Counting Triangles under Updates in Worst-Case Optimal Time

Ahmet Kara
Department of Computer Science, University of Oxford, Oxford, UK
ahmet.kara@cs.ox.ac.uk

Hung Q. Ngo
RelationalAI, Inc., Berkeley, CA, USA
hung.ngo@relational.ai

Milos Nikolic[1]
School of Informatics, University of Edinburgh, Edinburgh, UK
milos.nikolic@ed.ac.uk

Dan Olteanu
Department of Computer Science, University of Oxford, Oxford, UK
dan.olteanu@cs.ox.ac.uk

Haozhe Zhang
Department of Computer Science, University of Oxford, Oxford, UK
haozhe.zhang@cs.ox.ac.uk

Abstract

We consider the problem of incrementally maintaining the triangle count query under single-tuple updates to the input relations. We introduce an approach that exhibits a space-time tradeoff such that the space-time product is quadratic in the size of the input database and the update time can be as low as the square root of this size. This lowest update time is worst-case optimal conditioned on the Online Matrix-Vector Multiplication conjecture.

The classical and factorized incremental view maintenance approaches are recovered as special cases of our approach within the space-time tradeoff. In particular, they require linear-time maintenance under updates, which is suboptimal. Our approach can also count all triangles in a static database in the worst-case optimal time needed for enumerating them.

2012 ACM Subject Classification Theory of computation → Database query processing and optimization (theory); Information systems → Database views; Information systems → Data streams

Keywords and phrases incremental view maintenance, amortized analysis, data skew

Related Version An extended version of this work is available online [13].

Funding This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 682588. Kara acknowledges funding from Fondation Wiener Anspach.

Acknowledgements Olteanu would like to thank Nicole Schweikardt for the connection between the Online Matrix-Vector Multiplication conjecture and the triangle query.

1 Introduction

We consider the problem of incrementally maintaining the result of the triangle count query

$$Q() = \sum_{a \in \mathsf{Dom}(A)} \sum_{b \in \mathsf{Dom}(B)} \sum_{c \in \mathsf{Dom}(C)} R(a,b) \cdot S(b,c) \cdot T(c,a) \tag{1}$$

[1] Work performed while being at the University of Oxford.

under single-tuple updates to the relations R, S, and T with schemas (A, B), (B, C), and (C, A), respectively. The relations are given as functions mapping tuples over relation schemas to tuple multiplicities. A single-tuple update $\delta R = \{ (\alpha, \beta) \mapsto m \}$ to relation R maps the tuple (α, β) to a nonzero multiplicity m, which is positive for inserts and negative for deletes.

The triangle query and its counting variant have served as a milestone for worst-case optimality of join algorithms in the centralized and parallel settings and for randomized approximation schemes for data processing. They serve as the workhorse showcasing suboptimality of mainstream join algorithms used currently by virtually all commercial database systems. For a database $\mathbf{D}$ consisting of R, S, and T, standard binary join plans implementing these queries may take $O(|\mathbf{D}|^2)$ time, yet these queries can be solved in $\mathcal{O}(|\mathbf{D}|^{\frac{3}{2}})$ time [2]. This observation motivated a new line of work on worst-case optimal algorithms for arbitrary join queries [18]. The triangle query has also served as a yardstick for understanding the optimal communication cost for parallel query evaluation in the Massively Parallel Communication model [15]. The triangle count query has witnessed the development of randomized approximation schemes with increasingly lower time and space requirements, e.g., [9].

A worst-case optimal result for incrementally maintaining the exact triangle count query has so far not been established. Incremental maintenance algorithms may benefit from a good range of processing techniques whose flexible combinations may make it harder to reason about optimality. Such techniques include algorithms for aggregate-join queries with low complexity developed for the non-incremental case [17]; pre-materialization of views that reduces maintenance of the query to that of simpler subqueries [14]; and delta processing that allows to only compute the change in the result instead of the entire result [7].

