Subgraph Enumeration in Optimal I/O Complexity

Shiyuan Deng
The Chinese University of Hong Kong, China

Yufei Tao
The Chinese University of Hong Kong, China

Abstract

Given a massive data graph $G = (V, E)$ and a small pattern graph Q, the goal of *subgraph enumeration* is to list all the subgraphs of G isomorphic to Q. In the external memory (EM) model, it is well-known that every indivisible algorithm must perform $\Omega(\frac{|E|^\rho}{M^{\rho-1}B})$ I/Os in the worst case, where M represents the number of words in (internal) memory, B denotes the number of words in a disk block, and ρ is the fractional edge covering number of Q. It has been a longstanding open problem to design an algorithm to match this lower bound. The state of the art is an algorithm in ICDT'23 that achieves an I/O complexity of $O(\frac{|E|^\rho}{M^{\rho-1}B} \log_{M/B} \frac{|E|}{B})$ with high probability. In this paper, we remove the $\log_{M/B} \frac{|E|}{B}$ factor, thereby settling the open problem when randomization is permitted.

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.2024.21

Funding This work was supported in part by GRF projects 14207820, 14203421, and 14222822 from HKRGC.

1 Introduction

This paper revisits *subgraph enumeration*, a classical problem that has been extensively studied in computer science (see [2, 3, 5–8, 10, 13–19, 27, 30, 32, 40] and the references therein) and has many applications in database systems. The input is a simple undirected graph $G = (V, E)$ called the *data graph* and another simple undirected graph $Q = (V_Q, E_Q)$ called the *pattern*. The objective is to find all the pattern's *occurrences* in the data graph, or specifically, all the subgraphs[1] of G that are isomorphic to Q. We consider that (i) Q is connected (i.e., only a single connected component), and (ii) G has no isolated vertices (i.e., vertices with no incident edges), implying $|V| \leq 2|E|$.

We investigate the problem in the *external memory* (EM) model [1], the de facto model for studying I/O-efficient algorithms. A machine is equipped with M words of memory and an unbounded disk that has been partitioned into disjoint *blocks*, each comprising B words. The values of M and B satisfy $M \geq 2B$. An *I/O operation* either reads a disk block into memory or overwrites a disk block with B memory words. The *cost* of an algorithm is defined as the number of I/Os performed. CPU calculation is for free.

We require Q to have a constant size, namely, $|V_Q| = O(1)$ (otherwise, even detecting whether G contains at least one occurrence of Q is NP-hard [9]). Conversely, the data graph G is presumed to have a gigantic size $|E| > M$ and is provided under the adjacency format across $O(|E|/B)$ blocks. An algorithm should report each occurrence G_{sub} of Q by "emission". Specifically, the algorithm has the discretion to *emit* G_{sub} – for free – at any moment when all the edges of E_{sub} are memory resident, as long as G_{sub} is emitted only once throughout

[1] A *subgraph* of G is defined to be a graph $G_{sub} = (V_{sub}, E_{sub})$ satisfying $V_{sub} \subseteq V$ and $E_{sub} \subseteq E$.

the algorithm. Although not required to output the result to the disk, such an algorithm can be easily modified to do so in $O(\text{OUT}/B)$ extra I/Os, where OUT is the total number of occurrences found.

Our discussion will focus on *indivisible* algorithms (also called *tuple-based* algorithms), which operate under the constraint that one I/O can bring $O(B)$ edges into the memory, effectively prohibiting any compression tricks that attempt to pack $\omega(B)$ edges into B words. Nearly all the existing subgraph enumeration algorithms are indivisible. Consequently, discerning the optimal I/O complexity attainable by this class provides valuable insight into the problem's characteristics.

Previous Results. An imperative parameter characterizing the computational hardness of subgraph enumeration is the *fractional edge covering number* ρ of the pattern $Q = (V_Q, E_Q)$. To understand this concept, imagine assigning, for each edge $e = \{X, Y\} \in E_Q$, a non-negative *weight* w_e such that, for each vertex $X \in V_Q$, the edges incident on X have a total weight at least 1, i.e., $\sum_{e \in E_Q : X \in e} w_e \geq 1$. Then, the value of ρ is the smallest sum $\sum_{e \in E_Q} w_e$ that can be achieved by all possible weight assignments. As a well-established lower bound [20, 23, 36], every (deterministic or randomized) indivisible algorithm needs $\Omega(|E|^\rho/(M^{\rho-1}B))$ I/Os to solve the subgraph enumeration problem in the worst case.

Designing an algorithm to match this lower bound has been an intriguing endeavor. Next, we provide a chronicle of the milestones in the literature. In 2014, Pagh and Silvestri [36] studied the subgraph enumeration problem for Q = triangle (i.e., 3-clique) and presented two algorithms: the first is randomized and guarantees the I/O complexity $O(|E|^{1.5}/(\sqrt{M}B))$ in expectation, while the second is deterministic and ensures an I/O cost of $O(|E|^{1.5}/(\sqrt{M}B) \cdot \log_{M/B} \frac{|E|}{B})$. In 2015, Hu, Qiao, and Tao [22] (long version in [23]) managed to improve the deterministic bound to $O(|E|^{1.5}/(\sqrt{M}B))$. Because $\rho = 1.5$ for Q = triangle, the algorithm of [22] is asymptotically optimal (and so is the randomized algorithm of [36] in expected performance).

In 2016, Hu and Yi [20] considered the scenario where the pattern Q is an (arbitrary) tree and gave a deterministic algorithm with I/O complexity $O(\frac{|E|^\rho}{M^{\rho-1}B} \log_{M/B} \frac{|E|}{B})$. In 2017, Ketsman and Suciu [25] obtained an algorithm that, given an arbitrary pattern Q, solves the subgraph enumeration problem in $O(\frac{|E|^\rho}{M^{\rho-1}B} \operatorname{polylog} |E|)$ I/Os with high probability (i.e., with probability at least $1 - 1/|E|^c$, where c can be an arbitrarily large constant decided before running the algorithm), provided that $M \geq |E|^{\Omega(1)}$. In 2020, Tao [38] found a simpler algorithm attaining the same I/O complexity as [25]. Another main contribution of [38] is its establishment of the *isolated cartesian product theorem* (the ICP-theorem) (see also [26]), which serves as the main weapon in analyzing the performance of a method called *heavy-light decomposition*. This method underlies the algorithm of [38] and all the subsequent works (including ours) on the topic.

In 2023, Deng, Silvestri, and Tao [12] presented new progress that came close to achieving the optimal I/O bound for a general pattern $Q = (V_Q, E_Q)$. Their algorithm finishes in $O(\frac{|E|^\rho}{M^{\rho-1}B} + \frac{|E|^{k/2}}{M^{k/2-1}B} \log_{M/B} \frac{|E|}{B})$ I/Os with high probability, where the parameter k represents the size of V_Q. In general, we know $\rho \geq k/2$ [37]. For a pattern Q where ρ is strictly greater than $k/2$, the I/O cost of [12] can be simplified to the optimal bound $O(|E|^\rho/(M^{\rho-1}B))$. However, when $\rho = k/2$, the I/O bound becomes $O(\frac{|E|^\rho}{M^{\rho-1}B} \log_{M/B} \frac{|E|}{B})$, which is away from optimality by a $\log_{M/B} \frac{|E|}{B}$ factor. It is worth mentioning that the relationship $\rho = k/2$ is satisfied by many patterns Q, important examples of which include k-cycles and k-cliques.

Unlike in the EM model, subgraph enumeration has been well understood in the traditional RAM model, where the problem can be settled in $O(|E|^\rho)$ time [34,35,39], which is known to be worst-case optimal; see also [4, 11, 24, 28, 29, 31, 33] for algorithms with $O(|E|^\rho \operatorname{polylog} |E|)$

Table 1 A summary of the previous results and ours.

pattern Q	I/O cost in big-O	source	remark
triangle	$\lvert E\rvert^{1.5}/(\sqrt{M}B)$ expected	[36]	
triangle	$\frac{\lvert E\rvert^{1.5}}{\sqrt{M}B}\log_{M/B}\frac{\lvert E\rvert}{B}$	[36]	
triangle	$\lvert E\rvert^{1.5}/(\sqrt{M}B)$	[21, 23]	
acyclic	$\frac{\lvert E\rvert^{\rho}}{M^{\rho-1}B}\log_{M/B}\frac{\lvert E\rvert}{B}$	[20]	
arbitrary	$\frac{\lvert E\rvert^{\rho}}{M^{\rho-1}B}\cdot \text{polylog}\,\lvert E\rvert$ w.h.p.	[25, 26, 38]	needs $M \geq \lvert E\rvert^{\Omega(1)}$
arbitrary	$\frac{\lvert E\rvert^{\rho}}{M^{\rho-1}B}\log_{M/B}\frac{\lvert E\rvert}{B}$ w.h.p.	[12]	
arbitrary	$\boldsymbol{\lvert E\rvert^{\rho}/(M^{\rho-1}B)}$ **w.h.p.**	**ours**	**optimal**

time. In general, it is non-trivial to translate a RAM algorithm into an efficient external memory counterpart: a direct translation of the solutions in [4, 11, 24, 28, 29, 31, 33–35, 39] will result in a huge I/O complexity of $\Omega(|E|^{\rho})$. Indeed, the aforementioned subgraph enumeration algorithms in EM harbor numerous ideas that are purposed for accessing data in *blocks* and are thus not required in RAM.

Our Results. This paper's main contribution is an algorithm that solves the subgraph enumeration problem in $O(|E|^{\rho}/(M^{\rho-1}B))$ I/Os with high probability (and, hence, also in expectation). Although this improves the result of [12] by "only" a logarithmic factor, our algorithm is the first whose I/O complexity matches the lower bound $\Omega(|E|^{\rho}/(M^{\rho-1}B))$ for any pattern Q. See Table 1 for a comparison between our result and the existing ones.

Math Conventions. For an integer $x \geq 1$, the notation $[x]$ 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 $\text{sort}(n) = O(1 + \lceil \frac{n}{B} \rceil \log_{M/B} \lceil \frac{n}{B} \rceil)$. We use double curly braces to represent multi-sets, e.g., $\{\{1, 1, 1, 2, 2, 3\}\}$ is a multi-set with 6 elements. Symbol $\mathbb{N}$ represents the set of integers.

2 The Heavy-Light Algorithmic Framework

This section will introduce the *heavy-light decomposition* technique, a framework that transforms subgraph enumeration into a join on binary relations. To lay the groundwork for a formal discussion, we will begin by reviewing the relevant concepts of joins and their connections to subgraph enumeration in Section 2.1. Then, Section 2.2 will delve into the specifics of the decomposition technique at a level sufficient for our technical development.

2.1 Reducing Subgraph Enumeration to Joins

Let **att** represent an arbitrary infinite set where each element is called an *attribute*. Given any set $U \subseteq \textbf{att}$, we define a *tuple* over U as a function $\boldsymbol{t} : U \to \mathbb{N}$. Given any subset U_{sub} of U, we designate $\boldsymbol{t}[U_{sub}]$ as the tuple $\boldsymbol{t_{sub}}$ over U_{sub} such that $\boldsymbol{t_{sub}}(X) = \boldsymbol{t}(X)$ holds true for every $X \in U_{sub}$.

