T because $\chi_1 \setminus \{G\} \subseteq \chi_2$, the *FEC* of χ_2 is $\gamma = \{e_2\}$. Fig. 3(h) shows the entire tree decomposition T constructed. Here, an FHD width can be computed based on γ's using $\gamma_i = \rho^*(\chi_i)$ (refer to Eq. (4) by replacing λ with γ), for each



(a) H_π^1 (eliminate G) (b) H_π^2 (eliminate F) (c) H_π^3 (eliminate H) (d) H_π^4 (eliminate B)

(e) H_π^5 (eliminate C) (f) H_π^6 (eliminate A) (g) H_π^7 (eliminate D) (h) H_π^8 (elimiate E)

Figure 3: An elimination order (G, F, H, B, C, A, D, E) the hypergraph $\mathcal{H}$ in Fig. 1

bag χ_i, for $1 \leq i \leq \pi$, and $\mathsf{fhtw}(\mathcal{H})$ can be identified if it explores all linear orders.

3.2 The Order Independent

We show that the elimination order plays an important role in FHD. With the goal of designing a *DP* algorithm to solve FHD, we explore the order independent property of the elimination order. In brief, with this property, two partial hypergraphs constructed are identical even if they are constructed with different elimination orders. Below, we first discuss how and why a *DP* algorithm works on a simple case of FHD: computing $\mathsf{tw}(G)$ for a graph $G = (V, E)$. Note that G is a special hypergraph in which every hyperedge has 2 vertices. Then we discuss how to extend it to compute fhtw for a hypergraph. The equation used is given as follows

$$TW(S) = \min_{\pi \in \prod(V)} \max_{v \in S} |Q(\pi_{<,v}, v)| \quad (7)$$

with which it computes all subsets $S \subseteq V$ using *DP*. Here, $\prod(V)$ is the set of all linear orders (permutations) for a set of vertices, V. For a given linear order π, $\pi_{<,v}$ is the set of vertices that appear before v in the order π, and $Q(S, v)$ is a non-empty set of vertices, S $(\subset \mathcal{V})$, such that any vertex $w \neq v$ in $\mathcal{V} \setminus S$ and v can be connected by a path over the induced subgraph $G[S \cup \{v, w\}]$. As observed in Eq. (7), the complexity of computing $Q(\pi_{<,v}, v)$ is high as it needs to enumerate all such linear orders. To reduce such complexity, in [8], it is proved that it does not need to compute all linear orders (Eq. (7)). To make it efficient, in [8], it shows that it can compute $Q(\pi_{<,v}, v)$ using a set, i.e., $Q(S \setminus \{v\}, v)$, instead of all linear orders in the set in computing treewidth for G.

$$TW(S) = \min_{v \in S} \max\{TW(S \setminus \{v\}), |Q(S \setminus \{v\}, v)|\}$$

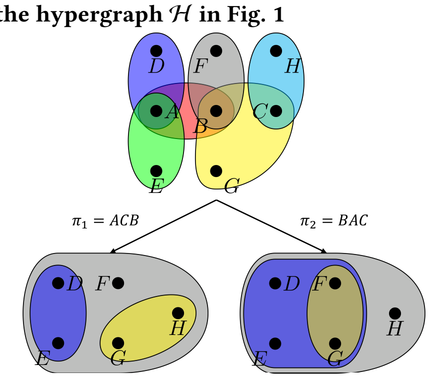


Figure 4: A hypergraph eliminates $\{A, B, C\}$ in two orders

We present the result of this property in a Lemma below, which is not given explicitly in [8].

Lemma 3.1: *Let $G = (V, E)$ be a graph, and $\pi_1, \pi_2 \in \prod(S)$ be any two different linear orders on subset $S \subseteq V$. The simple graphs obtained by the two elimination orders of vertices, π_1 and π_2, are identical such that* $\mathsf{elim}(G, \pi_1) = \mathsf{elim}(G, \pi_2)$.

The *DP* algorithm is given in [8] in $O^*(2^n)$ time complexity and $O^*(2^n)$ space complexity[1] where $n = |V|$, which is too high to deal with a large graph G. In [8], a divide-&-conquer algorithm is given based on the same property in Lemma 3.1.

However, the similar observation by Lemma 3.1 for G does not hold for hypergraphs. In other words, different elimination orders can result in different hypergraphs. We show it using an example.

[1]The O^*-notation suppresses all polynomial factors in O-notation.

Example 3.2: Consider a set of vertices, $S = \{A, B, C\}$, in a hypergraph $\mathcal{H}$ shown in Fig. 4 with two elimination orders, $\pi_1 = ACB$ and $\pi_2 = BAC$. The hypergraph by π_1 adds hyperedges $e_1 = \{B, D, E\}$, $e_2 = \{B, G, H\}$, and $e_3 = \{D, E, F, G, H\}$ to $\mathcal{H}$, whereas the hypergraph by π_2 adds hyperedges $e'_1 = \{A, C, F, G\}$, $e'_2 = \{C, D, E, F, G\}$, and $e'_3 = \{D, E, F, G, H\}$ to $\mathcal{H}$. The two hypergraphs obtained via the two different elimination orders are not identical.

We need a condition to compute FHD efficiently for two hypergraphs obtained via two different orders to be identical.

The width of a bag: We identify the key factor hidden in Lemma 3.1. In doing so, we re-examine the sequence of $|V|$ graphs, $(G^0_\pi, G^1_\pi, \cdots, G^n_\pi)$ for $n = |V|$, with a linear order π. We observe that the width of a bag at a node in (T, χ) for a graph, G, is the cardinality of the bag minus one, where the bag consists of the closed neighbors of the eliminated vertex from G. That is, the width of the bag χ_i at node t_i created for π_i in T for G^i_π is equal to the degree of the eliminated vertex π_i such that

$$\omega(\chi_i) = |N_{G^{i-1}_\pi}[\pi_i]| - 1 = d_{G^{i-1}_\pi}(\pi_i) \tag{8}$$

Here, $N_{G^{i-1}_\pi}[\pi_i]$ is the closed neighborhood of the vertex π_i, and $d_{G^{i-1}_\pi}(\pi_i)$ is the degree of π_i in the graph G^{i-1}_π. The vertex has the same degree which is irrelevant to the elimination order used over the same subset of vertices.

The vertex-width: We define a similar concept over a hypergraph called vertex-width. In a hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$, the vertex-width of a vertex $v \in \mathcal{V} = (\mathcal{V}, \mathcal{E})$ is the weight of minimum *FEC* of its closed neighborhood, i.e.,

$$\omega_{\mathcal{H}}(v) = \rho^*(N_{\mathcal{H}}[v], \mathcal{E}) \tag{9}$$

Consider the sequence of $|\mathcal{V}|$ hypergraphs, $(\mathcal{H}^0_\pi, \mathcal{H}^1_\pi, \cdots, \mathcal{H}^{|\pi|}_\pi)$, in terms of a linear order π. As discussed in the forward construction of a tree decomposition, T, a node t_i is created for π_i in T with the bag of $\chi_i = N_{\mathcal{H}^{i-1}_\pi}[\pi_i]$. With Eq. (6), we have the width of χ_i as $\omega(\chi_i) = \rho^*(N_{\mathcal{H}^{i-1}_\pi}[\pi_i], \mathcal{E})$.

Definition 3.1: Let $\mathcal{H}^i_\pi = (\mathcal{V}^i_\pi, \mathcal{E}^i_\pi)$ be a partial eliminated hypergraph over a partial elimination order π that eliminates vertices in $\{\pi_1, \pi_2, \cdots, \pi_i\}$ from $\mathcal{H}$ in order. The vertex-width of a vertex v in a partial eliminated hypergraph $\mathcal{H}^i_\pi$ is the minimum *FEC* of its closed neighborhood over the hyperedges in $\mathcal{E}$, i.e.,

$$\omega_{\mathcal{H}_\pi}(v) = \rho^*(N_{\mathcal{H}^i_\pi}[v], \mathcal{E}) \tag{10}$$

It is important to note that the vertex-width of a vertex v given in Eq. (6) allows us to compute $\omega(\chi_i) = \rho^*(\chi_i, \mathcal{E})$ for the input hypergraph as well as any partial eliminated hypergraphs using $\mathcal{E}$, even though the hyperedges of a partial eliminated hypergraph are different from the input hypergraph.

Example 3.3: Consider the hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ in Fig. 4 with two different elimination orders, $\pi_1 = ACB$ and $\pi_2 = BAC$. The vertex-width of G in the 2 partial eliminated hypergraphs, $\mathcal{H}_{\pi_1}$ and $\mathcal{H}_{\pi_2}$, is $\omega_{\mathcal{H}_{\pi_1}}(G) = \omega_{\mathcal{H}_{\pi_2}}(G) = \rho^*(\{D, E, F, G, H\}) = 5$.

Like Lemma 3.1 which gives a sufficient condition to compute TD for a graph G, we give a similar but bf weaker sufficient condition to compute FHD for a hypergraph $\mathcal{H}$. We observe that the hyperedges of a partial eliminated hypergraph, $\mathcal{H}^i_\pi$, is different from that in the original hypergraph $\mathcal{H}$; however, for FHD, the vertex-width, $\omega(\cdot)$, in a partial eliminated hypergraph is computed by the original hyperedges in $\mathcal{H}$. With it, we consider if we can derive some condition like the one used in Eq. (8) regarding TD for a graph, which says that the 'degree' of a vertex remains the same for different elimination orders. Here, we consider two vertex-widths, $\omega_{\mathcal{H}_{\pi_1}}(v)$ and $\omega_{\mathcal{H}_{\pi_2}}(v)$, for any two partial elimination orders, π_1 and π_2, over the same subset $\mathcal{V}'$ ($\subseteq \mathcal{V}$). If two vertex-widths are equal (e.g., $\omega_{\mathcal{H}_{\pi_1}}(v) = \omega_{\mathcal{H}_{\pi_2}}(v)$) by Eq. (10), then it implies $\rho^*(N_{\mathcal{H}^i_{\pi_1}}[v], \mathcal{E}) = \rho^*(N_{\mathcal{H}^i_{\pi_2}}[v], \mathcal{E})$. If two vertex-widths are equal because $N_{\mathcal{H}^i_{\pi_1}}[v] = N_{\mathcal{H}^i_{\pi_2}}[v]$, then it implies that, for any vertex v, they have the same neighbors on different eliminated hypergraphs. In other words, their primal graphs must be identical. The weaker sufficient condition we find is that two primal graphs with different linear orders over the same subset need to be identical for their vertex-widths to be the same. This condition is similar to the condition on graph (Lemma (3.1)). We give a lemma below.

