# Enumerating Subgraphs of Constant Sizes in External Memory

**Shiyuan Deng**
The Chinese University of Hong Kong, China

**Francesco Silvestri**
University of Padova, Italy

**Yufei Tao**
The Chinese University of Hong Kong, China

## Abstract

We present an indivisible I/O-efficient algorithm for *subgraph enumeration*, where the objective is to list all the subgraphs of a massive graph $G := (V, E)$ that are isomorphic to a pattern graph $Q$ having $k = O(1)$ vertices. Our algorithm performs $O(\frac{|E|^{k/2}}{M^{k/2-1}B} \log_{M/B} \frac{|E|}{B} + \frac{|E|^{\rho}}{M^{\rho-1}B})$ I/Os with high probability, where $\rho$ is the fractional edge covering number of $Q$ (it always holds $\rho \geq k/2$, regardless of $Q$), $M$ is the number of words in (internal) memory, and $B$ is the number of words in a disk block. Our solution is optimal in the class of indivisible algorithms for all pattern graphs with $\rho > k/2$. When $\rho = k/2$, our algorithm is still optimal as long as $M/B \geq (|E|/B)^{\epsilon}$ for any constant $\epsilon > 0$.

**2012 ACM Subject Classification** Theory of computation → Graph algorithms analysis; Information systems → Join algorithms

**Keywords and phrases** Subgraph Enumeration, Conjunctive Queries, External Memory, Algorithms

**Digital Object Identifier** 10.4230/LIPIcs.ICDT.2023.4

**Funding** Shiyuan Deng and Yufei Tao were supported in part by GRF Projects 14207820, 14203421, and 14222822 from HKRGC. Francesco Silvestri was supported in part by PRIN 20174LF3T8 AHeAD and Uni-Impresa Big-Mobility projects, and by the Italian National Center for HPC, Big Data, and Quantum Computing.

## 1 Introduction

*Subgraph enumeration* is the problem of listing all the subgraphs of a data graph $G := (V, E)$ that are isomorphic to a pattern graph $Q$. It is fundamental to a wide range of applications and has been extensively studied in computer science; see [2,3,5–8,10–12,14–18,24,26,29,38] for entry points into the literature. The problem is NP-hard [9] when $Q$ is allowed to have arbitrarily many vertices. In practice, however, a pattern $Q$ of interest is often *considerably* smaller than the data graph $G$ and usually remains the same even as $G$ increases in size over time. For these reasons, research in recent years has focused on pattern graphs $Q$ having $O(1)$ vertices. In the random access machine (RAM) model, numerous (subgraph enumeration) algorithms [4,25,28,30–33,37] have been discovered to achieve worst-case optimal running time (sometimes up to an $O(\text{polylog}\,|E|)$ factor) on such pattern graphs.

RAM algorithms, designed to minimize CPU time, are ill-fitted for *massive* graphs $G$ that cannot be stored in a machine's (main) memory and thus must reside at least partially in the disk. In those environments, the efficiency bottleneck is no longer CPU time, but instead, the number of *I/O accesses* transferring data between the disk and memory. As data graphs' volume continues to outgrow commodity machines' memory capacity, designing I/O-efficient solutions to subgraph enumeration has been a critical challenge. This work will present new progress in tackling the challenge that brings us close to unraveling the problem's I/O complexity.

26th International Conference on Database Theory (ICDT 2023).
Editors: Floris Geerts and Brecht Vandevoort; Article No. 4; pp. 4:1–4:20
LIPICS Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

## 1.1 Problem Definitions and Complexity Parameters

This subsection will formally define the subgraph enumeration problem, the computation model assumed, and the parameters used to characterize algorithm efficiency.

**Subgraph Enumeration.** We are given a simple undirected graph $G := (V, E)$ called the *data graph* and a simple undirected graph $Q := (V_Q, E_Q)$ called the *pattern graph*. We require that $Q$ should have a constant number $k := |V_Q|$ of vertices (and hence, $|E_Q| = O(k^2) = O(1)$). A *subgraph* of $G$ is defined as a simple undirected graph $G_{sub} := (V_{sub}, E_{sub})$ where $V_{sub} \subseteq V$ and $E_{sub} \subseteq E$. We call $G_{sub}$ an *occurrence* of $Q$ if the former is isomorphic to the latter. The goal of the subgraph enumeration problem is to enumerate all the occurrences of $Q$.

We assume that $Q$ is connected (i.e., it has only one connected component), and $G$ has no isolated vertices (i.e., vertices with no incident edges). Isolated vertices cannot participate in any occurrence and, thus, can be safely deleted. As such, $|V| \leq 2|E|$ always holds.

**Computation Model.** We will investigate the problem in the *external memory* (EM) model [1], the de-facto model for studying I/O-efficient algorithms. Under this model, a machine is equipped with $M$ words of *memory* and a disk of an unbounded size that has been formatted into *blocks* of $B$ words. The values of $M$ and $B$ satisfy $M \geq 2B$. A *disk I/O* – henceforth, simply *I/O* – either reads a block of data from the disk into memory or conversely writes $B$ words from memory into a disk block. The *cost* of an algorithm is defined as the number of I/Os performed (CPU computation is for free).

For subgraph enumeration, the data graph is provided under the adjacency format in $O(|E|/B)$ disk blocks, and the pattern graph is stored in memory using $O(1)$ words. In early research (see [21, 34] and the references therein), an algorithm was required to write all the occurrences of $Q$ to the disk. However, as the number of occurrences can be gigantic, the cost of result outputting alone may dominate an algorithm's total cost, thus hampering an effective investigation into the problem's intrinsic I/O complexity. Moreover, in some applications, disk materialization may not even be the intended approach for result reporting, e.g., an algorithm could be instructed to send out all the occurrences by network. For these reasons, the mainstream research nowadays strips off the outputting cost by introducing an *emit*(.) function. Once an occurrence of $Q$ – say, $G_{sub}$ – has been found, the algorithm can invoke *emit*($G_{sub}$) to report $G_{sub}$ *for free*; the algorithm is said to have *emitted* $G_{sub}$ in that case. The algorithm must ensure that every occurrence should be emitted exactly once. It is worth mentioning that an algorithm designed to work with an *emit*(.) function can be easily adapted to produce the result in the disk with $O(\mathrm{OUT}/B)$ extra I/Os, where OUT is the total number of occurrences.

We will concentrate on the class of *indivisible* algorithms (sometimes referred to as *tuple-based* algorithms). Such an algorithm adheres to the constraint that each I/O can bring $O(B)$ edges into memory (this rule prevents, for example, encoding tricks that can compress $\omega(B)$ edges into memory). Furthermore, to emit an occurrence $G_{sub}$, an indivisible algorithm is required to have loaded all the edges of $G_{sub}$ in memory simultaneously. Although the indivisible class does not capture all possible algorithms, it encapsulates all existing subgraph enumeration algorithms we are aware of, with a single exception (to be discussed in Section 1.2). Hence, understanding the optimal I/O complexity achievable by this class offers meaningful insight into the problem's characteristics.

**Fractional Edge Covering Numbers.** Next, we will introduce the *fractional edge covering number*, a notion from graph theory that plays an imperative role in characterizing the I/O complexity of subgraph enumeration. As before, let $Q := (V_Q, E_Q)$ be the input pattern graph. Define $W$ as a function that maps each edge $e \in E_Q$ to a real-valued weight $W(e) \geq 0$. We call $W$ a *fractional edge covering* of $Q$ if it holds for every vertex $v \in V_Q$ that:

$$\sum_{e \in E_Q : e \text{ incident to } v} W(e) \geq 1.$$

We refer to $\sum_{e \in E_Q} W(e)$ as the *total weight* of $W$. The *fractional edge covering number* of $Q$, denoted as $\rho$, is the smallest total weight of all the fractional edge coverings of $Q$.

**Complexity Parameters and Math Conventions.** We will measure the I/O cost of a subgraph enumeration algorithm using five parameters: $|E|$, $\rho$, $k$, $M$, and $B$. Whenever a random event is said to hold "with high probability" – w.h.p. for short – we require the event to hold with probability at least $1 - 1/|E|^{\xi}$, where $\xi$ can be set to an arbitrarily large constant. For an integer $x \geq 1$, $[x]$ represents the set $\{1, 2, ..., x\}$. We define $\text{sort}(n)$ to be the I/O complexity of sorting $n$ elements; it is known [1] that $\text{sort}(n) = O(\lceil \frac{n}{B} \rceil \log_{M/B} \lceil \frac{n}{B} \rceil)$.

## 1.2 Previous Work

In the EM model, research on subgraph enumeration started with *triangle enumeration*, where the pattern graph is $Q :=$ 3-clique, a.k.a., triangle. Pagh and Silvestri [35] gave a randomized algorithm that can emit the triangles of a data graph $G := (V, E)$ in $O(|E|^{1.5}/(\sqrt{M}B))$ I/Os in expectation. They also de-randomized their algorithm to obtain a deterministic I/O bound of $O(\frac{|E|^{1.5}}{\sqrt{M}B} \log_{M/B} \frac{|E|}{B})$ [35]. Their result was later improved by Hu, Qiao, and Tao [20], who managed to emit all triangles deterministically in $O(|E|^{1.5}/(\sqrt{M}B))$ I/Os. The result of [20] matches an I/O lower bound of $\Omega(|E|^{1.5}/(\sqrt{M}B))$ [21, 35] on indivisible algorithms.

The lower bound argument of [21, 35] can be extended [19, 20] to show that, for any pattern graph $Q$ (of a constant size), every indivisible algorithm – no matter randomized or deterministic – must perform $\Omega(|E|^{\rho}/(M^{\rho-1}B))$ I/Os in the worst case to emit all the occurrences of $Q$, where $\rho$ is the fractional edge covering number of $Q$ (for triangle, $\rho = 1.5$). Matching this lower bound for arbitrary $Q$ has been an intriguing open problem. In [19], Hu and Yi developed a deterministic algorithm that achieves an I/O complexity of $O(\frac{|E|^{\rho}}{M^{\rho-1}B} \cdot \log_{M/B} \frac{|E|}{B})$ for any acyclic pattern graph $Q$; their algorithm, however, does not work for cyclic $Q$. In [27], Koutris, Beame, and Suciu presented a technique that can convert an algorithm from the so-called *massively parallel computation* (MPC) model to an algorithm in EM. By combining their technique with a recent MPC algorithm of Ketsman, Suciu, and Tao [23], one can obtain a randomized EM algorithm that can solve, w.h.p., the subgraph enumeration problem for any pattern graph $Q$ in $O(\frac{|E|^{\rho}}{M^{\rho-1}B} \cdot \text{polylog}\,|E|)$ I/Os, as long as $M \geq |E|^c$ where $c < 1$ is a positive constant dependent on $Q$.

All the above algorithms are indivisible. Outside the indivisible class, we are aware of only one algorithm due to Eppstein et al. [13], which is designed for triangle enumeration. Their (randomized) solution guarantees an expected I/O cost of $O(\text{sort}(\alpha|V|) + \text{sort}(|E| \lceil \frac{\alpha \log wlen}{wlen} \rceil) + \text{sort}(\text{OUT}))$ where OUT is the number of occurrences, *wlen* is the number of bits in a word, and $\alpha$ is the arboricity value of $G$ (the algorithm was designed for writing all occurrences to the disk; it can also be deployed for result *emission*, but the I/O complexity does not decrease). The value of $\alpha$ falls between 1 and $O(\sqrt{E})$. Compared to the aforementioned indivisible solutions [20, 35] to triangle enumeration, the algorithm of [13] may have a lower I/O complexity when $\alpha$ and OUT are sufficiently small.