A relation is a set R comprising tuples over the same set U of attributes. The set U is called the *schema* of R, a fact we denote as $U = schema(R)$. We say that R is *unary* if $|schema(R)| = 1$, or *binary* if $|schema(R)| = 2$. Given an integer value x and an attribute $X \in schema(R)$, the *degree of* x *under* X *in* R is the number of tuples $\boldsymbol{t} \in R$ satisfying

$\boldsymbol{t}(X) = x$. We say that the attribute X has *degree* at most λ in R if all values have degrees bounded by λ under X in R. For each attribute $X \in \mathit{schema}(R)$, define its *active domain under* R – represented as $\mathbf{adom}_R(X)$ – as $\Pi_X(R)$, where Π is the standard projection operator in relational algebra.

A *join* is formalized as a set $\mathcal{Q}$ of relations. If we denote $\mathit{schema}(\mathcal{Q}) = \bigcup_{R\in\mathcal{Q}} \mathit{schema}(R)$, the join result – denoted as $\mathit{join}(\mathcal{Q})$ – can be defined as a relation over $\mathit{schema}(\mathcal{Q})$:

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

The *input size* N of $\mathcal{Q}$ is the total number of tuples in all the relations of Q, namely, $N = \sum_{R\in\mathcal{Q}} |R|$. The join $\mathcal{Q}$ is *schema-clean* if it has no two relations such that the schema of one relation contains that of the other. For each attribute $X \in \mathit{schema}(\mathcal{Q})$, define its *active domain under* $\mathcal{Q}$ – represented as $\mathbf{adom}_{\mathcal{Q}}(X)$ – as $\bigcup_{R\in\mathcal{Q}:X\in\mathit{schema}(R)} \mathbf{adom}_R(X)$, namely, the set of values that appear under X in at least one relation of $\mathcal{Q}$.

Every join $\mathcal{Q}$ defines a *schema graph*, which is a hypergraph $\mathcal{G} = (\mathcal{X}, \mathcal{E})$ where $\mathcal{X} = \mathit{schema}(\mathcal{Q})$ and $\mathcal{E} = \{\!\{\mathit{schema}(R) \mid R \in \mathcal{Q}\}\!\}$. Throughout the paper, we will refer to the elements in $\mathcal{X}$ as "attributes" and the elements in $\mathcal{E}$ as "hyperedges". Note that $\mathcal{E}$ is a multi-set: even if two relations $R_1, R_2 \in \mathcal{Q}$ share a common schema, we still treat $\mathit{schema}(R_1)$ and $\mathit{schema}(R_2)$ as different hyperedges in $\mathcal{E}$. As another noteworthy remark, if a relation $R \in \mathcal{Q}$ is unary, the hyperedge $\mathit{schema}(R) \in \mathcal{E}$ has only one attribute. Finally, we say that two attributes $X_1, X_2 \in X$ are *adjacent* in $\mathcal{G}$ if they co-appear in at least one hyperedge in $\mathcal{E}$.

In EM, the *join enumeration* problem is formalized as follows. Let $\mathcal{Q}$ be a schema-clean join whose schema graph $\mathcal{G} = (\mathcal{X}, \mathcal{E})$ has $O(1)$ attributes. Initially, the tuples of each relation $R \in \mathcal{Q}$ – which we call the *raw tuples* – are provided in $O(\lceil |R|/B \rceil)$ consecutive blocks on the disk, and $\mathcal{G}$ is provided in memory using $O(1)$ words. Similar to subgraph enumeration, an algorithm is not required to write the result to the disk. Instead, it suffices to do the reporting through emission. Specifically, for each tuple $\boldsymbol{t} \in \mathit{join}(\mathcal{Q})$, the algorithm can *emit* $\boldsymbol{t}$ for free at any moment when the raw tuple $t[\mathit{schema}(R)]$ exists in memory for every $R \in \mathcal{Q}$, but $\boldsymbol{t}$ should be emitted only once throughout the algorithm. A randomized algorithm is said to guarantee an I/O bound "with high probability" if the bound holds asymptotically with a probability at least $1 - 1/N^c$, where c can be any arbitrarily large constant decided before running the algorithm. We consider only *indivisible* algorithms where each I/O can bring only $O(B)$ raw tuples into memory.

The following lemma was proved in [12]:

▶ **Lemma 1** ([12]). *Consider any input to subgraph enumeration with data graph $G = (V, E)$ and pattern graph Q. It is possible to construct a join $\mathcal{Q}$ with schema graph Q and input size $N = \Theta(|E|)$ such that, if we can emit all the result tuples of $\mathcal{Q}$ using an indivisible algorithm in T_{join} I/Os with high probability, then we can design an indivisible algorithm to solve the original subgraph enumeration problem using $T_{join} + O(\lceil |E|/B \rceil)$ I/Os with high probability.*

As implied by the lemma's statement, the join $\mathcal{Q}$ constructed contains only binary relations, all of which have distinct schemas. Henceforth, we will concentrate on devising an algorithm to process any schema-clean join $\mathcal{Q}$ on binary relations in $T_{join} = O(N^\rho/(M^{\rho-1}B))$ I/Os with high probability, where ρ is the fractional edge-covering number of the schema graph of $\mathcal{Q}$. Once this is done, we will have obtained a subgraph enumeration algorithm that achieves the I/O complexity $O(|E|^\rho/(M^{\rho-1}B))$ with high probability. In the rest of the paper, we consider that $\mathcal{Q}$ has at least two relations (if $\mathcal{Q}$ has only one relation, the join requires no I/Os because the relation itself is the join result).

2.2 Heavy-Light Decomposition

Given a join $\mathcal{Q}$, the *heavy-light decomposition method* conceptually partitions the tuples $\boldsymbol{t}$ in the join result $join(\mathcal{Q})$ by (i) the number of "high-degree" values used in $\boldsymbol{t}$ and (ii) the specific attributes under which those values appear. The method then concentrates on computing the result tuples in each partition separately. Presented below is a version of the method that was introduced in [38] and deployed in the subsequent works [12, 26].

Suppose that we are given a join $\mathcal{Q}$ where there are at least two relations, all the relations are binary, and their schemas are distinct. Denote by $\mathcal{G} = (\mathcal{X}, \mathcal{E})$ the schema graph of $\mathcal{Q}$. For each hyperedge $e \in \mathcal{E}$, we use R_e to represent the (only) relation in Q whose schema is e.

Define

$$\lambda = \sqrt{NM} \tag{1}$$

where N is the input size of $\mathcal{Q}$ and M is the memory size. An integer x is *heavy* if the degree of x under an attribute of any relation is no less than λ; otherwise, the integer is *light*. The total number of heavy integer values is bounded by $O(N/\lambda)$.

Configurations. Fix an arbitrary subset $\mathcal{H} \subseteq \mathcal{X}$. We call each attribute of $\mathcal{H}$ a *heavy* attribute. A *configuration* of $\mathcal{H}$ is a tuple $\boldsymbol{\eta}$ over $\mathcal{H}$ such that $\boldsymbol{\eta}(X)$ is a heavy value for every $X \in \mathcal{H}$. We denote by $config(\mathcal{H})$ the set of "active configurations" $\boldsymbol{\eta}$ satisfying $\boldsymbol{\eta}[e] \in R_e$ for every hyperedge $e \in \mathcal{E}$ subsumed by $\mathcal{H}$ (i.e., $e \subseteq \mathcal{H}$); note that if no such hyperedges exist, then every configuration of $\mathcal{H}$ is active. It is easy to see that $|config(\mathcal{H})| = O((N/\lambda)^{|\mathcal{H}|}) = O((N/M)^{|\mathcal{H}|/2})$, plugging in the value of λ in (1).

Residual Joins. Fix any subset $\mathcal{H} \subseteq \mathcal{X}$ and any active configuration $\boldsymbol{\eta} \in config(\mathcal{H})$. Next, we will formulate a "residual join" to compute the tuples $\boldsymbol{u} \in join(\mathcal{Q})$ that "agree" with $\boldsymbol{\eta}$ on every heavy attribute. We say that a hyperedge $e \in \mathcal{E}$ is *relevant* to $\mathcal{H}$ if $e \setminus \mathcal{H} \neq \emptyset$, namely, e is not subsumed by $\mathcal{H}$. For every relevant $e \in \mathcal{E}$, define $R_e(\boldsymbol{\eta})$ as the relation that comprises every tuple $\boldsymbol{t} \in R_e$ satisfying

- $\boldsymbol{t}(X) = \boldsymbol{\eta}(X)$ for all $X \in e \cap \mathcal{H}$;
- $\boldsymbol{t}(X)$ is light for every $X \in e \setminus \mathcal{H}$.

Namely, $\boldsymbol{t}(X)$ must take *the* heavy value $\boldsymbol{\eta}(X)$ if X is heavy; otherwise, $\boldsymbol{t}(X)$ must be light.

Once $R_e(\boldsymbol{\eta})$ is ready, we can define the *residual relation of* e *under* $\boldsymbol{\eta}$ – denoted as $R'_e(\boldsymbol{\eta})$ – via a simple projection[2]:

$$R'_e(\boldsymbol{\eta}) = \Pi_{e \setminus \mathcal{H}}(R_e(\boldsymbol{\eta})). \tag{2}$$

Now, by putting together the residual relations defined by all the relevant hyperedges e, we obtain the *residual join* of $\boldsymbol{\eta}$ – denoted as $\mathcal{Q}'(\boldsymbol{\eta})$ – which can be formalized as

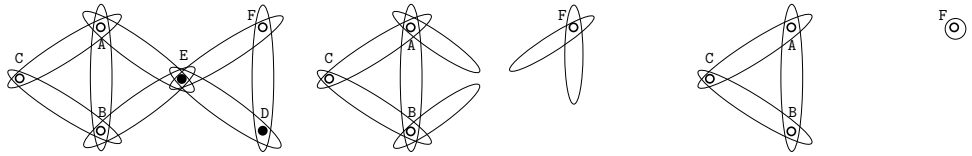
$$\mathcal{Q}'(\boldsymbol{\eta}) = \{R'_e(\boldsymbol{\eta}) \mid e \in \mathcal{E}, e \text{ relevant to } \mathcal{H}\}. \tag{3}$$

We will use $N_{\boldsymbol{\eta}}$ to represent the input size of $\mathcal{Q}'(\boldsymbol{\eta})$, that is

$$N_{\boldsymbol{\eta}} = \sum_{R \in \mathcal{Q}'(\boldsymbol{\eta})} |R|. \tag{4}$$

It is important to note that the relations in $\mathcal{Q}'(\boldsymbol{\eta})$ consist solely of light values.

[2] Strictly speaking, a tuple in $R'_e(\boldsymbol{\eta})$ may no longer be a raw tuple (recall that raw tuples are those in the input relations of $\mathcal{Q}$). This would create an issue when the need arises to verify our algorithms' indivisibility. The issue, however, can be easily fixed by "augmenting" each tuple $\boldsymbol{u} \in R'_e(\boldsymbol{\eta})$ with $\boldsymbol{\eta}$, thereby recording the raw tuple of R_e corresponding to $\boldsymbol{u}$. In the subsequent discussion, we will no longer be concerned with such (pedantic) details, except to mention that all our join algorithms can be implemented in an indivisible manner without affecting the claimed I/O complexities.



(a) Original schema graph $\mathcal{G}$. (b) Residual sch. graph $\mathcal{G}'$ of $\mathcal{H}$. (c) Simplified residual sch. graph $\mathcal{G}''$ of $\mathcal{H}$.

Figure 1 Illustration of the heavy-light decomposition method ($\mathcal{H} = \{\mathtt{D}, \mathtt{E}\}$).