Lemma 3.2: *Let $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ be a hypergraph. For any two different partial elimination order over $\mathcal{V}'$ ($\subseteq \mathcal{V}$), π_1 and π_2, its partial eliminated hypergraphs, $\mathcal{H}_{\pi_1}$ and $\mathcal{H}_{\pi_2}$, have the same primal graph, i.e., $G(\mathcal{H}_{\pi_1}) = G(\mathcal{H}_{\pi_2})$.*

Proof Sketch: As the elimination operation on $\mathcal{V}'$ only affects the adjacency of $N_{\mathcal{H}}(\mathcal{V}')$, we focus on the adjacency of $N_{\mathcal{H}}(\mathcal{V}')$, and consider its two cases. The first case is when $\mathcal{V}'$ is connected. Eliminating $\mathcal{V}'$ from $\mathcal{H}$ makes the vertices in $N_{\mathcal{H}}(\mathcal{V}')$ pairwise adjacent. The resulting primal graph is $G(\mathsf{elim}(\mathcal{H}, \mathcal{V}')) = (\mathcal{V} \setminus \mathcal{V}', \mathcal{E} \cup clique(N_{\mathcal{H}}(\mathcal{V}')))$. We prove it as follows. If $\mathcal{V}'$ is a chain $v_1, v_2, \cdots, v_{|\mathcal{V}'|}$ and we eliminate v_i at some step. Let v_l, v_r be the vertices of v_i adjacent to the left and right on the chain, eliminating v_i makes the neighbors of v_i, $N_{\mathcal{H}}(v_i)$, pairwise adjacent. This is due to the fact that v_l and v_r are adjacent to v_i, v_l and v_r become adjacent, and also adjacent to $N_{\mathcal{H}}(v_i)$, such that $N_{\mathcal{H}}(v_l) = N_{\mathcal{H}}(v_l) \cup N_{\mathcal{H}}(v_i), N_{\mathcal{H}}(v_r) = N_{\mathcal{H}}(v_r) \cup N_{\mathcal{H}}(v_i)$. This means the adjacency on v_i spreads to v_l and v_r and the chain cannot be broken. In addition, in the last elimination, $N_{\mathcal{H}}(\mathcal{V}')$ will spread to the last vertex and become pairwise adjacent. For general cases, for any $u, v \in N_{\mathcal{H}}(S)$, there must be a path $v_1, ..., v_k \in \mathcal{V}'$ and $u \in N_{\mathcal{H}}(v_1), v \in N_{\mathcal{H}}(v_k)$. This reduce to the chain case where u and v are adjacent. The second case is when $\mathcal{V}'$ is composed of k connected components $C_1, \cdots, C_k$. Eliminating $\mathcal{V}'$ from $\mathcal{H}$ will make the vertices in each $N_{\mathcal{H}}(C_1), \cdots, N_{\mathcal{H}}(C_k)$ pairwise adjacent. The resuting primal graph is $G(\mathsf{elim}(\mathcal{H}, \mathcal{V}')) = (\mathcal{V} \setminus \mathcal{V}', \mathcal{E} \cup clique(N_{\mathcal{H}}(C_1)) \cup \cdots \cup clique(N_{\mathcal{H}}(C_k)))$. And the elimination operation does not affect the disconnected vertices. We can deal with each C_i independently. □

3.3 A *DP* algorithm for FHD over hypergraphs

Let $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ be a hypergraph, $\prod(\mathcal{V})$ be the set of all linear orders of $\mathcal{H}$, and let χ_{π_i} and γ_{π_i} be the bag and the corresponding minimum *FEC* for π_i. Note that the weight of the minimum *FEC* on the bag χ_{π_i}, weight(γ_{π_i}) is equal to the vertex-width of π_i on the partial eliminated hypergraph $\mathcal{H}_{\pi_{i-1}}$ (Eq. (10)). The $\mathsf{fhtw}(\mathcal{H})$ can be formulated as follows.

$$
\begin{aligned}
\mathsf{fhtw}(\mathcal{H}) &= \min_{\pi\in\prod(\mathcal{V})}\ \max_{i=1,\cdots,|\mathcal{V}|} \mathsf{weight}(\gamma_{\pi_i}) \\
&= \min_{\pi\in\prod(\mathcal{V})}\ \max_{i=1,\cdots,|\mathcal{V}|} w_{\mathcal{H}_{\pi_{i-1}}}(\pi_i)
\end{aligned}
\tag{11}
$$

With Eq. (11), it needs to enumerate all $|\mathcal{V}|!$ linear orders (permutations), whose computational complexity is unacceptable. To speed up the computation of Eq. (11), we compute the width of the tree decomposition for the partial eliminated hypergraph. Let $\mathcal{V}' \subseteq \mathcal{V}$ be a subset of vertices, and π be a partial order of $\prod(\mathcal{V}')$ over $\mathcal{V}'$. We have

$$
\mathsf{width}(\mathcal{H},\pi) = \max_{i=1,\cdots,|\mathcal{V}'|} w_{\mathcal{H}_{\pi_{i-1}}}(\pi_i) \tag{12}
$$

Here, the partial width on $\mathcal{V}'$ can be derived as follows.

$$
\begin{aligned}
\mathsf{fhtw}(\mathcal{H},\mathcal{V}') &= \min_{\pi\in\prod(\mathcal{V}')} \mathsf{width}(\mathcal{H},\pi) \\
&= \min_{\pi\in\prod(\mathcal{V}')}\ \max_{i=1,\cdots,|\mathcal{V}'|} w_{\mathcal{H}_{\pi_{i-1}}}(\pi_i)
\end{aligned}
\tag{13}
$$

Let $H_{\mathsf{elim}(\mathcal{V}')}$ be the partial eliminated hypergraph from which the vertices of $\mathcal{V}'$ have been eliminated, and let $\mathcal{H}_{\mathsf{elim}(\mathcal{V}')+\pi_i}$ be the partial eliminated hypergraph obtained by first eliminating $\mathcal{V}'$ followed by eliminating one more vertex π_i. The width of such a partial eliminated hypergraph is given as follows.

$$
\mathsf{fhtw}(H_{\mathsf{elim}(\mathcal{V}')},\mathcal{V}\setminus\mathcal{V}') = \min_{\pi\in\prod(\mathcal{V}\setminus\mathcal{V}')}\ \max_{i=1,\cdots,|\mathcal{V}\setminus\mathcal{V}'|} w_{\mathcal{H}_{\mathcal{V}'+\pi_{1\sim i-1}}}(\pi_i) \tag{14}
$$

Let the size of a subset of $\mathcal{V}'$ be k for $1 \le k \le |\mathcal{V}|$. The problem of computing $\mathsf{fhtw}(\mathcal{H})$ becomes to compute $\binom{|\mathcal{V}|}{k}$ pairs of subproblems $(\mathsf{fhtw}(\mathcal{H},\mathcal{V}'), \mathsf{fhtw}(\mathcal{H}_{\mathsf{elim}(\mathcal{V}')},\mathcal{V}\setminus\mathcal{V}'))$. And Eq. (11) can be represented as follows.

$$
\mathsf{fhtw}(\mathcal{H}) = \min_{\mathcal{V}'\subseteq\mathcal{V},|\mathcal{V}'|=k} \max\{\mathsf{fhtw}(\mathcal{H},\mathcal{V}'), \mathsf{fhtw}(\mathcal{H}_{\mathsf{elim}(\mathcal{V}')},\mathcal{V}\setminus\mathcal{V}')\} \tag{15}
$$

With Lemma (3.2), the problem of $\mathsf{fhtw}(\mathcal{H})$ and the subproblems $\mathsf{fhtw}(\mathcal{H},\mathcal{V}')$ and $\mathsf{fhtw}(\mathcal{H}_{\mathsf{elim}(\mathcal{V}')},\mathcal{V}\setminus\mathcal{V}')$ are in the same form. A recursive algorithm can be given. We represent the general form of the problem as follows.

$$
\mathsf{fhtw}(\mathcal{H}_{\mathsf{elim}(\mathcal{V}_1)},\mathcal{V}_2),\ \text{with } \mathcal{V}_1,\mathcal{V}_2 \subseteq \mathcal{V} \text{ and } \mathcal{V}_1\cap\mathcal{V}_2 = \emptyset
$$

and solve Eq. (15) as follows.

$$
\begin{aligned}
\mathsf{fhtw}(\mathcal{H}_{\mathsf{elim}(\mathcal{V}_1)},\mathcal{V}_2) = \min_{\mathcal{V}'\subseteq\mathcal{V}_2,|\mathcal{V}'|=k} \max\{&\mathsf{fhtw}(\mathcal{H}_{\mathsf{elim}(\mathcal{V}_1)},\mathcal{V}'), \\
&\mathsf{fhtw}(\mathcal{H}_{\mathsf{elim}(\mathcal{V}_1\cup\mathcal{V}')},\mathcal{V}_2\setminus\mathcal{V}')\}
\end{aligned}
\tag{16}
$$

The time complexity to compute Eq. (16) with different k conforms to the following recursive formula.

$$
T(n) = \binom{n}{k}(T(k) + T(n-k) + p(n)) \tag{17}
$$

Let $k = \lfloor \frac{n}{2} \rfloor$, we can get a $O^*(4^n)$ algorithm with no extra space requirement. When $k = n-1$, we can obtain a $O^*(2^n)$ *DP* algorithm by maintaining $\mathsf{fhtw}(\mathcal{H},\mathcal{V}')$.

