Towards Parameterized Hardness on Maintaining Conjunctive Queries

QICHEN WANG, Nanyang Technological University, Singapore

We investigate the fine-grained complexity of dynamically maintaining the result of fixed self-join free conjunctive queries under single-tuple updates. Prior work shows that free-connex queries can be maintained in update time $O(|D|^{\delta})$ for some $\delta \in [0.5, 1]$, where $|D|$ is the size of the current database. However, a gap remains between the best known upper bound of $O(|D|)$ and lower bounds of $\Omega(|D|^{0.5-\epsilon})$ for any $\epsilon \geqslant 0$.

We narrow this gap by introducing two structural parameters to quantify the dynamic complexity of a conjunctive query: the height k and the dimension d. We establish new fine-grained lower bounds showing that any algorithm maintaining a query with these parameters must incur update time $\Omega(|D|^{1-1/\max(k,d)-\epsilon})$, unless widely believed conjectures fail. These yield the first super-$\sqrt{|D|}$ lower bounds for maintaining free-connex queries, and suggest the tightness of current algorithms when considering arbitrarily large k and d.

Complementing our lower bounds, we identify a data-dependent parameter, the generalized H-index $h(D)$, which is upper bounded by $|D|^{1/d}$, and design an efficient algorithm for maintaining star queries, a common class of height 2 free-connex queries. The algorithm achieves an instance-specific update time $O(h(D)^{d-1})$ with linear space $O(|D|)$. This matches our parameterized lower bound and provides instance-specific performance in favorable cases.

CCS Concepts: • **Theory of computation → Database query processing and optimization (theory)**.

Additional Key Words and Phrases: conjunctive query, fine-grained complexity, query maintenance, dynamic algorithms

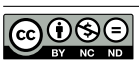
ACM Reference Format:
Qichen Wang. 2026. Towards Parameterized Hardness on Maintaining Conjunctive Queries. *Proc. ACM Manag. Data* 4, 2 (PODS), Article 121 (May 2026), 26 pages. https://doi.org/10.1145/3801917

1 Introduction

Answering conjunctive queries under updates is a fundamental task in database theory and systems, with applications to incremental view maintenance [13, 24, 27, 33, 35, 36], complex event processing [34], and streaming processing [10]. In the fully dynamic setting, a fixed self-join free conjunctive query[1] Q is given together with an initially empty database D. The database then undergoes a sequence of tuple insertions and deletions. The goal is to maintain a data structure so that the current query result $Q(D)$ can be retrieved at any time. In particular, let $\omega(Q)$ be the *optimal* static exponent, meaning that for every database instance D, all algorithms that can evaluate query Q over D requires runtime $\Omega(|D|^{\omega(Q)-\epsilon} + |Q(D)|)$ for some constant $\epsilon \geqslant 0$. We are interested in the dynamic algorithms where, after each update, the maintained data structure supports

[1]In the remainder of this paper, we consider only "self-join free" conjunctive queries and will omit "self-join free" from the text. A conjunctive query with self-joins may be easier than the associated self-join free conjunctive query, where the associated self-join free query is obtained by giving distinct relational symbols to all relations [12]. While some lower bounds have been established on queries with self-joins (e.g., Theorem 1.2 in [9] on maintaining Boolean queries with self-joins), the hardness results for queries with self-joins are not currently settled in general for static and dynamic evaluations.

Author's Contact Information: Qichen Wang, Nanyang Technological University, Singapore, qichen.wang@ntu.edu.sg.



ACM 2836-6573/2026/5-ART121
https://doi.org/10.1145/3801917

retrieving $Q(D)$ with strictly smaller overhead than re-evaluating Q from scratch, i.e., in time $O(|D|^{\omega(Q)-\epsilon} + |Q(D)|)$ for some constant $\epsilon > 0$. Here, $|D|$ denotes the size of the current database, and $|Q(D)|$ denotes the size of the current query result. We measure efficiency by the amortized update time t_u for processing a single insertion or deletion, as well as the enumeration delay. Ideally, the enumeration delay should be constant, i.e., independent of $|D|$.

Despite this restriction, the dynamic problem remains challenging. Recent works have established nontrivial upper and lower bounds. The optimal static exponent $\omega(Q)$ provides a straightforward lower bound $\Omega(|D|^{\omega(Q)-1-\epsilon})$ on the amortized update time [5, 23] via an insertion-only reduction. Specifically, if we construct a database instance D by inserting tuples one by one, a dynamic algorithm with amortized update time t_u and enumeration time $O(|D|^{\omega(Q)-\epsilon} + |Q(D)|)$ would allow evaluating Q in $O(|D| \cdot t_u + |D|^{\omega(Q)-\epsilon} + |Q(D)|)$. Consequently, for any $\gamma > 0$, we must have $t_u = \Omega(|D|^{\omega(Q)-1-\gamma})$ even under insertion-only updates; otherwise, the optimality of $\omega(Q)$ would be violated.Although the lower bound is tight [5, 23] for insertion-only update sequences on free-connex queries (i.e., acyclic queries with $\omega(Q) = 1$), it is not the best lower bound when considering deletions. Berkholz et al. [9] introduced the notion of q-hierarchical queries, a subset of free-connex queries, and proved a tight dichotomy: assuming the Online Matrix-Vector multiplication (OMv) conjecture, every conjunctive query that is not q-hierarchical cannot be maintained with constant update time while supporting constant delay enumeration. In particular, if a conjunctive query is not q-hierarchical, any dynamic algorithm must incur amortized $\Omega(|D|^{1/2-\epsilon})$ for any $\epsilon > 0$ update time to achieve constant delay enumeration. On the upper-bound side, multiple efforts have been made to efficiently maintain free-connex conjunctive queries [24, 33, 35]. They demonstrate that free-connex queries can be maintained in linear time $O(|D|)$ per update, while still supporting constant delay enumeration. Kara et al. [28] studies trade-offs between amortized update time t_u and enumeration delay t_d for hierarchical queries, another subset of acyclic queries that intersects free-connex queries. For free-connex hierarchical queries, they show a product trade-off $t_d \times t_u = O(|D|)$. On the other hand, since $\omega(Q) = 1$ for all free-connex queries, any non-constant enumeration delay will result in a larger overhead than re-evaluating the query when $|Q(|D|)| = \Omega(|D|)$. Together, these results imply that the best achievable exponent δ for free-connex queries currently lies between 0.5 and 1. Whether we can close this gap remains an open problem in this field.

1.1 Our contributions

In this work, we make progress toward this question by introducing parameterized lower bounds that refine the known dichotomy. We define two new structural parameters of a conjunctive query Q: the height (denoted k) and the dimension (denoted d). Intuitively, the height of Q measures the length of the longest chordless path in Q's relational hypergraph and will increase as we "scale out" the query with more relations. For a free-connex query Q, the height of Q is equal to the minimum height of all free-connex join trees for Q. On the other hand, the dimension of Q measures how the query "scales up" by adding more attributes per relation.[2] These parameters induce a natural hierarchy within conjunctive queries. See Figure 1 for the hierarchy proposed

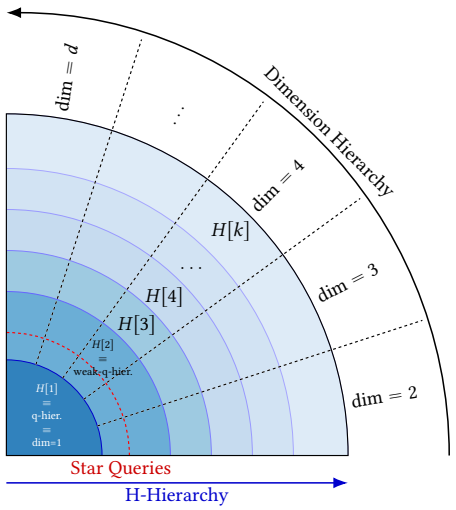


Fig. 1. Query Hierarchy with $\omega(Q) = 1$.

[2]Formal definitions of the height and dimension parameters are given in Sections 3 and 4, respectively.

in this paper on free-connex queries with $\omega(Q) = 1$. We then prove fine-grained lower bounds that depend on (k, d). In particular, we show:

Theorem 1.1 (**height-based lower bound**). *Let Q be a height k query with $k \geqslant 2$. Assuming combinatorial k-clique conjecture, no dynamic **combinatorial** algorithm can maintain Q with amortized update time $O(|D|^{(k-1)/k-\epsilon})$ and support constant delay enumeration for any constant $\epsilon > 0$.*

Theorem 1.2 (**Dimension-based lower bound**). *Let Q be a conjunctive query of dimension $d \geqslant 2$. Assuming the OuMv$_k$ conjecture, no dynamic algorithm can maintain Q with amortized update time $O(|D|^{(d-1)/d-\epsilon})$ and support constant delay enumeration for any constant $\epsilon > 0$.*

Theorem 1.1 shows that higher-height queries inherently require larger update costs: for height k one cannot beat the exponents $(k-1)/k$. Theorem 1.2 shows that queries of dimension d similarly cannot be maintained with an exponent below $(d-1)/d$. In addition, they recover the previous dichotomy from Berkholz et al. [9]:

Proposition 1.3. *For any conjunctive query Q, the following statements are equivalent: (1) Q is q-hierarchical; (2) Q has height $k = 1$; and (3) Q has dimension $d = 1$.*

These results also complement the $\omega(Q) - 1$ lower bounds when $\omega(Q) \leqslant 2$. When $\omega(Q) \leqslant 1.5$, the lower bounds from height and dimension dominate and any maintenance algorithm must incur amortized update time $\Omega\left(|D|^{1-1/\max(k,d)-\epsilon}\right)$, while when $1.5 < \omega(Q) \leqslant 2$, any maintenance algorithm must incur amortized update time $\Omega\left(|D|^{\delta-\epsilon}\right)$ with $\delta = 1 - 1/\max\left(k, d, \frac{1}{2-\omega(Q)}\right)$ and any constant $\epsilon > 0$. In particular, these results are the **first to provide super-square-root lower bounds for maintaining free-connex queries**, implying that a general $O(|D|^{1-\epsilon})$-time maintenance algorithm for all free-connex queries would violate these conjectures, resolving the open problem in the negative unless breakthrough algorithms for the underlying conjectured-hard problems are found, and thus **prove the optimality of current upper bound algorithms**.

On the algorithmic side, we complement our lower bounds with a new parameterized upper bound for star queries, a subset of height 2 queries with details given in Definition 4.5. Specifically:

Theorem 1.4 (**Algorithm for Star Queries**). *Let Q be a star query with dimension $d \geqslant 1$, and $j \in [0, d]$ output attributes in $\bar{x}_1$, D be the current database instance, and $h(D)$ be its* generalized H-index *(a measure of database densities, see Section 5). There is a dynamic algorithm with linear space $O(|D|)$ that maintains Q with an amortized update time of $O(h(D)^{d-1})$ and supports $O(h(D)^{d-j})$-delay enumeration.*

Theorem 1.4 shows that for those star queries that are widely used in analytical workloads, we can sometimes do better than the general $(d-1)/d$ bound by exploiting the instance-dependent H-index parameter. Meanwhile, $h(D) \leqslant |D|^{1/d}$ always holds; any algorithm that can maintain such queries in time $O(h(D)^{d-1-\epsilon})$ for any $\epsilon > 0$ would contradict Theorem 1.2 and OuMv$_k$ conjecture.

1.2 Organization

The rest of the paper is structured as follows. In Section 2, we introduce notation and recall the necessary background used in this paper. We also formally define the height and dimension parameters of a query. In Section 3, we study the height parameter, formalizing the notion of a height k query and proving Theorem 1.1 via a reduction from the Odd/Even Cycle detection problem. In Section 4 we study the dimension parameter and prove Theorem 1.2 via a reduction from the OuMv$_k$ problem. In Section 5 we present our dynamic algorithm for star queries and prove Theorem 1.4. We conclude in Section 6 with a discussion of related work and future directions, with proofs and additional examples in the appendix.

2 Preliminaries

2.1 Conjunctive Queries and Their Structural Properties

2.1.1 Acyclic Conjunctive Queries.

Conjunctive Queries (CQs) [4]. Let a database schema be defined by a set of attributes $\mathcal{V} = \{x_1, \ldots, x_m\}$ and a set of n relations $\mathcal{E} = \{R_1, \ldots, R_n\}$. Each relation $R_i \in \mathcal{E}$ has a schema $\bar{x}_i \subseteq \mathcal{V}$. A conjunctive query (CQ) is expressed as:

$$Q(\bar{y}) \leftarrow R_1(\bar{x}_1), \cdots, R_n(\bar{x}_n),$$

where $\bar{y} \subseteq \mathcal{V}$ is the set of output attributes. If $\bar{y} = \mathcal{V}$, the query is a full conjunctive query, and we can omit $\bar{y}$ from the head, denote it as $Q(*)$. If $\bar{y} = \emptyset$, the query is a Boolean conjunctive query, and we write it as $Q()$. We also use $(\mathcal{V}, \mathcal{E}, \bar{y})$ to represent a CQ.

For any attribute $x \in \mathcal{V}$, we let $\mathcal{E}[x] = \{R(\bar{x}) \in \mathcal{E} : x \in \bar{x}\}$ be the set of relations containing x. We also let $\mathcal{V}_\bullet = \{x \in \mathcal{V} : |\mathcal{E}[x]| = 1\}$ be the set of unique attributes, i.e., attributes that are not join attributes.

Hypergraphs and Acyclicity. The acyclicity of the associated relational hypergraph defines the acyclicity of a CQ Q.

Definition 2.1 (Relational Hypergraph). The relational hypergraph associated with a conjunctive query $Q = (\mathcal{V}, \mathcal{E}, \bar{y})$ is $\mathcal{H} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{E}$ is the set of hyperedges (relations) and $\mathcal{V}$ is the set of vertices (attributes). Each hyperedge $R_i \in \mathcal{E}$ contains the set of vertices $\bar{x}_i$.

A *primal graph* $G = (V, E)$ of $\mathcal{H}$ is a graph with $V = \mathcal{V}$, and an edge $(v, u) \in E$ if and only if there exists $R(\bar{x}) \in \mathcal{E}$, such that $v, u \in \bar{x}$. A *primal graph* over $V \subseteq \mathcal{V}$ is a graph $G(V) = (V, E)$, and an edge $(v, u) \in E$ if and only if there exists $R(\bar{x}) \in \mathcal{E}$, such that $v, u \in V \cap \bar{x}$.

There are multiple notions of acyclicity; in this paper, we consider the standard notion of α-acyclicity. A CQ Q is α-acyclic if and only if it admits a *generalized join tree* (GJT) $\mathcal{T}$ [35].

A GJT is a tree $\mathcal{T} = (\mathcal{V}(\mathcal{T}), \mathcal{E}(\mathcal{T}))$ where each node $v \in \mathcal{V}(\mathcal{T})$ corresponds to either an *input relation* $R_i(\bar{x}_i) \in \mathcal{E}$, or a *generalized relation* $[\bar{x}]$ with $[\bar{x}] \subseteq \bar{x}_i$ for some $R_i(\bar{x}_i)$. The GJT $\mathcal{T}$ satisfy:

- **Cover:** For every relation $R_i(\bar{x}_i) \in \mathcal{E}$, there exists a node $v \in \mathcal{V}(\mathcal{T})$ such that $v = R_i$. Furthermore, all leaf nodes of $\mathcal{T}$ must be input relations from $\mathcal{E}$.
- **Connect:** For any attribute $x \in \mathcal{V}$, the set of nodes $\{v \in \mathcal{V}(\mathcal{T}) : x \in v\}$ induces a connected subtree in $\mathcal{T}$.