1.1 Existing Incremental View Maintenance (IVM) Approaches

The problem of incrementally maintaining the triangle count has received a fair amount of attention. Existing exact approaches require at least linear time in worst case. After each update to a database $\mathbf{D}$, the naïve approach joins the relations R, S, and T in time $\mathcal{O}(|\mathbf{D}|^{\frac{3}{2}})$ using a worst-case optimal algorithm [2, 18] and counts the result tuples. The number of distinct tuples in the result is at most $|\mathbf{D}|^{\frac{3}{2}}$, which is a well-known result by Loomis and Whitney from 1949 (see recent notes on the history of this result [17]). The classical first-order IVM [7] computes on the fly a delta query δQ per single-tuple update δR to relation R (or any other relation) and updates the query result:

$$\delta Q() = \delta R(\alpha, \beta) \cdot \sum_{c \in \mathsf{Dom}(C)} S(\beta, c) \cdot T(c, \alpha), \qquad Q() = Q() + \delta Q().$$

The delta computation takes $\mathcal{O}(|\mathbf{D}|)$ time since it needs to intersect two lists of possibly linearly many C-values that are paired with β in S and with α in T (i.e., the multiplicity of such pairs in S and T is nonzero). The recursive IVM [14] speeds up the delta computation by precomputing three auxiliary views representing the update-independent parts of the delta queries for updates to R, S, and T:

$$V_{ST}(b, a) = \sum_{c \in \mathsf{Dom}(C)} S(b, c) \cdot T(c, a)$$
$$V_{TR}(c, b) = \sum_{a \in \mathsf{Dom}(A)} T(c, a) \cdot R(a, b)$$
$$V_{RS}(a, c) = \sum_{b \in \mathsf{Dom}(B)} R(a, b) \cdot S(b, c).$$

These three views take $\mathcal{O}(|\mathbf{D}|^2)$ space but allow to compute the delta query for single-tuple updates to the input relations in $\mathcal{O}(1)$ time. Computing the delta $\delta Q() = \delta R(\alpha, \beta) \cdot V_{ST}(\beta, \alpha)$ requires just a constant-time lookup in V_{ST}; however, maintaining the views V_{RS} and V_{TR}, which refer to R, still requires $\mathcal{O}(|\mathbf{D}|)$ time. The factorized IVM [19] materializes only one of the three views, for instance, V_{ST}. In this case, the maintenance under updates to R takes $\mathcal{O}(1)$ time, but the maintenance under updates to S and T still takes $\mathcal{O}(|\mathbf{D}|)$ time.

Further exact IVM approaches focus on acyclic conjunctive queries. For free-connex acyclic conjunctive queries, the dynamic Yannakakis approach allows for enumeration of result tuples with constant delay under single-tuple updates [11]. For databases with or without integrity constraints, it is known that a strict, small subset of the class of acyclic conjunctive queries admit constant-time update, while all other conjunctive queries have update times dependent on the size of the input database [4, 5].

Further away from our line of work is the development of dynamic descriptive complexity, starting with the DynFO complexity class and the much-acclaimed result on FO expressibility of the maintenance for graph reachability under edge inserts and deletes, cf. a recent survey [20]. The k-clique query can be maintained under edge inserts by a quantifier-free update program of arity $k-1$ but not of arity $k-2$ [22].

A distinct line of work investigates randomized approximation schemes with an arbitrary relative error for counting triangles in a graph given as a stream of edges, e.g., [3, 12, 6, 16, 8]. Each edge in the data stream corresponds to a tuple insert, and tuple deletes are not considered. The emphasis of these approaches is on space efficiency, and they express the space utilization as a function of the number of nodes and edges in the input graph and of the number of triangles. The space utilization is generally sublinear but may become superlinear if, for instance, the number of edges is greater than the square root of the number of triangles. The update time is polylogarithmic in the number of nodes in the graph.

A complementary line of work unveils structure in the PTIME complexity class by giving lower bounds on the complexity of problems under various conjectures [10, 21].

▶ **Definition 1** (Online Matrix-Vector Multiplication (OMv) [10]). *We are given an $n \times n$ Boolean matrix $\mathbf{M}$ and receive n column vectors of size n, denoted by $\mathbf{v}_1, \ldots, \mathbf{v}_n$, one by one; after seeing each vector $\mathbf{v}_i$, we output the product $\mathbf{M}\mathbf{v}_i$ before we see the next vector.*

▷ Conjecture 2 (OMv Conjecture, Theorem 2.4 in [10]). For any $\gamma > 0$, there is no algorithm that solves OMv in time $\mathcal{O}(n^{3-\gamma})$.

The OMv conjecture has been used to exhibit conditional lower bounds for many dynamic problems, including those previously based on other popular problems and conjectures, such as 3SUM and combinatorial Boolean matrix multiplication [10]. This also applies to our triangle count query: For any $\gamma > 0$ and database of domain size n, there is no algorithm that incrementally maintains the triangle count under single-tuple updates with arbitrary preprocessing time, $\mathcal{O}(n^{1-\gamma})$ update time, and $\mathcal{O}(n^{2-\gamma})$ answer time, unless the OMv conjecture fails [4].