Lemma 3.3: *Let $\mathcal{H} = (\mathcal{V},\mathcal{E})$ be a hypergraph and for any non-empty $\mathcal{V}' \subseteq \mathcal{V}$, we have*

$$
\begin{aligned}
\mathsf{fhtw}(\mathcal{H},\mathcal{V}') &= \min_{v\in\mathcal{V}'} \max\{\mathsf{fhtw}(\mathcal{H},\mathcal{V}'\setminus\{v\}), \omega_{\mathcal{H}_{\mathsf{elim}(\mathcal{V}'\setminus\{v\})}}(v)\} \\
&= \min_{v\in\mathcal{V}'} \max\{\mathsf{fhtw}(\mathcal{V},\mathcal{V}'\setminus\{v\}), \rho^*(N_{\mathcal{H}_{\mathsf{elim}(\mathcal{V}'\setminus\{v\})}}[v],\mathcal{E})\}
\end{aligned}
\tag{18}
$$

We show a new *DP* algorithm, DP4FHD, in Algorithm 1. It takes a hypergraph, $\mathcal{H} = (\mathcal{V},\mathcal{E})$ as input, and outputs a tree decomposition by FHD with two procedures, DP4ORDER and BuildFHD. In

Algorithm 1: DP4FHD ($\mathcal{H} = (\mathcal{V},\mathcal{E})$)

1 **Main**
2 $\pi(\mathcal{V}) \leftarrow$ DP4ORDER($\mathcal{H}$);
3 **return** BuildFHD ($\mathcal{H},\pi(\mathcal{V})$);
4 **Procedure** DP4ORDER ($\mathcal{H} = (\mathcal{V},\mathcal{E})$)
5 Initialize $\mathsf{fhtw}(\mathcal{H},\cdot)$ with $+\infty$;
6 $\mathsf{fhtw}(\mathcal{H},\emptyset) \leftarrow -\infty, \pi(\emptyset) \leftarrow ()$;
7 **for** $i = 1$ *to* $|\mathcal{V}|$ **do**
8 **for** *each* $\mathcal{V}' \subseteq \mathcal{V}$ *with* $|\mathcal{V}'| = i$ **do**
9 **for** *each* $v \in \mathcal{V}'$ **do**
10 **if** $\max\{\mathsf{fhtw}(\mathcal{V}'\setminus\{v\}, \rho^*(N_{\mathsf{elim}(\mathcal{H},\mathcal{V}')}[v],\mathcal{E}))\} <$ $\mathsf{fhtw}(\mathcal{H},\mathcal{V}')$ **then**
11 $\mathsf{fhtw}(\mathcal{H},\mathcal{V}') \leftarrow$ $\max\{\mathsf{fhtw}(\mathcal{V}'\setminus\{v\}, \rho^*(N_{\mathsf{elim}(\mathcal{H},\mathcal{V}')}[v],\mathcal{E}))\}$;
12 $\pi(\mathcal{V}') \leftarrow \pi(\mathcal{V}'\setminus\{v\}).concate(v)$;
13 **return** $\pi(\mathcal{V})$;
14 **Procedure** BuildFHD ($\mathcal{H} = (\mathcal{V},\mathcal{E}), \pi = (\pi_1,\pi_2,...,)$)
15 Initialize $linked(\cdot)$ with $False$;
16 $T \leftarrow \emptyset$; $\mathcal{H}^0_\pi \leftarrow \mathcal{H}$;
17 **for** $i = 1$ *to* $|\pi|$ **do**
18 $\mathcal{H}^i_\pi \leftarrow \mathsf{elim}(\mathcal{H}^{i-1}_\pi,\pi_i)$;
19 $\chi_i \leftarrow N_{\mathcal{H}^{i-1}_\pi}[\pi_i]$; $\gamma_i \leftarrow FEC(\chi_i)$; $V(T) \leftarrow V(T)\cup\{\chi_i\}$;
20 **for** $j = 1$ *to* $i-1$ **do**
21 **if** $linked(j) = False$ *and* $\chi_j\setminus\{\pi_j\} \subseteq \chi_i$ **then**
22 $E(T) \leftarrow E(T)\cup\{(\chi_i,\chi_j)\}, linked(j) \leftarrow True$;
23 **return** (T,χ,γ);

DP4ORDER (line 4-13), it determines an elimination order $\pi(\mathcal{V})$. In BuildFHD (line 14-24), it builds FHD following the elimination order. First, we discuss DP4ORDER which is based on Lemma 3.3. We compute $\mathsf{fhtw}(\mathcal{H},\mathcal{V}')$ for each $\mathcal{V}' \subseteq \mathcal{V}$ (line 8) while increasing the size of $\mathcal{V}'$ (line 7). During the *DP* process, $\pi(\mathcal{V}')$ maintains the corresponding partial elimination order of $\mathcal{V}'$ (line 12) and the elimination order returned is the order, $\pi(\mathcal{V})$, for $\mathcal{H}$ (line 13). Next, we discuss BuildFHD. BuildFHD is a forward construction algorithm of building FHD by elimination order as discussed in Section 3.1. Here, $linked(\cdot)$ indicates whether a node is linked or unlinked and is initialized with $False$, T is a tree, and $\mathcal{H}^i_\pi$ is the partial eliminated hypergraph starting from 0 (line 15-17). It forward construct FHD from π_1 to $\pi_{|\pi|}$ (line 17-22). In each iteration (line 17-22), $\mathcal{H}^i_\pi$, χ_i, and γ_i computed before adding a node χ_i into the tree T (line 19). Then, for every $j < i$ (line 20), it checks if the j-th node is unlinked and $\chi_j\setminus\{\pi_j\} \subseteq \chi_i$ (line 21). If it is true, then it adds an edge (χ_i,χ_j) into the tree T and marks the j-th node linked (line 22). Finally, it returns a $FHD = (T,\chi,\gamma)$, which is the output of DP4FHD.

We discuss the time/space complexity of DP4FHD. The computation of minimum *FEC* is a LP problem solvable in polynomial time. The time complexity of DP4FHD is related to the number of variables used in *DP* which is related to $|\mathcal{E}|$ and $|\mathcal{V}|$. For time complexity, assume we need $T(|\mathcal{E}|,|\mathcal{V}'|)$ time to compute the minimum *FEC* of $\mathcal{V}'$. Then DP4FHD needs $O(|\mathcal{V}|2^{|\mathcal{V}|}\cdot T(|\mathcal{E}|,|\mathcal{V}|))$ time, because we do at most $|\mathcal{V}|$ step to compute for each subset $\mathcal{V}'$. For space complexity, DP4FHD only needs to keep the result of the subset $\mathcal{V}'$ with definitely size k for computing the $k+1$ subsets. Thus the space complexity of DP4FHD is $O(\binom{|\mathcal{V}|}{\frac{|\mathcal{V}|}{2}})$.

4 THE BOUNDS ON FHD

We give a *DP* algorithm, DP4FHD, for FHD. However, its complexity is too high to be used for computing FHD in practice. In this section, we discuss several upper/lower bounds to be used to prune in the *DP* algorithm.

4.1 Upper Bounds

Given a hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$, an upper bound for $\mathsf{fhtw}(\mathcal{H})$ (Eq. (11)) or $\mathsf{fhtw}(H_{\mathsf{elim}(\mathcal{V}')})$ (Eq. (14)) can be computed by selecting vertices in a linear order for all vertices in $\mathcal{V}$ iteratively, and such an upper bound computed can be used for a hypergraph or a partial eliminated hypergraph. In brief, it initializes a linear order π as empty, and then it appends a new unselected vertex into π following some strategy one-by-one until all vertices are appended into π. We introduce two such strategies below.

The min-width strategy: This strategy computes the fractional edge cover (*FEC*) of a bag constructed by each vertex, and selects the vertex that is with the min-width (i.e. min *FEC*) to eliminate. The strategy maintains the min *FEC* for all bags constructed by vertices, and recomputes some of vertices when a vertex is selected to be eliminated. A heap is used to maintain the min-width.

The min-fill strategy: This strategy avoids adding hyperedges to the hypergraph, and selects the vertex with the smallest number of non-adjacent neighbors pairs. A heap is used to maintain the smallest fill-in in each iteration.

4.2 Lower Bounds

We give three lower bounds that only depend on the vertex-width in a hypergraph. There are two trivial lower bounds, $\delta(\mathcal{H})$ and $\delta_2(\mathcal{H})$, based on the vertex-width $\omega(v)$ (Eq. (10)) for a hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$.

$$\begin{aligned} \delta(\mathcal{H}) &= \min_{v \in \mathcal{V}} \omega(v) \\ \delta_2(\mathcal{H}) &= \min_{v \in V} \{\omega(v) | v \in \mathcal{V} \land \exists u \in \mathcal{V} : \omega(u) \le \omega(v)\} \end{aligned}$$

Here, $\delta(\mathcal{H})$ and $\delta_2(\mathcal{H})$ are the smallest and the second smallest vertex-width of a vertex in $\mathcal{H}$.

First, we give a lower bound, $\gamma_R(\mathcal{H})$, for a hypergraph $\mathcal{H}$ by extending the Ramachandramurthi's bound given for treewidth over graphs.

$$\gamma_R(\mathcal{H}) = \begin{cases} \rho^*(\mathcal{V}, \mathcal{E})(Eq.\ (5)), & \text{if } \mathcal{H} \text{ is a hyperclique} \\ \min\limits_{u,v \in \mathcal{V}, u \notin N_{\mathcal{H}}[v]} \max\{\omega(u), \omega(v)\}, & otherwise \end{cases}$$

We show that the $\mathsf{fhtw}(\mathcal{H})$ is at least $\gamma_R(\mathcal{H}) \ge \delta_2(\mathcal{H}) \ge \delta(\mathcal{H})$.