To "visualize" the schemas of the relations in $\mathcal{Q}'(\boldsymbol{\eta})$, define $\mathcal{G}' = (\mathcal{X}', \mathcal{E}')$ as the hypergraph obtained by discarding the heavy attributes from $\mathcal{G} = (\mathcal{X}, \mathcal{E})$, namely, $\mathcal{X}' = \mathcal{X} \setminus \mathcal{H}$ and $\mathcal{E}' = \{\!\{e \setminus \mathcal{H} \mid e \in \mathcal{E} \text{ and } e \setminus \mathcal{H} \neq \emptyset\}\!\}$ (note: $\mathcal{E}'$ is a multi-set). This $\mathcal{G}'$ – called the *residual schema graph* of $\mathcal{H}$ – is the schema graph of $\mathcal{Q}'(\boldsymbol{\eta})$; this is true regardless of $\boldsymbol{\eta}$.

▶ **Example 2.** *For an illustration, Figure 1a presents the schema graph $\mathcal{G} = (\mathcal{X}, \mathcal{E})$ of a join $\mathcal{Q}$, where $\mathcal{X} = \{\mathtt{A}, \mathtt{B}, \mathtt{C}, \mathtt{D}, \mathtt{E}, \mathtt{F}\}$ and $\mathcal{E} = \{\mathtt{AB}, \mathtt{AC}, \mathtt{AE}, \mathtt{BC}, \mathtt{BE}, \mathtt{DE}, \mathtt{DF}, \mathtt{EF}\}$. Set $\mathcal{H} = \{\mathtt{D}, \mathtt{E}\}$; the two heavy attributes $\mathtt{D}$ and $\mathtt{E}$ are colored black in the figure. Let $\boldsymbol{\eta}$ be an arbitrary active configuration of $\mathcal{H}$. All the hyperedges are relevant to $\mathcal{H}$, except $\mathtt{DE}$. The residual relation of $\mathtt{AB}$ under $\boldsymbol{\eta}$ – that is, $R'_{\mathtt{AB}}(\boldsymbol{\eta})$ – is the same as $R_{\mathtt{AB}}(\boldsymbol{\eta})$, which in turn is simply the relation $R_{\mathtt{AB}}$ in $\mathcal{Q}$. The residual relation of $\mathtt{AE}$ under $\boldsymbol{\eta}$ – that is, $R'_{\mathtt{AE}}(\boldsymbol{\eta})$ – has only one attribute $\mathtt{A}$ and contains every light value $\mathtt{a}$ such that $R_{\mathtt{AE}}$ has a tuple $\boldsymbol{u}$ with $\boldsymbol{u}(\mathtt{A}) = \mathtt{a}$ and $\boldsymbol{u}(\mathtt{E}) = \boldsymbol{\eta}(\mathtt{E})$. Figure 1b shows the residual schema graph $\mathcal{G}'$, which is the schema graph of the residual join $\mathcal{Q}'(\boldsymbol{\eta})$ of $\boldsymbol{\eta}$.*

By considering all subsets $\mathcal{H} \subseteq \mathcal{X}$, we have

$$join(\mathcal{Q}) = \bigcup_{\mathcal{H}} \Big(\bigcup_{\boldsymbol{\eta} \in config(\mathcal{H})} join(\mathcal{Q}'(\boldsymbol{\eta})) \times \{\boldsymbol{\eta}\} \Big). \tag{5}$$

This transforms the evaluation of the original join $\mathcal{Q}$ into computing the residual joins.

Simplified Residual Joins. Again, fix any subset $\mathcal{H} \subseteq \mathcal{X}$ and one arbitrary configuration $\boldsymbol{\eta} \in config(\mathcal{H})$. One issue arising from computing the residual join $\mathcal{Q}'(\boldsymbol{\eta})$ is that the join may not be schema-clean, i.e., $\mathcal{Q}'(\boldsymbol{\eta})$ may contain two relations where one relation's schema encompasses that of the other. The second phase of the heavy-light decomposition framework is to convert each $\mathcal{Q}'(\boldsymbol{\eta})$ to its schema-clean version. Next, we explain how this is done.

Define $\mathcal{L} = \mathcal{X} \setminus \mathcal{H}$; we will call each attribute in $\mathcal{L}$ a *light* attribute (as opposed to the heavy attributes in $\mathcal{H}$). We say that a light attribute $X \in \mathcal{L}$ is a *border attribute* if it is adjacent in $\mathcal{G}$ to at least one heavy attribute in $\mathcal{H}$. For each such attribute X, define

$$R''_X(\boldsymbol{\eta}) = \bigcap_{e \in \mathcal{E}: X \in e} \Pi_X(R'_e(\boldsymbol{\eta})). \tag{6}$$

To understand the intuition behind, here we look at all the edges (of $\mathcal{G}$) containing X and examine their residual relations under $\boldsymbol{\eta}$. The relation $R''_X(\boldsymbol{\eta})$ collects every integer that appears under attribute X in all those residual relations.

Now consider each hyperedge $e = \{X_1, X_2\}$ in $\mathcal{G}$ where both X_1 and X_2 are light attributes. Note that e is also a hyperedge in $\mathcal{E}'$. Construct a relation $R''_e(\boldsymbol{\eta}) \subseteq R'_e(\boldsymbol{\eta})$ as follows:

- If X_1 and X_2 are both border attributes, then $R''_e(\boldsymbol{\eta}) = R'_e(\boldsymbol{\eta}) \bowtie R''_{X_1}(\boldsymbol{\eta}) \bowtie R''_{X_2}(\boldsymbol{\eta})$, where $\bowtie$ is the standard natural join operator in relational algebra;
- 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 $R''_e(\boldsymbol{\eta}) = R'_e(\boldsymbol{\eta})$.

It is safe to discard the tuples in $R'_e(\boldsymbol{\eta}) \setminus R''_e(\boldsymbol{\eta})$ because they cannot contribute to the result of the residual join $\mathcal{Q}'(\boldsymbol{\eta})$. We call $R''_e(\boldsymbol{\eta})$ the *simplified residual relation of e under $\boldsymbol{\eta}$*.

A border attribute $X \in \mathcal{X}$ may become *isolated* in $\mathcal{G}'$, meaning that X is not adjacent to any other vertex in $\mathcal{G}'$ (i.e., X appears only in unary hyperedges of $\mathcal{E}'$). This happens when X is not adjacent to any light attribute in $\mathcal{G}$ (i.e., X is adjacent to only heavy attributes in $\mathcal{G}$). Let $\mathcal{I}$ be the set of isolated attributes in $\mathcal{G}'$.

The *simplified residual join* of $\boldsymbol{\eta}$ – denoted as $\mathcal{Q}''(\boldsymbol{\eta})$ – can now be formalized as

$$\mathcal{Q}''(\boldsymbol{\eta}) \quad = \quad \{R''_e(\boldsymbol{\eta}) \mid \text{binary } e \in \mathcal{E}'\} \cup \{R''_X(\boldsymbol{\eta}) \mid X \in \mathcal{I}\}. \tag{7}$$

To visualize the schemas of the relations in $\mathcal{Q}''(\boldsymbol{\eta})$, define $\mathcal{G}'' = (\mathcal{X}'', \mathcal{E}'')$ where $\mathcal{X}'' = \mathcal{X}' = \mathcal{L}$ and $\mathcal{E}''$ contains (ii) all the binary edges of $\mathcal{E}'$ and (ii) one unary edge for each isolated attribute. This $\mathcal{G}''$ is the schema graph of $\mathcal{Q}''(\boldsymbol{\eta})$, regardless of $\boldsymbol{\eta} \in \mathit{config}(\mathcal{H})$.

▶ **Example 3.** *Let us continue the discussion in Example 2. In Figure 1a, $\mathcal{L} = \{\mathtt{A}, \mathtt{B}, \mathtt{C}, \mathtt{F}\}$, and among these attributes, $\mathtt{A}$, $\mathtt{B}$ and $\mathtt{F}$ are border attributes, but only $\mathtt{F}$ is an isolated attribute. The relation $R''_{\mathtt{A}}$ is the intersection of $\Pi_{\mathtt{A}}(R'_{\mathtt{AB}}(\boldsymbol{\eta})) = \Pi_{\mathtt{A}}(R_{\mathtt{AB}})$, $\Pi_{\mathtt{A}}(R'_{\mathtt{AC}}(\boldsymbol{\eta})) = \Pi_{\mathtt{A}}(R_{\mathtt{AC}})$, and $R'_{\mathtt{AE}}(\boldsymbol{\eta})$ (recall that $R'_{\mathtt{AE}}(\boldsymbol{\eta})$ contains only one attribute, i.e., $\mathtt{A}$). The simplified residual relation of $\mathtt{AB}$ – that is, $R''_{\mathtt{AB}}(\boldsymbol{\eta})$ – comprises every tuple $\boldsymbol{u} \in R'_{\mathtt{AB}}(\boldsymbol{\eta})$ such that $\boldsymbol{u}(\mathtt{A}) \in R''_{\mathtt{A}}$ and $\boldsymbol{u}(\mathtt{B}) \in R''_{\mathtt{B}}$. The simplified residual relation of $\mathtt{BC}$ – that is, $R''_{\mathtt{BC}}(\boldsymbol{\eta})$ – comprises every tuple $\boldsymbol{u} \in R'_{\mathtt{BC}}(\boldsymbol{\eta})$ such that $\boldsymbol{u}(\mathtt{B}) \in R''_{\mathtt{B}}$. Figure 1c shows the simplified residual schema graph $\mathcal{G}''$, which is the schema graph of the simplified residual join $\mathcal{Q}''(\boldsymbol{\eta})$ of $\boldsymbol{\eta}$.*

It is easy to verify that $\mathit{join}(\mathcal{Q}''(\boldsymbol{\eta})) = \mathit{join}(\mathcal{Q}'(\boldsymbol{\eta}))$. Combining this fact with (5), we have

$$\mathit{join}(\mathcal{Q}) = \bigcup_{\mathcal{H}} \Big(\bigcup_{\boldsymbol{\eta} \in \mathit{config}(\mathcal{H})} \mathit{join}(\mathcal{Q}''(\boldsymbol{\eta})) \times \{\boldsymbol{\eta}\} \Big). \tag{8}$$

which transforms the evaluation of the original join $\mathcal{Q}$ into computing $\mathit{join}(\mathcal{Q}''(\boldsymbol{\eta})) \times \{\boldsymbol{\eta}\}$ for all $\boldsymbol{\eta} \in \mathit{config}(\mathcal{H})$ and $\mathcal{H} \subseteq \mathcal{X}$.

A main contribution of [12] is the following lemma.

▶ **Lemma 4** ([12]). *Let $\mathcal{Q}$ be a schema-clean join on at least two binary relations whose schema graph is $\mathcal{G} = (\mathcal{X}, \mathcal{E})$. Let $k = |\mathcal{X}|$, N be the input size of $\mathcal{Q}$, and ρ be the fractional edge covering number of $\mathcal{G}$. Suppose that, for any $\mathcal{H} \subseteq \mathcal{X}$, using $T_{\mathcal{H}}$ I/Os in total we can produce*

- *$\mathit{config}(\mathcal{H})$ in consecutive disk blocks;*
- *each input relation of $\mathcal{Q}''(\boldsymbol{\eta})$ in consecutive disk blocks for every active configuration $\boldsymbol{\eta} \in \mathit{config}(\mathcal{H})$.*

Then, there is an algorithm for computing $\mathit{join}(\mathcal{Q})$ using $\sum_{\mathcal{H} \subseteq \mathcal{X}} T_{\mathcal{H}} + O(N^{\rho}/(M^{\rho-1}B))$ I/Os.