This definition generalizes traditional join trees, which are special cases where all nodes in $\mathcal{V}(\mathcal{T})$ correspond to input relations in $\mathcal{E}$.

2.1.2 Updates, Enumeration and Model of Computation.

We assume an initially empty database. We consider a dynamic setting where the database undergoes a sequence of single-tuple updates. Each update consists of either inserting or deleting a tuple $t \in \mathrm{dom}(\bar{x})$ from a relation $R(\bar{x})$. We adhere to set semantics: inserting a tuple t that already exists in R, or deleting a tuple t that is not present in R, results in no change to the database instance. While a specific tuple value may be inserted or deleted multiple times throughout the sequence, we treat each operation as a distinct event.

Updates are processed sequentially; the maintenance procedure for a single update must conclude before the next update is received. Following any update, the system may allow enumeration of the query result. We define the *enumeration delay* as the maximum time elapsed between the start of the enumeration and the output of the first tuple, between the output of any two consecutive tuples, and between the output of the last tuple and the termination of the procedure. We focus on *data complexity*, treating the query size as a constant.

Our complexity analysis is based on the standard word RAM model of computation under the uniform cost measure. We assume a machine word size of $w = \Omega(\log N)$ bits for an input of size N. In this model, fundamental operations, including arithmetic operations, bitwise logic, concatenation, and direct memory access, are executed in $O(1)$ time.

2.1.3 Free-connex CQs. The definition of acyclicity for conjunctive queries does not consider the influence of output attributes. However, if we also take the output attributes $\bar{y}$ into account, we can refine the notion of acyclicity to define the class of *free-connex* queries. Specifically, an acyclic CQ $Q = (\mathcal{V}, \mathcal{E}, \bar{y})$ is *free-connex* if the derived query $Q' = (\mathcal{V}, \mathcal{E} \cup \{\bar{y}\}, \bar{y})$, obtained by treating the set of output attributes $\bar{y}$ as an additional relation, is also acyclic. Equivalently, an acyclic CQ Q is free-connex if it admits a free-connex join tree.

free-connex join trees. A GJT $\mathcal{T}$ is a *free-connex join tree* if it satisfies the standard **Cover** and **Connect** properties, as well as the following three additional properties:

- **Above:** No input relation node $v \in \mathcal{V}(\mathcal{T})$ is an ancestor of any generalized relation node.
- **Guard:** For any generalized relation node $v \in \mathcal{V}(\mathcal{T})$, all its children $v_c \in \mathcal{V}(\mathcal{T})$ satisfy $v \subseteq v_c$.
- **Connex:** There exists a *connex set* of nodes $\mathcal{V}_{\mathsf{con}}(\mathcal{T}) \subseteq \mathcal{V}(\mathcal{T})$ such that: (i) $\mathcal{V}_{\mathsf{con}}(\mathcal{T})$ induces a connected subtree of $\mathcal{T}$ that contains the root of $\mathcal{T}$; (ii) for each node $v \in \mathcal{V}_{\mathsf{con}}(\mathcal{T})$ with parent v_p, their shared attributes are all output attributes, i.e., $(v \cap v_p) \subseteq \bar{y}$; and (iii) the union of attributes in the connex set covers all output attributes, i.e., $\bar{y} \subseteq \bigcup_{v \in \mathcal{V}_{\mathsf{con}}(\mathcal{T})} v$.

The class of free-connex queries attracts significant attention because only free-connex queries can be evaluated in linear time or enumerated in constant delay with linear preprocessing time, assuming the Boolean matrix multiplication conjecture and Hyperclique conjecture.

Conjecture 2.2 (Boolean Matrix Multiplication Conjecture (BMM)). *The multiplication of two $n \times n$ boolean matrices cannot be done in time $O(n^2)$.*

Conjecture 2.3 (HyperClique Conjecture (HyperClique)). *For any integer $k \geqslant 3$, it is not possible to determine the existence of a k hyperclique in a $(k-1)$-uniform hypergraph with m hyperedges in time $O(m)$.*

Theorem 2.4 ([8]). *A CQ Q can be evaluated in linear time $O(|D| + |Q(D)|)$ if and only if Q is free-connex, assuming BMM and HyperClique.*

Theorem 2.5 ([8]). *A CQ Q can be enumerated with $O(1)$ delay after a linear-time $O(|D|)$ preprocessing step if and only if Q is free-connex, assuming BMM and HyperClique.*

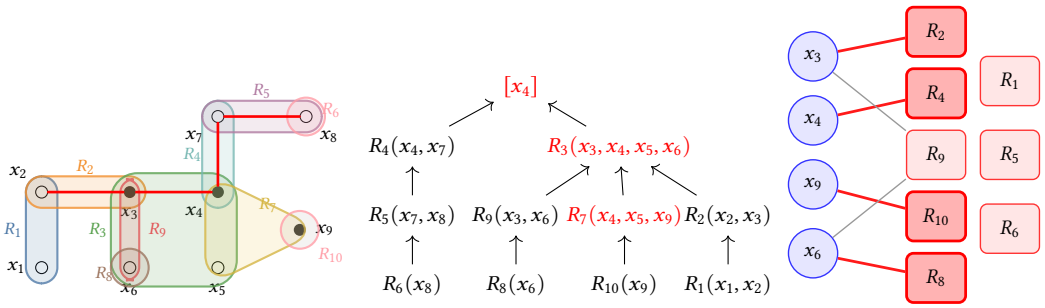
These theorems also suggest that a CQ Q has $\omega(Q) = 1$ if and only if Q is free-connex, assuming BMM. Such dichotomies also extend to dynamic settings with insertion-only update sequences.

Theorem 2.6 ([23, 24, 35]). *Given any insertion-only update sequence, a CQ Q can be maintained in amortized $O(1)$ time while supporting $O(1)$-delay enumeration if and only if Q is free-connex, assuming BMM and HyperClique.*

Example 2.7. Consider the free-connex query

$$\begin{aligned} Q_1(x_3, x_4, x_9) \leftarrow & R_1(x_1, x_2), R_2(x_2, x_3), R_3(x_3, x_4, x_5, x_6), R_4(x_4, x_7), \\ & R_5(x_7, x_8), R_6(x_8), R_7(x_4, x_5, x_9), R_8(x_6), R_9(x_3, x_6), R_{10}(x_9). \end{aligned}$$

Figure 2a shows its relational hypergraph with output attributes marked in solid dots, and Figure 2b shows a possible generalized join tree, where all nodes in red form the connex subtree.



(a) Relational Hypergraph (b) Generalized Join Tree (c) $\mathcal{N}$ and $\mathcal{V}_k(\mathcal{N})$ for $\{x_5\}$

Fig. 2. Illustration of Example 2.7

2.1.4 Fine-grained Classification of Free-connex Queries. In the static setting, every free-connex query can be efficiently evaluated and enumerated. However, Berkholz et al. [9] showed that in the dynamic setting, understanding the hardness of maintaining queries requires a more fine-grained classification. In particular, they identified the class of *q-hierarchical queries*, a subset of free-connex queries that can be efficiently maintained.

q-hierarchical CQs. A CQ $Q = (\mathcal{V}, \mathcal{E}, \bar{y})$ is q-hierarchical if it satisfies the two conditions: (1) for any pair of attributes $x_1, x_2 \in \mathcal{V}$, either $\mathcal{E}[x_1] \subseteq \mathcal{E}[x_2]$ or $\mathcal{E}[x_2] \subseteq \mathcal{E}[x_1]$ or $\mathcal{E}[x_1] \cap \mathcal{E}[x_2] = \emptyset$; and (2) if $x_1 \in \bar{y}$ and $\mathcal{E}[x_1] \subsetneq \mathcal{E}[x_2]$, then $x_2 \in \bar{y}$.

Theorem 2.8 ([9],Theorem 1.1). *Given any update sequence, a CQ Q can be maintained in $O(1)$ time while supporting $O(1)$-delay enumeration if and only if Q is q-hierarchical, assuming OMv Conjecture.*

Weak-q-hierarchical CQs. The q-hierarchical class is quite restrictive; for example, even a simple query $Q(x_1) \leftarrow R_1(x_1, x_2), R_2(x_2)$ is not q-hierarchical. Wang et al. [23, 35] slightly extended this class by defining *weak-q-hierarchical* queries, which can be efficiently maintained under certain practical conditions, such as first-in-first-out (FIFO) update sequences. To define weak-q-hierarchical queries, we need to introduce the notions of *ear* and *skeleton*.

The concept of ears was first introduced by the GYO algorithm [20, 38], where a relation R is an ear if there exists another relation R' that contains all non-unique attributes of R. The algorithm tests α-acyclicity of a CQ by iteratively removing all unique attributes and relations whose attributes are contained in another relation, or equivalently, iteratively removing ears from the query, and then checking whether the query becomes empty. If the query is reduced to the empty set, one can construct a valid join tree for the query by connecting each removed ear R to its anchor R'.

Hu and Wang [23] extended this notion to account for output attributes, referring to such relations as *reducible relations*. Under this extension, the generalized GYO reduction proceeds as follows: iteratively removing a unique attribute if it is not an output attribute, or if it belongs to a relation R whose non-unique attributes are all output attributes. One then removes every relation whose attributes are contained in another relation. Based on this generalized reduction, we now give the formal definition of an ear for a free-connex query.

Definition 2.9 (Ears). In a free-connex query $Q = (\mathcal{V}, \mathcal{E}, \bar{y})$, a relation $R(\bar{x}) \in \mathcal{E}$ is called an ear of the query if there exists another relation $R'(\bar{x}') \in \mathcal{E}$, such that either (i) $\bar{x} \backslash \mathcal{V}_\bullet \subseteq \bar{x}'$ and $\bar{x} \cap \mathcal{V}_\bullet \subseteq \mathcal{V} \backslash \bar{y}$; or (ii) $\bar{x} \backslash \mathcal{V}_\bullet \subseteq \bar{x}' \cap \bar{y}$. We refer to R' as an anchor of R.

The *skeleton* of a query Q, denoted $\mathcal{E}_*$, is the subset of relations remaining after recursively removing an ear from $\mathcal{E}$ until no further removal is possible. This reduction is summarized in

Algorithm 1: REDUCTION($Q = (\mathcal{V}, \mathcal{E}, \bar{y})$)

$\mathcal{E}_e \leftarrow$ {all ears in $\mathcal{E}$};
while *there exists* $R(\bar{x}), R'(\bar{x}') \in \mathcal{E}$ *s.t.* $R \in \mathcal{E}_e$ *and* R' *is its anchor* **do**
  $\mathcal{E} \leftarrow \mathcal{E} - \{R\}$
return $\mathcal{E}$;

Algorithm 1 and is similar to the GYO algorithm, except that the set of ears is computed once at the start and held fixed while removing them from $\mathcal{E}$. As a consequence, the skeleton is guaranteed to be non-empty, as removing an ear requires an anchor in the remaining query. We further define the concept of residual query for a given free-connex query:

Definition 2.10 (Residual Query). Given a free-connex query Q with its skeleton $\mathcal{E}_*$, where $\mathcal{V}_* = \{x \mid \exists R(\bar{x}) \in \mathcal{E}_*, x \in \bar{x}\}$ is the set of attributes in $\mathcal{E}_*$, and $\bar{y}_* = \{x \mid x \in \bar{y} \wedge \exists R(\bar{x}) \in \mathcal{E}_*, x \in \bar{x}\}$ is the set of output attributes in $\mathcal{E}_*$, the residual query Q_* for Q is $\mathcal{E}_*$-induced query $Q_* \leftarrow (\mathcal{V}_*, \mathcal{E}_*, \bar{y}_*)$.

It is possible that a pair of relations R_i, R_j is both an ear relation and the anchor of the other, making the skeletons, and thus the residual queries, not unique. However, the underlying hypergraphs for different residual queries should be isomorphism and the hardness of these residual queries remains the same:

LEMMA 2.11. *Let* $Q_1 = (\mathcal{V}_1, \mathcal{E}_1, \bar{y}_1)$ *and* $Q_2 = (\mathcal{V}_2, \mathcal{E}_2, \bar{y}_2)$ *are two residual queries for a given free-connex query* Q*, then for every* $R(\bar{x}) \in \mathcal{E}_1$*, it is either (1)* $R(\bar{x}) \in \mathcal{E}_2$*, or (2) there exists* $R'(\bar{x}') \in \mathcal{E}_2$*, such that* $R'(\bar{x}') \notin \mathcal{E}_1$*, and* $\bar{x} \backslash \mathcal{V}_\bullet = \bar{x}' \backslash \mathcal{V}_\bullet$.

PROOF. By definition, every relation that is not an ear must remain in the skeleton. Hence, any ear with a non-ear anchor must be removed. Now consider two residual queries Q_1 and Q_2. If $R(\bar{x}) \in \mathcal{E}_1$ but $R(\bar{x}) \notin \mathcal{E}_2$, then R must be an ear of the original query Q whose anchors are also ears of Q. Since R is removed in Q_2 but remains in Q_1, there must exist an anchor $R' \in \mathcal{E}_2$ of R such that $R' \notin \mathcal{E}_1$. By the same argument, because $R' \notin \mathcal{E}_1$, it must have an anchor $R'' \in \mathcal{E}_1$. The anchor relation is transitive: if R_i is an anchor of R_j and R_k is an anchor of R_i, then R_k is also an anchor of R_j. Therefore, we must have $R'' = R$; otherwise, by transitivity, R would also have an anchor in $\mathcal{E}_1$ different from itself, and thus should have been removed from $\mathcal{E}_1$, a contradiction. It follows that R and R' are anchors of each other, and hence $\bar{x} \backslash \mathcal{V}_\bullet^1 = \bar{x}' \backslash \mathcal{V}_\bullet^2$. □

Example 2.12. Considering Q_1 from Example 2.7. Relation $R_1, R_6, R_8, R_9, R_{10}$ are ears, with R_2 be the anchor of R_1, R_5 be the anchor of R_6, R_4 be the anchor of R_8 and R_9, R_7 be the anchor of R_{10}. Its residual query is

$$Q_1'(x_3, x_4, x_9) \leftarrow R_2(x_2, x_3), R_3(x_3, x_4, x_5, x_6), R_4(x_4, x_7), R_5(x_7, x_8), R_7(x_4, x_5, x_9).$$

Definition 2.13. A free-connex query Q is weak-q-hierarchical if Q is not q-hierarchical and its residual query Q_* is q-hierarchical.

Free-connex join tree height. The classification of free-connex queries is closely tied to the height of their free-connex join trees [23, 35].

Definition 2.14. The *height* of a generalized join tree $\mathcal{T}$ is the maximum count of input relations on any path from a leaf to the root without considering the generalized relations.