Second, we give a lower bound based on hypergraph minor. Here, a hypergraph $\mathcal{H}'$ is a minor of another hypergraph $\mathcal{H}$, denoted $\mathcal{H}' \ll \mathcal{H}$, if $\mathcal{H}'$ can be obtained from $\mathcal{H}$ by a sequence of operations, namely, vertex deletion, edge contraction of two vertices that are contained in a common hyperedge, addition of a hyperedge e such that the set e induces a clique in the primal graph, and deletion of a proper subhyperedge. We explain the edge contraction in brief, where the others are self-explained. Let $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ be a hypergraph and $\mathcal{H}' = (\mathcal{V}', \mathcal{E}')$ be a hypergraph by contracting an edge $\{x, y\}$ in the primal graph $G(\mathcal{H})$ such that $\mathcal{V}' = (\mathcal{V} \setminus \{x, y\}) \cup \{v_{xy}\}$ and $\mathcal{E}' = \{e \in \mathcal{E} | e \cap \{x, y\} = \emptyset\} \cup \{(e \setminus \{x, y\}) \cup \{v_{xy}\} | e \in \mathcal{E}, e \cap \{x, y\} \ne \emptyset\}$, where v_{xy} is a new contracted vertex and every hyperedge containing either x or y is set to contain v_{xy}. We have $\mathsf{fhtw}(\mathcal{H}') \le \mathsf{fhtw}(\mathcal{H})$ if $\mathcal{H}' \ll \mathcal{H}$. The proof of $\mathsf{fhtw}(\mathcal{H}') \le \mathsf{fhtw}(\mathcal{H})$ is similar with the proof of $\mathsf{ghtw}(\mathcal{H}') \le \mathsf{ghtw}(\mathcal{H})$ given in [4]. Based on the width based lower bounds, we can have a new lower bound by looking at the maximum of this bound over all hypergraph minors. ❶ The contraction degeneracy of a hypergraph $\mathcal{H}$, denoted $\overline{\delta}(\mathcal{H})$, is the maximum of $\delta(\mathcal{H}')$ over all hypergraph minors $\mathcal{H}'$ of $\mathcal{H}$. ❷ The δ_2-contraction degeneracy of a hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ with $|\mathcal{V}| \ge 2$, denoted $\overline{\delta}_2(\mathcal{H})$, is the maximum of $\delta_2(\mathcal{H}')$ over all hypergraph minors $\mathcal{H}'$ of $\mathcal{H}$ with at least two vertices. ❸ The γ_R-contraction degeneracy of a hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ with $|\mathcal{V}| \ge 2$, denoted $\overline{\gamma}_R(\mathcal{H})$, is the maximum of $\gamma_R(H)$ over all hypergraph minors $\mathcal{H}'$ of $\mathcal{H}$ with at least two vertices.

For a hypergraph $\mathcal{H}$, $\mathsf{fhtw}(\mathcal{H})$ is at least $\overline{\gamma}_R(\mathcal{H}) \ge \overline{\delta}_2(\mathcal{H}) \ge \overline{\delta}(\mathcal{H})$. This can be proved based on (a) $\gamma_R(\mathcal{H}) \ge \delta_2(\mathcal{H}) \ge \delta(\mathcal{H})$, and (b) $\mathsf{fhtw}(\mathcal{H}') \le \mathsf{fhtw}(\mathcal{H})$ if $\mathcal{H}' \ll \mathcal{H}$. Note that this lower bound is held on hypergraphs, but not held on partial eliminated hypergraphs. That is, $\mathsf{fhtw}(\mathcal{H}'_{elim(\mathcal{V}')}, \mathcal{V} \setminus \mathcal{V}') \le \mathsf{fhtw}(\mathcal{H}_{elim(\mathcal{V}')}, \mathcal{V} \setminus \mathcal{V}')$ is not always held, if $\mathcal{H}'_{elim(\mathcal{V}')}$ is a hypergraph minor of a partial eliminated hypergraph $\mathcal{H}_{elim(\mathcal{V}')}$. But we can use it to compute the lower bound of $\mathsf{fhtw}(\mathcal{H}_{elim(\mathcal{V}')})$ if we treat a partial eliminated hypergraph as a hypergraph, and use the inequality $\mathsf{fhtw}(\mathcal{H}_{elim(\mathcal{V}')}) \le \mathsf{fhtw}(\mathcal{H}_{elim(\mathcal{V}')}, \mathcal{V} \setminus \mathcal{V}')$. It can be used as a lower bound of $\mathsf{fhtw}(\mathcal{H}_{elim(\mathcal{V}')}, \mathcal{V} \setminus \mathcal{V}')$.

We give a simple heuristic algorithm, named MMDH, in Algorithm 2 to compute the contraction degeneracy, which is based on the same algorithm for graphs. In Algorithm 2, it needs to select a neighbor w to which the minimum width vertex v is contracted based on heuristic strategy, including the min-width strategy, the max-width strategy, the least common vertices strategy and the least common edges strategy.

Algorithm 2: MMDH ($\mathcal{H} = (\mathcal{V}, \mathcal{E})$)

```
ω_max ← 0;
for i = 1 to |V| − 1 do
    v ← arg min_{u∈V} ω(u), ω_max ← max{ω_max, ω(v)};
    u ← select(H, v);
    H ← contract(H, {v, u});
return ω_max;
```

5 A BRANCH-&-BOUND ALGORITHM

We present a *DP* algorithm named DP4FHD in Algorithm 1, and discuss several upper/lower bounds in Section 4. In this section, we give a branch-&-bound algorithm, named BB4FHD, in Algorithm 3, which is based on the DP4FHD algorithm and the upper/lower bounds discussed. The upper/lower bounds used can be arbitrary algorithm in Section 4 and we will discuss them in Section 8.

In Algorithm 3, we compute fhtw for partial eliminated hypergraphs by pushing the vertex-width layer by layer, i.e., using the result on $\mathcal{V}'$ to update all $\mathcal{V}'$ extendable states with size of $|\mathcal{V}'|+1$. Note that, with branch-&-bound, we can prune those states that will not participate in the following processing. In BB4FHD, we use $|\mathcal{V}| + 1$ maps, $FHTW_0[\cdot], FHTW_1[\cdot], \cdots, FHTW_{|V|}[\cdot]$ to maintain the state in each layer with $\Pi[\cdot]$ to store the partial linear orders.

Algorithm 3: BB4FHD ($\mathcal{H} = (\mathcal{V}, \mathcal{E})$)

1 **Main**
2    $\pi(\mathcal{V}) \leftarrow$ BB4ORDER($\mathcal{H}$);
3   **return** BuildFHD ($\mathcal{H}, \pi(\mathcal{V})$) `/* Same as DP4FHD */`
4 **Procedure** BB4ORDER ($\mathcal{H} = (\mathcal{V}, \mathcal{E})$)
5   Initialize $FHTW_0[\cdot], ..., FHTW_{|V|}[\cdot], \Pi[\cdot]$ with $\emptyset$;
6   Initialize $\mathsf{fhtw}_{min}, \pi_{min}$ with upper($\mathcal{H}$);
7    $FHTW_0[\emptyset] \leftarrow$ lower($\mathcal{H}$);
8   **for** $i = 0$ *to* $|\mathcal{V}| - 1$ **do**
9   **for** *each state* $\mathcal{V}' \in FHTW_i$ **do**
10        $\pi \leftarrow \prod[\mathcal{V}']$;
11   **for** *each vertex* $v \in \mathcal{V} \setminus \mathcal{V}'$ **do**
12          $\mathcal{S} \leftarrow \mathcal{V}' \cup \{v\}, \pi' \leftarrow \pi.concate(v)$;
13          $\omega_v \leftarrow \omega_{\mathcal{H}_{\pi'}}(v), \omega_p \leftarrow \max\{FHTW_i[\mathcal{V}'], \omega_v\}$;
14          $low \leftarrow$ lower($H_{\pi'}$);
15   **if** $\max\{low, \omega_p\} \geq \mathsf{fhtw}_{min}$ **then**
16   continue;
17   **if** $\mathcal{S} \notin FHTW_{i+1}$ *or* $FHTW_{i+1}[\mathcal{S}] > \omega_p$ **then**
18            $FHTW_{i+1}[\mathcal{S}] \leftarrow \omega_p, \prod[\mathcal{S}] \leftarrow \pi'$;
19          $up, \pi_{up} \leftarrow$ upper($\mathcal{H}_{\pi'}$);
20   **if** $\max\{\omega_p, up\} < \mathsf{fhtw}_{min}$ **then**
21            $\mathsf{fhtw}_{min} \leftarrow \max\{\omega_p, up\}$;
22            $\pi_{min} \leftarrow \pi'.concate(\pi_{up})$;
23   **return** π_{min};

They are initialized with $\emptyset$ (line 5). The global optimal solution fhtw_{min} and its corresponding elimination order π_{min} is initialized with an upper bound heuristic (line 6). The *DP* process starts with the $\emptyset$ state and sets the lower bound of $\mathcal{H}$ as the initial width (line 7). Then, the algorithm computes fhtw for a partial eliminated hypergraph layer by layer, and enumerates the states in the layer to transfer (lines 8-22). For each state $\mathcal{V}'$, it enumerates all vertex $v \in \mathcal{V} \setminus \mathcal{V}'$ to extend state $\mathcal{S} = \mathcal{V}' \cup \{v\}$, and enlarges a partial linear order π' (line 12). Then it computes the vertex-width of the partial eliminated hypergraph by π' (Eq. (10)) in line 13. We prune it if its value is greater than or equal to the current fhtw_{min} (lines 15-16). We compute it for $\mathcal{S}$ and the current global optimal solution by the current solution (lines 17-22).

It is important to mention that the BB4FHD algorithm (Algorithm 3) is an **anytime** algorithm, and can terminate at any time, as it always maintains the current global optimal solution $\mathsf{fhtw}_{min}, \pi_{min}$.