The algorithm of [12] necessitates $\sum_{\mathcal{H} \subseteq \mathcal{X}} T_{\mathcal{H}} = O(\frac{N^{k/2}}{M^{k/2-1}B} \log_{M/B} \frac{N}{B})$ I/Os, as is the culprit behind the algorithm's sub-optimality. Our mission in this work is:

$$\text{reduce } \sum_{\mathcal{H} \subseteq \mathcal{X}} T_{\mathcal{H}} \text{ to } O(N^{k/2}/(M^{k/2-1}B)). \tag{9}$$

3 Binary-Relation Joins in a Small Domain

In this and the next sections, we will discuss two standalone problems that are intriguing in their own right. The solutions to those problems will play a crucial role in our approach to achieving the objective given in (9).

This section will study join enumeration in the special scenario where the number of distinct values is limited. The rest of the section serves as a proof of:

▶ **Lemma 5.** *Let $\mathcal{Q}$ be a schema-clean join on binary relations, and let $\mathcal{G} = (\mathcal{X}, \mathcal{E})$ be its schema graph. If $|\mathbf{adom}_{\mathcal{Q}}(X)| \leq D$ for every attribute $X \in \mathcal{X}$, we can emit all the result tuples of the join $\mathcal{Q}$ in $O((1 + \frac{D}{\sqrt{M}})^k \cdot \frac{M}{B} + \text{sort}(N))$ I/Os, where $k = |\mathcal{X}|$, N is the input size of $\mathcal{Q}$, M is the memory size, and B is the disk block size.*

Recall from Section 2 that $\mathbf{adom}_{\mathcal{Q}}(X)$ is the active domain of X under $\mathcal{Q}$. In general, the value of D (i.e., the maximum active domain size) can reach $\Omega(N)$, in which case Lemma 5 is not particularly useful: it is worse than the $O(N^\rho/(M^{\rho-1}B))$ I/O bound that will be ultimately established in this paper. However, the lemma's strength is reflected in the scenario where D is small. For example, for $D = \sqrt{N}$ and $k \geq 3$, the I/O complexity of Lemma 5 becomes $O(N^{k/2}/(M^{k/2-1}B))$, which is never worse, but can be considerably better, than $O(N^\rho/(M^{\rho-1}B))$ because the value of ρ falls in the range from $k/2$ to $k-1$ [37].

We now proceed to prove the lemma. For each attribute $X \in \mathcal{X}$, we obtain and sort the active domain $\mathbf{adom}_{\mathcal{Q}}(X)$ in $O(\text{sort}(N))$ I/Os. Then, by scanning (the sorted) $\mathbf{adom}_{\mathcal{Q}}(X)$, we partition the integer domain $\mathbb{N}$ into a set S^{intv}_X of disjoint intervals such that each interval covers $\lfloor\sqrt{M}/c\rfloor$ values of $\mathbf{adom}_{\mathcal{Q}}(X)$ – where c is a constant to be chosen later – except possibly one interval that may cover less values. The size of S^{intv}_X is $O(1 + D/\sqrt{M})$ because $|\mathbf{adom}_{\mathcal{Q}}(X)| \leq D$.

Let $X_1, X_2, ..., X_k$ be an arbitrary ordering of the k attributes in $\mathcal{X}$. Consider any hyperedge $e = \{X_i, X_j\}$ of $\mathcal{E}$ $(1 \leq i < j \leq k)$. Given an interval $I_i \in S^{intv}_{X_i}$ and an interval $I_j \in S^{intv}_{X_j}$, define

$$R_e(I_i, I_j) \quad = \quad \{\boldsymbol{u} \in R_e \mid \boldsymbol{u}(X_i) \in I_i \text{ and } \boldsymbol{u}(X_j) \in I_j\}.$$

Recall that R_e is the (only) relation in $\mathcal{Q}$ with schema e. As I_i (resp., I_j) covers $\lfloor\sqrt{M}/c\rfloor$ values of $\mathbf{adom}_{\mathcal{Q}}(X_i)$ (resp., $\mathbf{adom}_{\mathcal{Q}}(X_j)$), the set $R_e(I_i, I_j)$ can contain at most $(\lfloor\sqrt{M}/c\rfloor)^2 \leq M/c^2$ tuples. For each interval combination $(I_1, I_2, ..., I_k) \in S^{intv}_{X_1} \times S^{intv}_{X_2} \times ... \times S^{intv}_{X_k}$, define a sub-join

$$\mathcal{Q}(I_1, I_2, ..., I_k) \quad = \quad \{R_e(I_i, I_j) \mid e = \{X_i, X_j\} \in \mathcal{E}\}.$$

It is easy to verify that

$$join(\mathcal{Q}) \quad = \bigcup_{(I_1, I_2, ..., I_k) \in S^{intv}_{X_1} \times S^{intv}_{X_2} \times ... \times S^{intv}_{X_k}} join(\mathcal{Q}(I_1, I_2, ..., I_k)). \tag{10}$$

Let us assume, for now, that, given a hyperedge $e = \{X_i, X_j\} \in \mathcal{E}$, any interval $I_i \in S^{intv}_{X_i}$, and any interval $I_j \in S^{intv}_{X_j}$, we can load $R_e(I_i, I_j)$ from the disk into memory using $O(M/B)$ I/Os. Fix an arbitrary interval combination $(I_1, I_2, ..., I_k) \in S^{intv}_{X_1} \times S^{intv}_{X_2} \times ... \times S^{intv}_{X_k}$. If we add up the size of $R_e(I_i, I_j)$ for all hyperedges $e = \{X_i, X_j\} \in \mathcal{E}$, the total size is at most $|\mathcal{E}| \cdot M/c^2$, which is at most M by setting the constant c sufficiently large. Hence, in $O(M/B) \cdot |\mathcal{E}| = O(M/B)$ I/Os, we can load into memory the set $R_e(I_i, I_j)$ of every hyperedge $e = \{X_i, X_j\} \in \mathcal{E}$. This permits us to emit the result tuples of $\mathcal{Q}(I_1, I_2, ..., I_k)$ with no more I/Os. As the cartesian product $S^{intv}_{X_1} \times S^{intv}_{X_2} \times ... \times S^{intv}_{X_k}$ has a size of $O((1 + \frac{D}{\sqrt{M}})^k)$, by virtue of (10) we can emit all the result tuples in $\mathcal{Q}$ using $O((1 + \frac{D}{\sqrt{M}})^k \cdot \frac{M}{B})$ I/Os.

It remains to explain how to ensure that $R_e(I_i, I_j)$ can always be loaded into memory using $O(M/B)$ I/Os. This requires only $O((1 + \frac{D}{\sqrt{M}})^2 \cdot \frac{1}{B} + \text{sort}(N))$ extra I/Os to prepare the input relations of $\mathcal{Q}$ appropriately. The details are standard and moved to Appendix A. With that, we conclude the proof of Lemma 5.

4 Batched Two-Way Semi-Join Reductions

This section will study an interesting variant of the traditional "semi-join" problem. This variant is extracted from a sub-problem that will arise in Section 5 when we discuss how to fulfill the objective outlined in (9).

We are given a binary relation R with schema $\{X, Y\}$, together with ℓ unary relations $S_1, S_2, ..., S_\ell$ with schema $\{X\}$ and another ℓ unary relations $T_1, T_2, ..., T_\ell$ with schema $\{Y\}$. Both attributes X and Y have degrees at most λ in R, whereas every S_i and every T_i ($i \in [\ell]$) have been sorted by X and Y, respectively. For each $i \in [\ell]$, define a join

$$\mathcal{Q}_i \quad = \quad \{R, S_i, T_i\} \tag{11}$$

whose result $join(Q_i)$ is a subset of R, including every tuple $\boldsymbol{u} \in R$ with $\boldsymbol{u}(X) \in S_i$ and $\boldsymbol{u}(Y) \in T_i$. Our objective is to compute, for every $i \in [\ell]$, the result $join(\mathcal{Q}_i)$ and write it to consecutive blocks on the disk. We will refer to the above as the *batched two-way semi-join reduction problem*.

The problem admits a "textbook solution" that computes each $join(\mathcal{Q}_i)$ individually as follows. Specifically, we can first sort R on attribute X and compute $R' = R \ltimes S_i$ by scanning R and S_i synchronously. Then, we sort R' on attribute Y and compute $R'' = R' \ltimes T_i$ by scanning R' and T_i synchronously. The relation R'' is the join result $join(\mathcal{Q}_i)$. Executing these steps for each $i \in [\ell]$ necessitates a total I/O cost of $O(\ell \cdot \text{sort}(|R|) + \sum_{i=1}^{\ell} \lceil (|S_i| + |T_i|)/B \rceil)$. On the other hand, we will prove:

▶ **Lemma 6.** *Batched two-way semi-join reduction can be solved with an I/O complexity*

$$O\left(\left(\frac{|R|}{\lambda} + \frac{\lambda}{M}\right)\frac{\sum_{i=1}^{\ell}|S_i| + |T_i|}{B} + \ell \cdot \left\lceil \frac{|R|}{B} \right\rceil + \frac{\ell \cdot |R|^2 M}{\lambda^2 \cdot B} + \text{sort}(|R|)\right).$$

The above holds for arbitrary integers λ and ℓ (which may not be constants).

The lemma is most interesting under $\lambda = \sqrt{|R| \cdot M}$, in which case the I/O complexity can be simplified to $O(\sqrt{|R|/M} \cdot (\sum_{i=1}^{\ell}|S_i| + |T_i|)/B + \ell \cdot |R|/B + \text{sort}(|R|))$. In the "lopsided situation" where $\sum_{i=1}^{\ell}|S_i| + |T_i| \leq \ell\sqrt{|R| \cdot M}$ – that is, on average each unary relation has a size $O(\sqrt{|R| \cdot M}) = O(\lambda)$ – the I/O complexity becomes $O(\ell \cdot \lceil |R|/B \rceil + \text{sort}(|R|))$, which improves the aforementioned textbook solution by a factor of $O(\log_{M/B}(|R|/B))$ for large ℓ.

The rest of the section serves as a proof of Lemma 6. We start by partitioning the integer domain $\mathbb{N}$ into a set S_X^{intv} of intervals such that

- $|S_X^{intv}| = O(|R|/\lambda)$ and
- for each interval $I \in S_X^{intv}$, the relation R has $O(\lambda)$ tuples $\boldsymbol{u}$ with $\boldsymbol{u}(X) \in I$.

Symmetrically, obtain a set S_Y^{intv} of intervals satisfying two analogous conditions with respect to Y. The intervals in S_X^{intv} and S_Y^{intv} can be obtained in $O(\text{sort}(|R|))$ I/Os[3].

[3] We will explain this only for S_X^{intv} due to symmetry. Sort and group the tuples of R by their X-values. Each group has a size of at most λ. Next, we will scan the groups in ascending order of their X-values and, in doing so, divide the tuples of R using special tokens. First, place a token before the first group and then start the scan. Every time an entire group of tuples has been scanned, place another token at the end of the group if we have seen at least λ tuples since the last token. Finally, place another token at the end of the last group. It is easy to see that there can be at most 2λ tuples between any two consecutive tokens and the number of tokens is $O(|R|/\lambda)$. The desired set S_X^{intv} of intervals can then be easily determined based on the tokens.