LEMMA 2.15 ([23], LEMMA 2.2 AND LEMMA 3.7). *Let* Q *be a free-connex conjunctive query.* Q *is q-hierarchical if and only if it admits a height* 1 *free-connex join tree, and* Q *is weak-q-hierarchical if and only if it admits a height* 2 *free-connex join tree.*

In this work, we refine the classification of conjunctive queries by extending the notions of height. When the query is free-connex, we show that a query has height k exactly when it admits a height k free-connex join tree but admits no height $k-1$ free-connex join tree. Under this view, the q-hierarchical and weak-q-hierarchical classes correspond precisely to the height 1 and height 2 queries, respectively.

2.2 k-Cycle Detection Problem

Finding cycles in graphs is a fundamental problem in algorithmic graph theory. Given a directed graph $G = (V, E)$ with n vertices and m edges, a k-cycle is a sequence of k distinct vertices $(v_1, v_2, \ldots, v_k)$ such that $(v_i, v_{i+1}) \in E$ for every $1 \leqslant i < k$, and $(v_k, v_1) \in E$. The k-cycle detection problem asks whether a given graph contains a k-cycle. This problem is NP-complete in general. For example, when $k = n$, it reduces to the Hamiltonian cycle problem. On the other hand, it is fixed-parameter tractable (FPT) when parameterized by k, meaning it is polynomial-time solvable for each fixed $k = O(1)$.

The k-cycle detection problem also serves as a core problem in fine-grained complexity. In particular, the special case $k = 3$ (triangle detection) has been extensively used to establish conditional lower bounds for database query processing [22, 31, 37]. However, beyond triangles, the general k-cycle problem is rarely discussed in the context of database queries. Lincoln and Vyas [29] have proved the lower bounds for detecting odd cycles and even cycles on sparse graphs based on the combinatorial k-clique conjecture.

Conjecture 2.16 (Combinatorial k-Clique Conjecture [1]). *There is no **combinatorial** algorithm that can detect a k-clique in a graph with n vertices and $O(n^2)$ edges in $O(n^{k-\epsilon})$ time, for any constant $\epsilon > 0$ and any integer $k \geqslant 3$*

Theorem 2.17 (Odd Cycle [29], Theorem 15). *Given a directed graph $G = (V, E)$ with $|V| = n$ and $|E| = m$, and $m = n^{1+2/(k-1)}$, no **combinatorial** algorithm can detect the existence of an odd k-cycle for any odd integer $k \geqslant 3$ in G in time $O(m^{2k/(k+1)-\epsilon})$ for any constant $\epsilon > 0$, assuming combinatorial k-clique conjecture.*

Theorem 2.18 (Even Cycle [29], Corollary 16). *Given a directed graph $G = (V, E)$ with $|V| = n$ and $|E| = m$, and $m = n^{k/(k-2)}$, no **combinatorial** algorithm can detect the existence of a even k-cycle for any even integer $k \geqslant 4$ in G in time $O(m^{(2k-2)/k-\epsilon})$ for any constant $\epsilon > 0$, assuming combinatorial k-clique conjecture.*

Note that both the odd and even cycle lower bounds assume sparse graphs. This restriction is necessary to prove lower bounds for database queries, as we measure complexity in terms of the input size $O(|D|)$ rather than the domain size. On the upper bound side, Alon et al. [7] gave an algorithm running for detecting a k-cycle with running time $O(m^{2-(1+\lceil k/2 \rceil^{-1})/(k+1)})$. Later, Dahlgaard et al. [15] improved the running time for finding even cycles in undirected graphs to $O(m^{2k/(k+2)})$. Furthermore, Lincoln et al. [30] provided a reduction that connects the complexity of detecting an undirected even cycle to that of detecting a directed odd cycle.

We will now briefly review relevant techniques from these works that are useful for our proofs.

Color-Coding. The primary challenge in detecting k-cycles is ensuring that all k vertices in the cycle are distinct. Color-coding is a randomized technique introduced by Alon et al. [6] that addresses this challenge. The core idea is to assign each vertex $v \in V$ a random color from a set of k colors. A *colorful* k-cycle, one in which all k vertices have distinct colors, is by definition a simple k-cycle. If the graph contains a k-cycle, then with some probability, the vertices of that cycle will be colorful under the random color assignment. By repeating this randomized process a sufficient number of times, any existing k-cycle can be found with high probability.

This randomized procedure can be derandomized using a (n, k)-perfect hash family [32].

Definition 2.19 ((n, k)-Perfect Hash Family). A (n, k)-perfect hash family is a set of hash functions H mapping a universe of n items to a range of k values such that for every subset S of items with $|S| = k$, there exists at least one hash function $f \in H$ that is injective on S (i.e., f maps the k distinct items in S to k distinct values).

Given such a family H, one can iterate over each $f \in H$ to color the vertices of G and then run the cycle-detection algorithm on the colored graph. The family H guarantees that for every k-cycle in G, at least one hash function $f \in H$ makes the cycle colorful. A construction for H exists such that $|H| = k^{O(1)} \log(n)$ [32], ensuring the derandomized algorithm remains FPT. In this paper, we treat color-coding and perfect hashing as black boxes that we can employ when needed.

2.3 (Generalized) Online Matrix-Vector Multiplications

Online matrix-vector multiplication (OMv) [3] is a fundamental problem for understanding the hardness of online problems. Formally, given an $n \times n$ Boolean matrix M and a sequence of n-dimensional vectors $\{v_1, \cdots, v_n\}$ is given in order. The task is to answer the matrix-vector multiplication Mv_i for every v_i before receiving the next vertex v_{i+1}. This leads to the well-known *Online Matrix-Vector Conjecture.*

Conjecture 2.20 (OMv Conjecture). *There is no algorithm for the OMv problem with a sequence of n vectors that runs in time $O(n^{3-\epsilon})$ for any constant $\epsilon > 0$.*

The conjectured hardness result implies lower bounds on a variety of dynamic problems, which also lead to fine-grained complexity results for answering conjunctive queries [9, 23, 27].

Jin and Xu [26] generalized the problem to a tensor version, and proposed the parameterized OuMv$_k$. Given a tensor $M \in [n]^k$, for every round i, a sequence of k n-dimensional vectors $\{u^{(1)}, u^{(2)}, \cdots, u^{(k)}\}$ is given, the task is to answer whether $u^{(1)} \times u^{(2)} \times \cdots \times u^{(k)}$ has a non-empty intersection with M, before receiving the next round of vectors.

Conjecture 2.21 (OuMv$_k$ Conjecture). *There is no algorithm for the OuMv$_k$ problem with n^γ queries with linear pre-processing time and $O(n^{\gamma+k-\epsilon})$ total query time for any $\gamma, \epsilon > 0$.*

In this paper, we apply the generalized OuMv$_k$ conjecture to provide tighter lower bounds for maintaining CQ under updates.

3 Maintenance Hardness when Scaling Out

From the examples of q-hierarchical and weak-q-hierarchical queries, we observe a pattern: when scaling out the query by adding additional relations, the hardness of maintaining the query increases. In this section, we investigate how maintenance hardness changes as we scale out.

3.1 Cycle Detection and Conjunctive Queries

We first start by drawing the connection between answering conjunctive queries with k cycle detection, which identifies three classes of hardness queries, Q_{2k-1}, Q_{2k}, and Q'_k.

Lemma 3.1. *Consider the query with arbitrary output attributes $\bar{y}$:*

$$Q_{2k-1}(\bar{y}) \leftarrow R_1(x_1), R_2(x_1, x_2), \cdots, R_{2k-2}(x_{2k-3}, x_{2k-2}), R_{2k-1}(x_{2k-2}).$$

If Q_{2k-1} can be maintained in amortized $O(|D|^{(k-1)/k-\epsilon})$ time while supporting $O(|D|^{1-\epsilon})$-delay enumeration for any constant $\epsilon > 0$, the combinatorial k-clique conjecture failed.

Proof. Given a graph $G = (V, E)$ with m edges and n vertices with $n = m^{(k-1)/k}$, we first find an $(n, 2k-1)$-perfect hash family H, and for every $f \in H$, we color every vertex using the hash function f and thus partition the graph into $2k-1$ disjoint subgraphs as $V = V^1 \cup V^2 \cup \cdots \cup V^{2k-1}$, where $V^x \leftarrow \{v \in V | f(v) = x\}$. Let P be the set of all $(2k-1, 2k-1)$-permutations, and let p be any of such permutation, for any f, p, we create the update sequence as follows:

(1) We assign $R_i = \{(u, v) \in E | f(u) = p(i-1), f(v) = p(i)\}$ and we insert all such edges into R_i for all $i \in [2, 2k-2]$
(2) For every $v \in V$ such that $f(v) = p(2k-1)$, we insert all $\{(u) | (v, u) \in E, f(u) = p(1)\}$ into R_1, and all $\{(u) | (u, v) \in E, f(u) = p(2k-2)\}$ into R_{2k-1}.
(3) We try to enumerate Q_{2k-1} on the current database instance. If there exists any results, then there exists a $2k-2$ path $x_1, x_2, \cdots, x_{2k-2}$, because the color coding guarantee that for any pair of x_i, x_j, $x_i \neq x_j$. In addition, we know that there exist two edges (v, x_1) and (x_{2k-2}, v) with v having a different color from any x_i. Therefore, $(v, x_1, \cdots, x_{2k-2})$ form a length $2k-1$-cycle, and we can terminate by reporting $2k-1$ cycle detected.
(4) Otherwise, we delete all tuples from R_1 and R_{2k-1}, and repeat (2) - (4) until no further v exists. If that is the case, we can conclude a $2k-1$ cycle does not exist for the given f, p.

The above update sequence inserts every edge into the database once, and deletes every edge from the database at most once, making a total of $O(m)$ updates; therefore, if such a query can be maintained in $O(|D|^{(k-1)/k-\epsilon})$ time, maintaining the entire update sequence takes a total running time of $O(m^{(k-1)/k-\epsilon} \times m) = O(m^{(2k-1)/k-\epsilon})$ time. By setting $k' = 2k-1$, $O(m^{(2k-1)/k-\epsilon}) = O(m^{2k'/(k'+1)-\epsilon})$. Meanwhile, a total of $O(n)$ enumeration operations will be issued during the above procedures. If the enumeration delay is $O(m^{1-\epsilon})$, then for the enumeration procedure, we would need to spend $O(m^{1-\epsilon} \times m^{(k-1)/k}) = O(m^{2k'/(k'+1)-\epsilon})$ time, summing up the two parts resulting in a total running time of $O(m^{2k'/(k'+1)-\epsilon})$ for the above procedures.

Note that we also need to verify all possible color coding and permutations to ensure we don't miss any cycles in the graph. The total running time for maintaining all $f \in H$ and $p \in P$ takes

$$O(\underbrace{\log m}_{\text{size of } H} \times \underbrace{(k')!}_{\text{Number of permutations}} \times \underbrace{m^{2k'/(k'+1)-\epsilon}}_{\text{Maintenance cost for each } p,f}) = \underbrace{O(m^{2k'/(k'+1)-\epsilon'})}_{k'=O(1),\log m \text{ suppressed in } \epsilon'}.$$

Once finished, we can determine if the given graph G contains a $k' = 2k-1$ cycle. Since the above procedure is combinatorial, it provides a combinatorial algorithm for solving the odd cycle detection problem. However, Theorem 2.17 suggests that there is no combinatorial algorithm that can detect such a cycle in time $O(m^{2k'/(k'+1)-\epsilon})$ unless the combinatorial k-clique conjecture failed. Thus, the existence of such a maintenance method also suggests the conjecture failed. □

Lemma 3.2. *Consider the query with arbitrary output attributes $\bar{y}$:*

$$Q_{2k}(\bar{y}) \leftarrow R_1(x_1), R_2(x_1, x_2), \cdots, R_{2k-1}(x_{2k-2}, x_{2k-1}), R_{2k}(x_{2k-1}).$$

If Q_{2k} can be maintained in amortized $O(|D|^{(k-1)/k-\epsilon})$ time while supporting $O(|D|^{1-\epsilon})$-delay enumeration for any constant $\epsilon > 0$, the combinatorial k-clique conjecture failed.

Proof. Given a graph $G = (V, E)$ with m edges, $n = m^{(k-1)/k}$ and $k' = 2k$, we construct the same update sequence as for the odd cycle case. Now the total update time would be $O(m^{(2k-1)/k-\epsilon}) = O(m^{(2k'-2)/k'-\epsilon})$ and the enumeration time would also be $O(m^{(2k-1)/k-\epsilon}) = O(m^{(2k'-2)/k'-\epsilon})$ for any color coding $f \in H$ and permutation p, which would return a combinatorial algorithm for detecting even cycle in time $O(m^{(2k'-2)/k'-\epsilon})$. Combining with Theorem 2.18, the existence of the maintenance algorithm suggests the combinatorial k-clique conjecture failed. □

Lemma 3.3. *Consider the query*

$$Q_k(x_k) \leftarrow R_1(x_1), R_2(x_1, x_2), \cdots, R_k(x_{k-1}, x_k).$$

If Q_k can be maintained in amortized $O(|D|^{(k-1)/k-\epsilon})$ time while supporting $O(|D|^{1/k-\epsilon})$ delay enumeration for any constant $\epsilon > 0$, the combinatorial k-clique conjecture failed.

Proof. Given a graph $G = (V, E)$ with m edges, we maintain two $Q_k(x_k)$ queries as

$$Q_k(x_k) \leftarrow R_1(x_1), R_2(x_1, x_2), \cdots, R_k(x_{k-1}, x_k)$$
$$Q'_k(x_k) \leftarrow R_{2k-1}(x_{2k-2}), R_{2k-2}(x_{2k-3}, x_{2k-2}), \cdots, R_k(x_{k-1}, x_k).$$

The update sequence is the same as the proof for Lemma 3.1; the only difference is in the enumeration procedure. Once we finish every round of updates, we enumerate both Q_k and Q'_k and check if $Q_k \cap Q'_k$ is empty. If it is not empty, then a $k' = 2k - 1$ cycle is detected. The output size is at most $n = m^{(k-1)/k}$. Therefore, the delay should be $O(|D|^{1/k-\epsilon})$ for some $\epsilon > 0$ to guarantee strictly smaller overhead than re-evaluation. When the amortized update time is $O(|D|^{(k-1)/k-\epsilon})$, the total running time becomes $O(m^{(2k-1)/k-\epsilon}) = O(m^{2k'/(k'+1)-\epsilon})$, and the above procedure provides a combinatorial algorithm for odd k' cycle detection problem . Combining with Theorem 2.17, the existence of the maintenance algorithm suggests the combinatorial k-clique conjecture failed. □

3.2 Chordless Path and Heights

The queries presented in Section 3.1 suggest that the hardness increases when more relations are involved. To quantify the influence, we introduce a structure parameter on the relational hypergraph, *height*, for any CQ Q, based on the chordless path on the relation hypergraph.

Given an undirected graph $G = (V, E)$, a *chordless path* [21] on G is a sequence of distinct vertices $P = \{v_1, \cdots, v_{n+1}\}$ and an edge sequence $P_e = \{e_1, \cdots, e_n\}$, such that $e_i = (v_i, v_{i+1})$ for every $i \in [1, n]$, and the subgraph $G_P \leftarrow (V_p = P, E_p = \{(v_i, v_j) \in E, v_i, v_j \in P\})$ induced by P is the path itself. In other words, for any $i, j \in [1, n+1]$ with $|i - j| > 1$, $(v_i, v_j) \notin E$.