**Table 1** Comparison of our and previous results on subgraph enumeration.

| pattern $Q$ | I/O cost in big-$O$ | source | remark |
|---|---|---|---|
| triangle | $\|E\|^{1.5}/(\sqrt{M}B)$ expected | [35] | rand. |
| triangle | $\frac{\|E\|^{1.5}}{\sqrt{M}B} \log_{M/B} \frac{\|E\|}{B}$ | [35] | det. |
| triangle | $\|E\|^{1.5}/(\sqrt{M}B)$ | [20] | det. |
| triangle | $\text{sort}(\alpha\|V\|) + \text{sort}(\|E\|\lceil \frac{\alpha \log wlen}{wlen} \rceil)$<br>$+\text{sort}(\text{OUT})$ expected | [13] | $\alpha :=$ arboricity of $G$<br>$wlen :=$ word length<br>rand., outside indivisible class |
| acyclic | $\frac{\|E\|^{\rho}}{M^{\rho-1}B} \log_{M/B} \frac{\|E\|}{B}$ | [19] | det. |
| arbitrary | $\frac{\|E\|^{\rho}}{M^{\rho-1}B} \cdot \text{polylog}\,\|E\|$ w.h.p. | [23] | rand., needs $M \geq \|E\|^{c}$ for some<br>$Q$-dependent constant $c \in (0,1)$ |
| arbitrary | $\frac{\|E\|^{k/2}}{M^{k/2-1}B} \log_{\frac{M}{B}} \frac{\|E\|}{B} + \frac{\|E\|^{\rho}}{M^{\rho-1}B}$<br>w.h.p. and expected | ours | rand., optimal when $\rho > \frac{k}{2}$<br>or $\frac{M}{B} \geq (\frac{\|E\|}{B})^{\epsilon}$ for any constant $\epsilon > 0$ |

## 1.3 Our Contributions

The main result of this paper is:

▶ **Theorem 1.** *Let $G := (V, E)$ be a simple undirected graph with no isolated vertices. Let $Q := (V_Q, E_Q)$ be a simple undirected connected pattern graph with $k := O(1)$ vertices. When $|E| \geq M$, there is an algorithm in EM that, with high probability, emits every occurrence of $Q$ in $G$ exactly once with $O(\frac{|E|^{k/2}}{M^{k/2-1}B} \log_{M/B} \frac{|E|}{B} + \frac{|E|^{\rho}}{M^{\rho-1}B})$ I/Os, where $\rho$ is the fractional edge covering number of $Q$, $M$ is the number of words in memory, and $B$ is the number of words in a disk block. The same I/O complexity holds also in expectation.*

The theorem applies to all $M$ and $B$ satisfying $M \geq 2B$. The value of $\rho$ is at least $k/2$ for all pattern graphs $Q$, but can reach $k-1$ for some $Q$ [36]. Our algorithm is indivisible; when $\rho > k/2$, its I/O complexity becomes $O(\frac{|E|^{\rho}}{M^{\rho-1}B})$, matching the indivisible lower bound $\Omega(\frac{|E|^{\rho}}{M^{\rho-1}B})$ (see Section 1.2). When $\rho = k/2$, the algorithm is still optimal as long as $M/B \geq (|E|/B)^{\epsilon}$ for an arbitrarily small constant $\epsilon > 0$ (a condition likely to hold in reality). Table 1 presents a comparison between our and previous results.

# 2 Preliminaries

We will cast subgraph enumeration as a join problem for two reasons. First, it permits us to simplify presentation by leveraging relational algebra's expressive power. Second, our algorithm has a crucial connection to the isolated cartesian product theorem recently developed by Ketsman, Suciu, and Tao [23], which is stated on joins and still lacks an intuitive interpretation on graphs currently. In Section 2.1, we will define the relevant concepts of joins, formulate the join enumeration problem in EM, and review a textbook join algorithm. In Section 2.2, we will explain how to reduce subgraph enumeration to joins. Finally, in Section 2.3, we will introduce a concentration bound that will be useful in our analysis.

## 2.1 Joins on Binary Relations

**Joins.** Define **att** as an arbitrary finite set of *attributes*. A *tuple* over a set $U \subseteq \textbf{att}$ of attributes is a function $\boldsymbol{t} : U \rightarrow \textbf{dom}$, where **dom** is an arbitrary infinite set. For any $U_{sub} \subseteq U$, we define $\boldsymbol{t}[U_{sub}]$ as the tuple $\boldsymbol{t_{sub}}$ over $U_{sub}$ such that $\boldsymbol{t_{sub}}(X) = \boldsymbol{t}(X)$ for every

$X \in U_{sub}$. A *relation* is a set $R$ of tuples over the same set $U$ of attributes; the *schema* of $R$ – denoted as $schema(R)$ – is $U$. $R$ is *unary* if $schema(R)$ has one attribute, or *binary* if $schema(R)$ has two attributes.

We define a *join* as a set $\mathcal{Q}$ of relations. Let $schema(\mathcal{Q}) := \bigcup_{R \in \mathcal{Q}} schema(R)$. The join result, denoted as $join(\mathcal{Q})$, is a relation over $schema(\mathcal{Q})$ that can be formalized as

$$join(\mathcal{Q}) := \{\text{tuple } \boldsymbol{t} \text{ over } schema(\mathcal{Q}) \mid \forall R \in \mathcal{Q} : \boldsymbol{t}[schema(R)] \in R\}.$$

The *input size* of $\mathcal{Q}$ is defined as $\sum_{R \in \mathcal{Q}} |R|$, namely, the total number of tuples in all relations. We will call $\mathcal{Q}$ a *binary join* if all its relations are binary.

**Schema Graphs.** We define the *schema graph* of a join $\mathcal{Q}$ as the hypergraph $\mathcal{G} := (\mathcal{X}, \mathcal{E})$ where $\mathcal{X} := schema(\mathcal{Q})$ and $\mathcal{E} := \{schema(R) \mid R \in \mathcal{Q}\}$. We will consistently refer to the vertices in $\mathcal{X}$ as "attributes" and to the elements in $\mathcal{E}$ as "hyperedges". Note that $\mathcal{G}$ is a "hyper"-graph, rather than just a "graph", because each of its hyperedges may not have exactly two attributes (e.g., if a relation $R \in \mathcal{Q}$ is unary, then the hyperedge $schema(R) \in \mathcal{E}$ has only one attribute). Moreover, $\mathcal{G}$ may have identical hyperedges (this happens when two relations in $\mathcal{Q}$ have the same schema). A hyperedge $e \in \mathcal{E}$ is *unary* if $|e| = 1$, or *binary* if $|e| = 2$. Two vertices $X_1 \in X$ and $X_2 \in X$ are *adjacent* in $\mathcal{G}$ if there exists a hyperedge $e \in \mathcal{E}$ containing both $X_1$ and $X_2$.

A function $W$ mapping each hyperedge $e \in \mathcal{E}$ to a non-negative real value $W(e)$ is called a *fractional edge covering* of $\mathcal{G}$ if it satisfies the following condition: for every attribute $X \in \mathcal{X}$, $\sum_{e \in \mathcal{E}: X \in e} W(e) \geq 1$, namely, the weights of all the hyperedges containing $X$ add up to at least 1. The *total weight* of $W$ is defined as $\sum_{e \in \mathcal{E}} W(e)$. The *fractional edge covering number* of $\mathcal{G}$ is the smallest total weight of all the fractional edge coverings of $\mathcal{G}$.

**Active Domains and Degrees.** Let $\mathcal{Q}$ be a binary join with schema graph $\mathcal{G} := (\mathcal{X}, \mathcal{E})$. For each attribute $X \in \mathcal{X}$, we define the *active domain* of $X$ as $\textbf{adom}(X) := \bigcup_{R \in \mathcal{Q}: X \in schema(R)} \{\boldsymbol{t}(X) \mid \boldsymbol{t} \in R\}$. Henceforth, we will take the view that the attributes in $schema(\mathcal{Q})$ have mutually disjoint active domains (this loses no generality because one can conceptually prefix each value with an attribute name, if necessary). Define the *combined active domain* of $\mathcal{Q}$ as

$$\textbf{adom} := \bigcup_{X \in schema(\mathcal{Q})} \textbf{adom}(X). \tag{1}$$

Fix any attribute $X \in \mathcal{X}$ and any value $v \in \textbf{adom}(X)$. We define the *degree* of $v$ as

$$\max_{R \in \mathcal{Q}: X \in schema(R)} |\{\boldsymbol{u} \in R \mid \boldsymbol{u}(X) = v\}|.$$

Intuitively, the degree tells us at most how many tuples can carry value $v$ under attribute $X$ in a relation of $\mathcal{Q}$. Moreover, define

$$\text{degree of } \mathcal{Q} := \max_{v \in \textbf{adom}} \text{degree of } v. \tag{2}$$

**Join Result Enumeration in EM.** Let $\mathcal{Q}$ be a binary join with input size $N := \sum_{R \in \mathcal{Q}} |R|$ and schema graph $\mathcal{G}$. We will study the evaluation of $\mathcal{Q}$ under the EM model, assuming that $\mathcal{G}$ has $O(1)$ attributes. At the beginning of an algorithm, each relation $R \in \mathcal{Q}$ is stored in $O(|R|/B)$ consecutive blocks in the disk, and $\mathcal{G}$ is stored in memory using $O(1)$ words. Result reporting is done through a special function *emit*(.): every time the algorithm finds a tuple $\boldsymbol{t} \in join(\mathcal{Q})$, it can *emit* $\boldsymbol{t}$ for free by calling *emit*($\boldsymbol{t}$). Every tuple in $join(\mathcal{Q})$ should be emitted exactly once. If the algorithm is randomized, we will use the statement "an event holds with high probability (w.h.p.)" to state that the event holds with probability at least $1 - 1/N^\xi$, where $\xi$ can be an arbitrarily large constant.

**Blocked Nested Loop (BNL).** This textbook algorithm works for arbitrary joins:

▶ **Lemma 2.** *Let $\mathcal{Q}$ be a join with $r = O(1)$ input relations. The BNL algorithm emits every tuple of join($\mathcal{Q}$) exactly once in $O(\lceil \frac{N^r}{M^{r-1}B} \rceil)$ I/Os, where $N := \sum_{R \in \mathcal{Q}} |R|$, $M$ is the number of words in memory, and $B$ is the number of words in a disk block.*

The proof is trivial and omitted. BNL will serve as a building block in our algorithms.

## 2.2 Reduction from Subgraph Enumeration to Binary Joins

We can convert subgraph enumeration to binary-join evaluation with no degradation in terms of worst-case I/O complexity. Consider an instance of subgraph enumeration with data graph $G := (V, E)$ and pattern graph $Q := (V_Q, E_Q)$. We create a binary join $\mathcal{Q}$ on $|E_Q|$ relations by executing the following steps for each edge $\{X_1, X_2\} \in E_Q$ (where $X_1$ and $X_2$ are distinct vertices in $V_Q$):

- Add a relation $R$ to $\mathcal{Q}$ with schema $schema(R) := \{X_1, X_2\}$.
- For each edge $\{u, v\} \in E$ (where $u$ and $v$ are distinct vertices in $V$), define a tuple $\boldsymbol{t_1}$ with $\boldsymbol{t_1}(X_1) := u$ and $\boldsymbol{t_1}(X_2) := v$, and another tuple $\boldsymbol{t_2}$ with $\boldsymbol{t_2}(X_1) := v$ and $\boldsymbol{t_2}(X_2) := u$. Add both $\boldsymbol{t_1}$ and $\boldsymbol{t_2}$ to $R$.

The above conversion has several properties. First, the schema graph $\mathcal{G} := (\mathcal{X}, \mathcal{E})$ of $\mathcal{Q}$ is isomorphic to the pattern graph $Q$. Second, each relation $R \in \mathcal{Q}$ has $2|E|$ tuples such that the input size of $\mathcal{Q}$ is $2|E| \cdot |E_Q| = \Theta(|E|)$. Third, the relations in $\mathcal{Q}$ have distinct schemas.

The lemma below, which is proved in Appendix A, shows that an efficient algorithm for evaluating $\mathcal{Q}$ implies an efficient algorithm for performing subgraph enumeration on $G$.

▶ **Lemma 3.** *Consider any input to the subgraph enumeration problem with data graph $G$ and pattern graph $Q$. Let $\mathcal{Q}$ be the join constructed in the way explained above. If we have an algorithm to emit all the tuples of join($\mathcal{Q}$) in $T$ I/Os w.h.p., then we can emit every occurrence of $Q$ in $G$ exactly once using $T + O(\lceil |E|/B \rceil)$ I/Os w.h.p..*

By virtue of the above lemma, we will turn our attention to joins on binary relations.

## 2.3 A Concentration Bound under Partial Dependence

Next, we will review a Chernoff-like result due to Janson [22]. Let $X_1, X_2, ..., X_n$ be random variables satisfying $X_i - \mathbf{E}[X_i] \le 1$ for all $i \in [n]$; these variables may *not* follow the same distribution. Suppose that we are also given a dependency graph $G_{dep}$ with $\{X_1, X_2, ..., X_n\}$ as the vertex set. $G_{dep}$ must fulfill the following *independence requirement*: for any $S \subseteq \{X_1, X_2, ..., X_n\}$ and any vertex $X_i \notin S$ (for some $i \in [n]$), if $X_i$ is not adjacent to any vertex in $S$, then $X_i$ is independent of the joint distribution of the variables in $S$. In Theorem 2.3 of [22], Janson proved:

▶ **Lemma 4** ( [22]). *Set $X := \sum_{i=1}^n X_i$, $\mu := \mathbf{E}[X]$, and $\sigma$ to any value at least $\sum_{i=1}^n \mathbf{Var}(X_i)$. Define $\Delta$ to be the maximum vertex degree in $G_{dep}$. Then, for any $\epsilon > 0$, it holds that*

$$\Pr[X \ge (1+\epsilon)\mu] \quad \le \quad \exp\left(-\frac{8\epsilon^2 \cdot \mu^2}{25\Delta(\sigma + \epsilon \cdot \mu/3)}\right). \tag{3}$$

# 3 An EM Algorithm for Binary Joins of Bounded Degrees

This section serves as a proof of:

▶ **Lemma 5.** *Consider a binary join $\mathcal{Q}$ whose relations have distinct schemas. Let $\mathcal{G} := (\mathcal{X}, \mathcal{E})$ be the schema graph of $\mathcal{Q}$, and set $N := \sum_{R \in \mathcal{Q}} |R|$ and $k := |\mathcal{X}|$. Fix any value $\lambda \geq \sqrt{NM}$, where $M$ is the number of words in memory. If $N \geq M$ and $\mathcal{Q}$ has a degree at most $\lambda$, there is an EM algorithm that, with probability at least $1 - 1/\lambda^{\xi}$, emits every tuple of join($\mathcal{Q}$) exactly once in $O(\lambda^k/(M^{k-1}B))$ I/Os, where $\xi$ can be an arbitrarily large constant, and $B$ is the number of words in a disk block.*

Note that the success probability is expressed using $\lambda$ rather than $N$. This will be an essential feature in Section 4 where we utilize the lemma as a subroutine to tackle general binary joins. To prove Lemma 5, we consider only $k \geq 3$; if $k = 2$, $\mathcal{G}$ has only two attributes – namely, $\mathcal{Q}$ has only one single relation – in which case we can trivially emit the tuples of *join*($\mathcal{Q}$) exactly once in $O(N/B)$ I/Os. When $k \geq 3$, it holds that $\text{sort}(N) = O(N^{1.5}/(\sqrt{M}B)) = O(\lambda^k/(M^{k-1}B))$.