For any interval $I_X \in S_X^{intv}$, any interval $I_Y \in S_Y^{intv}$, and any $i \in [\ell]$, define

$$\begin{aligned} R(I_X, I_Y) &= \{\boldsymbol{u} \in R \mid \boldsymbol{u}(X) \in I_X \text{ and } \boldsymbol{u}(Y) \in I_Y\} \\ S_i(I_X) &= \{\boldsymbol{u} \in X \mid \boldsymbol{u}(X) \in I_X\} \\ T_i(I_Y) &= \{\boldsymbol{u} \in Y \mid \boldsymbol{u}(Y) \in I_Y\}. \end{aligned}$$

After $O(\ell + \frac{|R|^2}{\lambda^2 \cdot B} + \text{sort}(|R|) + \frac{\ell \cdot |R|}{\lambda B} + \sum_{i=1}^{\ell}(|S_i| + |T_i|)/B)$ I/Os, we can store each

- $R(I_X, I_Y)$ in consecutive blocks whose starting address can be located in one I/O;
- $S_i(I_X)$ in consecutive blocks whose starting address can be located in one I/O;
- $T_i(I_Y)$ in consecutive blocks whose starting address can be located in one I/O.

The details are standard and moved to Appendix B.

For each $i \in [\ell]$, we initialize an empty disk file for $\mathcal{Q}_i$ and will gradually populate the file with tuples of $join(\mathcal{Q}_i)$ until eventually the file's content is exactly $join(\mathcal{Q}_i)$. Motivated by the fact

$$join(\mathcal{Q}_i) = \bigcup_{(I_X, I_Y) \in S_X^{intv} \times S_Y^{intv}} R(I_X, I_Y) \bowtie S_i(I_X) \bowtie T_i(I_Y) \tag{12}$$

we process each interval pair $(I_X, I_Y) \in S_X^{intv} \times S_Y^{intv}$ separately and, in doing so, append $R(I_X, I_Y) \bowtie S_i(I_X) \bowtie T_i(I_Y)$ to the file of $\mathcal{Q}_i$ for every $i \in [\ell]$.

We now elaborate how to process a pair $(I_X, I_Y) \in S_X^{intv} \times S_Y^{intv}$. Conceptually, partition $R(I_X, I_Y)$ (arbitrarily) into *chunks*, each of which contains $M - B \geq M/2$ tuples except possibly the last chunk. For each chunk, we first read it into memory using $O(M/B)$ I/Os; let R_{chunk} be the set of tuples in that chunk. Keeping R_{chunk} in memory, we then – for each $i \in [\ell]$ in turn – compute and append $R_{chunk} \bowtie S_i(I_X) \bowtie T_i(I_Y)$ to the file of $\mathcal{Q}_i$ using $O(\frac{|S_i(I_X)|+|T_i(I_Y)|+M}{B})$ I/Os. For this purpose, we scan S_i in its entirety using one block of memory. As soon as a tuple $\boldsymbol{v} \in S_i(I_X)$ is brought into memory, it is compared to all the tuples in R_{chunk}. In doing so, for each tuple $\boldsymbol{u} \in R_{chunk}$, we track whether any tuple $\boldsymbol{v} \in S_i(I_X)$ with $\boldsymbol{u}(X) = \boldsymbol{v}(X)$ has been seen; if so, we "mark" $\boldsymbol{u}$. Similarly, by scanning $T_i(I_Y)$ once, we can detect, for each tuple $\boldsymbol{u} \in R_{chunk}$, whether $T_i(I_Y)$ has any tuple $\boldsymbol{v}$ with $\boldsymbol{u}(Y) = \boldsymbol{v}(Y)$; if so, we "mark" $\boldsymbol{u}$. The tuples of $R_{chunk} \bowtie S_i(I_X) \bowtie T_i(I_Y)$ are exactly those in R_{chunk} that receive two marks (one from scanning $S_i(I_X)$ and the other from $T_i(I_Y)$). These tuples are then written to the disk in $O(M/B)$ I/Os. In the entire processing of (I_X, I_Y), $S_i(I_X)$ and $T_i(I_Y)$ are scanned $O(\lceil |R(I_X, I_Y)|/M \rceil)$ times, i.e., the number of chunks.

By executing the above algorithm for every (I_X, I_Y), we produce the result of $\mathcal{Q}_i$ on the disk for all $i \in [\ell]$ with an I/O complexity:

$$O\Big(\sum_{(I_X, I_Y) \in S_X^{intv} \times S_Y^{intv}} \left\lceil \frac{|R(I_X, I_Y)|}{M} \right\rceil \cdot \sum_{i=1}^{\ell} \frac{|S_i(I_X)| + |T_i(I_Y)| + M}{B} \Big) \tag{13}$$

In Appendix C, we show how to relate the above to the degree threshold λ and prove

$$(13) = O\Big(\left(\frac{|R|}{\lambda} + \frac{\lambda}{M} \right) \frac{\sum_{i=1}^{\ell} |S_i| + |T_i|}{B} + \frac{\ell \cdot |R|}{B} + \frac{\ell \cdot |R|^2 M}{\lambda^2 \cdot B} \Big). \tag{14}$$

With this, we can then conclude the proof of Lemma 6.

5 I/O-Efficient Heavy-Light Decomposition

We now return to processing a schema-clean join consisting purely of binary relations and will accomplish the mission described in (9). This section effectively proves the lemma below.

▶ **Lemma 7.** *Let $\mathcal{Q}$ be a schema-clean join on at least two binary relations, whose schema graph is $\mathcal{G} = (\mathcal{X}, \mathcal{E})$. Let $k = |\mathcal{X}|$, and let N be the input size of $\mathcal{Q}$. Fix an arbitrary $\mathcal{H} \subseteq \mathcal{X}$. In $T_{\mathcal{H}} = O(N^{k/2}/(M^{k/2-1}B))$ I/Os, we can produce*

- *config($\mathcal{H}$) in consecutive disk blocks;*
- *each input relation of $\mathcal{Q}''(\boldsymbol{\eta})$ in consecutive disk blocks for every active configuration $\boldsymbol{\eta} \in$ config($\mathcal{H}$).*

Because $\mathcal{X}$ has $2^k = O(1)$ subsets $\mathcal{H}$, the above lemma indicates $\sum_{\mathcal{H}\subseteq\mathcal{X}} T_{\mathcal{H}} = O(N^{k/2}/(M^{k/2-1}B))$, as needed in (9). We will explain in Section 5.1 how to generate config($\mathcal{H}$) and then in Section 5.2 how to create the input relations of the simplified residual joins $\mathcal{Q}''(\boldsymbol{\eta})$ for all $\boldsymbol{\eta} \in$ config($\mathcal{H}$). As will be clear later, the core of the former (resp., latter) task boils down to the problem tackled in Section 3 (resp., 4).

Preprocessing. For each attribute $X \in \mathcal{X}$, we use $\mathbf{adom}^{\mathrm{H}}_{\mathcal{Q}}(X)$ (resp., $\mathbf{adom}^{\mathrm{L}}_{\mathcal{Q}}(X)$) to represent the set of heavy (resp., light) values in $\mathbf{adom}_{\mathcal{Q}}(X)$. Recall that, for each hyperedge $e = \{X, Y\} \in \mathcal{E}$, the symbol R_e denotes the only relation in $\mathcal{Q}$ whose schema is e. We define:

$$R_e^{\mathrm{HH}} = \{\boldsymbol{u} \in R_e \mid \boldsymbol{u}(X) \in \mathbf{adom}^{\mathrm{H}}_{\mathcal{Q}}(X) \text{ and } \boldsymbol{u}(Y) \in \mathbf{adom}^{\mathrm{H}}_{\mathcal{Q}}(Y)\}. \quad (15)$$

$$R_e^{\mathrm{LL}} = \{\boldsymbol{u} \in R_e \mid \boldsymbol{u}(X) \in \mathbf{adom}^{\mathrm{L}}_{\mathcal{Q}}(X) \text{ and } \boldsymbol{u}(Y) \in \mathbf{adom}^{\mathrm{L}}_{\mathcal{Q}}(Y)\}. \quad (16)$$

If e contains a heavy attribute $X \in \mathcal{H}$ and a light attribute $Y \in \mathcal{L}$ (recall: $\mathcal{L} = \mathcal{X} \setminus \mathcal{H}$), we refer to $e \in \mathcal{E}$ as a *crossing hyperedge*. Given a crossing hyperedge e, we define:

$$R_e^{\mathrm{HL}} = \{\boldsymbol{u} \in R_e \mid \boldsymbol{u}(X) \in \mathbf{adom}^{\mathrm{H}}_{\mathcal{Q}}(X) \text{ and } \boldsymbol{u}(Y) \in \mathbf{adom}^{\mathrm{L}}_{\mathcal{Q}}(Y)\}. \quad (17)$$

It is rudimentary to use sort(N) I/Os to produce on the disk:

- $\mathbf{adom}^{\mathrm{H}}_{\mathcal{Q}}(X)$ in sorted order for every $X \in \mathcal{X}$;
- $\mathbf{adom}^{\mathrm{L}}_{\mathcal{Q}}(X)$ in sorted order for every $X \in \mathcal{X}$;
- R_e^{HH} for every $e \in \mathcal{E}$;
- R_e^{LL} for every $e \in \mathcal{E}$;
- R_e^{HL} for every crossing hyperedge $e \in \mathcal{E}$.

We remark that $\mathrm{sort}(N) = O(\frac{N}{B}\log_{M/B}\frac{N}{B}) = O(\frac{N}{B}\log_{M/B}\frac{N}{M}) = O(\frac{N}{B}\sqrt{\frac{N}{M}}) = O(\frac{N^{k/2}}{M^{k/2-1}B})$, where the last step used the fact $k \geq 3$ (because $\mathcal{Q}$ has at least two relations and is schema-clean). In other words, sorting is within the target budget of Lemma 7.

5.1 Generating *config*($\mathcal{H}$)

Let us first explain how to compute config($\mathcal{H}$) under the condition $\mathcal{H} = \mathcal{X}$. In such case, we construct a join

$$\mathcal{Q}_{\mathcal{H}} = \{R_e^{\mathrm{HH}} \mid e \in \mathcal{E}\}.$$

Note that config($\mathcal{H}$) is precisely the result join($\mathcal{Q}_{\mathcal{H}}$) of $\mathcal{Q}_H$. The input relations of $\mathcal{Q}_{\mathcal{H}}$ contain only values classified as "heavy" for the original join $\mathcal{Q}$. Hence, for every attribute $X \in \mathcal{X}$, it holds that $\mathbf{adom}_{\mathcal{Q}_{\mathcal{H}}}(X)$ – the active domain of X under $\mathcal{Q}_{\mathcal{H}}$ – has size $O(N/\lambda)$, which is $O(\sqrt{N/M})$ given the heavy-value threshold λ in (1). We compute join($\mathcal{Q}_H$) – namely, config($\mathcal{H}$) – using Lemma 5 (plugging in $D = O(\sqrt{N/M})$). The I/O cost is

$$O\Big(\Big(1+\frac{\sqrt{N/M}}{\sqrt{M}}\Big)^k\cdot\frac{M}{B}+\text{sort}(N)\Big)+O\Big(\frac{|\mathit{config}(\mathcal{H})|}{B}\Big) \tag{18}$$

where the term $O(|\mathit{config}(\mathcal{H})|/B)$ does not come from Lemma 5, but is instead due to the need of writing $\mathit{config}(\mathcal{H})$ to the disk. Applying the relationship $|\mathit{config}(\mathcal{H})| = O((N/M)^{|\mathcal{H}|/2}) = O((N/M)^{k/2})$, the reader can easily confirm that (18) is bounded by $O(N^{k/2}/(M^{k/2-1}B))$.