Problem reduction techniques: The time complexity of Algorithm 3 is exponentially related to the problem size. To reduce the problem size without losing the optimality is profitable. We simply introduce 6 reduction rules following the same ideas used in [13], which are given for preprocessing only as the approach taken in [13] is an *SMT* encoding method.

Given a hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$, the 6 rules are given to reduce $\mathcal{H}$ below. **Rule-1** A hyperedge can be removed if it is a subset of another hyperedge. **Rule-2** A vertex only belongs to one hyperedge can be removed. **Rule-3** Two vertices, u and v, belong to the same class if they have the same closed neighborhood, i.e., $N_{\mathcal{H}}[v] = N_{\mathcal{H}}[u]$. We only need to keep one vertex in a class. **Rule-4** A vertex $v \in \mathcal{V}$ is a simplicial vertex if the neighborhood of v forms a clique in the primal graph of $G(\mathcal{H})$, which can be removed. **Rule-5** The connectivity and biconnectivity on hypergraph is the same to its primal graph. For different biconnected components, we can solve the elimination order problem separately. **Rule-6** For any hyperclique $\mathcal{H}_C$ of $\mathcal{H}$, there must be a bag that contains $\mathcal{H}_C$ in the tree decomposition of $\mathcal{H}$. Thus, we can select a hyperclique $\mathcal{H}_C = \{v_1, ..., v_k\}$ and place it at the last of the order, i.e, $\pi = (\cdots, v_1, \cdots, v_k)$.

6 THE FHD WIDTH COMPUTATION

The computation of FHD width (e.g., $\rho^*(\mathcal{V}, \mathcal{E})$ (Eq. 5)) is a fundamental operation, and is frequently used in our branch-&-bound algorithm. We explain it below. In our algorithm, we need to deal with the input hypergraph, $\mathcal{H} = (\mathcal{V}, \mathcal{E})$, any partial eliminated hypergraph $\mathcal{H}_\pi = (\mathcal{V}_\pi, \mathcal{E}_\pi)$, and other $\mathcal{H}' = (\mathcal{V}', \mathcal{E}')$, for example hypergraph minors, sub-hypergraph, etc. The computation of vertex-width (Eq. (10)) for a vertex v is to be one of $\rho^*(N_{\mathcal{H}}(v), \mathcal{E})$, $\rho^*(N_{\mathcal{H}_\pi}(v), \mathcal{E})$, $\rho^*(N'_{\mathcal{H}}(v), \mathcal{E}')$, and is needed in the preprocessing stage, in *DP* (or branch-&-bound) processing, in computing upper/lower bounds. The cost of computing FHD width is high due to the fact that it is a linear programming (LP) problem.

We find that most of FHD width computation is $\rho^*(\cdot, \mathcal{E})$. In other words, it uses the same set of $\mathcal{E}$ to compute $\rho^*(\mathcal{S}, \mathcal{E})$ for different $\mathcal{S}$ from time to time frequently. In the view of linear programming, we convert a hyperedge, e_i, and set $\mathcal{S}$ to 0-1 vector form (e.g, $\mathcal{V} = \{1, 2, 3, 4, 5\}, e_i = \{1, 2, 4\} \rightarrow (1, 1, 0, 1, 0)^T$), then $\rho^*(\cdot, \mathcal{E})$ can be written as this type of linear programming:

$$\begin{aligned} & minimize \quad ||\gamma||_1 \\ & \text{subject to} \quad \mathcal{E}\gamma = \begin{bmatrix} e_1 & \cdots & e_{|\mathcal{V}|} \end{bmatrix} \gamma \geq \mathcal{S}, \quad \mathbf{0} \leq \gamma \leq \mathbf{1} \end{aligned} \tag{19}$$

Here, $\mathcal{E}$ is in the form of matrix, $\mathcal{S}$ is in the form of vector for the input set $\mathcal{S}$, and γ is a vector where every element in γ is between 0 and 1. With Eq. (19), we can build an index to accelerate this process for different $\mathcal{S}$ with preprocessing. In doing so, we construct a lattice for various of sets of vertices and the same set of hyperedges. We give a lemma below.

Lemma 6.1: *Let $\mathcal{V}$ be the universe set, $\mathcal{E}$ be a collection of subsets of $\mathcal{V}$, and $\rho^*(\mathcal{S}, \mathcal{E})$ be the minimum fractional edge-over (FEC) of $\mathcal{S}$ regarding $\mathcal{E}$. Then, for any other $\mathcal{S}' \subseteq \mathcal{V}$, we have ① $\rho^*(\mathcal{S}, \mathcal{E}) \leq \rho^*(\mathcal{S}', \mathcal{E})$ if $\mathcal{S} \subseteq \mathcal{S}'$, and ② $\rho^*(\mathcal{S}, \mathcal{E}) \geq \rho^*(\mathcal{S}', \mathcal{E})$ if $\mathcal{S} \supseteq \mathcal{S}'$.*

The lemma obviously holds and it guides us to make use of the bounds computed for a new *FEC* width computation request. Let $\mathbb{S} = \{\mathcal{S}_1, \mathcal{S}_2, \cdots\}$ be the collection of sets we have already computed and their width $\rho^*(\mathcal{S}_i, \mathcal{E})$ have been stored. Then for a new *FEC* width computation $\rho^*(\mathcal{S}, \mathcal{E})$, we search the lower and upper bound as follows: ① $upper(\mathcal{S}) = \min_{\mathcal{S}' \in \mathbb{S} \wedge \mathcal{S} \subseteq \mathcal{S}'} \rho^*(\mathcal{S}', \mathcal{E})$, and ② $lower(\mathcal{S}) = \max_{\mathcal{S}'' \in \mathbb{S} \wedge \mathcal{S} \supseteq \mathcal{S}''} \rho^*(\mathcal{S}'', \mathcal{E})$.

The queries of lower/upper bounds are known as a n-dimensional partial order maintenance problem. It has been studied when $\mathbb{S}$ is static or when the queries/updates of $\mathbb{S}$ are already known that come offline. In the literature, we have not yet found any efficient methods to address the dynamic update with online queries. In this work, we use a k-d tree to support the lower/upper bound queries.

The lower/upper bounds of an FHD width may speed up the LP computation. We substitute them as known conditions into the formula (e.g, add an addition condition $\mathrm{lower}(\mathcal{S}) \leq ||\gamma||_1 \leq \mathrm{upper}(\mathcal{S})$ into Eq. (19)). On the other hand, we can skip some width computation in Algorithm 3 by combining the bounds with the current global optimal solution fhtw_{min} and $FHTW_i[\mathcal{S}]$.

7 RELATED WORK

We discuss hypergraph decompositions: TD, GHD, and FHD together with HD. Note that the bounded TD is too loose to be used. For computing GHD, a basic search algorithm based on properties BIP and BMIP is proposed in [15], which is to deal with some restricted class of conjunctive queries, and several algorithms with some heuristic to speed up computing such properties are given in [14]. In addition, a parallel algorithm is given in [21]. The *SMT* approach discussed is the only practical algorithm to compute FHD [13] for all conjunctive queries.

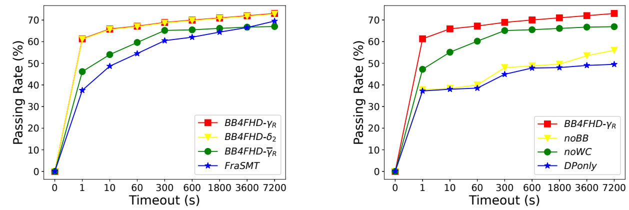
As both GHD and FHD are known NP-complete to check (e.g., CHECK(,k)), HD is proposed in [20] under a special condition following GHD. It is known that HD width, denoted by htw($\mathcal{H}$), can be checked (e.g., CHECK(HD, k)) in polynomial time instead of NP-complete for GHD and FHD. As it is one in polynomial time, htw($\mathcal{H}$) is greater than ghtw($\mathcal{H}$), which is greater than fhtw($\mathcal{H}$). Hypertree decomposition (HD) has been extensively studied as an effective alternative instead of GHD and FHD. To compute HD, a backtracking-based search algorithm, named det-k-decomp, by exploring separators to check is implemented and reported in [22], and a parallel search algorithm with searching balanced separator, named log-k-decomp, can be found in [19]. All the existing algorithms for hypertree decompositions (HD, GHD, FHD) are based on CHECK(,) while exploring all separators.

8 EXPERIMENTAL STUDIES

We have conducted extensive experimental studies using *Hyperbench* benchmark [14]. *Hyperbench* contains 3,648 hypergraphs (called instances) including *CQ*s and *CSP*s collected from various sources. We use the full hypergraphs used in [19, 21]. As reported in *Hyperbench*, the statistics on the known min-widths, w_{min}, are as follows. There are 710, 596, 310, 385, 450, 496, and 702 instances which are with the known min-width, 1, 2, 3, 4, 5, 6, and > 6, respectively. Since the existing fhtw results are not complete, we use the smaller of the known ghtw and htw [1] as min-width.

We study our anytime branch-&-bound algorithm, BB4FHD, to compute fhtw for a hypergraph, which can find an approximate fhtw in a limited time, and can find the exact fhtw if time is allowed. We discuss the upper/lower bound used in BB4FHD testing. First, for the upper bound, we use the smallest among all the strategies discussed in Section 4, because all the strategies can be computed in linear time. Second, for the lower bound, we compare the width-based bounds and the minor-based bounds. For the width-based bounds, we test δ_2 and γ_R. Here, δ_2 can be computed in linear time, whereas γ_R needs to be computed with additional sorting. For the minor-based bounds, we report the min-width strategy with $\overline{\gamma}_R$, as all of the minor-based strategies are similar. Note that $\overline{\gamma}_R$ needs to compute using $\rho^*(,)$ (Eq. 6) which is costly. Below, we use BB4FHD-X where X is one of the three, namely, δ_2, γ_R, and $\overline{\gamma}_R$. We implemented our algorithms in C++ and complied it by GCC 8.5.0 with -O2 flag.