## 3.1 An Algorithmic Framework

We will describe a high-level framework for evaluating the join $\mathcal{Q}$ in Lemma 5. Depending on $M$, we will instantiate the framework differently in Sections 3.2 and 3.3, which together will make a complete algorithm with the guarantees in Lemma 5.

**Coloring.** Set $r := |\mathcal{Q}|$, i.e., the number of relations in $\mathcal{Q}$ (also the number of hyperedges in $\mathcal{E}$). Furthermore, define

$$s := \lceil \lambda/M \rceil \tag{4}$$

and assume that we are given a function

$$\Gamma : \mathbf{adom} \to [s]. \tag{5}$$

Recall that **adom** is the combined active domain of $\mathcal{Q}$; see (1). We will refer to each possible output of $\Gamma$ as a *color*; in other words, $\Gamma$ maps each value of **adom** to a color in $[s]$. We will also assume that a *coloring step* has been performed to color all the tuples by $\Gamma$; namely, for every relation $R \in \mathcal{Q}$, any tuple $\boldsymbol{t} \in R$, and each attribute $X \in \textit{schema}(R)$, the color $\Gamma(\boldsymbol{t}(X))$ is stored together with $\boldsymbol{t}$ (this means two extra words for each tuple). The choice of $\Gamma$ (henceforth named the *coloring function*), as well as the coloring step, is the key to instantiating our algorithmic framework.

**Color Schemes.** We can divide the join result *join*($\mathcal{Q}$) by how the tuples therein are colored by $\Gamma$. We say that two tuples $\boldsymbol{t_1}$ and $\boldsymbol{t_2}$ in *join*($\mathcal{Q}$) have the same *color scheme* if $\Gamma(\boldsymbol{t_1}(X)) = \Gamma(\boldsymbol{t_2}(X))$ for every attribute $X \in \mathcal{X}$. Formally, a *color scheme* is a function

$$\gamma : \mathcal{X} \to [s]. \tag{6}$$

As each attribute can be colored any value in $[s]$, there are in total $s^{|\mathcal{X}|} = s^k$ color schemes. Every color scheme $\gamma$ spawns a join of its own. For each relation $R \in \mathcal{Q}$, define

$$R_\gamma := \{\boldsymbol{t} \in R \mid \Gamma(\boldsymbol{t}(X)) = \gamma(X) \text{ for all } X \in \textit{schema}(R)\}.$$

Intuitively, $R_\gamma$ is the subset of tuples in $R$ that are colored by $\Gamma$ in a way consistent with $\gamma$. We can now define a join induced by $\gamma$:

$$\mathcal{Q}_\gamma := \{R_\gamma \mid R \in \mathcal{Q}\}.$$

The sets $join(\mathcal{Q}_\gamma)$ of all color schemes $\gamma$ are mutually disjoint and their union is $join(\mathcal{Q})$.

**Algorithm.** In Appendix B, we show that, after a preprocessing step with I/O cost $O(\text{sort}(N))$, we can store the input relations of $\mathcal{Q}_\gamma$ – for every color scheme $\gamma$ – in consecutive disk blocks. Then, for each $\gamma$, we deploy the BNL algorithm of Lemma 2 to emit the tuples of $join(\mathcal{Q}_\gamma)$. This completes the algorithm for evaluating $\mathcal{Q}$.

**Analysis.** By Lemma 2, the BNL execution of all $s^k$ color schemes incurs a total I/O cost of

$$O\left(\sum_\gamma \left\lceil \frac{N_\gamma}{M} \right\rceil^{r-1} \left\lceil \frac{N_\gamma}{B} \right\rceil\right) = O\left(s^k + \sum_\gamma \frac{{N_\gamma}^r}{M^{r-1}B}\right) = O\left(s^k + \frac{\sum_\gamma \sum_{R\in\mathcal{Q}} |R_\gamma|^r}{M^{r-1}B}\right) \quad (7)$$

where $N_\gamma$ is the input size of $\mathcal{Q}_\gamma$, and the second equality used the fact that $|\mathcal{Q}|$ has only a constant number of relations.

To facilitate the analysis of (7), let us impose an arbitrary ordering on the attributes in $\mathcal{X}$; we use the notation $X_1 < X_2$ to denote the fact that attribute $X_1 \in \mathcal{X}$ precedes another attribute $X_2 \in \mathcal{X}$ in the ordering. Fix any two colors $c_1 \in [s]$ and $c_2 \in [s]$. For each relation $R \in \mathcal{Q}$ whose $schema(R)$ has attributes $X_1$ and $X_2$ with $X_1 < X_2$, define

$$R_{c_1,c_2} := \{\boldsymbol{t} \in R \mid \Gamma(\boldsymbol{t}(X_1)) = c_1 \text{ and } \Gamma(\boldsymbol{t}(X_2)) = c_2\};$$

namely, $R_{c_1,c_2}$ includes every tuple of $R$ that receives colors $c_1$ and $c_2$ on attributes $X_1$ and $X_2$, respectively. We can now derive:

$$(7) = O\Big(s^k + \frac{\sum_{R\in\mathcal{Q}} \sum_\gamma |R_\gamma|^r}{M^{r-1}B}\Big) = O\Big(s^k + \frac{s^{k-2}}{M^{r-1}B} \sum_{R\in\mathcal{Q}} \sum_{c_1,c_2\in[s]} |R_{c_1,c_2}|^r\Big) \quad (8)$$

where the second equality holds because each pair $(c_1, c_2)$ is relevant to $s^{k-2}$ color schemes.

Given the value of $s$ in (4), the term $s^k$ is $O((\lambda/M)^k) = O(\lambda^k/(M^{k-1}B))$. What is non-trivial is to argue that the term $\frac{s^{k-2}}{M^{r-1}B} \sum_{R\in\mathcal{Q}} \sum_{c_1,c_2\in[s]} |R_{c_1,c_2}|^r$ can also be bounded by $O(\lambda^k/(M^{k-1}B))$. We will do so by choosing the coloring function $\Gamma$ carefully according to the memory size $M$.

## 3.2 When $M = O(\lambda/\log^2 \lambda)$

We will first explain how to choose the coloring function $\Gamma$ to ensure that the algorithm described in Section 3.1 performs $O(\lambda^k/(M^{k-1}B))$ I/Os in expectation. Then, we will slightly modify the algorithm to achieve the same I/O complexity with probability at least $1 - 1/\lambda^\xi$.

**Choice of $\Gamma$.** We decide $\Gamma$ by independently mapping each value of **adom** to a color chosen uniformly at random from $[s]$. This $\Gamma$ can be stored as a list of (value, color) pairs in $O(N/B)$ blocks. The coloring step (as defined in Section 3.1) can then be performed in sort($N$) I/Os.

**Identically-Colored Subsets.** Let us first study a probability question that arises from our analysis. Take an arbitrary relation $R \in \mathcal{Q}$. Let $X_1$ and $X_2$ be the two attributes in *schema*$(R)$; w.l.o.g., assume $X_1 < X_2$ (recall from Section 3.1 that we have imposed an arbitrary ordering on the attributes). Given an integer $i \in [r]$, define an *$i$-subset* of $R$ to be a subset $S \subseteq R$ with $|S| = i$. We say that $S$ is *identically colored* if all the tuples in $S$ belong to the same color scheme; in other words, for any $\boldsymbol{t_1}, \boldsymbol{t_2} \in S$, it holds that $\Gamma(\boldsymbol{t_1}(X_1)) = \Gamma(\boldsymbol{t_2}(X_1))$ and $\Gamma(\boldsymbol{t_1}(X_2)) = \Gamma(\boldsymbol{t_2}(X_2))$. Define:

$$Y_i \quad := \quad \text{the number of identically colored } i\text{-subsets of } R. \tag{9}$$

Note that $Y_i$ is a random variable because its value varies with $\Gamma$. We want to understand how large $Y_i$ is in expectation. The lemma below provides an answer.

▶ **Lemma 6.** $\mathbf{E}[Y_i] = O(\lambda^2 \cdot M^{i-2})$ *for each* $i \in [r]$.

**Proof.** Recall from the statement of Lemma 5 that $\lambda \geq \sqrt{NM}$. Next, we will use induction to prove the claim "$\mathbf{E}[Y_i] \leq (4r^2 \cdot M)^{i-1}N$ for all $i \in [r]$", which will establish the lemma because $M^{i-1}N \leq \lambda^2 \cdot M^{i-2}$. For $i = 1$, $\mathbf{E}[Y_1]$ is trivially bounded by $N$; hence, the claim holds at $i = 1$.

Assuming the claim's correctness for $i = j - 1$ where $j \geq 2$, next we give the proof for $i = j$. Consider, w.l.o.g., $|R| \geq j$ (otherwise, $Y_j = 0$ and the claim is vacuously true). Given a $(j-1)$- or $j$-subset $S$ of $R$, we define $Z(S)$ to be 1 if $S$ is identically colored; otherwise, $Z(S) := 0$. Impose an arbitrary ordering on the tuples of $R$. Given a $j$-subset $S_j$ of $R$, we can list the tuples of $S_j$ in ascending order as $\boldsymbol{t_1}, \boldsymbol{t_2}, ..., \boldsymbol{t_j}$. We call $S_j$ an *extension* of the $(j-1)$-subset $S_{j-1} := \{\boldsymbol{t_1}, \boldsymbol{t_2}, ..., \boldsymbol{t}_{j-1}\}$. Next, we discuss the relationship between $\Pr[Z(S_{j-1}) = 1]$ and $\Pr[Z(S_j) = 1]$ by distinguishing three cases.

- Case 1: $\boldsymbol{t}_j(X_1) \in \Pi_{X_1}(S_{j-1})$ and $\boldsymbol{t}_j(X_2) \in \Pi_{X_2}(S_{j-1})$. Clearly, $Z(S_j)$ always equals $Z(S_{j-1})$ and, hence, $\Pr[Z(S_{j-1}) = 1] = \Pr[Z(S_j) = 1]$.
- Case 2: $\boldsymbol{t}_j(X_1) \in \Pi_{X_1}(S_{j-1})$ but $\boldsymbol{t}_j(X_2) \notin \Pi_{X_2}(S_{j-1})$. First note that if $Z(S_{j-1})$ is 0, so must be $Z(S_j)$. Consider now $Z(S_{j-1}) = 1$; hence, $\Gamma$ maps all the values in $\Pi_{X_2}(S_{j-1})$ to the same color, say, $c \in [s]$. $Z(S_j) = 1$ if and only if $\Gamma(\boldsymbol{t}_j(X_2)) = c$. Thus, $\Pr[Z(S_j) = 1] = \Pr[Z(S_{j-1}) = 1] \cdot \Pr[\Gamma(\boldsymbol{t}_j(X_2)) = c \mid Z(S_{j-1}) = 1] = \Pr[Z(S_{j-1}) = 1]/s$.
- Case 3: $\boldsymbol{t}_j(X_1) \notin \Pi_{X_1}(S_{j-1})$ but $\boldsymbol{t}_j(X_2) \in \Pi_{X_2}(S_{j-1})$. This is symmetric to Case 2, and we also have $\Pr[Z(S_j) = 1] = \Pr[Z(S_{j-1}) = 1]/s$.
- Case 4: $\boldsymbol{t}_j(X_1) \notin \Pi_{X_1}(S_{j-1})$ and $\boldsymbol{t}_j(X_2) \notin \Pi_{X_2}(S_{j-1})$. Again, if $Z(S_{j-1})$ is 0, so must be $Z(S_j)$. When $Z(S_{j-1}) = 1$, $\Gamma$ maps (i) all the values in $\Pi_{X_1}(S_{j-1})$ to the same color, say, $c_1 \in [s]$, and (ii) all the values in $\Pi_{X_2}(S_{j-1})$ to the same color, say, $c_2 \in [s]$. $Z(S_j) = 1$ if and only if $\Gamma(\boldsymbol{t}_j(X_1)) = c_1$ and $\Gamma(\boldsymbol{t}_j(X_2)) = c_2$. Hence, $\Pr[Z(S_j) = 1] = \Pr[Z(S_{j-1}) = 1] \cdot \Pr[\Gamma(\boldsymbol{t}_j(X_1)) = c_1, \Gamma(\boldsymbol{t}_j(X_2)) = c_2 \mid Z(S_{j-1}) = 1] = \Pr[Z(S_{j-1}) = 1]/s^2$.