While the path length in graphs is naturally defined by the number of edges (n), generalizing this to hypergraphs requires caution because a single hyperedge may contain an arbitrary number of vertices, and given the sequence P, the edge sequence P_e may not be unique. We propose the following generalization:

Definition 3.4 (Chordless Path on Hypergraph). Given a hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$, we define the chordless path on the hypergraph $\mathcal{H}$ to be a sequence of distinct vertices P, and an *edge sequence* P_e of the chordless path P to be a sequence of hyperedges. A chordless path P is

- a length 1 chordless path if $P \leftarrow \{\emptyset\}$, and $P_e \leftarrow \{e_1\}$ as an edge sequence of P with an arbitrary hyperedge e_1
- a length 2 chordless path if $P \leftarrow \{v_1\}$ and there exists two hyperedges e_1, e_2 with $v_1 \in e_1 \cap e_2$. All such pair of hyperedges form an edge sequence $P_e \leftarrow \{e_1, e_2\}$ of P
- a length k chordless path for $k \geqslant 3$ if $P \leftarrow \{v_1, \cdots, v_{k-1}\}$ such that
  (1) **(Adjacency)** For every $i \in [1, k-2]$, there exists a hyperedge $e_{i+1} \in E$ with both $v_i, v_{i+1} \in e$;
  (2) **(No shortcuts)** For any $i, j \in [1, k-1]$ with $|i - j| > 1$, there is no $e \in E$ with both $v_i, v_j \in e$;
  (3) **(Endpoints)** there exist $e_1, e_k \in E$, such that $v_1 \in e_1$ and $v_2 \notin e_1$, and $v_{k-1} \in e_k$ and $v_{k-2} \notin e_k$.
  Any $P_e \leftarrow \{e_1, \cdots, e_k\}$ that satisfies the above conditions is an edge sequence of P. Note that all hyperedges in an edge sequence are also distinct due to the "No shortcuts" requirement, and such an edge sequence may not be unique for a given hypergraph.

A chordless path P is *maximum* if there is no strictly longer chordless path P' with $P \subsetneq P'$. This definition aligns with the standard graph definition when all hyperedges are standard edges, and

the sequence P is defined as the path's internal vertices $v_2, ..., v_n$. The 'Adjacency' and 'No shortcuts' requirements are both naturally generalized from the standard graph. However, only one vertex is kept for the two 'endpoint' hyperedges due to the possible existence of unary hyperedges.

In addition, for non-full and non-Boolean queries, we further define the concept of a q-chordless path as a chordless path with only non-output attributes in P, while one 'endpoint' hyperedge contains output attributes. This generalizes the notions of q-cores and hook cores from [9, 23]. The length 1 chordless path is also a q-chordless path, with the edge sequence $\{e_1\}$ satisfying $e_1 \cap \bar{y} \neq \emptyset$ and $e_1 \backslash \bar{y} \neq \emptyset$, i.e., e_1 that contains both output and non-output attributes. A length k ($k \geqslant 2$) q-chordless path is a chordless path $P \leftarrow \{v_1, \cdots, v_{k-1}\}$ on $\mathcal{H}$, such that

(5) **(Non-output Vertices)** for all $x_i \in P$, $x_i \notin \bar{y}$, i.e. all attributes in P are not output attributes;
(6) **(Output Endpoint)** there exists a hyperedge $R(\bar{x})$ with $x_1 = \bar{x} \cap P$, and $\bar{x} \cap \bar{y} \neq \emptyset$, i.e., $\bar{x}$ contains both output attributes and an endpoint x_1 of P. The edge sequence of a q-chordless path must contain one of such hyperedges R.

Example 3.5. Consider Q_1 from Example 2.7. We can find a chordless path $P \leftarrow \{x_2, x_3, x_4, x_7, x_8\}$ of length 6 on its relational hypergraph (which is marked in red in Figure 2a) with $P_e \leftarrow \{R_1, R_2, R_3, R_4, R_5, R_6\}$. Note that x_1 cannot serve as one endpoint because there is no relation that contains only x_1 but no other $x \in P$. Additionally, we cannot include x_6 and x_5 in the chordless path between x_3 and x_4, as R_3 can serve as a shortcut.

Furthermore, we can find a length 3 q-chordless path $P \leftarrow \{x_7, x_8\}$ with edge sequence $P_3 \leftarrow \{R_4, R_5, R_6\}$, such that R_4 is the output endpoint with output attribute x_4.

With the chordless path, we can now define the height of a query:

Definition 3.6. Given a conjunctive query Q, let $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ be Q's relational hypergraph, query Q is a height k query if $\mathcal{H}$ contains no chordless path with length greater than $2k$, $\mathcal{H}$ contains no q-chordless path with length greater than k, and

- $\mathcal{H}$ contains a length $2k$ or length $2k-1$ chordless path; and/or
- $\mathcal{H}$ contains a length k q-chordless path.

Lemma 3.7. *A conjunctive query Q is height 1 if and only if it is q-hierarchical.*

When limiting the query to free-connex, we can relate the query height to the free-connex join tree height. The key idea is that removing all ear nodes from Q produces a residual query Q' whose height drops by one. Moreover, a free-connex join tree for Q' can be lifted to a free-connex join tree for Q by re-attaching the removed ears and increasing the tree height by one. Since a q-hierarchical (height 1) query admits a height 1 free-connex join tree, an induction on the height shows that every height k query admits a height k free-connex join tree.

On the other hand, every chordless path in the relational hypergraph of Q must be realized within any free-connex join tree, which rules out the existence of a height $k-1$ free-connex join tree for a height k query. Putting the two directions together yields the following lemma with the detail proof given in Appendix B.

Lemma 3.8. *Given a height k free-connex query Q, Q admits a free-connex join tree of height k, but not of height $k-1$.*

3.3 Proof of Theorem 1.1

After identifying a length $2k$ or $2k-1$ chordless path in height k query, or a length k q-chordless path in the non-full and non-boolean height k query, we can now draw a connection between

height k queries and cycle detection problem using the chordless path and its induced query, and prove Theorem 1.1 for the hardness of maintaining a height k query.

Given a height k query $Q = (\mathcal{V}, \mathcal{E}, \bar{y})$, we can then simulate Q_{2k-1}, Q_{2k}, or Q'_k. Without loss of generality, we use Q_{2k-1} as an example. Given an update sequence S for Q_{2k-1}, we first identify a length $2k-1$ chordless path $P = \{x_1, \cdots, x_{k-2}\}$ from Q, which corresponds to all attributes in Q_{2k-1} . For every $v \in \mathcal{V}$ such that $v \notin P$, we let their domain be a special value $\perp$. We can then create an equivalent update sequence S' for Q, such that for every $R(\bar{x}) \in \mathcal{E}$, we create the updates for R by the following rules:

- $\bar{x} \cap P = \varnothing$**.** We insert the tuple $(\perp, \cdots, \perp)$ for R at the beginning of all updates;
- $\bar{x} \cap P = \{x_i\}$**, where** x_i **is not an endpoint of** P**.** We insert the tuple $(u, \perp, \cdots, \perp)$ by assigning $x_i = u$ for all $u \in \mathrm{dom}(x_i)$ and all other attributes to $\perp$ at the beginning of all updates;
- $\bar{x} \cap P = \{x_i\}$**, where** x_i **is an endpoint of** P**.** When there is an update (u) to R_1 (or R_{k-1}) for Q_{2k-1}, we update $(u, \perp, \cdots, \perp)$ from R with $x_1 = u$ (or $x_{k-2} = u$) and all other attributes to $\perp$;
- $\bar{x} \cap P = \{x_i, x_{i+1}\}$. When there is an update (u, v) to R_i, we insert or delete $(u, v, \perp, \cdots, \perp)$ from R with $x_i = u$ and $x_{i+1} = v$ and all other attributes to $\perp$.

Due to the property of a chordless path, there is no R with more than two attributes in P, or $\bar{x} \cap P = \{x_i, x_j\}$ with $|j - i| > 1$. It is not hard to see that there is a one-to-one mapping for the query result Q and the Q_{2k-1}, and the result can be easily generalized to Q_{2k} and Q'_k. Therefore, given any update sequence S for Q_{2k-1}, Q_{2k} or Q'_k, we can create an equivalent update sequence S' on Q as long as Q has a length $2k-1$, $2k$ chordless path, or a length k q-chordless path, thus maintaining a height k query should be as hard as maintaining Q_{2k-1}, Q_{2k} or Q'_k.

4 Maintenance Hardness when Scaling Up

In Section 3, we analyzed the parameterized hardness of extending a query with additional relations. It is natural to ask whether two queries of the same height can still differ in maintenance difficulty due to the structure of their join attributes or unique output attributes[3]. To capture this effect, we introduce a second parameter, the *dimension* of a query. To define the dimension, we first formalize the notion of a *connected subset* on a query.

Definition 4.1 (Connected Subset). Given a conjunctive query $Q = (\mathcal{V}, \mathcal{E}, \bar{y})$, let $\mathcal{V}_c \subseteq \mathcal{V} \backslash \mathcal{V}_\bullet$ be a subset of join attributes. We define the induced active relations $\mathcal{E}_c = \{R(\bar{x}) \in \mathcal{E} \mid \bar{x} \cap \mathcal{V}_c \neq \varnothing\}$. To refine the definition, when $\mathcal{V}_c = \{\varnothing\}$, every $R_i(\bar{x}_i) \in \mathcal{E}$ can be the active relation $\mathcal{E}_c = \{R_i\}$.

We say $\mathcal{V}_c$ is a *connected subset* if the primal graph $G(\mathcal{V}_c)$ is connected. Let $\mathcal{V}(\mathcal{E}_c) = \cup_{R(\bar{x}) \in \mathcal{E}_c} \bar{x}$ denote the set of attributes appearing in $\mathcal{E}_c$. We define the set of *active attributes* as $\mathcal{V}_\star = \mathcal{V}(\mathcal{E}_c) \backslash \mathcal{V}_c$.

Definition 4.2 (Dimension). Let $Q = (\mathcal{V}, \mathcal{E}, \bar{y})$ be a conjunctive query. Let $\mathcal{V}_c \subseteq \mathcal{V}$ be a connected subset with induced relations $\mathcal{E}_c$. Let the remaining relations be $\mathcal{E}_r = \mathcal{E} \backslash \mathcal{E}_c$.

A subset of relations $\mathcal{N} \subseteq \mathcal{E}_r$ is called a *distinct neighbor set* if there exists an injective mapping $\pi : \mathcal{N} \to \mathcal{V}_\star$ such that for every $R(\bar{x}) \in \mathcal{N}$, $\pi(R) \in \bar{x} \cap \mathcal{V}_\star$. We refer to the image of this mapping, $\mathcal{V}_k(\mathcal{N}) = \{\pi(R) \mid R \in \mathcal{N}\}$, as a *key set*. Note that multiple key sets may exist for a fixed $\mathcal{N}$.

The *local dimension* of $\mathcal{V}_c$, denoted $\dim(\mathcal{V}_c)$, is defined as:

$$\dim(\mathcal{V}_c) = \max_{\mathcal{N} \subseteq \mathcal{E}_r,\ \mathcal{V}_k(\mathcal{N})} \begin{cases} |\mathcal{V}_k(\mathcal{N})| & \text{if } \mathcal{V}_c \cap \bar{y} \neq \varnothing \text{ or } \mathcal{V}_k(\mathcal{N}) \subseteq \bar{y}, \\ |\mathcal{V}_k(\mathcal{N}) \cup (\bar{y} \cap \mathcal{V}_\star)| & \text{otherwise.} \end{cases}$$

Intuitively, if $\mathcal{V}_c$ contains output attributes, or $\mathcal{V}_k(\mathcal{N})$ contains only output attributes, we only count the contribution of $\mathcal{V}_k(\mathcal{N})$. Otherwise, we additionally count any other active output attributes.

[3]For unique non-output attributes, we can project those attributes out without affecting the query result.

The *dimension* of the query Q is the maximum local dimension over all connected subsets:

$$\dim(Q) = \max_{\mathcal{V}_c \subseteq \mathcal{V}} \dim(\mathcal{V}_c).$$

We now establish a connection between the query dimension and the hierarchical structure.

Lemma 4.3. *A CQ is q-hierarchical if and only if* $\dim(\mathcal{V}_c) = 1$ *for all* $\mathcal{V}_c \subseteq \mathcal{V}$.

Combining with Lemma 3.7, we are able to show the equivalence between dimension-1 queries/height 1 queries/q-hierarchical queries and prove Proposition 1.3.

Example 4.4. Consider query Q_1 from Example 2.7. It is a dimension-4 query with $\dim(\{x_5\}) = 4$. Here, the connected set is $\mathcal{V}_c = \{x_5\}$ and the induced relations are $\mathcal{E}_c = \{R_3, R_7\}$. Figure 2c illustrates a distinct neighbor set $\mathcal{N} = \{R_2, R_4, R_8, R_{10}\}$ maximizing the dimension. The corresponding key set $V_k(\mathcal{N})$ includes $\{x_3, x_4, x_6, x_9\}$.

4.1 Star Queries

We link a dimension-d conjunctive query to the hardness of the Generalized Online Matrix-Vector Multiplication (OuMv$_k$) problem using the construct of *Star Queries*, where there exists a *center* relation that is connected to a set of *satellite* relations. All these satellite relations do not share any common attributes, whereas the center relation does share common attributes with each of them. Without loss of generality, we consider star queries in the following form:

Definition 4.5 (Star Queries). A query $Q = (\mathcal{V}, \mathcal{E}, \bar{y})$ is a *star query* if it is of the form:

$$Q(x_{d-j+1}, \ldots, x_d) \leftarrow R_1(x_1, \cdots, x_d), R_2(x_1), \ldots, R_{k+1}(x_k)$$

for any $0 \leqslant d - j \leqslant k \leqslant d$, where R_1 is the *center* relation, $R_2, \ldots, R_{k+1}$ are *satellite* relations, d is the dimension of the query and j is the number of output attributes. There are two special cases: $j = d, k = d$ (Full query) or $j = 0, k = d$ (Boolean query with no output attributes).

Star-shaped schemas are ubiquitous in data warehouse design: the center relation R_1 is the fact table, while all other satellite relations are dimension tables. In practice, these joins are usually between primary and foreign keys. Although we do not impose key constraints in the query definition, our lower bound instances respect them; meanwhile, prior work shows that, under restricted update sequences such as first-in-first-out, imposing key constraints can change the hardness of maintaining these sequences [36].

Any star query is a height 2 query: a height 2 free-connex join tree $\mathcal{T}$ can be obtained by taking R_1 as the root node and all satellites as leaf nodes attached to R_1. However, we can show that their lower bound can exceed the $O(\sqrt{|D|})$ bound established in Theorem 1.1:

Lemma 4.6. *Consider the star query:* $Q_d() \leftarrow R_1(x_1, \ldots, x_d), R_2(x_1), \ldots, R_{d+1}(x_d)$. *This query has a dimension of d. Unless the OuMv$_k$ Conjecture fails,* Q_d *cannot be maintained with amortized* $O(|D|^{(d-1)/d-\epsilon})$ *time and* $O(1)$*-delay enumeration for any constant* $\epsilon > 0$.