We compare BB4FHD with one exact algorithm and three approximate algorithm: ❶ the exact algorithm FraSMT [13] (which is the state-of-the-art *SMT* encoding method), ❷ a theoretical approximation algorithm ApproxFHD [29], ❸ an approximate algorithm ImproveHD [14], and ❹ an approximate algorithm FracImprove [14].



(a) Exact Testing **(b) Ablation Study**

Figure 5: Pass rates with different timeouts

Table 1: FraSMT **vs** BB4FHD-γ_R**: # of solved, and time (sec.)**

$\mathcal{H} = (\mathcal{V}, \mathcal{E})$			**Fractional Hypertree Decomposition** FHD							
Origin of Instances	**Size of Instances**	**Instances in Group**	FraSMT				BB4FHD-γ_R			
			#solved	**avg**	**max**	**stdev**	**#solved**	**avg**	**max**	**stdev**
Real World	$75 < \lvert\mathcal{E}\rvert \le 100$	405	97	842.0	7,153.3	1,579.0	47	2,325.6	7,187.4	2,614.0
	$50 < \lvert\mathcal{E}\rvert \le 75$	514	356	587.2	7,052.0	1,691.0	315	7.7	2,036.0	115.7
	$10 < \lvert\mathcal{E}\rvert \le 50$	369	234	120.0	943.0	619.8	232	24.6	4,968.0	327.0
	$\lvert\mathcal{E}\rvert \le 10$	915	913	0.2	5.7	0.3	915	0.0	0.0	0.0
Synthetic	$\lvert\mathcal{E}\rvert > 100$	66	11	1,067.7	5,802.8	1,697.7	9	708.9	5,262.1	1,726.1
	$75 < \lvert\mathcal{E}\rvert \le 100$	422	345	1,266.7	7,120.8	1,927.1	274	581.0	6,815.2	1,427.0
	$50 < \lvert\mathcal{E}\rvert \le 75$	215	202	320.5	5,858.0	877.5	215	0.1	0.3	0.1
	$10 < \lvert\mathcal{E}\rvert \le 50$	647	285	413.6	7143.1	1171.0	565	157.3	6,588.8	608.2
	$\lvert\mathcal{E}\rvert \le 10$	95	95	8.5	329.1	44.3	95	0.3	25.7	2.6
Total		3,648	2,538	374.8	7153.3	1203.2	2,667	140.0	7,187.4	736.5
CQs		1,113	1,113	0.3	43.6	1.7	1,113	0.0	0.5	0.0

We test FraSMT using the optimal parameters given in [13] with the *SMT* solver Z3. ApproxFHD needs to guess a lower bound w_l, and outputs a FHD width in $O(w_l^3)$ if it is found. If no width is found, it concludes that fhtw > w_l. Here, we provide the best lower bound of each hypergraph for ApproxFHD. ImproveHD and FracImprove are a "fractional improvement" algorithm based on HD, and need to guess an upper bound w_u to check. We provide the smallest upper bound w_{min} we have in *Hyperbench*. The output of FracImprove is the best with a minimum fractional improvement at 0.1 and 0.5.

All experiments are conducted on a machine with an AMD 2.7Ghz CPU (96 cores) and 512GB main memory running Linux.

HyperBench Testing: Exact. We have conducted a complete *Hyperbench* test and record the pass rate of the instances using timeouts of 1, 10, 60, 300, 600, 1,800, 3,600, and 7,200 (sec). Here, the pass rate is the proportion of the number of instances in which the optimal fhtw can be calculated among all instances. We report the results by BB4FHD-δ_2, BB4FHD-γ_R, BB4FHD-$\overline{\gamma}_R$, and FraSMT in Fig. 5a. As illustrated in Fig. 5a, BB4FHD-δ_2 and BB4FHD-γ_R outperform BB4FHD-$\overline{\gamma}_R$ and FraSMT. The pass rates of BB4FHD-γ_R is slightly better than BB4FHD-δ_2.

Our branch-&-bound algorithms can finish at anytime or even when the timeout is small. When the timeout is 1s and 10s, BB4FHD-γ_R can pass approximately 20% more instances in *Hyperbench* than FraSMT. Below, we report BB4FHD-γ_R as it outperforms the other BB4FHD variants.

We compare FraSMT and BB4FHD-γ_R with the same timeout of 7,200s, and summarize the results in Table 1. Here, we follow the experiments reported in [19] to divide the instances in *Hyperbench* by size and origin. The instances are divided into two categories: real world applications and synthetic generated. For each category, the number of instances, $|\mathcal{E}|$, is further divided. We report the numbers of solved instances passed in each range as #solved and the running times (avg, max, stdev).

Overall, BB4FHD-γ_R passes more instances (2,667 out of 3,648, 73.1%) than FraSMT (2,538 out of 3,648, 69.5%) in total and the

Table 2: The average width reduction

w_{min}	BB4FHD-γ_R		FraSMT		ApproxFHD		ImproveHD		FracImprove	
	Total	**avg.**	**Total**	**avg.**	**Total**	**avg.**	**Total**	**avg.**	**Total**	**avg.**
2	**260**	0.48	257	0.48	231	0.48	186	0.48	240	0.48
3	**162**	0.62	158	0.62	134	0.62	141	0.62	153	0.62
4	**103**	0.67	88	0.67	74	0.54	73	0.66	95	0.63
5	**245**	0.60	233	0.54	31	0.98	42	0.93	220	0.52
6	**321**	0.75	179	0.60	28	0.59	155	0.50	276	0.53
> 6	**501**	1.55	30	0.73	193	1.05	286	1.31	426	1.08

Table 3: The width reduction of BB4FHD-γ_R (1 sec)

Width Reduction	ApproxFHD		ImproveHD		FracImprove	
	Total	**avg.**	**Total**	**avg.**	**Total**	**avg.**
≥ 1	1898	13.55	53	2.11	17	3.42
$[0.5, 1)$	102	0.61	487	0.59	69	0.59
$(0, 0.5)$	128	0.22	336	0.22	423	0.19
total(> 0)	2128	12.13	876	0.64	509	0.35
$= 0$	1287	0	1855	0	540	0
< 0	0	0	379	-1.06	74	-0.08
timeout	221	*	526	*	2513	*

average time by BB4FHD-γ_R is 2.7x faster than FraSMT. BB4FHD-γ_R outperforms FraSMT significantly in solving small and medium-sized instances ($|\mathcal{E}| \leq 50$). But due to its high time complexity, BB4FHD-γ_R performs worse on large-sized instances compared to FraSMT. The *SMT* encoding method can solve some large-sized instances with application-derived characteristics We will study it future to accelerate *DP* algorithms based on such characteristics.

We also report the results for *CQ*s (in total 1,113 instances). These instances are relatively simple in *Hyperbench*. Among 1,113 instances, the largest fhtw does not exceed 3, and most belong to the acyclic hypergraph. Both FraSMT and BB4FHD-γ_R solve all instances. BB4FHD-γ_R is much more efficient than FraSMT. BB4FHD-γ_R solves all instances in 0.5 seconds, whereas FraSMT takes over 40 seconds on some, and the running time varies.

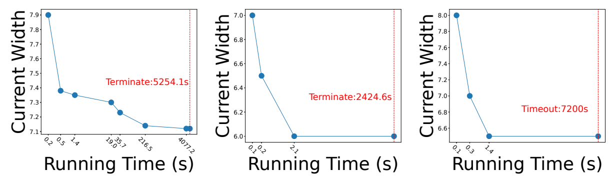
HyperBench Testing: Width Reduction. We compare BB4FHD-γ_R with all baseline algorithms in a timeout of 2 hours, and report the fractional width reduction based on known min width w_{min} in Table 2. Here, "Total" is the total number of instances corresponding to the w_{min} in *Hyperbench*, and "avg." is the width reduction on average, where the larger is better. BB4FHD-γ_R can reduce width in all cases on the largest number of instances, and the average width reduction is also the largest in almost all cases. BB4FHD-γ_R has advantages when dealing with instances with large width ($w_{min} > 5$), while FraSMT will become less reliable when the problem is complex.

ApproxFHD can quickly output results on all instances with an average time 0.42s. This is because of its very loose approximation bounds, and it simply chooses not to decompose on more than 90% of instances. ImproveHD and FracImprove need to compute an HD with a given width first, cannot be efficient on average time 405.79s and 5061.06s, and will not output results on all instances (94.87% and 37.75% of instances have output), respectively. Here, HD is computed with the best known width known in *Hyperbench*. In practice, there is no such width to assist HD construction.

HyperBench Testing: Approximate. We compare the approximate performance of BB4FHD-γ_R with the three approximate algorithms, ApproxFHD, ImproveHD, and FracImprove. We consider query optimization scenarios, to set the timeout of all algorithms to

Table 4: A case study with 3 hypergraphs

$\mathcal{H} = (\mathcal{V}, \mathcal{E})$	$\lvert\mathcal{V}\rvert$	$\lvert\mathcal{E}\rvert$	**Degree**	**Types**
rand_q0239	77	45	11	CQ Random
Pi-40-10-07948-40-76	40	98	11	CSP Application
Pi-40-10-07948-40-13	40	98	9	CSP Application



(a) rand_q0239 **(b) Pi-40-10-07948-40-76** **(c) Pi-40-10-07948-40-13**

Figure 6: Anytime width update

1 second and compare the width output for each instance. BB4FHD-γ_R achieves best approximate results on nearly all instances. Specifically, BB4FHD-γ_R can achieve better (possibly the same) approximate results on 3,219 instances, of which 1,164 results are strictly better than those of the other comparison algorithms. And the two numbers are 1,267 and 0, 2,234 and 352, 618 and 44 in ApproxFHD, ImproveHD, and FracImprove. BB4FHD-γ_R provides results for all instances, while ApproxFHD, ImproveHD, and FracImprove timeout on 221, 526, 2,513 instances, respectively. This demonstrates the great potential of our approach in practical scenarios.