Denote by $\mathcal{S}_j$ and $\mathcal{S}_{j-1}$ the set of all $j$- and $(j-1)$-subsets of $R$, respectively. We bound $\mathbf{E}[Y_j] = \sum_{S_j \in \mathcal{S}_j} \Pr[Z(S_j) = 1]$ using a charging argument. Each $S_j \in \mathcal{S}_j$ is the extension of a unique $(j-1)$-subset $S_{j-1}$. If $S_j$ is a Case-1 extension, we charge a weight of 1 on $S_{j-1}$; if $S_j$ is a Case -2 or -3 extension, we charge a weight of $1/s$ on $S_{j-1}$; if $S_j$ is a Case -4 extension, we charge a weight of $1/s^2$ on $S_{j-1}$. Note that each $S_{j-1} \in \mathcal{S}_{j-1}$ can be charged more than once because $S_{j-1}$ can have multiple extensions. The above discussion implies

$$\mathbf{E}[Y_j] = \sum_{S_j \in \mathcal{S}_j} \Pr[Z(S_j) = 1] = \sum_{S_{j-1} \in \mathcal{S}_{j-1}} \Pr[Z(S_{j-1}) = 1] \cdot \text{total weight charged on } S_{j-1}. \tag{10}$$

To analyze how much total weight can be charged on a $(j-1)$-subset $S_{j-1}$ of $R$, observe:

- $S_{j-1}$ has at most $(j-1)^2$ extensions of Case 1. Such an extension must add to $S_{j-1}$ a tuple $\boldsymbol{t}_j$ satisfying $\boldsymbol{t}_j(X_1) \in \Pi_{X_1}(S_{j-1})$ and $\boldsymbol{t}_j(X_2) \in \Pi_{X_2}(S_{j-1})$. Since each of $\Pi_{X_1}(S_{j-1})$ and $\Pi_{X_2}(S_{j-1})$ has size at most $j-1$, at most $(j-1)^2$ tuples can be selected as $\boldsymbol{t}_j$.
- $S_{j-1}$ has at most $(j-1)\lambda$ extensions of Case 2. Such an extension must add to $S_{j-1}$ a tuple $\boldsymbol{t}_j \in R$ satisfying $\boldsymbol{t}_j(X_1) = v$, for some $v \in \Pi_{X_1}(S_{j-1})$. As the degree of $v$ is at most $\lambda$ (by definition of $\lambda$), at most $\lambda$ tuples in $R$ can be selected as $\boldsymbol{t}_j$. The bound $(j-1)\lambda$ thus follows from the fact $|\Pi_{X_1}(S_{j-1})| \le j-1$.
- Symmetrically, $S_{j-1}$ has at most $(j-1)\lambda$ extensions of Case 3.
- Trivially, $S_{j-1}$ has at most $N$ extensions of Case 4.

It thus follows that the total weight charged on $S_{j-1}$ is at most $(j-1)^2 + \frac{2(j-1)\lambda}{s} + \frac{N}{s^2}$, which is at most $4r^2M$ given the value of $s$ in (4). We can then obtain from (10):

$$\mathbf{E}[Y_j] \le \sum_{S_{j-1} \in \mathcal{S}_{j-1}} \Pr[Z(S_{j-1}) = 1] \cdot (4r^2M) = \mathbf{E}[Y_{j-1}] \cdot (4r^2M) \le (4r^2M)^{j-1}N$$

where the last inequality used our inductive assumption $\mathbf{E}[Y_{j-1}] \le (4r^2M)^{j-2}N$. ◀

**I/O Cost in Expectation.** We now proceed to analyze the expected I/O cost of the algorithm in Section 3.1. The lemma below is essentially a corollary of Lemma 6.

▶ **Lemma 7.** *For any $R \in \mathcal{Q}$, $\mathbf{E}[\sum_{c_1,c_2 \in [s]} |R_{c_1,c_2}|^r] = O(\lambda^2 \cdot M^{r-2})$.*

**Proof.** Because $r$ is a constant, $\sum_{c_1,c_2 \in [s]} |R_{c_1,c_2}|^r = O(s^2 + \sum_{c_1,c_2 \in [s]:|R_{c_1,c_2}| \ge r} \binom{|R_{c_1,c_2}|}{r})$, where the term $O(s^2)$ accounts for the at most $s^2$ pairs of $(c_1, c_2)$ satisfying $|R_{c_1,c_2}| < r$.[1] Observe that $\sum_{c_1,c_2 \in [s]:|R_{c_1,c_2}| \ge r} \binom{|R_{c_1,c_2}|}{r}$ is exactly $Y_r$ as defined in (9). Hence, $\mathbf{E}[\sum_{c_1,c_2 \in [s]} |R_{c_1,c_2}|^r] = \mathbf{E}[O(s^2 + Y_r)] = O(\lambda^2 \cdot M^{r-2})$ (here, we applied Lemma 6 and the value of $s$ in (4)). ◀

Hence, the term $\frac{s^{k-2}}{M^{r-1}B} \sum_{R \in \mathcal{Q}} \sum_{c_1,c_2 \in [s]} |R_{c_1,c_2}|^r$ has an expectation of $O(\frac{s^{k-2}}{M^{r-1}B} \cdot \lambda^2 \cdot M^{r-2}) = O(\lambda^k/(M^{k-1}B))$. We can now conclude that the algorithm in Section 3.1 has an expected I/O cost of $O(\lambda^k/(M^{k-1}B))$ overall.

**Achieving High Probability.** Our analysis indicates that the I/O cost is $O(\lambda^k/(M^{k-1}B))$ as long as $\sum_{c_1,c_2 \in [s]} |R_{c_1,c_2}|^r = O(\lambda^2 \cdot M^{r-2})$ for every $R \in \mathcal{Q}$. By Markov inequality, the probability for $\sum_{c_1,c_2 \in [s]} |R_{c_1,c_2}|^r$ to exceed $2r \cdot \mathbf{E}[\sum_{c_1,c_2 \in [s]} |R_{c_1,c_2}|^r] = O(\lambda^2 \cdot M^{r-2})$ is at most $1/(2r)$. The union bound then assures us that, with probability at least $1/2$, $\sum_{c_1,c_2 \in [s]} |R_{c_1,c_2}|^r = O(\lambda^2 \cdot M^{r-2})$ holds for all the $r$ relations $R \in \mathcal{Q}$. Once the coloring function $\Gamma$ has been chosen, by sorting, we can obtain the precise value $\sum_{c_1,c_2 \in [s]} |R_{c_1,c_2}|^r$ for every $R \in \mathcal{Q}$ in sort($N$) I/Os. As long as any $\sum_{c_1,c_2 \in [s]} |R_{c_1,c_2}|^r$ falls out of $O(\lambda^2 \cdot M^{r-2})$, we repeat from scratch by choosing another $\Gamma$. It takes $O(\log \lambda)$ repeats to ensure $\sum_{c_1,c_2 \in [s]} |R_{c_1,c_2}|^r = O(\lambda^2 \cdot M^{r-2})$ for all $R \in \mathcal{Q}$ with probability at least $1 - 1/\lambda^\xi$ for an arbitrarily large constant $\xi$. With the above modification, our algorithm has an I/O cost $O(\text{sort}(N) \cdot \log \lambda + \lambda^k/(M^{k-1}B))$ with probability at least $1 - 1/\lambda^\xi$. The complexity is $O(\lambda^k/(M^{k-1}B))$ as long as $M = O(\lambda/\log^2 \lambda)$.

[1] Every such pair can contribute at most $(r-1)^r = O(1)$ to $\sum_{c_1,c_2 \in [s]} |R_{c_1,c_2}|^r$.

### 3.3 When $M = \Omega((\lambda \log \lambda)^{2/3})$

This subsection will present another instantiation of the framework in Section 3.1 that runs in $O(\lambda^k/(M^{k-1}B))$ I/Os with probability at least $1 - 1/\lambda^\xi$ when $M = \Omega((\lambda \log \lambda)^{2/3})$. Combining this instantiation with the one in Section 3.2 proves Lemma 5.

**Choice of $\Gamma$.** We classify a value $v \in \mathbf{adom}$ as a *low-degree* value if its degree is less than $\lambda/\sqrt{M}$, or a *high-degree* value otherwise (review Section 2.1 for the notion "degree"). Let $\mathbf{adom}_{lo}$ (resp. $\mathbf{adom}_{hi}$) be the set of low- (resp. high-) degree values. We can obtain the degrees of all values in $\mathbf{adom}$ – hence, $\mathbf{adom}_{lo}$ and $\mathbf{adom}_{hi}$ – in $O(\text{sort}(N))$ I/Os.

Different strategies are deployed to map $\mathbf{adom}_{lo}$ and $\mathbf{adom}_{hi}$ to $[s]$. For $\mathbf{adom}_{lo}$, we independently map each value therein to a color chosen from $[s]$ uniformly at random. The strategy for $\mathbf{adom}_{hi}$ is, however, deterministic. By scanning $\mathbf{adom}_{hi}$ once in $O(N/B)$ I/Os, we can break $\mathbf{adom}_{hi}$ into at most $N/\lambda$ disjoint *groups* such that, for each group, the total degree of all the values therein is at most $5\lambda$.[2] Note that, as a value in $\mathbf{adom}_{hi}$ has degree at least $\lambda/\sqrt{M}$, each group contains at most $\frac{5\lambda}{\lambda/\sqrt{M}} = 5\sqrt{M}$ values. Moreover, since $\lambda \geq \sqrt{NM}$, there can be no more than $N/\lambda \leq \lambda/M \leq s$ groups. We treat each group as a distinct color, and define function $\Gamma_2 : \mathbf{adom}_{hi} \to [s]$ that maps a value $v \in \mathbf{adom}_{hi}$ to color $c \in [s]$ if $v$ appears in the $c$-th group. Functions $\Gamma_1$ and $\Gamma_2$ together define the coloring function $\Gamma$ in (5). The coloring step (defined in Section 3.1) can then be performed with sorting in $O(\text{sort}(N))$ I/Os.

**Analysis.** Next, we analyze the I/O cost of our algorithm in Section 3.1, given the above choice of $\Gamma$. Our objective is to prove that (8) is bounded by $O(\lambda^k/(M^{k-1}B))$. The lemma below establishes a crucial fact towards that purpose.

▶ **Lemma 8.** *When $M = \Omega((\lambda \log \lambda)^{2/3})$, $|R_{c_1,c_2}| = O(M)$ holds with probability at least $1 - 1/\lambda^{\xi'}$ for any relation $R \in \mathcal{Q}$ and any colors $c_1, c_2 \in [s]$, where $\xi'$ can be an arbitrarily large constant.*

**Proof.** Let $X_1, X_2$ be the attributes in *schema*$(R)$ such that $X_1 < X_2$ (recall from Section 3.1 that we have imposed an arbitrary total order on attributes). Divide $R_{c_1,c_2}$ into four subsets:

- $R^{lo,lo}_{c_1,c_2}$, the set of tuples $\boldsymbol{t} \in R_{c_1,c_2}$ such that $\boldsymbol{t}(X_1)$ and $\boldsymbol{t}(X_2)$ are both in $\mathbf{adom}_{lo}$;
- $R^{lo,hi}_{c_1,c_2}$, the set of tuples $\boldsymbol{t} \in R_{c_1,c_2}$ such that $\boldsymbol{t}(X_1) \in \mathbf{adom}_{lo}$ but $\boldsymbol{t}(X_2) \in \mathbf{adom}_{hi}$;
- $R^{hi,lo}_{c_1,c_2}$, the set of tuples $\boldsymbol{t} \in R_{c_1,c_2}$ such that $\boldsymbol{t}(X_1) \in \mathbf{adom}_{hi}$ but $\boldsymbol{t}(X_2) \in \mathbf{adom}_{lo}$;
- $R^{hi,hi}_{c_1,c_2}$, the set of tuples $\boldsymbol{t} \in R_{c_1,c_2}$ such that $\boldsymbol{t}(X_1)$ and $\boldsymbol{t}(X_2)$ are both in $\mathbf{adom}_{hi}$.

We will show that each subset has size $O(M)$ with probability at least $1 - 1/(4\lambda^{\xi'})$, which is sufficient for proving the lemma.

The case of $R^{hi,hi}_{c_1,c_2}$ is the easiest. Every tuple $\boldsymbol{t} \in R^{hi,hi}_{c_1,c_2}$ must set $\boldsymbol{t}(X_1)$ to a high-degree value from color (a.k.a. group) $c_1$ and $\boldsymbol{t}(X_2)$ to a high-degree value from color (a.k.a. group) $c_2$. As mentioned, every group has at most $5\sqrt{M}$ values. Hence, $|R^{hi,hi}_{c_1,c_2}| \leq 25M$.