The case where $\mathcal{H} \subset \mathcal{X}$ can be dealt with in a more conventional manner:

1. $\mathit{CPadom}^{\text{H}} = \bigtimes_{X\in\mathcal{H}} \mathbf{adom}^{\text{H}}_{\mathcal{Q}}(X)$, i.e., the cartesian product of the heavy-value sets $\mathbf{adom}^{\text{H}}_{\mathcal{Q}}(X)$ for all the heavy attributes $X \in \mathcal{H}$.
2. $R^* = \mathit{CPadom}^{\text{H}}$

/* henceforth, we will regard R^* as a relation with schema $\mathcal{H}$ */

3. **for** each hyperedge $e \in \mathcal{E}$ such that $e \subseteq \mathcal{H}$ **do**
4. $\quad R^* = R^* \ltimes R_e$ (semi-join)

The final R^* is the $\mathit{config}(\mathcal{H})$ we aim to compute. Regarding the cost, Line 1 can be implemented[4] in $O(|\mathit{CPadom}^{\text{H}}|/B)$ I/Os (dominated by the cost of writing $\mathit{CPadom}^{\text{H}}$ to the disk), where $|\mathit{CPadom}^{\text{H}}| = O((\sqrt{N/M})^{|\mathcal{H}|})$. With sorting, we can perform Lines 2-4 in $O(\text{sort}(|\mathit{CPadom}^{\text{H}}|) + \text{sort}(N))$ I/Os. The total overhead is thus bounded by

$$O\Big(\Big(\frac{N}{M}\Big)^{|\mathcal{H}|/2}\log_{M/B}\frac{N}{M}+\text{sort}(N)\Big) = O\Big(\Big(\frac{N}{M}\Big)^{k/2}+\text{sort}(N)\Big) = O(N^{k/2}/(M^{k/2-1}B)) \tag{19}$$

I/Os, where the first equality holds because $|\mathcal{H}| < |\mathcal{X}| = k$.

5.2 Generating the Input Relations for the Simplified Residual Joins

We now proceed to explain how to create the input relations of $\mathcal{Q}''(\boldsymbol{\eta})$ for every active configuration $\boldsymbol{\eta} \in \mathit{config}(\mathcal{H})$. It suffices to consider $\mathcal{H} \subset \mathcal{X}$ (if $\mathcal{H} = \mathcal{X}$, every $\mathcal{Q}''(\boldsymbol{\eta})$ is empty, i.e., there are no input relations at all). The relationship $\mathcal{H} \subset \mathcal{X}$ implies a useful property: $O(\text{sort}(|\mathit{config}(\mathcal{H})|) = O(\text{sort}((N/M)^{|\mathcal{H}|/2}) = O(N^{k/2}/(M^{k/2-1}B))$, following the same derivation as in (19).

Input Relations of Residual Joins. We start by generating $R_e(\boldsymbol{\eta})$ – the subset of the relation R_e that "agrees" with $\boldsymbol{\eta}$; see definition in Section 2.2 – for every hyperedge $e \in \mathcal{E}$ relevant to $\mathcal{H}$ (i.e., e is not subsumed by $\mathcal{H}$, as defined in Section 2.2) and active configuration $\boldsymbol{\eta} \in \mathit{config}(\mathcal{H})$. This is, in fact, trivial if e is disjoint with $\mathcal{H}$ (i.e., both attributes in e are light), in which case $R_e(\boldsymbol{\eta})$ is simply the relation R^{LL}_e in (16), which has already been obtained.

It remains to consider the crossing hyperedges $e \in \mathcal{E}$ (which contain one heavy attribute and one light attribute). W.l.o.g., let $e = \{X, Y\}$ where $X \in \mathcal{H}$ and $Y \in \mathcal{L}$. Imagine dividing R^{HL}_e into groups according to the X-values of the tuples therein. Then, for each active configuration $\boldsymbol{\eta} \in \mathit{config}(\mathcal{H})$, the set $R_e(\boldsymbol{\eta})$ is exactly the group having the X-value $\boldsymbol{\eta}(X)$. Motivated by this, we sort R^{HL}_e (defined in (17)) by attribute X in $O(\text{sort}(N))$ I/Os and sort the active configurations $\boldsymbol{\eta} \in \mathit{config}(\mathcal{H})$ by $\boldsymbol{\eta}(X)$ in $O(\text{sort}(|\mathit{config}(\mathcal{H})|) = O(N^{k/2}/(M^{k/2-1}B))$ I/Os. Then, by going through R^{HL}_e and $\mathit{config}(\mathcal{H})$ synchronously once, we can, for each $\boldsymbol{\eta} \in \mathit{config}(\mathcal{H})$, identify the starting disk address of $R_e(\boldsymbol{\eta})$ and store this address along with $\boldsymbol{\eta}$.

[4] Using the textbook algorithm "blocked nested loop".

Now that we have created $R_e(\boldsymbol{\eta})$ on the disk for every relevant hyperedge $e \in \mathcal{E}$ and active configuration $\boldsymbol{\eta} \in \mathit{config}(\mathcal{H})$, we can acquire the residual relation $R'_e(\boldsymbol{\eta})$ via a projection, as indicated in (2), which requires only a single scan of $R_e(\boldsymbol{\eta})$. This generates the input relations of the residual joins $\mathcal{Q}'(\boldsymbol{\eta})$ of all $\boldsymbol{\eta} \in \mathit{config}(\mathcal{H})$. The overall I/O cost is bounded by $O(N^{k/2}/(M^{k/2-1}B))$.

While the above algorithm fulfills the objective of creating the input relations of all residual joins, we will modify it slightly to better facilitate the subsequent algorithm design. The purpose of these modifications is to ensure that, for every active configuration $\boldsymbol{\eta} \in \mathit{config}(\mathcal{H})$ and every hyperedge $e \in \mathcal{E}$ containing a border attribute[5] $Y \in \mathcal{L}$, there should be a copy of $R'_e(\boldsymbol{\eta})$ on the disk that has been sorted on attribute Y. To that end, we change the aforementioned method for computing $R_e(\boldsymbol{\eta})$ as follows:

- If e is disjoint with $\mathcal{H}$, sort R_e^{LL} on Y in $O(\text{sort}(N))$ I/Os. Remember that $R_e^{\text{LL}} = R_e(\boldsymbol{\eta})$ regardless of $\boldsymbol{\eta} \in \mathit{config}(\mathcal{H})$.
- If e is a crossing hyperedge of the form $\{X, Y\}$ where $X \in \mathcal{H}$, instead of sorting R_e^{HL} only by attribute X as described earlier, we perform the sorting first by X and then break ties by Y. After that, the relations $R_e(\boldsymbol{\eta})$ of all $\boldsymbol{\eta} \in \mathit{config}(\mathcal{H})$ can still be obtained by going through (the sorted) R_e^{HL} and (the sorted) $\mathit{config}(\mathcal{H})$ synchronously once, but the tuples of each $R_e(\boldsymbol{\eta})$ are now sorted on attribute Y.

Every relation $R'_e(\boldsymbol{\eta})$ is still computed by taking a projection of $R_e(\boldsymbol{\eta})$ as before. The overall I/O complexity remains as $O(N^{k/2}/(M^{k/2-1}B))$.

Input Relations of Simplified Residual Joins. Next, for each border attribute $X \in \mathcal{X}$ and every active configuration $\boldsymbol{\eta} \in \mathit{config}(\mathcal{H})$, we compute the set $R''_X(\boldsymbol{\eta})$ defined in (6). This is the intersection of all the $\Pi_X(R'_e(\boldsymbol{\eta}))$, where e ranges over all the hyperedges in $\mathcal{G}'$ (defined in Section 2.2) containing X. Since (i) we have prepared a copy of $R'_e(\boldsymbol{\eta})$ sorted by X on the disk and (ii) $R'_e(\boldsymbol{\eta})$ has a size at most $N_{\boldsymbol{\eta}}$ (see (4)), the intersection can be computed in $O(\lceil N_{\boldsymbol{\eta}}/B \rceil)$ I/Os; note that the $R''_X(\boldsymbol{\eta})$ thus computed is sorted on X and has a size at most $N_{\boldsymbol{\eta}}$. The total cost for obtaining the sets $R''_X(\boldsymbol{\eta})$ of all $\boldsymbol{\eta} \in \mathit{config}(\mathcal{H})$ is bounded by

$$\sum_{\boldsymbol{\eta} \in \mathit{config}(\mathcal{H})} O(\lceil N_{\boldsymbol{\eta}}/B \rceil) \quad = \quad |\mathit{config}(\mathcal{H})| + \sum_{\boldsymbol{\eta} \in \mathit{config}(\mathcal{H})} O(N_{\boldsymbol{\eta}}/B) \tag{20}$$

Deng et al. showed [12, Lemma 11] that

$$\sum_{\boldsymbol{\eta} \in \mathit{config}(\mathcal{H})} N_{\boldsymbol{\eta}} = O(N^{k/2}/M^{k/2-1}).$$

Therefore:

$$(20) \quad = \quad O(N^{k/2}/M^{k/2}) + O(N^{k/2}/(M^{k/2-1}B)) = O(N^{k/2}/(M^{k/2-1}B)). \tag{21}$$

In the scenario where every hyperedge $e \in \mathcal{E}$ contains at most one light attribute, the set $\mathcal{I}$ of isolated attributes is exactly the set of border attributes, and hence, the simplified residual join $\mathcal{Q}''(\eta)$ of each $\boldsymbol{\eta} \in \mathit{config}(\mathcal{H})$ – see (7) – consists of $\{R''_X(\boldsymbol{\eta}) \mid X \text{ is a border attribute}\}$. We are therefore done with generating the input relations of all the simplified residual joins.

In the final segment of our proof (for Lemma 7), we consider that $\mathcal{E}$ has at least one edge $e = \{X_1, X_2\}$ where both X_1 and X_2 are light attributes. Note that this means $|\mathcal{H}| \leq k - 2$. Next, given an arbitrary such hyperedge $e = \{X_1, X_2\}$, we will explain how to produce on the disk the simplified residual relations $R''_e(\boldsymbol{\eta})$ under all active configurations $\boldsymbol{\eta} \in \mathit{config}(\mathcal{H})$ with $O(N^{k/2}/(M^{k/2-1}B))$ I/Os in total.

[5] As defined in Section 2.2, a border attribute is a light attribute that co-appears with a heavy attribute in some hyperedge of $\mathcal{E}$.

This is trivial if neither X_1 nor X_2 is a border attribute – in this case, $R''_e(\boldsymbol{\eta}) = R'_e(\boldsymbol{\eta})$, which has already been computed. Now consider the case where e has only one border attribute, say, X_1 (thus X_2 is not a border attribute; nonetheless, remember that X_2 is a light attribute). Earlier we have prepared a copy of $R'_e(\boldsymbol{\eta}) = R_e^{\text{LL}}$ sorted on X_1, as well as $R''_{X_1}(\boldsymbol{\eta})$, which is also sorted on X_1. Because the sizes of both $R'_e(\boldsymbol{\eta})$ and $R''_{X_1}(\boldsymbol{\eta})$ are at most $N_{\boldsymbol{\eta}}$, we can compute $R''_e(\boldsymbol{\eta}) = R'_e(\boldsymbol{\eta}) \bowtie R''_{X_1}(\boldsymbol{\eta})$ in $O(\lceil N_\eta/B \rceil)$ I/Os. Hence, the simplified residual relations $R''_e(\boldsymbol{\eta})$ under all $\boldsymbol{\eta} \in \mathit{config}(\mathcal{H})$ can be obtained in $O(\sum_{\boldsymbol{\eta} \in \mathit{config}(\mathcal{H})} O(\lceil N_{\boldsymbol{\eta}}/B \rceil))$ I/Os, which is $O(N^{k/2}/(M^{k/2-1}B))$; see (20) and (21).