Proof. We reduce the OuMv$_k$ problem to maintaining Q_d. Let M be an d-dimensional Boolean tensor of size $n \times \cdots \times n$, and let $u_1, \ldots, u_d$ be a stream of n^γ vector sets. Since $n = |D|^{1/d}$, when $\gamma \geqslant d - 1$, the total input size is bounded by $(|D|^{1/d}) \times (|D|^{1/d})^\gamma = |D|^{(\gamma+1)/d}$. Given an OuMv$_k$ instance, we can solve the problem with the following reduction:

(1) Encode tensor M into R_1: insert tuple $(t_1, \ldots, t_d)$ into R_1 if and only if $M[t_1, \ldots, t_d] = 1$.
(2) For each round j of vector updates and queries:
   - For each $i \in [1, d]$, update R_{i+1} to represent vector $u_i^{(j)}$ by inserting (x_i) if $u_i^{(j)}[x_i] = 1$.

- Enumerate the result of Q_d. The result is non-empty if and only if M has a non-empty intersection with $u_1^{(j)} \times \cdots \times u_d^{(j)} = 1$.
- Delete tuples from $R_2, \ldots, R_{d+1}$ to prepare for round $j+1$.

If Q_d can be maintained in $O(|D|^{(d-1)/d-\epsilon})$ time per update, the total update time would be $O(|D|^{(d-1)/d+(\gamma+1)/d-\epsilon}) = O(|D|^{1+\gamma/d-\epsilon}) = O(n^{d+\gamma-d\epsilon})$, and the enumeration takes $O(1)$ per query, which thus solves OuMv$_k$ problem in time $O(n^{d+\gamma-\epsilon})$ and violates OuMv$_k$ Conjecture. □

Lemma 4.7. *Consider the star query $Q'_d(x_{d-j+1}, \cdots, x_d) \leftarrow R_1(x_1, x_2, \cdots, x_d), R_2(x_1), \cdots, R_{k+1}(x_k)$, for any $0 \leqslant d-j \leqslant k \leqslant d$. Unless OuMv$_k$ Conjecture failed, Q'_d cannot be maintained in $O(|D|^{(d-1)/d-\epsilon})$ time while supporting $O(|D|^{(d-j)/d-\epsilon})$-delay enumeration for any $\epsilon > 0$.*

Proof. We adapt the construction from the proof for Q_d. In this scenario, enumeration requires outputting slices of the tensor M. For every output tuple $(t_{d-j+1}, \ldots, t_d)$, a valid join result exists for the first k dimensions. Consequently, for each candidate output, we only need to verify if the intersection of the remaining dimensions ($k+1$ to d) exists in $u_{k+1} \times \cdots \times u_d$. We can construct a hash index on the Cartesian product $u_{k+1} \times \cdots \times u_d$ to perform this check in $O(1)$ time per entry. Computing this index takes $O(|D|^{(d-k)/d})$ time per round, and performing the distinct projection on Q'_d to the remaining dimensions raises an additional $O(|D|^{(k+j-d)/d})$ delay.

If the amortized update time were $O(|D|^{(d-1)/d-\epsilon})$ and enumeration delay for enumerating $\pi_{x_{k+1},\cdots,x_d} Q'_d$ were $O(|D|^{(d-j)/d-\epsilon} \times |D|^{(k+j-d)/d}) = O(|D|^{k/d-\epsilon})$, since there are total $O(|D|^{(d-k)/d})$ output per round, the total running time would be $O(n^{\gamma+d-1-\epsilon'})$, which again violates the conjecture. □

4.2 Proof of Theorem 1.2

We prove the theorem via a reduction from the hardness of the Star Queries (Q_d and Q'_d) to an arbitrary query Q with $\dim(Q) = d$. Let $\mathcal{V}_c$ be the connected subset, with $\mathcal{E}_c$ the induced active relations, $\mathcal{N}$ the distinct neighbor set, and $\mathcal{V}_k(\mathcal{N})$ the key set that attains the maximum dimension (i.e., $\dim(\mathcal{V}_c) = d$). We construct an instance of Q to simulate the hard query Q_d (or Q'_d) as follows.

Let M be the input tensor and $u_1, \ldots, u_d$ be the input vectors for the OuMv$_k$ problem instance associated with Q_d. We map this instance to Q through the following steps:

(1) We map the d dimensions to the attributes in Q. If $\mathcal{V}_c \cap \bar{y} \neq \varnothing$ or $\mathcal{V}_k(\mathcal{N}) \subseteq \bar{y}$, we simply map the dimensions $x_1, \ldots, x_d$ to the attributes in $\mathcal{V}_k(\mathcal{N})$. Otherwise, we map the first $x_1, \cdots, x_k$ to the attributes in $\mathcal{V}_k(\mathcal{N})$, and the remaining dimensions $x_{k+1}, \ldots, x_d$ to $(\bar{y} \cap \mathcal{V}_\star) \backslash \mathcal{V}_k(\mathcal{N})$. All attributes involved in this mapping are assigned the domain $[n]$.

(2) The attributes in the connected subset $\mathcal{V}_c$ serve as "glue" to reconstruct the tensor M. We assign their domain to be $[N]$, where $N = n^d$. Let $\mathcal{V}'$ be the set of all active attributes in the reduction: $\mathcal{V}' = \mathcal{V}_c \cup \mathcal{V}_k(\mathcal{N})$ or $\mathcal{V}' = \mathcal{V}_c \cup \mathcal{V}_k(\mathcal{N}) \cup (\bar{y} \cap \mathcal{V}_\star)$ depends on $\mathcal{V}_c$ and $\mathcal{V}_k(\mathcal{N})$. Any attributes $v \in \mathcal{V} \backslash \mathcal{V}'$ are fixed to a constant value $\perp$.

We define the update sequence for $R(\bar{x})$ for all $\bar{x} \cap \mathcal{V}' \neq \varnothing$ as follows:

(3) **Tensor Encoding:** Consider relations where $|\bar{x} \cap \mathcal{V}'| \geqslant 2$. These relations form the core structure. We index the non-zero entries of the tensor M with a unique identifier $ID \in [1, N]$. For each relation $R(\bar{x})$ with $\mathcal{V}_c \cap \bar{x} \neq \varnothing$, we project (without removing the duplicates) the tensor M onto the attributes $\bar{x} \backslash \mathcal{V}_c$, and set those variables in $\mathcal{V}_c$ to the corresponding tensor entry ID. If $R(\bar{x})$ contains *only* variables from $\mathcal{V}_c$ (i.e., $\bar{x} \cap \mathcal{V}' \subseteq \mathcal{V}_c$), we simply insert tuples $(ID, \ldots, ID)$ for all active IDs. This ensures the connectivity of $\mathcal{V}_c$ propagates the synchronization ID across the query. If $R(\bar{x})$ contains no variables from $\mathcal{V}_c$, we insert the distinct projection of the tensor M for attributes $\bar{x} \cap \mathcal{V}'$ without constraining the query outputs.

(4) **Neutralizing Unused Relations.** For any relation $R(\bar{x})$ with $\bar{x} \cap \mathcal{V}' = \varnothing$, we populate it with the single tuple $(\perp, \ldots, \perp)$ at the beginning of the stream. This ensures these relations do not restrict the join result.
(5) **Vector Updates:** After (3) and (4), we start to update vectors. Consider relations where $\bar{x} \cap \mathcal{V}' = \{v\}$ for some $v \in \mathcal{V}_k(\mathcal{N})$. By the definition of the Distinct Neighbor Set, such relations exist and uniquely correspond to specific dimensions. We map the update stream of vector u_i directly to this relation. Inserting a value c into this relation corresponds to setting $u_i[c] = 1$.

Correctness. The construction ensures that a join result exists if and only if there is a tuple identifier ID (representing a non-zero entry in M) such that all its coordinate values are present in the corresponding satellite relations (representing the vectors u_i being 1). This exactly reconstructs the condition $M(x_1, \ldots, x_d) \wedge \bigwedge u_i(x_i)$. The size of the database is dominated by M and the vector updates, preserving the $O(|D|^{(d-1)/d-\epsilon})$ lower bound.

5 Maintaining Star queries with Parameterized Complexity

In Sections 3 and 4, we established that a query Q with height k and dimension d cannot be maintained better than amortized $O\big(|D|^{1-1/\max\{k,d\}-\epsilon}\big)$ time. In this section, we provide a matching upper bound for the class of Star Queries. We introduce a data-dependent parameter, the *Generalized H-index*, and design an algorithm with amortized update time $O(h(D)^{\dim(Q)-1})$. Since $h(D)$ is naturally bounded by $|D|^{1/\dim(Q)}$, this proves the optimality of our lower bound for these queries.

5.1 Generalized H-index

The classical H-index is a widely used metric that measures both the productivity and the citation impact of a scholar's publications. A scholar has an H-index of 10 if they have at least 10 publications, each with at least 10 citations, and they do not have 11 publications with at least 11 citations. The same idea has been used in parameterized graph algorithms [17, 18], where one partitions the graph into a dense part (induced by the vertices with many neighbors) and a sparse remainder. However, the standard H-index is defined for binary relations such as graphs. We therefore introduce a generalized version to measure the density of a relation in a star query.

Definition 5.1 (Generalized H-index). Consider a star query Q with center relation $R_1(x_1, \ldots, x_d)$ and dimension $d = \dim(Q)$. For any attribute $x_i \in \{x_1, \ldots, x_d\}$, let $\deg_{R_1}(v)$ denote the frequency of a value v in the projection $\pi_{x_i} R_1$. We define the H-index of a specific attribute x_i, denoted $h_i(D)$, as the largest integer such that there are at least $h_i(D)$ distinct values in $\pi_{x_i} R_1$ with frequency at least $h_i(D)^{d-1}$. Formally:

$$h_i(D) = \max\left\{k \in \mathbb{N} \;\middle|\; \left|\left\{v \in \pi_{x_i} R_1 : \deg_{R_1}(v) \geqslant k^{d-1}\right\}\right| \geqslant k\right\}.$$

The *Generalized H-index* of the query Q over database D is defined as $h(D) = \max_{i=1}^{d} h_i(D)$.

Proposition 5.2. *For any database instance D, the generalized H-index satisfies $h(D) \leqslant |D|^{1/d}$.*

Proof. By definition, there exists an attribute x_i with at least $h(D)$ distinct values, each appearing in at least $h(D)^{d-1}$ tuples in R_1. These sets of tuples are disjoint (grouped by x_i), so $|R_1| \geqslant h(D) \times h(D)^{d-1} = h(D)^d$. Thus, $h(D) \leqslant |R_1|^{1/d} \leqslant |D|^{1/d}$. □

5.2 Algorithm Designs

Our maintenance strategy relies on a *Heavy-Light Decomposition* of the data space. Let $h(D)$ be the current generalized H-index. We classify values in the domain of each attribute x_i as *Heavy* if their degree exceeds $\tau_i = (h_i(D) + 1)^{d-1}$, and *Light* otherwise.

Partitioning. For each attribute x_i, we define the set of heavy keys $K_i^H = \{v \in \pi_{x_i} R_1 \mid \deg_{R_1}(v) \geqslant \tau_i\}$. We partition the center relation R_1 into 2^d logical partitions based on the heaviness of its attribute values. Each partition is indexed by a bit-vector $\mathbf{b} \in \{0,1\}^d$:

$$R_1^{(\mathbf{b})} = \left\{ t \in R_1 \;\middle|\; \forall i \in [1,d],\ \begin{cases} t[x_i] \in K_i^H & \text{if } \mathbf{b}[i] = 1 \text{ (Heavy)} \\ t[x_i] \notin K_i^H & \text{if } \mathbf{b}[i] = 0 \text{ (Light)} \end{cases} \right\}$$

Similarly, each satellite relation R_i is partitioned into $R_i^{(1)}$ and $R_i^{(0)}$. In addition, for x_{k+1} to x_d, let $k+1 \leqslant i \leqslant d$ we conceptually maintain a vector $R_{i+1}^{(1)} \leftarrow \pi_{x_i} R_1^{\mathbf{b}[i]=1}$ for those heavy value.

View Maintenance. We maintain the query Q as a union of 2^d subqueries $Q^{(\mathbf{b})}$, corresponding to $R_1^{(\mathbf{b})}$. For a fixed $\mathbf{b}$, let $I_H = \{i \mid \mathbf{b}[i] = 1\}$ be the indices of heavy attributes. We maintain:

(1) **Satellite Projections:** For every $R_i^{(\mathbf{b}[i])}$, we maintain its projection: $V_{R_i}^{(\mathbf{b}[i])} = \pi_{x_{i-1}} R_i^{(\mathbf{b}[i])}$.

(2) **Light-Filtered Core (V_S):** For the center relation $R_1^{(\mathbf{b})}$, we actively filter it by the *light* satellites. We maintain the semi-join view: $V_S^{(\mathbf{b})} = R_1^{(\mathbf{b})} \ltimes \left(\ltimes_{i \notin I_H} V_{R_{i+1}}^{(0)} \right)$.
Since light keys have a low degree, updates to light satellites are cheap to propagate.

(3) **Heavy Core (V_C):** We further project the light-filtered core onto the *heavy* attributes and filter by the Cartesian product of *heavy* satellites: $V_C^{(\mathbf{b})} = \left(\pi_{\{x_i\}_{i \in I_H}} V_S^{(\mathbf{b})} \right) \ltimes \left(\times_{i \in I_H} V_{R_{i+1}}^{(1)} \right)$.
V_C contains only tuples composed entirely of heavy keys that have survived all satellite filters.

Enumeration. To enumerate the result of Q, we iterate through all $\mathbf{b} \in \{0,1\}^d$. For each $\mathbf{b}$, we 1. Enumerate tuples $t_{core} \in V_C^{(\mathbf{b})}$. 2. Use t_{core} to probe the index of $V_S^{(\mathbf{b})}$ to find matching tuples $t_{full} \in R_1$ that survived the light filters. 3. For each t_{full}, extend it to the final result by joining with the satellites (using standard constant delay lookups). This process guarantees $O(1)$-delay enumeration because the query is free-connex [24, 35]. We then perform a distinct project on the output attributes $\bar{y}$, which raises an additional $O(|D|^{(d-j)/d})$ delay.

Rebalance. So far, we have assumed that the heavy/light partition is fixed, and $h(D)$ is also fixed. In the dynamic setting, however, the classification of a key as heavy or light, and the generalized H-index $h(D)$, may change over time as updates arrive. We handle these changes with a standard rebalancing technique similar to the textbook method for dynamic tables [14].

(1) **Key Reclassification:** We promote a key x_i to heavy only when its degree exceeds $2\tau_i$ and demote it only when it drops below $\frac{1}{2}\tau_i$. This ensures that between any two status changes of a key, $\Omega(h_i(D)^{d-1})$ updates involving that key must have occurred.
To perform such a rebalance, we first delete all associated tuples from R_1, and then all associated tuples from R_{i+1}, and reinsert them back to R_{i+1} and then R_1 with the updated label. These updates pay for the $O(h_i(D)^{d-1})$ work required to move tuples between partitions, as the frequency of such a key would be $O(h_i(D)^{d-1})$, and incur an amortized $O(1)$ cost per update.