[2] Add the next high-degree value $v$ to the current group as long as doing so will not push the group's total degree over $3\lambda$. Otherwise, start a new group with only $v$; the preceding group must have a total weight as least $2\lambda$ because the degree of $v$ is bounded by $\lambda$. If the last group has a total degree less than $2\lambda$, combine it with the previous group (if it exists), which will yield a group with total weight at most $5\lambda$. This way, we guarantee that either only a single group exists, or every group has a total weight at least $2\lambda$. As each tuple can contribute one to the degrees of at most two values in $\mathbf{adom}_{hi}$, the total weights of all the groups add up to at most $2N$. The number of groups is therefore at most $2N/(2\lambda) = N/\lambda$.

To analyze $R^{lo,lo}_{c_1,c_2}$, define $R^{lo,lo}$ to be the set of tuples $\boldsymbol{t} \in R$ such that $\boldsymbol{t}(X_1)$ and $\boldsymbol{t}(X_2)$ are both low-degree values. For each tuple $\boldsymbol{t} \in R^{lo,lo}$, introduce a random variable $Z_{\boldsymbol{t}}$ that equals 1 if $\boldsymbol{t} \in R^{lo,lo}_{c_1,c_2}$, or 0 otherwise. Our function $\Gamma_1$ ensures $\Pr[Z_{\boldsymbol{t}} = 1] = 1/s^2$ with variance $\mathbf{Var}(Z_{\boldsymbol{t}}) = \frac{1}{s^2} - \frac{1}{s^4}$. Define $Z := |R^{lo,lo}_{c_1,c_2}| = \sum_{\boldsymbol{t} \in R^{lo,lo}} Z_{\boldsymbol{t}}$. We will deploy Lemma 4 to analyze how likely $Z$ can deviate significantly from $\mathbf{E}[Z]$. For this purpose, create a dependency graph $G^{lo,lo}$ as follows. Each vertex of $G^{lo,lo}$ is the variable $Z_{\boldsymbol{t}}$ of a distinct tuple $\boldsymbol{t} \in R^{lo,lo}$. Two vertices $Z_{\boldsymbol{t_1}}$ and $Z_{\boldsymbol{t_2}}$ are adjacent in $G^{lo,lo}$ if and only if tuples $\boldsymbol{t_1}$ and $\boldsymbol{t_2}$ share the same value on attribute $X_1$ or $X_2$. It is easy to verify that $G^{lo,lo}$ fulfills the independence requirement described in Section 2.3 and has a maximum vertex degree at most $2\lambda/\sqrt{M}$ by definition of low-degree value. Now, apply Lemma 4 with $\mu := \mathbf{E}[Z] = |R^{lo,lo}|/s^2$, $\sigma := |R^{lo,lo}|/s^2 > \sum_{\boldsymbol{t} \in R^{lo,lo}} \mathbf{Var}(Z_{\boldsymbol{t}})$, $\Delta := 2\lambda/\sqrt{M}$, and $\epsilon := Ms^2/|R^{lo,lo}|$. The application yields $\Pr[Z \geq 2M] \leq \exp(-\Theta(1) \cdot \frac{M^{1.5}}{\lambda})$, which is at most $1/(4\lambda^{\xi'})$ as long as $M = \Omega((\lambda \log \lambda)^{2/3})$.

The analysis of $R^{hi,lo}_{c_1,c_2}$ and $R^{lo,hi}_{c_1,c_2}$ is similar and deferred to Appendix C. ◀

We now return to our algorithm's I/O cost in (8). As mentioned before, $s^k$ is bounded by $O(\lambda^k/(M^{k-1}B))$. By the above lemma, with probability at least $1 - 1/\lambda^{\xi}$ for an arbitrarily large constant $\xi$, $\frac{s^{k-2}}{M^{r-1}B} \sum_{R \in \mathcal{Q}} \sum_{c_1,c_2 \in [s]} |R_{c_1,c_2}|^r$ is bounded by $O(\frac{s^{k-2}}{M^{r-1}B} \cdot s^2 M^r) = O(\lambda^k/(M^{k-1}B))$, applying the value of $s$ in (4). We thus complete the proof of Lemma 5.

## 4 An EM Algorithm for Arbitrary Binary Joins

This section serves as a proof of:

▶ **Theorem 9.** *Consider a binary join $\mathcal{Q}$ whose relations have distinct schemas. Let $\mathcal{G} := (\mathcal{X}, \mathcal{E})$ be the schema graph of $\mathcal{Q}$, $k := |\mathcal{X}|$, and $N := \sum_{R \in \mathcal{Q}} |R|$. There is an algorithm in EM that, with high probability, emits every tuple of join($\mathcal{Q}$) exactly once in $O(\frac{N^{k/2}}{M^{k/2-1}B} \log_{M/B} \frac{N}{B} + \frac{N^{\rho}}{M^{\rho-1}B})$ I/Os, where $\rho$ is the fractional edge covering number of $\mathcal{G}$, $M$ is the number of words in memory, and $B$ is the number of words in a disk block.*

Theorem 1 follows from the above result and Lemma 3. Our solution can be regarded as an efficient EM translation of an MPC algorithm in [23]. The non-trivial part is to show that the I/O cost is as claimed. We will achieve the purpose by utilizing a mathematical property of binary joins recently revealed by the *isolated cartesian product theorem* [23].

We consider $|\mathcal{X}| \geq 3$; otherwise, $\mathcal{Q}$ has only one relation and the tuples of *join*($\mathcal{Q}$) can be emitted in $O(N/B)$ I/Os. For each hyperedge $e \in \mathcal{E}$, we will use $R_e$ to denote the (only) relation in $\mathcal{Q}$ with schema $e$. As before, let **adom** be the combined active domain of $\mathcal{Q}$. Henceforth, we will fix

$$\lambda := \sqrt{NM}. \tag{11}$$

### 4.1 Residual Joins

We say that a value $v \in$ **adom** is *heavy* if its degree is at least $\lambda$, or *light* otherwise. The number of heavy values is $O(N/\lambda)$. Let $\mathcal{H}$ be any subset of $\mathcal{X} :=$ *schema*($\mathcal{Q}$). A *configuration* of $\mathcal{H}$ is defined as a tuple $\boldsymbol{\eta}$ over $\mathcal{H}$ whose $\boldsymbol{\eta}(X)$ is heavy for every attribute $X \in \mathcal{H}$. Let *config*($\mathcal{H}$) be the set of all configurations $\boldsymbol{\eta}$ of $\mathcal{H}$ satisfying

$$\boldsymbol{\eta}[e] \in R_e \text{ for every } e \in \mathcal{E} \text{ such that } e \subseteq \mathcal{H}. \tag{12}$$

It is clear that

$$|config(\mathcal{H})| = O((N/\lambda)^{|\mathcal{H}|}) = O((N/M)^{|\mathcal{H}|/2}) \tag{13}$$

Fix any configuration $\boldsymbol{\eta} \in config(\mathcal{H})$. For each hyperedge $e \in \mathcal{E}$ satisfying $e \setminus \mathcal{H} \neq \emptyset$, we define relation $R_e(\boldsymbol{\eta})$ to be a subset of $R_e$ that includes every tuple $\boldsymbol{t} \in R_e$ satisfying (i) $\boldsymbol{t}(X) = \boldsymbol{\eta}(X)$ for all $X \in e \cap \mathcal{H}$; (ii) $\boldsymbol{t}(X)$ is light for every $X \in e \setminus \mathcal{H}$. Note that if $e \cap \mathcal{H} = \emptyset$, then $R_e(\boldsymbol{\eta}) = R_e$. Every such hyperedge $e$ has a *residual relation* $R'_e(\boldsymbol{\eta})$ defined as

$$R'_e(\boldsymbol{\eta}) := \Pi_{e \setminus \mathcal{H}}(R_e(\boldsymbol{\eta})). \tag{14}$$

The configuration $\boldsymbol{\eta}$ induces a *residual join* $\mathcal{Q}'(\boldsymbol{\eta})$ formalized as

$$\mathcal{Q}'(\boldsymbol{\eta}) := \{R'_e(\boldsymbol{\eta}) \mid e \in \mathcal{E}, e \setminus \mathcal{H} \neq \emptyset\} \tag{15}$$

whose input size is

$$N_{\boldsymbol{\eta}} := \sum_{R \in \mathcal{Q}'(\boldsymbol{\eta})} |R|. \tag{16}$$

▶ **Example 10.** Figure 1(a) shows the schema graph $\mathcal{G} := (\mathcal{X}, \mathcal{E})$ of a join $\mathcal{Q}$, where $\mathcal{X} := \{\texttt{A}, \texttt{B}, ..., \texttt{L}\}$ and the hyperedges in $\mathcal{E}$ are represented as ellipses. Set $\mathcal{H} := \{\texttt{E}, \texttt{F}, \texttt{I}\}$ and consider the configuration $\boldsymbol{\eta}$ with heavy values $\boldsymbol{\eta}(\texttt{E}) := \texttt{e}$, $\boldsymbol{\eta}(\texttt{F}) := \texttt{f}$, and $\boldsymbol{\eta}(\texttt{I}) := \texttt{i}$. The figure illustrates $\boldsymbol{\eta}$ by darkening vertices E, F, and I. Suppose $\boldsymbol{\eta} \in config(\mathcal{H})$, which means that $\boldsymbol{\eta}[\texttt{EF}]$ is a tuple in $R_{\texttt{EF}}$, and $\boldsymbol{\eta}[\texttt{EI}]$ is a tuple in $R_{\texttt{EI}}$. Relation $R_{\texttt{DE}}(\boldsymbol{\eta})$ includes all such tuples $\boldsymbol{t} \in R_{\texttt{DE}}$ that use value e for $\boldsymbol{t}(\texttt{E})$ and a light value for $\boldsymbol{t}(\texttt{D})$. The residual relation $R'_{\texttt{DE}}(\boldsymbol{\eta})$ is a unary relation that is the projection of $R_{\texttt{DE}}(\boldsymbol{\eta})$ on D. The reader can verify that the residual relations $R'_{\texttt{AD}}(\boldsymbol{\eta})$, $R'_{\texttt{DG}}(\boldsymbol{\eta})$, and $R'_{\texttt{DH}}(\boldsymbol{\eta})$ are identical to $R_{\texttt{AD}}$, $R_{\texttt{DG}}$, and $R_{\texttt{DH}}$, respectively. Edges EI and EF define no residual relations. ⌟

The lemma below, proved in Appendix D, will be useful in our analysis later.

▶ **Lemma 11.** *The statements below are true for every $\mathcal{H} \subseteq \mathcal{X}$:*

- $\sum_{\boldsymbol{\eta} \in config(\mathcal{H})} N_{\boldsymbol{\eta}} = O(N^{k/2}/M^{k/2-1})$;
- *in $O(\frac{N^{k/2}}{M^{k/2-1}B} \log_{M/B} \frac{N}{B})$ I/Os, we can ensure the following for all $\boldsymbol{\eta} \in config(\mathcal{H})$: each relation of $\mathcal{Q}'(\boldsymbol{\eta})$ is stored in consecutive disk blocks.*

It is easy to verify that

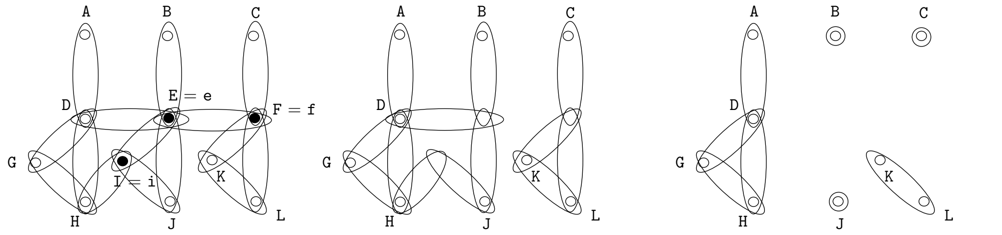
$$join(\mathcal{Q}) = \bigcup_{\mathcal{H}} \Big( \bigcup_{\boldsymbol{\eta} \in config(\mathcal{H})} join(\mathcal{Q}'(\boldsymbol{\eta})) \times \{\boldsymbol{\eta}\} \Big). \tag{17}$$

Next, we will present an algorithm that, given any $\mathcal{H} \subseteq \mathcal{X}$, emits each tuple of $\bigcup_{\boldsymbol{\eta} \in config(\mathcal{H})} join(\mathcal{Q}'(\boldsymbol{\eta})) \times \{\boldsymbol{\eta}\}$ exactly once. Executing the algorithm for all the $2^{|\mathcal{X}|} = O(1)$ subsets $\mathcal{H} \subseteq \mathcal{X}$ emits the entire $join(\mathcal{Q})$, with no tuple emitted twice.

## 4.2 Simplifying Residual Joins

Fix an arbitrary $\mathcal{H} \subseteq \mathcal{X}$ and define $\mathcal{L} := \mathcal{X} \setminus \mathcal{H}$. We will call the attributes in $\mathcal{H}$ and $\mathcal{L}$ as *heavy* and *light* attributes, respectively. An attribute $X \in \mathcal{X}$ is a *border* attribute if it is adjacent to at least one heavy attribute.