The most non-trivial situation arises when both X_1 and X_2 in e are border attributes. This is where the "batched two-way semi-join reductions" problem (we will abbreviate the problem name as "batched semi" from now on) discussed in Section 4 comes in. Let us construct an instance of that problem as follows:

- The value of ℓ in "batched semi" equals $|\mathit{config}(\mathcal{H})| = O((N/M)^{|\mathcal{H}|/2}) = O((N/M)^{k/2-1})$, applying $|\mathcal{H}| \leq k-2$.
- The binary relation R in "batched semi" corresponds to R_e^{LL}.
- The degree threshold λ in "batched semi" equals the value of λ in (1), which is $O(\sqrt{NM})$.
- The i-th ($i \in [\ell]$) unary relation S_i in "batched semi" corresponds to the set $R''_{X_1}(\boldsymbol{\eta})$ of the i-th[6] active configuration $\boldsymbol{\eta} \in \mathit{config}(\mathcal{H})$. Remember that $R''_{X_1}(\boldsymbol{\eta})$ has been sorted on X_1.
- The i-th ($i \in [\ell]$) unary relation T_i in "batched semi" corresponds to set $R''_{X_2}(\boldsymbol{\eta})$ of the i-th active configuration $\boldsymbol{\eta} \in \mathit{config}(\mathcal{H})$. Remember that $R''_{X_2}(\boldsymbol{\eta})$ has been sorted on X_2.

"Batched semi" computes the result $\mathit{join}(\mathcal{Q}_i)$ of the join $\mathcal{Q}_i$ defined in (11). Based on our construction, $\mathit{join}(\mathcal{Q}_i)$ equals $R_e^{\text{LL}} \bowtie R''_{X_1}(\boldsymbol{\eta}) \bowtie R''_{X_2}(\boldsymbol{\eta})$ – with $\boldsymbol{\eta}$ being the i-th configuration in $\mathit{config}(\mathcal{H})$ – which is exactly the simplified residual relation $R''_e(\boldsymbol{\eta})$ of e under $\boldsymbol{\eta}$. From Lemma 6, we can assert that the relations $R''_e(\boldsymbol{\eta})$ under all $\boldsymbol{\eta} \in \mathit{config}(\mathcal{H})$ can be computed with an I/O cost

$$
\begin{aligned}
& O\left(\left(\frac{N}{\lambda} + \frac{\lambda}{M}\right) \frac{\sum_{\boldsymbol{\eta} \in \mathit{config}(\mathcal{H})} |R''_{X_1}(\boldsymbol{\eta})| + |R''_{X_2}(\boldsymbol{\eta})|}{B} + \frac{\ell \cdot N}{B} + \frac{\ell \cdot N^2 M}{\lambda^2 \cdot B} + \text{sort}(N)\right) \\
= \; & O\left(\frac{\sqrt{N}}{\sqrt{M}B}\Big(\sum_{\boldsymbol{\eta} \in \mathit{config}(\mathcal{H})} (|R''_{X_1}(\boldsymbol{\eta})| + |R''_{X_2}(\boldsymbol{\eta})|)\Big) + \frac{\ell \cdot N}{B} + \text{sort}(N)\right) \\
= \; & O\left(\frac{\sqrt{N}}{\sqrt{M}B}\Big(\sum_{\boldsymbol{\eta} \in \mathit{config}(\mathcal{H})} (|R''_{X_1}(\boldsymbol{\eta})| + |R''_{X_2}(\boldsymbol{\eta})|)\Big) + \frac{N^{k/2}}{M^{k/2-1}B}\right) \qquad (22)
\end{aligned}
$$

where the last step applied $\ell = O((N/M)^{k/2-1})$.

▶ **Proposition 8.** $\sum_{\boldsymbol{\eta} \in \mathit{config}(\mathcal{H})} |R''_{X_1}(\boldsymbol{\eta})| = O\left(\sqrt{NM} \cdot \left(\frac{N}{M}\right)^{k/2-1}\right)$

Proof. Because X_1 is a border attribute, there must exist a heavy attribute $H \in \mathcal{H}$ such that $e = \{X_1, Y\}$ is a hyperedge in $\mathcal{E}$. By how $R''_{X_1}(\boldsymbol{\eta})$ is computed in (6), we know that $R''_{X_1}(\boldsymbol{\eta})$ is a subset of $R'_e(\boldsymbol{\eta})$. Furthermore, since $R'_e(\boldsymbol{\eta}) = \Pi_{e \setminus \mathcal{H}}(R_e(\boldsymbol{\eta}))$ (see (2)), it must hold that

$$|R''_{X_1}(\boldsymbol{\eta})| \leq |R'_e(\boldsymbol{\eta})| \leq |R_e(\boldsymbol{\eta})|.$$

[6] Here, impose an arbitrary ordering of $\mathit{config}(\mathcal{H})$.

Next, we will prove that

$$\sum_{\boldsymbol{\eta} \in config(\mathcal{H})} |R_e(\boldsymbol{\eta})| = O\left(\sqrt{NM} \cdot \left(\frac{N}{M}\right)^{k/2-1}\right) \tag{23}$$

which will then imply the claim in the proposition.

To that end, we break the left hand side of (23) as follows:

$$\sum_{\boldsymbol{\eta} \in config(\mathcal{H})} |R_e(\boldsymbol{\eta})| = \sum_{y \in \mathbf{adom}^H(Y)} \left(\sum_{\boldsymbol{\eta} \in config(\mathcal{H}): \boldsymbol{\eta}(Y) = y} |R_e(\boldsymbol{\eta})| \right). \tag{24}$$

Observe from the definition of $R_e(\boldsymbol{\eta})$ that $R_e(\boldsymbol{\eta})$ is simply $\sigma_{Y=y}(R_e)$ where σ is the standard *selection* operator in relational algebra. Hence, for any $y \in \mathbf{adom}^H(Y)$, the size $|R_e(\boldsymbol{\eta})|$ is the same for all $\boldsymbol{\eta} \in config(\mathcal{H})$ satisfying $\boldsymbol{\eta}(Y) = y$. As there are at most $O((N/M)^{(|\mathcal{H}|-1)/2})$ such active configurations $\boldsymbol{\eta}$, we can derive

$$\begin{aligned}
(24) &= O((N/M)^{(|\mathcal{H}|-1)/2}) \cdot \sum_{y \in \mathbf{adom}^H(Y)} |\sigma_{Y=y}(R_e)| \\
&\leq O((N/M)^{(|\mathcal{H}|-1)/2}) \cdot |R_e| \\
&= O\left(\frac{N^{(|\mathcal{H}|+1)/2}}{M^{(|\mathcal{H}|-1)/2}}\right) \\
(\text{as } |H| \leq k-2) &= O\left(\sqrt{NM} \cdot \left(\frac{N}{M}\right)^{k/2-1}\right)
\end{aligned}$$

as claimed in (23). ◀

A symmetric argument yields $\sum_{\boldsymbol{\eta} \in config(\mathcal{H})} |R''_{X_2}(\boldsymbol{\eta})| = O(\sqrt{NM} \cdot \left(\frac{N}{M}\right)^{k/2-1})$. Plugging these relationships into (22), we can derive

$$(22) = O\left(\frac{\sqrt{N}}{\sqrt{M}B} \cdot \sqrt{NM}\left(\frac{N}{M}\right)^{k/2-1} + \frac{N^{k/2}}{M^{k/2-1}B}\right) = O\left(\frac{N^{k/2}}{M^{k/2-1}B}\right).$$

We now conclude that the input relations of the simplified residual joins $\mathcal{Q}''(\boldsymbol{\eta})$ of all $\boldsymbol{\eta} \in config(\mathcal{H})$ can be computed in $O(N^{k/2}/(M^{k/2-1}B))$ I/Os in total. This completes the entire proof of Lemma 7.

6 Conclusions

By putting together Lemmas 1, 4, and 7, we have arrived at the main result of this paper:

▶ **Theorem 9.** *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. There is an algorithm in the external memory model that, with probability at least $1 - 1/|E|^c$ where c is an arbitrarily high constant decided before running the algorithm, emits every occurrence of Q in G exactly once with $O(|E|^\rho/(M^{\rho-1}B))$ I/Os, where ρ 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 optimally settles the subgraph enumeration problem in external memory when randomization is permitted. The main open problem left behind by this work is to achieve the I/O complexity $O(|E|^\rho/(M^{\rho-1}B))$ deterministically.

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. doi:10.1145/48529.48535.

2 Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *Journal of the ACM (JACM)*, 42(4):844–856, 1995. doi:10.1145/210332.210337.

3 Noga Alon, Raphael Yuster, and Uri Zwick. Finding and counting given length cycles. *Algorithmica*, 17(3):209–223, 1997. doi:10.1007/BF02523189.

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. doi:10.4230/LIPIcs.ICDT.2021.3.

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. doi:10.1137/1.9781611976465.138.

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. doi:10.1145/3125500.

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. doi:10.1007/978-3-662-43948-7_19.

8 N. Chiba and T. Nishizeki. Arboricity and subgraph listing algorithms. *SIAM Journal of Computing*, 14(1):210–223, 1985. doi:10.1137/0214017.

9 Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of ACM Symposium on Theory of Computing (STOC)*, pages 151–158, 1971. doi:10.1145/800157.805047.

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. doi:10.1145/3055399.3055502.

11 Shiyuan Deng, Shangqi Lu, and Yufei Tao. On join sampling and the hardness of combinatorial output-sensitive join algorithms. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, pages 99–111, 2023. doi:10.1145/3584372.3588666.

12 Shiyuan Deng, Francesco Silvestri, and Yufei Tao. Enumerating subgraphs of constant sizes in external memory. In *Proceedings of International Conference on Database Theory (ICDT)*, pages 4:1–4:20, 2023. doi:10.4230/LIPIcs.ICDT.2023.4.

13 David Eppstein. Arboricity and bipartite subgraph listing algorithms. *Information Processing Letters (IPL)*, 51(4):207–211, 1994. doi:10.1016/0020-0190(94)90121-X.

14 David Eppstein. Subgraph isomorphism in planar graphs and related problems. *J. Graph Algorithms Appl.*, 3(3):1–27, 1999. doi:10.7155/jgaa.00014.

15 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. doi:10.1007/978-3-642-17517-6_36.

16 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. doi:10.1137/140978211.

17 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. doi:10.1016/j.jcss.2011.10.001.

18 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. doi:10.1007/s00453-019-00552-1.

19 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. doi:10.1016/j.dam.2012.01.024.

20 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. doi:10.1145/2902251.2902292.

21 Xiaocheng Hu, Miao Qiao, and Yufei Tao. External memory stream sampling. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, pages 229–239, 2015. doi:10.1145/2745754.2745757.

22 Xiaocheng Hu, Miao Qiao, and Yufei Tao. Join dependency testing, loomis-whitney join, and triangle enumeration. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, pages 291–301, 2015. doi:10.1145/2745754.2745768.

23 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. doi:10.1016/j.jcss.2016.05.005.

24 Manas Joglekar and Christopher Re. It's all a matter of degree - using degree information to optimize multiway joins. *Theory Comput. Syst.*, 62(4):810–853, 2018. doi:10.1007/s00224-017-9811-8.