In Table 3, from a different view, we show how good BB4FHD-γ_R can be. Here, we take the width by ApproxFHD, ImproveHD, and FracImprove as basis, respectively. We report the width reduction of BB4FHD-γ_R on average ("avg."). We group the width reduction in "≥ 1", "$[0.5, 1)$", "$(0, 0.5)$", "$= 0$", and "< 0" groups, and we report the number of instances and the average width reduction in each group. Here, the group of "$= 0$" is the group that all have the same width computed, and the average reduction ("avg.") is 0. BB4FHD-γ_R outperforms one if the corresponding "avg." value is greater than 0. BB4FHD-γ_R is always and far better than ApproxFHD. FracImprove will timeout on most instances. Compared with ImproveHD, BB4FHD-γ_R can reduce the width on about 900 instances, excluding the 526 timeout instances.

The Anytime Algorithm. We study our anytime algorithm, BB4FHD, by case studies. We select three representative hypergraphs, namely, (Case-1) rand_q0239, (Case-2) Pi-40-10-07948-40-76, and (Case-3) Pi-40-10-07948-40-13 from *Hyperbench* with the consieration of the width and the distribution of the fhtw updated timeouts. The details of the three hypergraphs are shown in Table 4. For Case-1, the hypergraph is a randomly selected conjunctive query, and its fhtw is very fractional. For Case-2 and Case-3, the 2 hypergraphs are taken from *CSP* applications, and their fhtw is integer-like.

We show the current optimal fhtw changes with the running time on the three hypergraphs in Fig. 6. For Case-1, as shown in Fig 6a, the current optimal fhtw is updated in 7.90, 7.38, 7.35, 7.30, 7.23 7.14, and 7.12. The fhtw computed for this hypergraph is very fractional and the current optimal fhtw has been updated many times with minor differences in computing. Our BB4FHD can prune many in such fractional behavior. This is because such fractional behavior makes BB4FHD more likely to use the current fhtw to distinguish the different solutions and therefore achieve the high

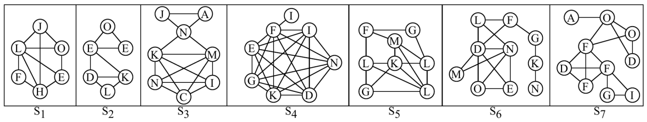


Figure 7: The subgraph queries

Table 5: Query evaluation time (ms) on EmptyHeaded

	EmptyHeaded native			EmptyHeaded + new FHD		
	FHD search	**Execution**	**Total**	**FHD search**	**Execution**	**Total**
S1	1,087.8	4,033.7	5,121.5	1.2	3,960.9	3,962.1
S2	776.9	8.8	785.7	1.2	8.3	9.5
S3	15,661.2	2,513.5	18,174.7	4.3	2,636.3	2,640.6
S4	>2h	NA	NA	2.6	4,169.7	4,172.3
S5	7,836.3	3,880.1	11,716.3	4.5	3,229.1	3,233.6
S6	8,778.5	11,538.4	20,316.9	2.4	11,357.7	11,360.1
S7	13,670.6	15,368.3	29,038.9	8.0	15,366.4	15,374.4
Q1	523.1	1,492.7	2,015.7	0.7	1,423.0	1,423.8
Q2	582.0	363.4	945.4	0.6	324.3	324.9
Q3	576.7	324.3	901.0	0.5	273.5	274.0
Q4	1,206.7	1,448.9	2,655.5	0.8	1,392.2	1,393.0
Q5	1,049.7	362.9	1,412.6	1.9	384.9	386.8

performance by pruning. Such very fractional cases mainly occur in very complex or randomly generated instances, and give BB4FHD the opportunity to solve larger-scale instances. For Case-2, as shown in Fig. 6b, it has only been updated 3 times, 7, 6.5, and 6. For Case-3, as shown in Fig. 6c, it has only been updated 3 times, 8, 7, and 6.5. The fhtw computed in the two hypergraphs is integer-like whose decimal precision is at most 0.5. This means the minimum *FEC* induced from the fhtw computation is rather simple. And this kind of integer-like fhtw is very detrimental to BB4FHD, due to the fact that the fhtw of the candidates are almost the same. We cannot prune some solutions by the width in BB4FHD. On the other hand, as shown in Fig. 6b, the anytime result is already the optimal at very beginning, even though it takes a long time to terminate. This indicates that the optimal integer-like fhtw can be easily obtained.

Ablation Study. In order to study the impacts of (a) Branch-&-Bound and (b) width computation optimization methods on top of the basic *DP* algorithm, namely BB4FHD-γ_R, we test three other cases. One is *DP* only, denoted as DPonly, one is to disable width computation optimization methods in BB4FHD-γ_R, denoted as noWC, and one is to disable Branch-&-Bound in BB4FHD-γ_R, denoted as noBB. We report the pass rates of the 4 algorithms on *Hyperbench* as shown in Fig. 5b. Both Branch-&-Bound and width computation optimization methods significantly affect the performance, while Branch-&-Bound has a greater impact. Specifically, under the 7,200s time limit set, noBB solves 2,042 instances with an average time of 422.69s, while BB4FHD-γ_R uses 0.07s; noWC solves 2,440 instances with an average time of 59.85s, while BB4FHD-γ_R uses 0.57s; DPonly solves 1,807 instances with an average time of 174.49s while BB4FHD-γ_R uses 0.01s.

Query Evaluation on Real Database Systems. We conduct query evaluation to test graph pattern queries and *SQL* queries on EmptyHeaded, PostgreSQL, and DuckDB. EmptyHeaded is a query processing engine that supports both FHD and WCOJ [31]. Its built-in query optimizer enumerates FHDs via a brute force search, then selects one min-width FHD together with other optimizations to generate a query plan. Instead, we generate 10 min-width FHDs with our FHD algorithm for EmptyHeaded to select. We compare the optimization time for searching FHDs and the execution time

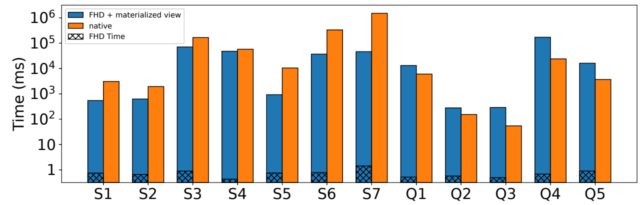


Figure 8: Query evaluation on PostgreSQL

between the native EmptyHeaded and the EmptyHeaded with new FHD algorithm. On the other hand, PostgreSQL/DuckDB do not support FHD and WCOJ. To simulate this, for an FHD plan, we first process every bag in the FHD plan as an *SQL* materialized view, and then join all such views by binary joins. It is worth mentioning that this results in significant additional I/O overhead in writing and reading materialized views.

The queries tested are as follows. ❶ The 7 graph pattern queries (Fig. 7) are two 6-vertex subgraph queries (S_1, S_2), three 8-vertex subgraph queries (S_3, S_4, S_5) and two 10-vertex subgraph queries (S_6, S_7) from DBLP dataset. Each query counts the number of subgraphs centered around every node in the data graph. ❷ The 5 cyclic *SQL* queries are taken from query5 (Q_1) in TPC-H, and query24a (Q_2), query24b (Q_3), query78 (Q_4) in TPC-DS, and query3 (Q_5) in LSQB. We generate data with SF=1 for LSQB and 1GB for TPC.

The EmptyHeaded results are shown in Table 5. For FHD searching time, our algorithm significantly outperforms the built-in algorithm. As all FHD plans are with the min width, the execution time are similar. The PostgreSQL results are shown in Fig. 8, due to space limit. The DuckDB shows the similar performance. For graph pattern queries ($S_1 \sim S_7$), PostgreSQL does not perform well for such complex cyclic queries. The FHD plan used achieves up to two orders of magnitude performance improvement. For *SQL* queries, the benefit of FHD is less obvious. There are two main reasons: fisrt, the simulation in PostgreSQL/DuckDB needs additional I/O overhead; second, FHD does not take data distribution into consideration, where the result for a bag in FHD used may generate very large intermediate relations (Q_4, Q_5). We will explore how to integrate FHD and WCOJ into a database system.

9 CONCLUSION

In this paper, we give a *DP* algorithm for computing FHD which is the first approach that is not based on CHECK(,). We give a branch-&-bound algorithm, BB4FHD, together with upper/lower bounds to accelerate the *DP* algorithm. Our BB4FHD algorithm is an anytime algorithm that can give a feasible solution at any time when it stops and can give a better solution if more time is allowed. With BB4FHD, we can compute the optimal fhtw for 2,667 out of the total number of 3,648 instances, which is better than FraSMT, which computes the optimal fhtw for 2,538. BB4FHD can compute the optimal fhtw for 84% instances within time of 1 second. We confirm that BB4FHD can compute fhtw efficiently and it leads to efficient query processing in database systems.

10 ACKNOWLEDGEMENT

This work was supported by the Research Grants Council of Hong Kong, China, No.14205520.

REFERENCES

[1] [n.d.]. Experimental Data for log-k-decomp. https://zenodo.org/record/6389816.

[2] Christopher Aberger, Andrew Lamb, Kunle Olukotun, and Christopher Re. 2018. LevelHeaded: A Unified Engine for Business Intelligence and Linear Algebra Querying. In *2018 IEEE 34th International Conference on Data Engineering (ICDE)*. 449–460.

[3] Christopher R. Aberger, Andrew Lamb, Susan Tu, Andres Nötzli, Kunle Olukotun, and Christopher Ré. 2017. EmptyHeaded: A Relational Engine for Graph Processing. *ACM Trans. Database Syst.* 42, 4, Article 20 (oct 2017), 44 pages.