By removing all the heavy attributes from $\mathcal{G} := (\mathcal{X}, \mathcal{E})$, we obtain a *residual graph* $\mathcal{G}' := (\mathcal{X}', \mathcal{E}')$ where $\mathcal{X}' := \mathcal{X} \setminus \mathcal{H}$ and $\mathcal{E}' := \{e \setminus \mathcal{H} \mid e \in \mathcal{E} \text{ and } e \setminus \mathcal{H} \neq \emptyset\}$. An attribute $X \in \mathcal{X}'$ is *isolated* if it is adjacent to no other attributes in $\mathcal{G}'$. Denote by $\mathcal{I}$ the set of isolated attributes. An isolated attribute is always a border attribute, but the reverse is not true.



(a) Schema graph $\mathcal{G}$ of $\mathcal{Q}$. (b) Residual graph $\mathcal{G}'$. (c) Simplified residual graph. $\mathcal{G}''$

**Figure 1** A running example.

▶ **Example 12.** Figure 1(b) shows the residual graph $\mathcal{G}' := (\mathcal{X}', \mathcal{E}')$ of the schema graph in Figure 1(a), after removing $\mathtt{E}$, $\mathtt{F}$, and $\mathtt{I}$, as well as the hyperedges that have become empty. The border attributes are $\mathtt{B}$, $\mathtt{C}$, $\mathtt{D}$, $\mathtt{H}$, $\mathtt{J}$, $\mathtt{K}$, and $\mathtt{L}$, while the set of isolated attributes is $\mathcal{I} = \{\mathtt{B}, \mathtt{C}, \mathtt{J}\}$. ⌟

Fix any configuration $\boldsymbol{\eta} \in \mathit{config}(\mathcal{H})$. $\mathcal{G}'$ is the schema graph of $\mathcal{Q}'(\boldsymbol{\eta})$ (true for all $\boldsymbol{\eta}$). For each border attribute $X \in \mathcal{X}'$, $\mathcal{G}'$ has at least one relation with schema $\{X\}$. Define:

$$R''_X(\boldsymbol{\eta}) := \bigcap_{e \in \mathcal{E}' : X \in e} \Pi_X(R'_e(\boldsymbol{\eta})) \tag{18}$$

where $R'_e(\boldsymbol{\eta})$ is defined in (14). Only the values in $R''_X(\boldsymbol{\eta})$ can contribute to $\mathit{join}(\mathcal{Q}'(\boldsymbol{\eta}))$.

▶ **Example 13.** Continuing on Example 12, consider again the residual graph $\mathcal{G}' := (\mathcal{X}', \mathcal{E}')$ in Figure 1(b). Set $X$ to the border attribute $\mathtt{D}$. The residual join $\mathcal{Q}'(\boldsymbol{\eta})$ has four relations whose schemas contain $\mathtt{D}$: unary relation $R'_{\mathtt{DE}}(\boldsymbol{\eta})$ and binary relations $R'_{\mathtt{AD}}(\boldsymbol{\eta})$, $R'_{\mathtt{DG}}(\boldsymbol{\eta})$, and $R'_{\mathtt{DH}}(\boldsymbol{\eta})$, all of which were explained in Example 10. $R''_{\mathtt{D}}(\boldsymbol{\eta})$ equals the intersection of $\Pi_{\mathtt{D}}(R'_{\mathtt{DE}}(\boldsymbol{\eta})) = R'_{\mathtt{DE}}(\boldsymbol{\eta})$, $\Pi_{\mathtt{D}}(R'_{\mathtt{AD}}(\boldsymbol{\eta}))$, $\Pi_{\mathtt{D}}(R'_{\mathtt{DG}}(\boldsymbol{\eta}))$, and $\Pi_{\mathtt{D}}(R'_{\mathtt{DH}}(\boldsymbol{\eta}))$. As another example, set $X$ to the isolated attribute $\mathtt{J}$, which appears in two hyperedges in $\mathcal{G}'$, both unary. $R''_{\mathtt{J}}(\boldsymbol{\eta})$ is the intersection of $\Pi_{\mathtt{J}}(R'_{\mathtt{IJ}}(\boldsymbol{\eta})) = R'_{\mathtt{IJ}}(\boldsymbol{\eta})$ and $\Pi_{\mathtt{J}}(R'_{\mathtt{EJ}}(\boldsymbol{\eta})) = R'_{\mathtt{EJ}}(\boldsymbol{\eta})$. ⌟

For every binary $e := \{X_1, X_2\} \in \mathcal{E}'$, we define a relation $R''_e(\boldsymbol{\eta}) \subseteq R'_e(\boldsymbol{\eta})$ as follows:

- If $X_1$ and $X_2$ are both border attributes, then $R''_e(\boldsymbol{\eta}) := R'_e(\boldsymbol{\eta}) \bowtie R''_{X_1}(\boldsymbol{\eta}) \bowtie R''_{X_2}(\boldsymbol{\eta})$;
- If only $X_1$ is a border attribute, then $R''_e(\boldsymbol{\eta}) := R'_e(\boldsymbol{\eta}) \bowtie R''_{X_1}(\boldsymbol{\eta})$;
- If only $X_2$ is a border attribute, then $R''_e(\boldsymbol{\eta}) := R'_e(\boldsymbol{\eta}) \bowtie R''_{X_2}(\boldsymbol{\eta})$;
- If neither is a border attribute, then $R''_e(\boldsymbol{\eta}) := R'_e(\boldsymbol{\eta})$.

Only the tuples in $R''_e(\boldsymbol{\eta})$ can contribute to $\mathit{join}(\mathcal{Q}'(\boldsymbol{\eta}))$.

▶ **Example 14.** Continuing on Example 13, if we set the hyperedge $e$ to $\mathtt{DH}$ in $\mathcal{E}'$, then $R''_e(\boldsymbol{\eta})$ contains only the tuples $\boldsymbol{t} \in R'_{\mathtt{DH}}(\boldsymbol{\eta})$ with $\boldsymbol{t}(\mathtt{D}) \in R''_{\mathtt{D}}(\boldsymbol{\eta})$ and $\boldsymbol{t}(\mathtt{H}) \in R''_{\mathtt{H}}(\boldsymbol{\eta})$. For another example, if $e := \mathtt{GH}$, then $R''_e(\boldsymbol{\eta})$ contains only the tuples $\boldsymbol{t} \in R'_{\mathtt{GH}}(\boldsymbol{\eta})$ with $\boldsymbol{t}(\mathtt{H}) \in R''_{\mathtt{H}}(\boldsymbol{\eta})$. ⌟

We can now define a *simplified residual join* induced by $\boldsymbol{\eta}$:

$$\mathcal{Q}''(\boldsymbol{\eta}) \quad := \quad \{R''_e(\boldsymbol{\eta}) \mid \text{binary } e \in \mathcal{E}'\} \cup \{R''_X(\boldsymbol{\eta}) \mid X \in \mathcal{I}\}. \tag{19}$$

Define $\mathcal{G}'' := (\mathcal{X}'', \mathcal{E}'')$ – the *simplified residual graph* – as the hypergraph where $\mathcal{X}'' := \mathcal{X}'$, and $\mathcal{E}''$ includes (i) all the binary edges in $\mathcal{E}'$ and (ii) a unary edge $\{X\}$ for every isolated attribute $X \in \mathcal{I}$. $\mathcal{G}''$ is the schema graph of $\mathcal{Q}''(\boldsymbol{\eta})$ for all $\boldsymbol{\eta} \in \mathit{config}(\mathcal{H})$.

▶ **Example 15.** Continuing on Ex.14, Figure 1(c) shows the simplified residual graph $\mathcal{G}''$. ⌟

It is rudimentary to verify several facts about the join $\mathcal{Q}''(\boldsymbol{\eta})$ in (19). First, its input size is at most that of $\mathcal{Q}'(\boldsymbol{\eta})$, which is $N_{\boldsymbol{\eta}}$; see (16). Second, its relations have distinct schemas. Third, as each relation of $\mathcal{Q}'(\boldsymbol{\eta})$ has been stored in consecutive disk blocks (Lemma 11), we can achieve the same for $\mathcal{Q}''(\boldsymbol{\eta})$ in $O(\text{sort}(N_{\boldsymbol{\eta}}))$ I/Os; doing so for all $\boldsymbol{\eta} \in \mathit{config}(\mathcal{H})$ requires

$$\sum_{\boldsymbol{\eta} \in \mathit{config}(\mathcal{H})} O(\text{sort}(N_{\boldsymbol{\eta}})) = O\Big(|\mathit{config}(\mathcal{H})| + \sum_{\boldsymbol{\eta} \in \mathit{config}(\mathcal{H})} \frac{N_{\boldsymbol{\eta}}}{B} \log_{\frac{M}{B}} \frac{N}{B}\Big) = O\Big(\frac{N^{k/2}}{M^{k/2-1}B} \log_{\frac{M}{B}} \frac{N}{B}\Big)$$

I/Os, where the last equality used (13) and the first bullet of Lemma 11. Fourth, $\mathit{join}(\mathcal{Q}''(\boldsymbol{\eta})) = \mathit{join}(\mathcal{Q}'(\boldsymbol{\eta}))$; hence, to process the original join $\mathcal{Q}$ in Theorem 9, it suffices to emit every result tuple of $\mathit{join}(\mathcal{Q}''(\boldsymbol{\eta}))$ exactly once, for all $\boldsymbol{\eta} \in \mathit{config}(\mathcal{H})$ and $\mathcal{H} \subseteq \mathcal{X}$.

## 4.3 Processing Simplified Residual Joins

Fix an arbitrary $\mathcal{H} \subseteq \mathcal{X}$, and define $\mathcal{L}$, $\mathcal{I}$, and $\mathcal{G}'' := (\mathcal{X}'', \mathcal{E}'')$ as in the previous subsection. Given an arbitrary $\boldsymbol{\eta} \in \mathit{config}(\mathcal{H})$, we will present an algorithm for processing the simplified residual join $\mathcal{Q}''(\boldsymbol{\eta})$. First, let us divide $\mathcal{Q}''(\boldsymbol{\eta})$ into $\mathcal{Q}''_{bin}(\boldsymbol{\eta}) := \{R''_e(\boldsymbol{\eta}) \mid \text{binary } e \in \mathcal{E}''\}$ and $\mathcal{Q}''_{iso}(\boldsymbol{\eta}) := \{R''_X(\boldsymbol{\eta}) \mid X \in \mathcal{I}\}$. It is clear that $\mathit{join}(\mathcal{Q}''(\boldsymbol{\eta})) = \mathit{join}(\mathcal{Q}''_{bin}(\boldsymbol{\eta})) \times \mathit{join}(\mathcal{Q}''_{iso}(\boldsymbol{\eta}))$.

▶ **Example 16.** Continuing on Example 15, $\mathcal{Q}''_{bin}(\boldsymbol{\eta})$ includes relations $R''_{\mathtt{AD}}(\boldsymbol{\eta})$, $R''_{\mathtt{DG}}(\boldsymbol{\eta})$, $R''_{\mathtt{DH}}(\boldsymbol{\eta})$, $R''_{\mathtt{GH}}(\boldsymbol{\eta})$, and $R''_{\mathtt{KL}}(\boldsymbol{\eta})$, while $\mathcal{Q}''_{iso}(\boldsymbol{\eta})$ includes $R''_{\mathtt{B}}(\boldsymbol{\eta})$, $R''_{\mathtt{C}}(\boldsymbol{\eta})$, and $R''_{\mathtt{J}}(\boldsymbol{\eta})$. ⌟

Observe that $\mathcal{Q}''_{bin}(\boldsymbol{\eta})$ is a binary join whose scheme graph has $|\mathcal{L} \setminus \mathcal{I}|$ attributes; furthermore, the relations of $\mathcal{Q}''_{bin}(\boldsymbol{\eta})$ contain only light values, implying that $\mathcal{Q}''_{bin}(\boldsymbol{\eta})$ has a degree at most $\lambda$. On the other hand, $\mathcal{Q}''_{iso}(\boldsymbol{\eta})$ is merely the cartesian product of all the (unary) relations therein. We process $\mathcal{Q}''(\boldsymbol{\eta})$ by integrating BNL with the algorithm in Lemma 5. Specifically, we chop each relation $R''_X(\boldsymbol{\eta}) \in \mathcal{Q}''_{iso}(\boldsymbol{\eta})$ into $O(\lceil |R''_X(\boldsymbol{\eta})|/M \rceil)$ disjoint subsets – called *chunks* – each of which fits in $M/(k+1)$ words. Define a *chunk combination* as a collection of $|\mathcal{Q}''_{iso}|$ chunks, each from a distinct relation in $\mathcal{Q}''_{iso}$. For every chunk combination, load the $|\mathcal{Q}''_{iso}|$ corresponding chunks in memory and then use the remaining at least $M/(k+1) = \Omega(M)$ words of memory to run the algorithm in Lemma 5. Every time the algorithm emits a tuple $\boldsymbol{t} \in \mathit{join}(\mathcal{Q}''_{bin}(\boldsymbol{\eta}))$, we emit all the tuples of $\mathit{join}(\mathcal{Q}''(\boldsymbol{\eta}))$ that can be produced by $\boldsymbol{t}$ and the memory-resident chunk data (this requires only CPU computation). As there are at most $O(\prod_{X \in \mathcal{I}} \lceil \frac{|R''_X(\boldsymbol{\eta})|}{M} \rceil)$ chunk combinations, the total I/O cost spent on $\mathcal{Q}''(\boldsymbol{\eta})$ is $O(\prod_{X \in \mathcal{I}} \lceil \frac{|R''_X(\boldsymbol{\eta})|}{M} \rceil \cdot \frac{\lambda^{|\mathcal{L} \setminus \mathcal{I}|}}{M^{|\mathcal{L} \setminus \mathcal{I}|-1}B})$ with probability at least $1 - 1/\lambda^{\xi'}$ for an arbitrarily large constant $\xi'$.