(2) **Global Parameter Update:** If the number of heavy keys in any dimension exceeds $2h(D)$, or if $h(D)$ decreases by half compared with the last parameter update, we rebuild the data structures with a new $h(D)$ that can be tracked in real time.
When a global rebalance happens, at most $O(h(D))$ keys need to be moved between the heavy and light sets, and in the worst case, it costs $O(|D|) = O(h(D)^d)$. However, to reach a state requiring a rebuild, $\Omega(h(D) \cdot h(D)^{d-1}) = \Omega(h(D)^d)$ updates must have occurred, e.g., at least $h(D)$ keys are moved from light to heavy, with every key's frequency changes from at most $h(D)^{d-1}$ in previous global rebalance to $2h(D)^{d-1}$; or at least $0.5h(D)$ heavy keys are moved

from heavy to light, with frequency changes from at least $h(D)^{d-1}$ in previous global rebalance to $0.5h(D)^{d-1}$. Distributing the cost over these updates yields an amortized cost of $O(1)$.

5.3 Complexity Analysis

It is not hard to see that the space requirement for the algorithm is linear to $O(|D|)$; therefore, to complete the proof of Theorem 1.4, it suffices to analyze the amortized update complexity. The update cost consists of three parts: maintenance cost, rebalancing cost, and the cost of tracking $h(D)$. In these three parts, we have also demonstrated that the rebalancing step incurs only an amortized constant cost per update; we now focus on the remaining two parts.

Maintenance cost. We analyze the cost of a single tuple update, assuming $h(D)$ is constant.

- **Update to Center R_1:** Inserting a tuple t into R_1 affects exactly one partition $R_1^{(\mathbf{b})}$. Checking the semi-joins for V_S and updating V_C takes $O(1)$ time per update with proper hash indices.
- **Update to Light Satellite $R_i^{(0)}$:** A tuple inserted here has a join key k with $\deg_{R_1}(k) < \tau_i$. This update may trigger insertions into $V_S^{(\mathbf{b})}$. Since the degree is bounded by $\tau_i = O(h_i(D)^{d-1})$, at most $O(h_i(D)^{d-1})$ tuples in R_1 are activated. Propagating these to V_C takes an additional constant time per R_1 tuple, making the total cost $O(h_i(D)^{d-1})$.
- **Update to Heavy Satellite $R_i^{(1)}$:** The join key k is heavy. This does not affect V_S but may update the Cartesian product $\times_{i\in I_H} V_{R_{i+1}}^{(1)}$. Since there are at most $h_i(D)$ heavy values per dimension, the number of the affected tuples in the Cartesian product is effectively bounded by $O(h(D)^{|I_H|-1})$, thus the update on V_C is upper bounded by $O(h(D)^{|I_H|-1}) \leqslant O(h(D)^{d-1})$.

Therefore, with a fixed heavy/light classification, the update cost is $O(h(D)^{d-1})$.

Tracking $h(D)$. One additional requirement for the maintenance algorithm is to track the change of $h_i(D)$ for all dimensions i and thus $h(D)$. To do that, for every dimension i, we maintain two group-by-count queries $V_{\mathsf{cnt}_i} = \pi_{x_i,\mathsf{count}(*)\text{ as }\mathsf{deg}} R_1$ and $V_{\mathsf{feq}_i} = \pi_{\mathsf{deg},\mathsf{count}(*)\text{ as }\mathsf{feq}} V_{\mathsf{cnt}_i}$. These two queries can be maintained in $O(1)$ time [13] under single-tuple update on R_1, and we can obtain the real-time h_i by maintaining the result $h_i = \max\left\{\mathsf{cnt} \,\middle|\, \sum_{i=\mathsf{cnt}}^{|D|^{1/\dim(Q)}} \left(V_{\mathsf{feq}_i}[\mathsf{cnt}].\mathsf{feq}\right) \geqslant \mathsf{cnt}^{\dim(Q)-1}\right\}$, which can also be maintained in $O(1)$ time by keeping a frequency histogram.

Combining all parts, the amortized update time per tuple update is $O\left(h(D)^{\dim(Q)-1}\right)$. Since we always have $h(D) \leqslant |D|^{1/\dim(Q)}$, this matches the lower bound from Theorem 1.2 for all star queries, and thus proves tightness for this class.

6 Additional Discussions and Future Works

In this work, we introduce two structural parameters, *height* and *dimension*, to establish parameterized lower bounds for maintaining conjunctive queries under updates. We shall note that *height* assumes the combinatorial k-clique conjecture, which only rules out the existence of a combinatorial algorithm. Although the notion of *combinatorial algorithm* is not formally defined, it is widely used across different research communities, including in database theory [16, 19, 31], due to the practical efficiency, elegance, and generalizability of combinatorial algorithms [2]. This also immediately raises an interesting research problem for both upper-bound and lower-bound designs: can we apply fast matrix multiplication to accelerate dynamic query evaluation?

Another critical question is whether a maintenance algorithm can be designed to match these lower bounds. To complement our lower bounds, we investigated the maintenance of *star queries*, a central subclass of analytical workloads. We established a parameterized upper bound based

on the *Generalized H-index* $H(D)$. It generalizes prior trade-off algorithms [27, 28] to an instance-dependent setting, demonstrating that efficiency often depends on data skew rather than worst-case size. The generalization of such techniques would benefit both theory and system research. On the other hand, it remains an open question to achieve a trade-off between the enumeration delay and maintenance cost, especially for queries with $\omega(Q) > 1$.

Furthermore, closing the gap for general queries remains a significant challenge. Specifically, new algorithmic designs are required to handle *Loomis-Whitney*-like queries [5, 27], which contain nested join keys that complicate the new maintenance techniques for star queries. Future avenues for research include extending both upper bounds and lower bounds to general scenarios, such as comparisons [25, 37], unions [11], and queries over specific update patterns [5, 23].

A Proof of Lemma 3.7

If Direction: Let $Q = (\mathcal{V}, \mathcal{E}, \bar{y})$ be a height 1 query. If Q is not q-hierarchical, then (1) there exists $x_1, x_2 \in \mathcal{V}$ such that $\mathcal{E}[x_1] \cap \mathcal{E}[x_2] \neq \emptyset$ and $\mathcal{E}[x_1] \backslash \mathcal{E}[x_2] \neq \emptyset$, $\mathcal{E}[x_2] \backslash \mathcal{E}[x_1] \neq \emptyset$; or (2) $x_1 \in \bar{y}$, $x_2 \notin \bar{y}$, while $\mathcal{E}[x_1] \subsetneq \mathcal{E}[x_2]$. However, for (1), let $e_1 \in \mathcal{E}[x_1] \cap \mathcal{E}[x_2]$, $e_2 \in \mathcal{E}[x_1] \backslash \mathcal{E}[x_2]$, $e_3 \in \mathcal{E}[x_2] \backslash \mathcal{E}[x_1]$, $P \leftarrow \{x_1, x_2\}$ forms a length 3 chordless path with $P_e \leftarrow \{e_2, e_1, e_3\}$ be one of its edge sequence, contradicting that Q is height 1; and for (2), let $e_1 \in \mathcal{E}[x_1] \cap \mathcal{E}[x_2]$, $e_2 \in \mathcal{E}[x_1] \backslash \mathcal{E}[x_2]$, then $P \leftarrow \{x_2\}$ is a length 2 q-chordless path, with $P_e \leftarrow \{e_1, e_2\}$ be an edge sequence with e_1 contains output attributes x_2, which also contradicts to Q is height 1. Therefore, the assumption that Q is not q-hierarchical is incorrect.

Only-if Direction: Similarly, let Q be a q-hierarchical query; we cannot find a length $\geqslant 3$ chordless path, otherwise we can find a pair of attributes x_1, x_2 such that $\mathcal{E}[x_1] \cap \mathcal{E}[x_2] \neq \emptyset$ and $\mathcal{E}[x_1] \backslash \mathcal{E}[x_2] \neq \emptyset$, $\mathcal{E}[x_2] \backslash \mathcal{E}[x_1] \neq \emptyset$. Similarly, we cannot find a length $\geqslant 2$ q-chordless path; otherwise, we can find attributes x_1, x_2 such that $x_1 \in \bar{y}$, $x_2 \notin \bar{y}$, while $\mathcal{E}[x_1] \subsetneq \mathcal{E}[x_2]$. Both cases violate the fact that Q is q-hierarchical; therefore, the query must have height 1.

B Proof of Lemma 3.8

To prove Lemma 3.8, we first demonstrate the recursive properties of the height definition.

Lemma B.1. *A free-connex query Q is a height k query for $k > 1$ if and only if its residual query Q' is a height $k-1$ query.*

Proof. The lemma is immediate for the base case when $k = 2$, i.e., the weak-q-hierarchical queries, as their residual queries are q-hierarchical. We now consider the general case $k > 2$. By definition of residual query and ears, given a free-connex join tree $\mathcal{T}$, each leaf node corresponds to an ear, and its parent node is the anchor of that ear. Therefore, we deduce two important facts: (1) if Q has a join tree of height k, then its residual query Q' has a join tree of height at most $k-1$. This is because removing all leaf nodes (ears) from $\mathcal{T}$ yields a join tree with height $k-1$. If necessary, further removing any remaining ears from the resulting tree can only decrease the height further. (2) Consequently, if the residual query Q' admits a free-connex join tree $\mathcal{T}'$ of height $k-1$, then we can construct a join tree for Q of height at most k by attaching each removed ear R as a child of its anchor R' in $\mathcal{T}'$. The resulting structure remains a valid free-connex join tree, satisfying the Connect, Cover, and Connex conditions without violating the Guard or Above conditions. Since adding these ears introduces only new leaves, the tree's height increases by 1; therefore, the reconstructed join tree for Q has height at most k.

We first prove the if direction. Suppose the residual query Q' is a height $k-1$ query. Assume Q admits a free-connex join tree $\mathcal{T}$ of height $\leqslant k-1$. Removing all ears from $\mathcal{T}$ yields a join tree for Q' of height at most $k-2$, by observation (1) above. This contradicts the assumption that Q' is a height $k-1$ query, since Q' would then have a join tree of height smaller than $k-1$. Therefore, Q

cannot have any free-connex join tree of height less than k. By observation (2), on the other hand, we know how to construct a height-k join tree for Q. Taken together, the minimum possible join tree height for Q is k. In other words, Q is a height k query.

For the "only-if" direction, assume Q is a height k query. By definition, every free-connex join tree of Q has height at least k, and there is some join tree $\mathcal{T}$ with height k. Remove all the leaf nodes from this $\mathcal{T}$ to obtain a join tree $\mathcal{T}'$ with height $k-1$, and further removing ears from $\mathcal{T}'$ will only decrease the height, making Q''s height at most $k-1$. Now, suppose for contradiction that Q' is not a height $k-1$ query. Then there exists a free-connex join tree for Q' of height at most $k-2$. By attaching the previously removed ears to this shorter join tree using the construction from observation (2), we could obtain a join tree for Q of height at most $k-1$. This would contradict the assumption that Q's height is k. Therefore, Q' must have minimal join tree height $k-1$, meaning Q' is a height $k-1$ query. □

Since every height 1 query is q-hierarchical and with a join tree of height 1 (Lemma 3.7), and let Q' be a residual query of Q by removing all ears from the query, and T' be a height k' free-connex join tree for Q', a height $k'+1$ free-connex join tree for Q can be constructed by adding each ear node as a leaf child under its anchor node and increasing the height of tree by 1. Therefore, we can construct a height k free-connex join tree for a height k query Q by the following steps: (1) we recursively removing all ear nodes and get residual queries $Q_2, \cdots, Q_k$, where Q_k has a height of 1; (2) we find the height 1 free-connex join tree T_k for Q_k; (3) for $Q_{k-1}, \cdots, Q_2, Q$, we add all ears back to T_k and obtain $T_{k-1}, \cdots, T_2, T$ with height $2, \cdots, k-1, k$.

In addition, to show that Q admits no height $k-1$ free-connex join tree, we prove that every chordless path must be realized within any free-connex join tree:

Lemma B.2. *Given a CQ $Q = (\mathcal{V}, \mathcal{E}, \bar{y})$, let $P \leftarrow \{v_1, \cdots, v_{k-1}\}$ be a length k chordless path on Q's relational hypergraph $\mathcal{H}$, for any join tree $\mathcal{T}$ of Q, let n_1 be a node with $v_1 \in n_1$ while $v_i \notin n_1$ for all $i \in [2, k-1]$, and n_l be a node with $v_{k-1} \in n_l$ while $v_i \notin n_l$ for all $i \in [1, k-2]$, then the path $P_{\mathcal{T}}$ between n_1 and n_l on the join tree includes all attributes in P. In addition, such a path has a length of at least k.*

Proof of Lemma B.2. We start by considering a length-2 path v_1, v_2. Because of the definition, we know that there exists e_1, e_2, e_3 such that $v_1 \in e_1$ and $v_2 \notin e_1$, $v_1, v_2 \in e_2$, and $v_2 \in e_3$ and $v_1 \notin e_3$. Because the Connect property of a join tree, let T_1 be the subtree induced by v_1, and T_2 be induced by v_2, since e_2 belongs to both T_1 and T_2, therefore, all nodes with v_1 or v_2 are also connected on $\mathcal{T}$. We let e_1 be n_1 and e_3 be n_2; there must exist a node n' on the path between n_1 and n_2 with $v_1, v_2 \in n'$, otherwise the path contains an addition node v_3 in between v_1 and v_2, combining the connect condition with e_2 creates a cycle, which violates the fact that $\mathcal{T}$ is a tree.

On the other hand, assuming for a length $k-1$ chordless path, the lemma holds, then for a length k chordless path, we can adopt a similar idea as for the length-2 case to show the connectness of $v_1, \cdots, v_{k-1}$-induced subtree. Therefore, let n_1 and n_k be two endpoints, the path between n_1 and n_k must cover $v_1, \cdots, v_{k-1}$, otherwise a cycle would be detected on the tree. By applying the induction, we can prove the lemma.

Because there is no shortcut, covering $v_1, \cdots, v_{k-1}$ requires at least $k-2$ nodes, and there are two endpoints for v_1 and v_{k-1}, making such a path have a length of at least k. □

Lemma B.3. *Given a CQ $Q = (\mathcal{V}, \mathcal{E}, \bar{y})$, let $P \leftarrow \{v_1, \cdots, v_{k-1}\}$ be a length k chordless path on Q's relational hypergraph $\mathcal{H}$, for any join tree $\mathcal{T}$ of Q, $\mathcal{T}$ has a height at least $\lceil \frac{k}{2} \rceil$.*

Proof of Lemma B.3. We can find a corresponding path $P_{\mathcal{T}}$ on any join tree $\mathcal{T}$ for the chordless path P, which has a length of at least k. Based on the definition of height, we can minimize the

height of the tree by placing the two endpoints to be leaf nodes of $\mathcal{T}$ and requiring the path to pass the root node. Assuming the distance between the two leaf nodes and the root node is balanced, the height would be at least $\lceil \frac{k}{2} \rceil$. □

Therefore, for a Boolean or full acyclic query of height k, all join trees for that query have a height of at least k. When considering non-full and non-Boolean free-connex queries, we can then demonstrate the connection between a q-chordless path and the height of a join tree.