25 Bas Ketsman and Dan Suciu. A worst-case optimal multi-round algorithm for parallel computation of conjunctive queries. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, pages 417–428, 2017. doi:10.1145/3034786.3034788.

26 Bas Ketsman, Dan Suciu, and Yufei Tao. A near-optimal parallel algorithm for joining binary relations. *Log. Methods Comput. Sci.*, 18(2), 2022. doi:10.46298/lmcs-18(2:6)2022.

27 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. doi:10.1145/2745754.2745776.

28 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. doi:10.1145/2967101.

29 Mahmoud Abo Khamis, Hung Q. Ngo, and Dan Suciu. What do shannon-type inequalities, submodular width, and disjunctive datalog have to do with one another? In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, pages 429–444, 2017. doi:10.1145/3034786.3056105.

30 Ton Kloks, Dieter Kratsch, and Haiko Müller. Finding and counting small induced subgraphs efficiently. *Information Processing Letters (IPL)*, 74(3-4):115–121, 2000. doi:10.1016/S0020-0190(00)00047-8.

31 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. doi:10.4230/LIPIcs.ICDT.2020.21.

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

33 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. doi:10.1145/2594538.2594547.

34 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. doi:10.1145/3180143.

35 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. doi:10.1145/2590989.2590991.

36 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. doi:10.1145/2594538.2594552.

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

38 Yufei Tao. A simple parallel algorithm for natural joins on binary relations. In *Proceedings of International Conference on Database Theory (ICDT)*, pages 25:1–25:18, 2020. doi:10.4230/LIPIcs.ICDT.2020.25.

39 Todd L. Veldhuizen. Triejoin: A simple, worst-case optimal join algorithm. In *Proceedings of International Conference on Database Theory (ICDT)*, pages 96–106, 2014. doi:10.5441/002/icdt.2014.13.

40 Virginia Vassilevska Williams and Ryan Williams. Finding, minimizing, and counting weighted subgraphs. *SIAM Journal of Computing*, 42(3):831–854, 2013. doi:10.1137/09076619X.

A Completing the Proof of Lemma 5

Focusing on an arbitrary hyperedge $e = \{X_i, X_j\} \in \mathcal{E}$, next we will explain how to process R_e in $O((1 + \frac{D}{\sqrt{M}})^2 \cdot \frac{1}{B} + \text{sort}(N))$ I/Os such that, given any interval $I_i \in S^{intv}_{X_i}$, and any interval $I_j \in S^{intv}_{X_j}$, we can load $R_e(I_i, I_j)$ into memory using $O(M/B)$ I/Os.

Recall that the intervals in $S^{intv}_{X_i}$ are disjoint. We order those intervals by their starting points; an interval has *rank* t if it has the t-th smallest starting point. Define ranks similarly for the intervals in $S^{intv}_{X_j}$.

For each tuple $\boldsymbol{u} \in R_e$, we associate it with an X_i*-interval rank*, which is the rank of the interval in $S^{intv}_{X_i}$ covering $\boldsymbol{u}(X_i)$. By sorting, we can obtain the X_i-interval ranks of all the tuples in R_e using $O(\text{sort}(N))$ I/Os. We also associate $\boldsymbol{u}$ with an X_j*-interval rank*, defined similarly with respect to X_j. The X_j-interval ranks of all the tuples in R_e can again be decided using $O(\text{sort}(N))$ I/Os.

We now sort the tuples of R_e by their X_i-interval ranks, breaking ties by X_j-interval ranks. After sorting, for any intervals $I_i \in S^{intv}_{X_i}$ and $I_j \in S^{intv}_{X_j}$, the $O(M)$ tuples in $R_e(I_i, I_j)$ have been grouped into $O(M/B)$ consecutive blocks on the disk. We bookmark the group's starting address such that, given the ranks of I_i and I_j, we can locate the group on the disk immediately. This requires materializing the starting addresses of all the $|S^{intv}_{X_i}| \cdot |S^{intv}_{X_j}| = O((1 + D/\sqrt{M})^2)$ groups in a two-dimensional array, which can be created in $O(1 + (1 + \frac{D}{\sqrt{M}})^2 \cdot \frac{1}{B} + \frac{N}{B})$ I/Os by scanning the (sorted) R_e once.

B Preprocessing the Input Relations for Proving Lemma 6

We will first explain how to preprocess R in $O((\frac{|R|}{\lambda})^2 \cdot \frac{1}{B} + \text{sort}(|R|))$ I/Os such that, for any interval $I_X \in S^{intv}_X$ and any interval $I_Y \in S^{intv}_Y$, we can ensure that (i) $R(I_i, I_j)$ is stored in consecutive blocks and (ii) the first block's address can be obtained in one I/O.

It suffices to re-apply the techniques illustrated in Appendix A. Sort the intervals of S^{intv}_X by their starting points and do the same for S^{intv}_Y, which requires $O(\text{sort}(|R|/\lambda))$ I/Os. An interval of S^{intv}_X (resp., S^{intv}_Y) is said to have *rank* t if it has the t-th smallest starting point among all the intervals in S^{intv}_X (resp., S^{intv}_Y). Associate each tuple $\boldsymbol{u} \in R$ with an X- (resp., Y-) *interval rank*, which is the rank of the interval in S^{intv}_X covering $\boldsymbol{u}(X)$ (resp., S^{intv}_Y covering $\boldsymbol{u}(Y)$). Sort the tuples of R by their X-interval ranks, breaking ties by their Y-interval ranks, after which the tuples in $R(I_X, I_Y)$ are placed in a sequence of consecutive blocks for any interval pair $(I_X, I_Y) \in S^{intv}_X \times S^{intv}_Y$. We record the sequence's starting address such that the sequence can be located immediately based on the ranks of I_X and I_Y. This requires materializing $|S^{intv}_X| \cdot |S^{intv}_Y| = O((|R|/\lambda)^2)$ starting addresses in a two-dimensional array, which can be created in $O(1 + (\frac{|R|}{\lambda})^2 \cdot \frac{1}{B} + \frac{|R|}{B})$ I/Os by scanning R once.

We now turn our attention to preprocessing $S_1, S_2, ..., S_\ell$. For each $i \in [\ell]$, recall that S_i has been sorted on attribute X. By scanning S_i synchronously with the (already sorted) S_X^{intv}, we can ensure that, for each interval $I_X \in S_X^{intv}$, the set $S_i(I_X)$ is stored in a sequence of consecutive blocks. We record the sequence's starting address such that the sequence can be located directly based on the rank of I_X. This requires materializing $|S_X^{intv}| = O(\lceil |R|/\lambda \rceil)$ starting addresses in an array, which can be created in $O(1 + \frac{|R|}{\lambda B} + \frac{|S_i|}{B})$ I/Os by scanning S_i once. The total I/O cost of preprocessing the S_i of all $i \in [\ell]$ this way is $O(\ell + \ell\frac{|R|}{\lambda B} + \sum_{i=1}^{\ell} |S_i|/B)$.

The preprocessing of $T_1, T_2, ..., T_\ell$ is similar and omitted.

C Proof of Equation (14)

We will prove the following relationship for each $i \in [\ell]$:

$$\begin{aligned} &\sum_{(I_X, I_Y) \in S_X^{intv} \times S_Y^{intv}} \left\lceil \frac{|R(I_X, I_Y)|}{M} \right\rceil \cdot \frac{|S_i(I_X)| + |T_i(I_Y)| + M}{B} \\ = \; & O\Big(\Big(\frac{|R|}{\lambda} + \frac{\lambda}{M} \Big) \frac{|S_i| + |T_i|}{B} + \frac{|R|}{B} + \frac{|R|^2 M}{\lambda^2 B} \Big). \end{aligned} \tag{25}$$

Equation (14) will then follow from a simple summation on all $i \in [\ell]$.

Clearly:

$$\begin{aligned} &\text{left hand side of (25)} \\ \leq \; & \sum_{(I_X, I_Y)} \left(1 + \frac{|R(I_X, I_Y)|}{M} \right) \cdot \frac{|S_i(I_X)| + |T_i(I_Y)| + M}{B} \\ = \; & \sum_{(I_X, I_Y)} \frac{|S_i(I_X)| + |T_i(I_Y)| + M}{B} + \sum_{(I_X, I_Y)} \frac{|R(I_X, I_Y)|}{M} \cdot \frac{|S_i(I_X)| + |T_i(I_Y)| + M}{B}. \end{aligned} \tag{26}$$

The first summation of (26) is easy to bound:

$$\begin{aligned} \sum_{(I_X, I_Y)} \frac{|S_i(I_X)| + |T_i(I_Y)| + M}{B} &= \sum_{(I_X, I_Y)} \frac{|S_i(I_X)|}{B} + \sum_{(I_X, I_Y)} \frac{|T_i(I_Y)|}{B} + \sum_{(I_X, I_Y)} \frac{M}{B} \\ &= \frac{|T_Y^{intv}| \cdot |S_i|}{B} + \frac{|S_X^{intv}| \cdot |T_i|}{B} + \frac{M}{B} \cdot |S_X^{intv}| \cdot |T_Y^{intv}| \\ &= O\left(\frac{|R|(|S_i| + |T_i|)}{\lambda \cdot B} + \frac{M}{B} \frac{|R|^2}{\lambda^2} \right) \end{aligned} \tag{27}$$

where the last step used $|S_X^{intv}| = O(|R|/\lambda)$ and $|T_Y^{intv}| = O(|R|/\lambda)$. To unfold the second summation of (26), let us first rearrange the terms:

$$\begin{aligned} &\sum_{(I_X, I_Y)} \frac{|R(I_X, I_Y)|}{M} \cdot \frac{|S_i(I_X)| + |T_i(I_Y)| + M}{B} \\ = \; & \sum_{(I_X, I_Y)} \frac{|R(I_X, I_Y)|}{M} \frac{|S_i(I_X)|}{B} + \sum_{(I_X, I_Y)} \frac{|R(I_X, I_Y)|}{M} \frac{|T_i(I_Y)|}{B} + \sum_{(I_X, I_Y)} \frac{|R(I_X, I_Y)|}{B} \end{aligned} \tag{28}$$

To facilitate the subsequent derivation, let us define

$$R(I_X, -) \;=\; \{\boldsymbol{u} \in R \mid \boldsymbol{u}(X) \in I_X\}$$

for any interval $I_X \in S_X^{intv}$. Note that the size of $R(I_X, -)$ is bounded by $O(\lambda)$ due to the way that S_X^{intv} is constructed. Equipped with this, we can derive

$$\begin{aligned}\sum_{(I_X,I_Y)} |R(I_X,I_Y)||S_i(I_X)| &= \sum_{I_X} |S_i(I_X)| \cdot \left(\sum_{I_Y} |R(I_X,I_Y)|\right) \\ &= \sum_{I_X} |S_i(I_X)| \cdot |R(I_X,-)| \\ &= O(\lambda) \cdot \sum_{I_X} |S_i(I_X)| = O(\lambda \cdot |S_i|). \end{aligned} \tag{29}$$

A symmetric analysis shows that $\sum_{(I_X,I_Y)} |R(I_X,I_Y)||T_i(I_Y)| = O(\lambda \cdot |T_i|)$. Utilizing also the obvious fact $\sum_{(I_X,I_Y)} |R(I_X,I_Y)| = |R|$, we obtain

$$(28) = O\left(\frac{\lambda \cdot |S_i|}{MB} + \frac{\lambda \cdot |T_i|}{MB} + \frac{|R|}{B}\right). \tag{30}$$

Equation (25) now follows from (26), (27) and (30).