Processing all $\boldsymbol{\eta} \in \mathit{config}(\mathcal{H})$ in the above manner incurs a total I/O cost of

$$O\Big(\frac{\lambda^{|\mathcal{L} \setminus \mathcal{I}|}}{M^{|\mathcal{L} \setminus \mathcal{I}|-1}B} \sum_{\boldsymbol{\eta} \in \mathit{config}(\mathcal{H})} \prod_{X \in \mathcal{I}} \Big\lceil \frac{|R''_X(\boldsymbol{\eta})|}{M} \Big\rceil\Big) = O\Big(\Big(\frac{N}{M}\Big)^{\frac{|\mathcal{L} \setminus \mathcal{I}|}{2}} \frac{M}{B} \sum_{\boldsymbol{\eta} \in \mathit{config}(\mathcal{H})} \prod_{X \in \mathcal{I}} \Big(\frac{|R''_X(\boldsymbol{\eta})|}{M} + 1\Big)\Big) \tag{20}$$

where the derivation applied the definition of $\lambda$ in (11).

▶ **Lemma 17.** $\sum_{\boldsymbol{\eta} \in \mathit{config}(\mathcal{H})} \prod_{X \in \mathcal{I}} \Big(\frac{|R''_X(\boldsymbol{\eta})|}{M} + 1\Big) = O((\frac{N}{M})^{\rho - \frac{|\mathcal{L} \setminus \mathcal{I}|}{2}})$, *where $\rho$ is the fractional edge covering number of the join $\mathcal{Q}$ stated in Theorem 9.*

**Proof.** For each $\boldsymbol{\eta} \in \mathit{config}(\mathcal{H})$ and any non-empty $\mathcal{J} \subseteq \mathcal{I}$, define $\mathit{CPsize}_{\mathcal{J}}(\boldsymbol{\eta}) := \prod_{X \in \mathcal{J}} |R''_X(\boldsymbol{\eta})|$. Crucially, observe that

$$\prod_{X \in \mathcal{I}} \Big(\frac{|R''_X(\boldsymbol{\eta})|}{M} + 1\Big) = 1 + \sum_{\mathcal{J} \subseteq \mathcal{I} : \mathcal{J} \neq \emptyset} \prod_{X \in \mathcal{J}} \frac{|R''_X(\boldsymbol{\eta})|}{M} = 1 + \sum_{\mathcal{J} \subseteq \mathcal{I} : \mathcal{J} \neq \emptyset} \frac{\mathit{CPsize}_{\mathcal{J}}(\boldsymbol{\eta})}{M^{|\mathcal{J}|}}.$$

Next, we will show that $\sum_{\boldsymbol{\eta} \in \mathit{config}(\mathcal{H})} 1$ and the term $\sum_{\boldsymbol{\eta} \in \mathit{config}(\mathcal{H})} \frac{\mathit{CPsize}_{\mathcal{J}}(\boldsymbol{\eta})}{M^{|\mathcal{J}|}}$ of each non-empty $\mathcal{J} \subseteq \mathcal{I}$ are all bounded by $O((N/M)^{\rho - \frac{|\mathcal{L} \setminus \mathcal{I}|}{2}})$, which will then establish Lemma 17 because there are $2^{|\mathcal{I}|} - 1 = O(1)$ choices for $\mathcal{J}$.

From (11) and (13), we know $\sum_{\boldsymbol{\eta} \in \mathit{config}(\mathcal{H})} 1 = |\mathit{config}(\mathcal{H})| = O((N/M)^{|\mathcal{H}|/2})$. As a well-known fact [36], the value of $\rho$ is at least $|\mathcal{X}|/2$, where $|\mathcal{X}|$ is the number of attributes in the schema graph of $\mathcal{Q}$. Because $|\mathcal{X}| = |\mathcal{H}| + |\mathcal{L}| \geq |\mathcal{H}| + |\mathcal{L} \setminus \mathcal{I}|$, we know $|\mathcal{H}|/2 \leq (|\mathcal{X}| - |\mathcal{L} \setminus \mathcal{I}|)/2 \leq \rho - \frac{|\mathcal{L} \setminus \mathcal{I}|}{2}$.

It remains to analyze the term $\sum_{\boldsymbol{\eta} \in \mathit{config}(\mathcal{H})} \frac{\mathit{CPsize}_{\mathcal{J}}(\boldsymbol{\eta})}{M^{|\mathcal{J}|}}$ for each non-empty $\mathcal{J} \subseteq \mathcal{I}$. We apply Lemma 11 of [23] – a weaker version of the isolated cartesian product theorem in [23] – which states $\sum_{\boldsymbol{\eta} \in \mathit{config}(\mathcal{H})} \mathit{CPsize}_{\mathcal{J}}(\boldsymbol{\eta}) = O((N/M)^{\rho - (|\mathcal{J}| + |\mathcal{L}|)/2} \cdot N^{|\mathcal{J}|})$. This yields

$$\sum_{\boldsymbol{\eta} \in \mathit{config}(\mathcal{H})} \frac{\mathit{CPsize}_{\mathcal{J}}(\boldsymbol{\eta})}{M^{|\mathcal{J}|}} = O\Big(\frac{(\frac{N}{M})^{\rho - \frac{|\mathcal{J}| + |\mathcal{L}|}{2}} \cdot N^{|\mathcal{J}|}}{M^{|\mathcal{J}|}}\Big) = O\Big(\Big(\frac{N}{M}\Big)^{\rho - \frac{|\mathcal{L} \setminus \mathcal{J}|}{2}}\Big) = O\Big(\Big(\frac{N}{M}\Big)^{\rho - \frac{|\mathcal{L} \setminus \mathcal{I}|}{2}}\Big)$$

where the derivation used the fact $\mathcal{J} \subseteq \mathcal{I} \subseteq \mathcal{L}$. ◀

Plugging the result of Lemma 17 into (20), we know that the simplified residual joins induced by all the $\boldsymbol{\eta} \in \mathit{config}(\mathcal{H})$ can be processed using $O(N^\rho/(M^{\rho-1}B))$ I/Os in total with probability at least $1 - |\mathit{config}(\mathcal{H})|/\lambda^{\xi'}$, namely, w.h.p. if we set $\xi'$ sufficiently large. Repeating the algorithm for all the $O(1)$ subsets $\mathcal{H} \subseteq \mathcal{X}$ settles the the original join $\mathcal{Q}$ in $O(N^\rho/(M^{\rho-1}B))$ w.h.p.. We thus complete the proof of Theorem 9.

## 5 Conclusions

This paper has presented new progress in designing I/O-efficient algorithms for subgraph enumeration, where the objective is to find all the occurrences of a pattern graph $Q$ having $k = O(1)$ vertices in a data graph $G := (V, E)$. Our algorithm guarantees an I/O complexity $O(\frac{|E|^{k/2}}{M^{k/2-1}B} \log_{M/B} \frac{|E|}{B} + \frac{|E|^\rho}{M^{\rho-1}B})$ with high probability, where $\rho \geq k/2$ is the fractional edge covering number of $Q$, $M$ is the number of words in memory, and $B$ is the number of words in a disk block. The algorithm matches an existing I/O lower bound of $\Omega(\frac{|E|^\rho}{M^{\rho-1}B})$ on the class of indivisible algorithms whenever $\rho > k/2$ or $M/B \geq (|E|/B)^\epsilon$ for any constant $\epsilon > 0$. The main open problem left behind by our work is to eliminate the $\log_{M/B}(|E|/B)$ factor altogether, thus obtaining an algorithm that matches the lower bound in all cases.

## References

1 Alok Aggarwal and Jeffrey Scott Vitter. The input/output complexity of sorting and related problems. *Communications of the ACM (CACM)*, 31(9):1116–1127, 1988.

2 Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *Journal of the ACM (JACM)*, 42(4):844–856, 1995.

3 Noga Alon, Raphael Yuster, and Uri Zwick. Finding and counting given length cycles. *Algorithmica*, 17(3):209–223, 1997.

4 Kaleb Alway, Eric Blais, and Semih Salihoglu. Box covers and domain orderings for beyond worst-case join processing. In *Proceedings of International Conference on Database Theory (ICDT)*, pages 3:1–3:23, 2021.

5 Suman K. Bera, Noujan Pashanasangi, and C. Seshadhri. Near-linear time homomorphism counting in bounded degeneracy graphs: The barrier of long induced cycles. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2315–2332, 2021.

6 Andreas Bjorklund, Petteri Kaski, and Lukasz Kowalik. Counting thin subgraphs via packings faster than meet-in-the-middle time. *ACM Transactions on Algorithms*, 13(4):48:1–48:26, 2017.

7 Andreas Bjorklund, Rasmus Pagh, Virginia Vassilevska Williams, and Uri Zwick. Listing triangles. In *Proceedings of International Colloquium on Automata, Languages and Programming (ICALP)*, pages 223–234, 2014.

8 N. Chiba and T. Nishizeki. Arboricity and subgraph listing algorithms. *SIAM Journal of Computing*, 14(1):210–223, 1985.

9 Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of ACM Symposium on Theory of Computing (STOC)*, pages 151–158, 1971.

10 Radu Curticapean, Holger Dell, and Dániel Marx. Homomorphisms are a good basis for counting small subgraphs. In *Proceedings of ACM Symposium on Theory of Computing (STOC)*, pages 210–223, 2017.

11 David Eppstein. Arboricity and bipartite subgraph listing algorithms. *Information Processing Letters (IPL)*, 51(4):207–211, 1994.

12 David Eppstein. Subgraph isomorphism in planar graphs and related problems. *J. Graph Algorithms Appl.*, 3(3):1–27, 1999.

13 David Eppstein, Michael T. Goodrich, Michael Mitzenmacher, and Manuel R. Torres. 2-3 cuckoo filters for faster triangle listing and set intersection. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, pages 247–260, 2017.

14 David Eppstein, Maarten Loffler, and Darren Strash. Listing all maximal cliques in sparse graphs in near-optimal time. In *International Symposium on Algorithms and Computation (ISAAC)*, volume 6506, pages 403–414, 2010.

15 Peter Floderus, Miroslaw Kowaluk, Andrzej Lingas, and Eva-Marta Lundell. Detecting and counting small pattern graphs. *SIAM J. Discret. Math.*, 29(3):1322–1339, 2015.

16 Fedor V. Fomin, Daniel Lokshtanov, Venkatesh Raman, Saket Saurabh, and B. V. Raghavendra Rao. Faster algorithms for finding and counting subgraphs. *Journal of Computer and System Sciences (JCSS)*, 78(3):698–706, 2012.

17 Pierre-Louis Giscard, Nils M. Kriege, and Richard C. Wilson. A general purpose algorithm for counting simple cycles and simple paths of any length. *Algorithmica*, 81(7):2716–2737, 2019.

18 Chinh T. Hoang, Marcin Kaminski, Joe Sawada, and R. Sritharan. Finding and listing induced paths and cycles. *Discrete Applied Mathematics*, 161(4-5):633–641, 2013.

19 Xiao Hu and Ke Yi. Towards a worst-case i/o-optimal algorithm for acyclic joins. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, pages 135–150, 2016.

20 Xiaocheng Hu, Miao Qiao, and Yufei Tao. I/O-efficient join dependency testing, loomis-whitney join, and triangle enumeration. *Journal of Computer and System Sciences (JCSS)*, 82(8):1300–1315, 2016.

21 Xiaocheng Hu, Yufei Tao, and Chin-Wan Chung. I/O-Efficient Algorithms on Triangle Listing and Counting. *ACM Transactions on Database Systems (TODS)*, 39(4):27:1–27:30, 2014.

22 Svante Janson. Large deviations for sums of partly dependent random variables. *Random Structures and Algorithms*, 24(3):234–248, 2004.

23 Bas Ketsman, Dan Suciu, and Yufei Tao. A near-optimal parallel algorithm for joining binary relations. *Log. Methods Comput. Sci.*, 18(2), 2022.

24 Mahmoud Abo Khamis, Hung Q. Ngo, Christopher Ré, and Atri Rudra. Joins via geometric resolutions: Worst-case and beyond. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, pages 213–228, 2015.

25 Mahmoud Abo Khamis, Hung Q. Ngo, Christopher Re, and Atri Rudra. Joins via geometric resolutions: Worst case and beyond. *ACM Transactions on Database Systems (TODS)*, 41(4):22:1–22:45, 2016.

26 Ton Kloks, Dieter Kratsch, and Haiko Müller. Finding and counting small induced subgraphs efficiently. *Information Processing Letters (IPL)*, 74(3-4):115–121, 2000.

27 Paraschos Koutris, Paul Beame, and Dan Suciu. Worst-case optimal algorithms for parallel query processing. In *Proceedings of International Conference on Database Theory (ICDT)*, pages 8:1–8:18, 2016.