Lemma B.4. *Given a free-connex $Q = (\mathcal{V}, \mathcal{E}, \bar{y})$, let $P \leftarrow \{v_1, \cdots, v_{k-1}\}$ be a length k q-chordless path on Q's relational hypergraph $\mathcal{H}$, for any free-connex join tree $\mathcal{T}$ of Q, $\mathcal{T}$ has a height at least k.*

Proof of Lemma B.4. Hu and Wang [23] shows that for a height 2 query with $\bar{y}$ to be non-empty or non-full, it is possible there is no length 3 or length 4 path, but contains a length 2 q-chordless path $P = \{x_1\}$ with $x_1 \notin \bar{y}$, $R_1(\bar{x}_1), R_2(\bar{x}_2)$ be two relations with $x_1 \in \bar{x}_1 \cap \bar{x}_2$, and $\exists x \in \bar{x}_1 \cap \bar{y}$. In that case, R_1 must be the parent node of R_2 on any free-connex join tree, otherwise violating the Connex Property of the join tree. On the other hand, if for a height k query Q, its residual query Q' satisfies $\mathcal{H}[Q']$ contains such a length $k-1$ q-chordless path, by definition of q-chordless path, let $R_{k-1}(\bar{x}_{k-1})$ be the relation with only x_{k-2} in the residual query Q', since R_{k-1} is not an ear in Q and R_{k-1} contains only non-output attributes, there exists an ear $R_k(\bar{x}_k)$ in Q with R_{k-1} be its anchor, and $\bar{x}_k \cap \bar{x}_{k-1} \cap P = \emptyset$ and $\bar{x}_k \cap \bar{x}_{k-1} \neq \emptyset$. Therefore, let $x_k \in \bar{x}_k \cap \bar{x}_{k-1}$ be any join attributes between R_{k-1} and R_k, we can find a length k q-chordless path with $P' \leftarrow P \cup \{x_k\}$, which completes the proof. □

Combining the two lemmas, we can show that a height k query cannot admit a height $k-1$ free-connex join tree.

C Proof of Lemma 4.3

For a q-hierarchical query Q, we can find that there is no $\mathcal{V}_c$ with $R_1, R_2 \in \mathcal{E}_c$ such that $\bar{x}_1 \cap \bar{x}_2 \notin \mathcal{V}_c$; otherwise the query won't be q-hierarchical.

Let $Q = (\mathcal{V}, \mathcal{E}, \bar{y})$ be a CQ, if any $\mathcal{V}_c \in \mathcal{V}$ has $\dim(\mathcal{V}_c) > 1$, consider the $\mathcal{N}$ and $\mathcal{V}_k(\mathcal{N})$ that would maximize the dimension of $\mathcal{V}_c$, if (1) $\mathcal{V}_c \cap \bar{y} \neq \emptyset$ or $\mathcal{V}_k(\mathcal{N}) \subseteq \bar{y}$, there exists two $R', R'' \in \mathcal{N}$, such that $x_1 \in \bar{x}'$ and $x_2 \in \bar{x}''$. If there exists $R \in \mathcal{E}_c$ with $x_1, x_2 \in \bar{x}$, then $\mathcal{E}[x_1] \cap \mathcal{E}[x_2] \neq \emptyset$, while $\mathcal{E}[x_1] \nsubseteq \mathcal{E}[x_2]$ and $\mathcal{E}[x_2] \nsubseteq \mathcal{E}[x_1]$, making the query non-q-hierarchical; otherwise, let $R_1, R_2 \in \mathcal{E}_c$ with $x_1 \in \bar{x}_1$ and $x_2 \in \bar{x}_2$, there must be $x_3 \in \mathcal{V}_c \cap \bar{x}_1 \cap \bar{x}_2$, and x_1/x_2 and x_3 will violate the q-hierarchical condition.

On the other hand, if (2) $\mathcal{V}_c$ are all non-output attributes, and $\mathcal{V}_k(\mathcal{N}) \backslash \bar{y} \neq \emptyset$, it is possible that there is a unique output attribute $x \in \mathcal{V}_\star$ that will contribute to the dimension. In that case, if $\mathcal{E}_c$ contains only one relation, we can find a non-output attribute $x' \in \mathcal{V}_k(\mathcal{N})$, we know $\mathcal{E}[x] \subsetneq \mathcal{E}[x']$, and $x \in \bar{y}$, but $x' \notin \bar{y}$, making the query non-q-hierarchical; otherwise, if $\mathcal{E}_c$ contains more than one relation, we can find a non-output attribute $x' \in \mathcal{V}_c$, and $\mathcal{E}[x] \subsetneq \mathcal{E}[x']$, and $x \in \bar{y}$, but $x' \notin \bar{y}$, making the query non-q-hierarchical.

For the only-if direction, since the query contains only dimension-1 $\mathcal{V}_c$, which holds for $\mathcal{V}_c = \emptyset$ and all $\mathcal{E}_c = \{R\}$, we can show that if, for all x_1, x_2 in $\mathcal{V}$, $\mathcal{E}[x_1] \cap \mathcal{E}[x_2] = \emptyset$ always holds, then all relations in the query are disconnected, and the query is q-hierarchical. If $\mathcal{E}[x_1] \cap \mathcal{E}[x_2] \neq \emptyset$, then $\mathcal{E}[x_1] \subseteq \mathcal{E}[x_2]$ or $\mathcal{E}[x_2] \subseteq \mathcal{E}[x_1]$, otherwise let $R \in \mathcal{E}[x_1] \cap \mathcal{E}[x_2]$, we can show that $\dim(\{\emptyset\}) \geqslant 2$ for $\mathcal{E}_c = \{R\}$, with a $\mathcal{N} \leftarrow \{R', R''\}$ with $R' \in \mathcal{E}[x_1] \backslash \mathcal{E}[x_2]$ and $R'' \in \mathcal{E}[x_2] \backslash \mathcal{E}[x_1]$ and use x_1 and x_2 be their distinct keys. In addition, if there exists $\mathcal{E}[x_1] \subsetneq \mathcal{E}[x_2]$, and $x_1 \in \bar{y}$, then it must have $x_2 \in \bar{y}$, otherwise for any R with both $x_1, x_2 \in \bar{x}$, when selecting $\{R'\}$ to be the distinct neighbour set $\mathcal{N}$ for $\mathcal{V}_c = \{\emptyset\}$ and $\mathcal{E}_c = \{R\}$, with $x_2 \in \bar{x}'$, $\mathcal{V}_k(\mathcal{N}) \nsubseteq \bar{y}$ because of x_2, and x_1 is a live output

attribute, making the dimension of $\mathcal{V}_c = \{\varnothing\}$ and $\mathcal{E}_c = \{R\}$ to be at least 2. Therefore, if the dimension of Q is 1, then the query is q-hierarchical.

D Additional Running Examples

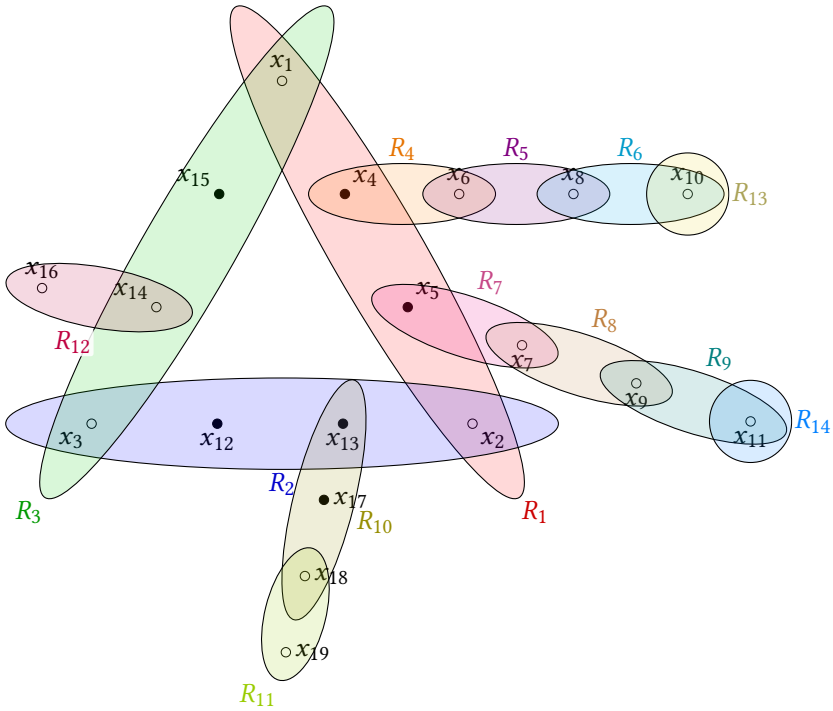


Fig. 3. Hypergraph for Q, where solid dots represent the output attributes.

Example D.1. Consider the query:

$$Q(x_4, x_5, x_{12}, x_{13}, x_{15}, x_{17}) \leftarrow R_1(x_1, x_4, x_5, x_2), R_2(x_3, x_{12}, x_{13}, x_2), R_3(x_1, x_{15}, x_{14}, x_3),$$
$$R_4(x_4, x_6), R_5(x_6, x_8), R_6(x_8, x_{10}), R_7(x_5, x_7), R_8(x_7, x_9), R_9(x_9, x_{11}),$$
$$R_{10}(x_{13}, x_{17}, x_{18}), R_{11}(x_{18}, x_{19}), R_{12}(x_{14}, x_{16}), R_{13}(x_{10}), R_{14}(x_{11})$$

Figure 3 illustrates its relational hypergraph. The query is a height 5 query, with the length-9 chordless path $x_{11}, x_9, x_7, x_5, x_4, x_6, x_8, x_{10}$, or a length-5 q-chordless path x_4, x_6, x_8, x_{10}. However, $x_{11}, x_9, x_7, x_5, x_2, x_3, x_1, x_4, x_6, x_8, x_{10}$ is not a chordless path due to the shortcut between x_4 and x_5.

When considering the dimension, the query is a dimension 6 query, with $\mathcal{V}_c = \{x_1, x_2, x_3\}$, $\mathcal{E}_c = \{R_1, R_2, R_3\}$, $\mathcal{N} = \{R_{12}, R_{10}, R_4, R_7\}$, $\mathcal{V}_k(N) = \{x_4, x_5, x_{13}, x_{14}\}$ and $\mathcal{V}_\star = \{x_4, x_5, x_{12}, x_{13}, x_{14}, x_{15}\}$. All vertices in $\mathcal{V}_\star$ contribute to the dimension, as $\mathcal{V}_\star \backslash \mathcal{V}_k(N) = \{x_{12}, x_{15}\} \subseteq \bar{y}$ and $\mathcal{V}_c \cap \bar{y} = \varnothing$. We cannot obtain a larger dimension by setting $\mathcal{V}_c = \{x_1, x_2, x_3, x_{13}\}$, $\mathcal{E}_c = \{R_1, R_2, R_3, R_{10}\}$ and $\mathcal{V}_\star = \{x_4, x_5, x_{12}, x_{14}, x_{15}, x_{17}, x_{18}\}$. Although the new $\mathcal{V}_\star$ contains more attributes, since $x_{13} \in \bar{y}$, we can only select $\mathcal{V}_k(\mathcal{N}) = \{x_4, x_5, x_{14}, x_{18}\}$ with $\mathcal{N} = \{R_4, R_7, R_{12}, R_{11}\}$ to count toward the dimension and $\dim(\mathcal{V}_c) = 4$ in this case.

When considering static database, the query can be evaluated in $O(|D|^{1.5} + |Q(D)|)$ by replacing $\{R_1, R_2, R_3\}$ with a bag $B_1(x_1, x_2, x_3, x_4, x_5, x_{12}, x_{13}, x_{14}, x_{15}) \leftarrow R_1, R_2, R_3$ and thus transforming the

query into free-connex. The running time suggests for this query, the width $\omega(Q) \leqslant 1.5$, and the lower bound for dynamic evaluation from the width would be $\Omega(|D|^{\omega(Q)-1}) = \Omega(|D|^{0.5})$, which is dominated by the bound from the dimension of the query.

Example D.2 (Running Example for Theorem 1.2). Consider Q_1 from Example 2.7, we now $\mathcal{V}_c = \{x_5\}$ with $\mathcal{E}_c = \{R_3, R_7\}$ to simulate an OuMv$_4$ problem. We select $\mathcal{N} \leftarrow \{R_2, R_4, R_{10}, R_8\}$ with the distinct join keys x_3, x_4, x_9, x_6 (See Figure 2c for the illustration). Given an OuMv$_4$ instance, let $M(a_1, a_2, a_3, a_4, l)$ be the sparse form of the tensor, with l being the tuple identifier. We create the update sequence to be: (1) We insert a special tuple $(\perp)$ into R_6 and $(\perp, \perp)$ into R_1 and R_5; (2) We insert all tuples in $\pi_{a_1,a_2,l,a_3}M$ into R_3, with $\{a_1 \to x_3, a_2 \to x_4, l \to x_5, a_3 \to x_6\}$; (3) We insert all tuples in $\pi_{a_2,l,a_4}M$ into R_7, with $\{a_2 \to x_4, l \to x_5, a_4 \to x_9\}$; (4) We insert all tuples in $\pi_{a_1,a_3}M$ into R_9 with $\{a_1 \to x_3, a_3 \to x_6\}$. After Step (4), we initialize the tensor and can start handling updates on each vector: (5) For every update (t) on u_3 u_4, we make the same update on R_8, R_{10}, respectively; (6) For every update (t) on u_2, we make the update $(t, \perp)$ on R_4; (7) For every update (t) on u_1, we make the update $(\perp, t)$ on R_2; (8) After updating i-th rounds of vectors, we enumerate Q_1. If there exists any query result, the intersection between M and $u_1^{(i)} \times u_2^{(i)} \times u_3^{(i)} \times u_4^{(i)}$ is non-empty.

Example D.3 (Maintaining a Star Query). Consider the dimension-4 star query:

$$Q(x_2, x_3, x_4) \leftarrow R_1(x_1, x_2, x_3, x_4), R_2(x_1), R_3(x_2), R_4(x_3).$$

Note that x_4 is an output attribute in R_1 that does not join with any satellite, but the attribute $x_1 \in \mathcal{V}_k(\mathcal{N})$ is a non-output attribute; thus, we can take x_4 into consideration for the dimension.

Database Instance (R_1). Let the current database instance D be such that the Generalized H-index is $h = 2$. The threshold to classify a key as **Heavy** is $\tau = (h+1)^{d-1} = 3^3 = 27$.

We construct R_1 with roughly 80 tuples distributed as follows:

- **Attribute x_1 Distribution:** Value `A` appears in **30 tuples** (e.g., tuples $t_1 \dots t_{30}$); Value `B` appears in **30 tuples** (e.g., tuples $t_{31} \dots t_{60}$); Value `C` appears in **10 tuples** (e.g., tuples $t_{61} \dots t_{70}$); Values `D-M` appear in **1 tuple** each. There are 2 values (`A`, `B`) with frequency $\geqslant 27$. Thus, x_1 supports $h = 2$. Since there are not 3 values with frequency $3^3 = 27$, it does not reach $h = 3$.
- **Attribute x_2 Distribution:** Value `X` appears in **40 tuples** (overlapping with x_1's `A` and `B` tuples); Value `Y`: appears in **35 tuples**; Other values appear $\leqslant 5$ times. Similar to x_1, values `X` and `Y` exceed the threshold 27 that are heavy.
- **Attribute x_3, x_4 Distribution:** All values are unique (frequency 1). Therefore, no value has a frequency that exceeds 27, and they are all Light.