[4] Isolde Adler, Tomas Gavenciak, and Tereza Klimosová. 2012. Hypertree-depth and minors in hypergraphs. *Theor. Comput. Sci.* 463 (2012), 84–95. https://doi.org/10.1016/j.tcs.2012.09.007

[5] Kamal Amroun, Zineb Habbas, and Wassila Aggoune-Mtalaa. 2016. A compressed generalized hypertree decomposition-based solving technique for non-binary constraint satisfaction problems. *AI Communications* 29, 2 (2016), 371–392.

[6] Albert Atserias, Martin Grohe, and Dániel Marx. 2008. Size Bounds and Query Plans for Relational Joins. In *2008 49th Annual IEEE Symposium on Foundations of Computer Science*. 739–748.

[7] Philip A. Bernstein and Nathan Goodman. 1981. Power of Natural Semijoins. *SIAM J. Comput.* 10, 4 (nov 1981), 751–771.

[8] Hans L. Bodlaender, Fedor V. Fomin, Arie M. C. A. Koster, Dieter Kratsch, and Dimitrios M. Thilikos. 2012. On Exact Algorithms for Treewidth. *ACM Trans. Algorithms* 9, 1, Article 12 (dec 2012), 23 pages.

[9] Angela Bonifati, Wim Martens, and Thomas Timm. 2017. An analytical study of large SPARQL query logs. *Proc. VLDB Endow.* 11, 2 (oct 2017), 149–161.

[10] Hubie Chen, Georg Gottlob, Matthias Lanzinger, and Reinhard Pichler. 2021. Semantic Width and the Fixed-Parameter Tractability of Constraint Satisfaction Problems. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence* (Yokohama, Yokohama, Japan) *(IJCAI'20)*. Article 239, 8 pages.

[11] Binyang Dai, Qichen Wang, and Ke Yi. 2023. SparkSQL+: Next-generation Query Planning over Spark. In *Companion of the 2023 International Conference on Management of Data* (Seattle, WA, USA) *(SIGMOD '23)*. Association for Computing Machinery, New York, NY, USA, 115–118.

[12] Ronald Fagin. 1983. Degrees of acyclicity for hypergraphs and relational database schemes. *J. ACM* 30, 3 (jul 1983), 514–550.

[13] Johannes K. Fichte, Markus Hecher, Neha Lodha, and Stefan Szeider. 2018. An SMT Approach to Fractional Hypertree Width. In *Principles and Practice of Constraint Programming*, John Hooker (Ed.). Springer International Publishing, Cham, 109–127.

[14] Wolfgang Fischl, Georg Gottlob, Davide Mario Longo, and Reinhard Pichler. 2020. HyperBench: A Benchmark and Tool for Hypergraphs and Empirical Findings. arXiv:2009.01769 [cs.DB]

[15] Wolfgang Fischl, Georg Gottlob, and Reinhard Pichler. 2018. General and Fractional Hypertree Decompositions: Hard and Easy Cases. In *Proceedings of the 37th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems* (Houston, TX, USA) *(PODS '18)*. Association for Computing Machinery, New York, NY, USA, 17–32.

[16] Lucantonio Ghionna, Luigi Granata, Gianluigi Greco, and Francesco Scarcello. 2007. Hypertree Decompositions for Query Optimization. In *2007 IEEE 23rd International Conference on Data Engineering*. 36–45.

[17] Georg Gottlob, Gianluigi Greco, Nicola Leone, and Francesco Scarcello. 2016. Hypertree Decompositions: Questions and Answers. In *Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2016, San Francisco, CA, USA, June 26 - July 01, 2016*, Tova Milo and Wang-Chiew Tan (Eds.). ACM, 57–74.

[18] Georg Gottlob, Gianluigi Greco, Nicola Leone, and Francesco Scarcello. 2016. Hypertree Decompositions: Questions and Answers *(PODS '16)*. Association for Computing Machinery, New York, NY, USA, 57–74.

[19] Georg Gottlob, Matthias Lanzinger, Cem Okulmus, and Reinhard Pichler. 2022. Fast Parallel Hypertree Decompositions in Logarithmic Recursion Depth. In *Proceedings of the 41st ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems* (Philadelphia, PA, USA) *(PODS '22)*. Association for Computing Machinery, New York, NY, USA, 325–336.

[20] Georg Gottlob, Nicola Leone, and Francesco Scarcello. 2002. Hypertree Decompositions and Tractable Queries. *J. Comput. System Sci.* 64, 3 (2002), 579–627.

[21] Georg Gottlob, Cem Okulmus, and Reinhard Pichler. 2020. Fast and Parallel Decomposition of Constraint Satisfaction Problems. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*, Christian Bessiere (Ed.). International Joint Conferences on Artificial Intelligence Organization, 1155–1162. Main track.

[22] Georg Gottlob and Marko Samer. 2009. A Backtracking-Based Algorithm for Hypertree Decomposition. *ACM J. Exp. Algorithmics* 13, Article 1 (feb 2009), 19 pages.

[23] Martin Grohe and Dániel Marx. 2014. Constraint Solving via Fractional Edge Covers. *ACM Trans. Algorithms* 11, 1, Article 4 (aug 2014), 20 pages.

[24] Xiao Hu, Stavros Sintos, Junyang Gao, Pankaj K. Agarwal, and Jun Yang. 2022. Computing Complex Temporal Join Queries Efficiently *(SIGMOD '22)*. Association for Computing Machinery, New York, NY, USA, 2076–2090.

[25] Xiao Hu and Qichen Wang. 2023. Computing the Difference of Conjunctive Queries Efficiently. 1, 2, Article 153 (jun 2023), 26 pages.

[26] Kyoungmin Kim, Jaehyun Ha, George Fletcher, and Wook-Shin Han. 2023. Guaranteeing the Õ(AGM/OUT) Runtime for Uniform Sampling and Size Estimation over Joins. In *Proceedings of the 42nd ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems* (Seattle, WA, USA) *(PODS '23)*. Association for Computing Machinery, New York, NY, USA, 113–125.

[27] Michael Langberg, Shi Li, Sai Vikneshwar Mani Jayaraman, and Atri Rudra. 2019. Topology Dependent Bounds For FAQs. In *Proceedings of the 38th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems* (Amsterdam, Netherlands) *(PODS '19)*. Association for Computing Machinery, New York, NY, USA, 432–449.

[28] Ling Liu, Ryen W. White, Amin Mantrach, Fabrizio Silvestri, Julian J. McAuley, Ricardo Baeza-Yates, and Leila Zia (Eds.). 2019. *The World Wide Web Conference, WWW 2019, San Francisco, CA, USA, May 13-17, 2019*. ACM.

[29] Dániel Marx. 2010. Approximating Fractional Hypertree Width. 6, 2, Article 29 (apr 2010), 17 pages.

[30] Thomas Neumann. 2009. Query simplification: graceful degradation for join-order optimization. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data* (Providence, Rhode Island, USA) *(SIGMOD '09)*. Association for Computing Machinery, New York, NY, USA, 403–414.

[31] Hung Q. Ngo, Ely Porat, Christopher Ré, and Atri Rudra. 2018. Worst-case Optimal Join Algorithms. *J. ACM* 65, 3, Article 16 (mar 2018), 40 pages. https://doi.org/10.1145/3180143

[32] Dan Olteanu and Jakub Závodný. 2015. Size Bounds for Factorised Representations of Query Results. *ACM Trans. Database Syst.* 40, 1, Article 2 (mar 2015), 44 pages.

[33] Marko Samer and Helmut Veith. 2009. Encoding Treewidth into SAT. In *Theory and Applications of Satisfiability Testing - SAT 2009*, Oliver Kullmann (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 45–50.

[34] Andre Schidler and Stefan Szeider. 2021. Computing Optimal Hypertree Decompositions with SAT. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, Zhi-Hua Zhou (Ed.). International Joint Conferences on Artificial Intelligence Organization, 1418–1424. Main Track.

[35] Yuchao Tao, Xi He, Ashwin Machanavajjhala, and Sudeepa Roy. 2020. Computing Local Sensitivities of Counting Queries with Joins. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data* (Portland, OR, USA) *(SIGMOD '20)*. Association for Computing Machinery, New York, NY, USA, 479–494.

[36] Robert E. Tarjan and Mihalis Yannakakis. 1984. Simple Linear-Time Algorithms to Test Chordality of Graphs, Test Acyclicity of Hypergraphs, and Selectively Reduce Acyclic Hypergraphs. *SIAM J. Comput.* 13, 3 (1984), 566–579.

[37] Qichen Wang and Ke Yi. 2022. Conjunctive Queries with Comparisons. In *Proceedings of the 2022 International Conference on Management of Data* (Philadelphia, PA, USA) *(SIGMOD '22)*. Association for Computing Machinery, New York, NY, USA, 108–121.

[38] Mihalis Yannakakis. 1981. Algorithms for Acyclic Database Schemes. In *Proceedings of the Seventh International Conference on Very Large Data Bases - Volume 7* (Cannes, France) *(VLDB '81)*. VLDB Endowment, 82–94.

[39] Hao Zhang, Jeffrey Yu, Yikai Zhang, Kangfei Zhao, and Hong Cheng. 2020. Distributed subgraph counting: a general approach. *Proceedings of the VLDB Endowment* 13 (08 2020), 2493–2507. https://doi.org/10.14778/3407790.3407840

[40] Hao Zhang, Jeffrey Xu Yu, Yikai Zhang, and Kangfei Zhao. 2022. Parallel Query Processing: To Separate Communication from Computation. In *Proceedings of the 2022 International Conference on Management of Data* (Philadelphia, PA, USA) *(SIGMOD '22)*. Association for Computing Machinery, New York, NY, USA, 1447–1461.

[41] Kamal Amroun Zineb Habbas and Daniel Singer. 2015. A Forward-Checking algorithm based on a Generalised Hypertree Decomposition for solving non-binary constraint satisfaction problems. *Journal of Experimental & Theoretical Artificial Intelligence* 27, 5 (2015), 649–671.