28 Gonzalo Navarro, Juan L. Reutter, and Javiel Rojas-Ledesma. Optimal joins using compact data structures. In *Proceedings of International Conference on Database Theory (ICDT)*, volume 155, pages 21:1–21:21, 2020.

29 Jaroslav Nesetril and Svatopluk Poljak. On the complexity of the subgraph problem. *Commentationes Mathematicae Universitatis Carolinae*, 26(2):415–419, 1985.

30 Hung Q. Ngo, Dung T. Nguyen, Christopher Re, and Atri Rudra. Beyond worst-case analysis for joins with minesweeper. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, pages 234–245, 2014.

31 Hung Q. Ngo, Ely Porat, Christopher Ré, and Atri Rudra. Worst-Case Optimal Join Algorithms: [Extended Abstract]. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, pages 37–48, 2012.

32 Hung Q. Ngo, Ely Porat, Christopher Re, and Atri Rudra. Worst-case optimal join algorithms. *Journal of the ACM (JACM)*, 65(3):16:1–16:40, 2018.

33 Hung Q. Ngo, Christopher Re, and Atri Rudra. Skew strikes back: new developments in the theory of join algorithms. *SIGMOD Rec.*, 42(4):5–16, 2013.

34 Anna Pagh and Rasmus Pagh. Scalable computation of acyclic joins. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, pages 225–232, 2006.

35 Rasmus Pagh and Francesco Silvestri. The input/output complexity of triangle enumeration. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, pages 224–233, 2014.

36 Edward R. Scheinerman and Daniel H. Ullman. *Fractional Graph Theory: A Rational Approach to the Theory of Graphs.* Wiley, New York, 1997.

37 Todd L. Veldhuizen. Triejoin: A simple, worst-case optimal join algorithm. In *Proceedings of International Conference on Database Theory (ICDT)*, pages 96–106, 2014.

38 Virginia Vassilevska Williams and Ryan Williams. Finding, minimizing, and counting weighted subgraphs. *SIAM Journal of Computing*, 42(3):831–854, 2013.

## A Proof of Lemma 3

Every tuple $\boldsymbol{t} \in join(\mathcal{Q})$ defines $|E_Q|$ edges in $G$ as follows: for every relation $R \in \mathcal{Q}$ with $schema(R) := \{X_1, X_2\}$, $\boldsymbol{t}$ defines an edge $\{\boldsymbol{t}(X_1), \boldsymbol{t}(X_2)\}$ in $G$. In general, for every occurrence $G_{sub}$ of $Q$ in $G$, there must be at least one tuple $\boldsymbol{t} \in join(\mathcal{Q})$ defining exactly the $|E_Q|$ edges in $G_{sub}$. The reverse, however, is not true: the $|E_Q|$ edges of a tuple $\boldsymbol{t} \in join(\mathcal{Q})$ may not always induce a subgraph of $G$ isomorphic to $Q$.

Every time a tuple $\boldsymbol{t} \in join(\mathcal{Q})$ is emitted, all the $|E_Q|$ edges defined by $\boldsymbol{t}$ are memory-resident. Hence, we can check for free if those edges induce a subgraph of $G$ isomorphic to $Q$. If the answer is negative, we ignore $\boldsymbol{t}$. Next, let us focus on the scenario where the $|E_Q|$ edges *do* induce a subgraph $G_{sub}$ isomorphic to $Q$. If we always emit $G_{sub}$ in such a case, we may risk emitting $G_{sub}$ multiple times because $join(\mathcal{Q})$ can contain multiple tuples all of which define the edges of $G_{sub}$. Let $S$ be the set of those tuples. To avoid duplicate emissions, a simple strategy is to impose an (arbitrary) ordering on $S$, and emit $G_{sub}$ only if $\boldsymbol{t}$ is the smallest tuple in $S$ according to the ordering. Whether $\boldsymbol{t}$ is indeed the smallest can be checked in memory with no extra I/Os. This is because $S$ is determined by the $|E_Q|$ edges defined by $\boldsymbol{t}$ and, hence, can be enumerated in memory for free.

It is clear from the above discussion that, apart from the initial construction of $\mathcal{Q}$ which incurs $O(\lceil |E|/B \rceil)$ I/Os, we can emit all the occurrences of $Q$ in $G$ with no more I/Os compared to evaluating $\mathcal{Q}$. This completes the proof of the lemma.

## B Obtaining the Input Relations of Each $\mathcal{Q}_\gamma$ in Section 3.1

Decide an arbitrary ordering on the attributes of $\mathcal{X}$; w.l.o.g., denote the attributes as $X_1, X_2, ..., X_k$ in ascending order. Every color scheme $\gamma$ can now be represented as a vector $(\gamma(X_1), \gamma(X_2), ..., \gamma(X_k))$. Let us impose a lexicographic order on the $s^k$ color schemes, viewing each $(\gamma(X_1), \gamma(X_2), ..., \gamma(X_k))$ as a $k$-character string. Let $R$ be a relation in $\mathcal{Q}$; w.l.o.g., assume that $schema(R) = \{X_i, X_j\}$ for some $i, j$ satisfying $1 \leq i < j \leq k$. In preprocessing, we sort the tuples $\boldsymbol{t} \in R$ by lexicographic order on $(\Gamma(\boldsymbol{t}(X_i)), \Gamma(\boldsymbol{t}(X_j))$ – viewing the pair as a 2-character string – and group those tuples by $(\Gamma(\boldsymbol{t}(X_i)), \Gamma(\boldsymbol{t}(X_j))$ in the disk. We do so for all the relations $R \in \mathcal{Q}$; the total preprocessing cost is $O(\text{sort}(N))$.

As mentioned in Section 4.1, we deploy BNL to evaluate the joins $\mathcal{Q}_\gamma$ induced by all the color schemes $\gamma$. We do so according to the lexicographic order on $(\gamma(X_1), \gamma(X_2), ..., \gamma(X_k))$. For each $\gamma$, every relation $R_\gamma \in \mathcal{Q}_\gamma$ is a group inside the relation $R \in \mathcal{Q}$ and, hence, has been stored in consecutive blocks. The groups of each relation $R \in \mathcal{Q}$ are accessed in the same lexicographic order determined in preprocessing. For each $\gamma$, the I/O cost of reading the input relations of $\mathcal{Q}_\gamma$ is dominated by that of BNL.

## C Completing the Proof of Lemma 8

Because the analysis of $R^{hi,lo}_{c_1,c_2}$ is symmetric to that of $R^{lo,hi}_{c_1,c_2}$, we will discuss only the former. Define $R^{hi,lo}_{c_1}$ as the set of tuples $\boldsymbol{t} \in R$ such that $\boldsymbol{t}(X_1)$ is a high-degree value mapped to color $c_1$, and $\boldsymbol{t}(X_2)$ is a low-degree value. Hence, $|R^{hi,lo}_{c_1}| \leq 5\lambda$ (because "group $c_1$" – see the creation of function $\Gamma_2$ in Section 3.3 – has a total degree at most $5\lambda$). For each tuple $\boldsymbol{t} \in R^{hi,lo}_{c_1}$, introduce a random variable $Z_{\boldsymbol{t}}$ that equals 1 if $\boldsymbol{t} \in R^{hi,lo}_{c_1,c_2}$, or 0 otherwise. Our function $\Gamma_1$ ensures $\Pr[Z_{\boldsymbol{t}} = 1] = 1/s$ and $\mathbf{Var}(Z_{\boldsymbol{t}}) = \frac{1}{s} - \frac{1}{s^2}$. Define $Z := |R^{hi,lo}_{c_1,c_2}| = \sum_{\boldsymbol{t} \in R^{hi,lo}_{c_1}} Z_{\boldsymbol{t}}$. Create a dependency graph $G^{hi,lo}_{c_1}$ as follows. Each vertex of $G^{hi,lo}_{c_1}$ is the variable $Z_{\boldsymbol{t}}$ of a distinct tuple $\boldsymbol{t} \in R^{hi,lo}_{c_1}$. Two vertices $Z_{\boldsymbol{t_1}}$ and $Z_{\boldsymbol{t_2}}$ are adjacent in $G^{hi,lo}_{c_1}$ if and only if tuples $\boldsymbol{t_1}$ and $\boldsymbol{t_2}$ share the same value on attribute $X_2$. It is easy to verify that $G^{hi,lo}_{c_1}$ fulfils the independence requirement described in Section 2.3 and has a maximum vertex degree at most $\lambda/\sqrt{M}$. Applying Lemma 4 with $\mu := \mathbf{E}[Z] = |R^{hi,lo}_{c_1}|/s$, $\sigma := |R^{hi,lo}_{c_1}|/s > \sum_{\boldsymbol{t} \in R^{hi,lo}_{c_1}} \mathbf{Var}(Z_{\boldsymbol{t}})$, $\Delta := \lambda/\sqrt{M}$, and $\epsilon := Ms/|R^{hi,lo}_{c_1}|$ yields $\Pr[Z \geq 6M] \leq \exp(-\Theta(1) \cdot \frac{M^{1.5}}{\lambda})$, which is at most $1/(4\lambda^{\xi'})$ as long as $M = \Omega((\lambda \log \lambda)^{2/3})$.

## D Proof of Lemma 11

The first statement follows directly from Lemma 6 of [23]. We will prove only the second statement. We first perform $O(\text{sort}(N))$ I/Os to obtain, for each attribute $X \in \mathcal{H}$, the list of all the $O(\sqrt{N/M})$ heavy values in the active domain of $\mathcal{X}$. Then, we compute the cartesian product – denoted as $S$ – of the $|\mathcal{H}|$ lists in $O(|S|/B)$ I/Os; note that $S$ is the set of all configurations. In another $O(\text{sort}(|S|))$ I/Os, we can remove from $S$ those configurations violating (12) and, thus, produce $config(\mathcal{H})$ in the disk. The cost so far is $O(\text{sort}(|S|)) = O(\frac{N^{k/2}}{M^{k/2-1}B} \log_{M/B} \frac{N}{B})$ because $|S| = O((\sqrt{N/M})^{|\mathcal{H}|})$.

Next, we explain how to generate the input relations of the residual joins $\mathcal{Q}'(\boldsymbol{\eta})$ for all $\boldsymbol{\eta} \in config(\mathcal{H})$. For this purpose, consider each hyperedge $e \in \mathcal{E}$ in turn (recall that $\mathcal{G} := (\mathcal{X}, \mathcal{E})$ is the schema graph of $\mathcal{Q}$). If $e \cap \mathcal{H} = \emptyset$, $R_e$ (i.e., the relation in $\mathcal{Q}$ with schema $e$) appears in all the residual joins $\mathcal{Q}'(\boldsymbol{\eta})$ where $\boldsymbol{\eta} \in config(\mathcal{H})$. As $R_e$ is already stored in consecutive blocks, this relation requires no more processing. If $e \subseteq \mathcal{H}$, then $R_e$ contributes nothing to residual joins. It remains to discuss the case where $e \cap \mathcal{H}$ has a single attribute.

W.o.l.g., assume that $e = \{X_1, X_2\}$ with $X_1 \notin \mathcal{H}$ but $X_2 \in \mathcal{H}$. Each tuple $\boldsymbol{t} \in R_e$ appears in the residual join of every configuration $\boldsymbol{\eta} \in \mathit{config}(\mathcal{H})$ satisfying $\boldsymbol{\eta}(X_2) = \boldsymbol{t}(X_2)$; the number of such $\boldsymbol{\eta}$ is $O((\sqrt{N/M})^{|\mathcal{H}|-1})$. To compute residual relations, we first sort both $R_e$ and $\mathit{config}(\mathcal{H})$ on attribute $X_2$ and then produce $R_e \bowtie \mathit{config}(\mathcal{H})$ in the disk by merging the two sorted lists; the cost is $O(\text{sort}(|\mathit{config}(\mathcal{H})|) + \text{sort}(N) + |R_e \bowtie \mathit{config}(\mathcal{H})|/B)$. Then, group the tuples $\boldsymbol{t} \in R_e \bowtie \mathit{config}(\mathcal{H})$ by $\boldsymbol{t}[\mathcal{H}]$, which can be done in $O(\text{sort}(|R_e \bowtie \mathit{config}(\mathcal{H})|))$ I/Os. Each group corresponds to a configuration $\boldsymbol{\eta} \in \mathit{config}(\mathcal{H})$ where $\boldsymbol{\eta} = \boldsymbol{t}[\mathcal{H}]$ for an arbitrary tuple $\boldsymbol{t}$ in the group (all tuples in the group share the same $\boldsymbol{t}[\mathcal{H}]$). The $X_1$-values of the group's tuples constitute the residual relation $R'_e(\boldsymbol{\eta})$. The total I/O cost is $O(\frac{N^{k/2}}{M^{k/2-1}B} \log_{M/B} \frac{N}{B})$ because $|R_e \bowtie \mathit{config}(\mathcal{H})| = O(N \cdot (\sqrt{N/M})^{|\mathcal{H}|-1})$ and $|\mathcal{H}| \le k - 1$ (recall that $X_1 \notin \mathcal{H}$).