Partitioning R_1. Given the threshold $\tau = 27$, the algorithm logically partitions R_1 into buckets $R_1^{(b_1 b_2 b_3 b_4)}$ with the following heavy/light keys:

Attribute	Heavy Keys	Light Keys
x_1	{`A`, `B`}	{`C`, `D`, ...}
x_2	{`X`, `Y`}	{...}
x_3	$\varnothing$	All values
x_4	$\varnothing$	All values

- **Partition** $(1, 1, 0, 0)$ **[Heavy x_1, Heavy x_2]:** This partition stores the Cartesian product of the dense keys of the data. It contains tuples like (`A`, `X`, ...), (`A`, `Y`, ...), (`B`, `X`, ...), and (`B`, `Y`, ...).
- **Partition** $(1, 0, 0, 0)$ **[Heavy x_1, Light x_2]:** Contains tuples where $x_1 =$ `A`, `B` but $x_2 \neq$ `X`, `Y`. Recall the x_2 key has frequency $\leqslant 5$ except for `X` and `Y`.
- **Partition** $(0, 1, 0, 0)$ **[Light x_1, Heavy x_2]:** Contains tuples where $x_1 \neq$ `A`, `B` but $x_2 =$ `X`, `Y` Recall `C` has frequency 10 and all keys in `D-M` have frequency 1.

- **Partition** $(0,0,0,0)$ **[All Light]:** Contains all remaining tuples, e.g., singletons like $(\mathtt{D},\mathtt{Z},\dots)$.
- **Other partitions:** For other partitions where x_3 is heavy or x_4 is heavy, they are all empty because none of these keys exist.

Maintenance Procedure. **Scenario A: Update on a Light Key.** Imagine a tuple $(\mathtt{C})$ is inserted into satellite $R_2(x_1)$. The algorithm checks if $\mathtt{C} \in K^H_{x_1}$. In this case, it is not. Therefore, the algorithm must scan the light partitions of R_1 (where $b_1 = 0$) for tuples matching $x_1 = \mathtt{C}$. We know from the distribution that C appears exactly 10 times. Since $10 < \tau = 27$, the scan cost is strictly bounded by $O((h+1)^{\dim(Q)-1})$.

Scenario B: Update on a Heavy Key. Imagine a tuple $(\mathtt{A})$ is inserted into satellite $R_2(x_1)$. The algorithm checks if $\mathtt{A} \in K^H_{x_1}$. In this case, it is **Heavy**. Therefore, the algorithm does *not* scan R_1 for A (which would cost 30 operations, potentially violating the strict bound if it were higher). Instead, it updates the Heavy View V_C. This view effectively tracks the Cartesian product of heavy keys: $\{\mathtt{A},\mathtt{B}\} \times \{\mathtt{X},\mathtt{Y}\}$. It marks $x_1 = \mathtt{A}$ as alive. The cost is proportional to the size of the Cartesian product of all heavy keys, which is limited by $h^{\dim(Q)-1} \approx 2^3 = 8$ operations.

Scenario C: Update on R_1. Imagine the tuple (a_1,a_2,a_3,a_4) is inserted into the center R_1. The algorithm checks if $a_1 \in K^H_{x_1}$, $a_2 \in K^H_{x_2}$, $a_3 \in K^H_{x_3}$, and $a_4 \in K^H_{x_4}$ and obtain the vector $\mathbf{b}$. After that, the algorithm inserts (a_1,a_2,a_3,a_4) into $R_1^{(\mathbf{b})}$, for $a_1,\cdots,a_3$, it checks if $a_i \in R_i^{(0)}$ when a_i is light value; if the tuple passes all these checks, the algorithm projects the tuple to heavy key only and propagates it to V_C. For such updates, the cost is constant.

Scenario D: Key Reclassification. Imagine the algorithm has deleted 17 tuples from R_1 with $x_1 = \mathtt{A}$, making the key A becomes light, while the frequency for X and Y is still higher than 27 after these deletions. Then the algorithm deletes the remaining 13 tuples with $x_1 = \mathtt{A}$ from R_1, and then all tuples with $x_1 = \mathtt{A}$ from R_2, the deletion takes $O(1)$ time per tuple in R_1, and the deletion in R_2 also takes $O(1)$ time due to no propagation can be triggered because A is fully removed from R_1. the algorithm then reinserts all tuples into R_2 with the light label, which takes $O(1)$ time because the current R_1 contains no $x_1 = \mathtt{A}$. After that, the algorithm finally reinserts all tuples into R_1 with the new label, taking $O(1)$ time per tuple. The total cost of the rebalance step is $O(13)$, and it takes 17 deletions to perform. By amortizing the cost into these deletions, we can see that the amortized cost is $O(1)$.

Scenario E: Global Parameter Update. Imagine after inserting 53 tuples with $x_1 = \mathtt{D}$ and 44 tuples with $x_1 = \mathtt{C}$ into R_1, it makes both C and D become heavy, and key reclassifications are done for these two keys. We assume these new tuples contain unique x_2, x_3, x_4 values, thus won't change the heavy/light classification for other keys. Now the number of heavy keys for x_1 is doubled. A global parameter update is required. By doing so, $h_1(D)$ becomes 3 while $h_2(D), h_3(D), h_4(D)$ remains unchanged. Thus, we only need to reclassify the heavy keys in x_1. All keys for x_1 are light because their frequencies are all smaller than $4^3 = 64$. The algorithm will first delete all heavy keys and reinstall them as light keys. The total cost for the rebalance can be $O(h(D)^{\dim(Q)})$, however, it is after $h(D) \times h(D)^{\dim Q-1}$ updates (h keys are added to heavy, while each key's frequency is change from at most $h(D)^{\dim(Q)-1}$ to at least $2h(D)^{\dim(Q)-1}$. Therefore, the amortized cost for such updates is $O(1)$.

Acknowledgments

This work is supported by the Ministry of Education, Singapore, under its Academic Research Fund Tier 1 (RS32/25), and by the Nanyang Technological University Startup Grant. The author would like to thank Ce Jin, Haozhe Zhang, and the anonymous reviewers for the insightful discussion and comments. The author would also like to thank Prof. Ke Yi for his continuous guidance.

References

[1] Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. 2015. If the Current Clique Algorithms are Optimal, So is Valiant's Parser. In *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, Venkatesan Guruswami (Ed.). IEEE Computer Society, 98–117. doi:10.1109/FOCS.2015.16

[2] Amir Abboud, Nick Fischer, and Yarin Shechter. 2024. Faster combinatorial k-clique algorithms. In *Latin American Symposium on Theoretical Informatics*. Springer, 193–206.

[3] Amir Abboud and Virginia Vassilevska Williams. 2014. Popular conjectures imply strong lower bounds for dynamic problems. In *2014 IEEE 55th Annual Symposium on Foundations of Computer Science*. IEEE, 434–443.

[4] Serge Abiteboul, Richard Hull, and Victor Vianu. 1995. *Foundations of databases*. Addison-Wesley Longman Publishing Co., Inc.

[5] Mahmoud Abo Khamis, Ahmet Kara, Dan Olteanu, and Dan Suciu. 2024. Insert-Only versus Insert-Delete in Dynamic Query Evaluation. *Proc. ACM Manag. Data* 2, 5, Article 219 (Nov. 2024), 26 pages. doi:10.1145/3695837

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

[7] Noga Alon, Raphael Yuster, and Uri Zwick. 1997. Finding and Counting Given Length Cycles. *Algorithmica* 17, 3 (1997), 209–223. doi:10.1007/BF02523189

[8] Guillaume Bagan, Arnaud Durand, and Etienne Grandjean. 2007. On Acyclic Conjunctive Queries and Constant Delay Enumeration. In *Computer Science Logic*. Springer Berlin Heidelberg, Berlin, Heidelberg, 208–222.

[9] Christoph Berkholz, Jens Keppeler, and Nicole Schweikardt. 2017. Answering Conjunctive Queries under Updates. In *Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems* (Chicago, Illinois, USA) *(PODS '17)*. Association for Computing Machinery, New York, NY, USA, 303–318. doi:10.1145/3034786.3034789

[10] Paris Carbone, Asterios Katsifodimos, Stephan Ewen, Volker Markl, Seif Haridi, and Kostas Tzoumas. 2015. Apache Flink: Stream and Batch Processing in a Single Engine. *IEEE Data Engineering Bulletin* 38, 4 (2015), 28–38.

[11] Nofar Carmeli and Markus Kröll. 2019. On the Enumeration Complexity of Unions of Conjunctive Queries. In *Proceedings of the 38th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*. ACM, 134–148.

[12] Nofar Carmeli and Luc Segoufin. 2023. Conjunctive Queries With Self-Joins, Towards a Fine-Grained Enumeration Complexity Analysis. 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, 277–289. doi:10.1145/3584372.3588667

[13] Rada Chirkova and Jun Yang. 2012. Materialized views. *Foundations and Trends® in Databases* 4, 4 (2012), 295–405.

[14] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. 2009. *Introduction to Algorithms* (3rd ed.). The MIT Press.

[15] Søren Dahlgaard, Mathias Bæk Tejs Knudsen, and Morten Stöckel. 2017. Finding even cycles faster via capped k-walks. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing* (Montreal, Canada) *(STOC 2017)*. Association for Computing Machinery, New York, NY, USA, 112–120. doi:10.1145/3055399.3055459

[16] Shiyuan Deng, Shangqi Lu, and Yufei Tao. 2023. On Join Sampling and the Hardness of Combinatorial Output-Sensitive Join Algorithms. 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, 99–111. doi:10.1145/3584372.3588666

[17] David Eppstein, Michael T. Goodrich, Darren Strash, and Lowell Trott. 2012. Extended dynamic subgraph statistics using h-index parameterized data structures. *Theoretical Computer Science* 447 (2012), 44–52. doi:10.1016/j.tcs.2011.11.034 Combinational Algorithms and Applications (COCOA 2010).

[18] David Eppstein and Emma S. Spiro. 2009. The h-Index of a Graph and Its Application to Dynamic Subgraph Statistics. In *Algorithms and Data Structures*, Frank Dehne, Marina Gavrilova, Jörg-Rüdiger Sack, and Csaba D. Tóth (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 278–289.

[19] Austen Z Fan, Paraschos Koutris, and Hangdong Zhao. 2023. The Fine-Grained Complexity of Boolean Conjunctive Queries and Sum-Product Problems. In *50th International Colloquium on Automata, Languages, and Programming (ICALP 2023)*. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 127–1.

[20] Marc H. Graham. 1979. *On the universal relation*. Technical Report. University of Toronto.

[21] Robert Haas and Michael Hoffmann. 2006. Chordless paths through three vertices. *Theoretical Computer Science* 351, 3 (2006), 360–371. doi:10.1016/j.tcs.2005.10.021 Parameterized and Exact Computation.

[22] Xiao Hu and Qichen Wang. 2023. Computing the Difference of Conjunctive Queries Efficiently. *Proc. ACM Manag. Data* 1, 2, Article 153 (June 2023), 26 pages. doi:10.1145/3589298

[23] Xiao Hu and Qichen Wang. 2025. Towards Update-Dependent Analysis of Query Maintenance. *Proc. ACM Manag. Data* 3, 2, Article 117 (June 2025), 25 pages. doi:10.1145/3725254

[24] Muhammad Idris, Martin Ugarte, and Stijn Vansummeren. 2017. The Dynamic Yannakakis Algorithm: Compact and Efficient Query Processing Under Updates. In *Proceedings of the 2017 ACM International Conference on Management of Data* (Chicago, Illinois, USA) *(SIGMOD '17)*. Association for Computing Machinery, New York, NY, USA, 1259–1274.

doi:10.1145/3035918.3064027

[25] Muhammad Idris, Martín Ugarte, Stijn Vansummeren, Hannes Voigt, and Wolfgang Lehner. 2018. Conjunctive queries with inequalities under updates. *Proc. International Conference on Very Large Data Bases* 11, 7 (2018), 733–745.

[26] Ce Jin and Yinzhan Xu. 2022. Tight dynamic problem lower bounds from generalized BMM and OMv. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing* (Rome, Italy) *(STOC 2022)*. Association for Computing Machinery, New York, NY, USA, 1515–1528. doi:10.1145/3519935.3520036

[27] Ahmet Kara, Hung Q. Ngo, Milos Nikolic, Dan Olteanu, and Haozhe Zhang. 2020. Maintaining Triangle Queries under Updates. *ACM Trans. Database Syst.* 45, 3, Article 11 (aug 2020), 46 pages. doi:10.1145/3396375

[28] Ahmet Kara, Milos Nikolic, Dan Olteanu, and Haozhe Zhang. 2020. Trade-offs in static and dynamic evaluation of hierarchical queries. In *Proceedings of the 39th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*. 375–392.

[29] Andrea Lincoln and Nikhil Vyas. 2020. Algorithms and Lower Bounds for Cycles and Walks: Small Space and Sparse Graphs. In *11th Innovations in Theoretical Computer Science Conference (ITCS 2020) (Leibniz International Proceedings in Informatics (LIPIcs), Vol. 151)*, Thomas Vidick (Ed.). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 11:1–11:17. doi:10.4230/LIPIcs.ITCS.2020.11

[30] Andrea Lincoln, Virginia Vassilevska Williams, and Ryan Williams. 2018. Tight hardness for shortest cycles and paths in sparse graphs. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM, 1236–1252.

[31] Stefan Mengel. 2025. Lower Bounds for Conjunctive Query Evaluation. In *Companion of the 44th Symposium on Principles of Database Systems* (Berlin, Germany) *(PODS '25)*. Association for Computing Machinery, New York, NY, USA, 5. doi:10.1145/3722234.3725824

[32] Moni Naor, Leonard J. Schulman, and Aravind Srinivasan. 1995. Splitters and Near-Optimal Derandomization. In *36th Annual Symposium on Foundations of Computer Science, FOCS 1995, Milwaukee, Wisconsin, USA, 23-25 October 1995*. IEEE Computer Society, 182–191. doi:10.1109/SFCS.1995.492475

[33] Milos Nikolic and Dan Olteanu. 2018. Incremental view maintenance with triple lock factorization benefits. In *Proc. ACM SIGMOD International Conference on Management of Data*. ACM, 365–380.

[34] Dante Pinto and Cristian Riveros. 2024. Complex event recognition meets hierarchical conjunctive queries. *CoRR* abs/2408.01652 (2024). arXiv:2408.01652 doi:10.48550/ARXIV.2408.01652

[35] Qichen Wang, Xiao Hu, Binyang Dai, and Ke Yi. 2023. Change Propagation Without Joins. *Proc. VLDB Endow.* 16, 5 (jan 2023), 1046–1058. doi:10.14778/3579075.3579080

[36] Qichen Wang and Ke Yi. 2020. Maintaining Acyclic Foreign-Key Joins under Updates. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 1225–1239.

[37] Qichen Wang and Ke Yi. 2025. Conjunctive Queries with Comparisons. *ACM Trans. Database Syst.* (Sept. 2025). doi:10.1145/3769424

[38] C. T. Yu and M. Z. Özsoyoğlu. 1979. An algorithm for tree-query membership of a distributed query. In *The IEEE Computer Society's Third International Computer Software and Applications Conference, COMPSAC 1979, 6-8 November, 1979, Chicago, Illinois, USA*. IEEE, 306–312. doi:10.1109/CMPSAC.1979.762509

Received December 2025; accepted February 2025