1.2 Our Contribution

This paper introduces IVM$^\epsilon$, an incremental view maintenance approach that maintains the triangle count in amortized sublinear time. Our main result is as follows:

▶ **Theorem 3.** *Given a database $\mathbf{D}$ and $\epsilon \in [0, 1]$, IVM$^\epsilon$ incrementally maintains the result of Query (1) under single-tuple updates to $\mathbf{D}$ with $\mathcal{O}(|\mathbf{D}|^{\frac{3}{2}})$ preprocessing time, $\mathcal{O}(|\mathbf{D}|^{\max\{\epsilon, 1-\epsilon\}})$ amortized update time, constant answer time, and $\mathcal{O}(|\mathbf{D}|^{1+\min\{\epsilon, 1-\epsilon\}})$ space.*

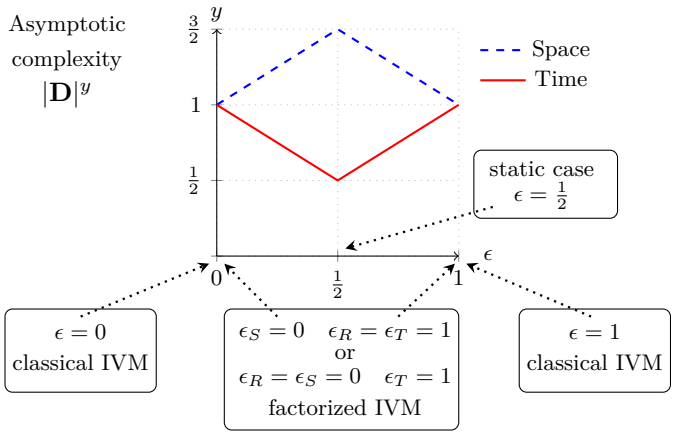

Figure 1 IVM$^\epsilon$'s space and amortized update time parameterized by ϵ. The classical IVM is recovered by setting $\epsilon \in \{0, 1\}$. The factorized IVM is recovered by setting $\epsilon_R \in \{0, 1\}$, $\epsilon_S = 0$, and $\epsilon_T = 1$ when V_{ST} is materialized (similar treatment when V_{RS} or V_{TR} is materialized). For $\epsilon = \frac{1}{2}$, IVM$^\epsilon$ counts all triangles in a static database in the worst-case optimal time for enumerating them.

The preprocessing time is for computing the triangle count on the initial database before the updates; if we start with the empty database, then this time is $\mathcal{O}(1)$. The IVM$^\epsilon$ approach exhibits a tradeoff between space and amortized update time, cf. Figure 1.

IVM$^\epsilon$ uses a data structure that partitions each input relation into a heavy part and a light part based on the degrees of data values. The degree of an A-value a in relation R is the number of B-values paired with a in R. The light part of R consists of all tuples (a, b) from R such that the degree of a in R is below a certain threshold that depends on the database size and ϵ. All other tuples are included in the heavy part of R. Similarly, the relations S and T are partitioned based on the degrees of B-values in S and C-values in T, respectively. The maintenance is adaptive in that it uses different evaluation strategies for different heavy-light combinations of parts of the input relations that overall keep the update time sublinear. Section 3 introduces this adaptive maintenance strategy.

As the database evolves under updates, IVM$^\epsilon$ needs to rebalance the heavy-light partitions to account for a new database size and updated degrees of data values. While this rebalancing may take superlinear time, it remains sublinear per single-tuple update. The update time is therefore amortized. Section 4 discusses the rebalancing strategy of IVM$^\epsilon$.

For $\epsilon = \frac{1}{2}$, IVM$^\epsilon$ achieves the lowest update time $\mathcal{O}(|\mathbf{D}|^{\frac{1}{2}})$ while requiring $\mathcal{O}(|\mathbf{D}|^{\frac{3}{2}})$ space. This update time is optimal conditioned on the OMv conjecture. For this, we specialize the lower bound result in [4] to refer to the size $|\mathbf{D}|$ of the database:

▶ **Proposition 4.** *For any $\gamma > 0$ and database $\mathbf{D}$, there is no algorithm that incrementally maintains the result of Query (1) under single-tuple updates to $\mathbf{D}$ with arbitrary preprocessing time, $\mathcal{O}(|\mathbf{D}|^{\frac{1}{2}-\gamma})$ amortized update time, and $\mathcal{O}(|\mathbf{D}|^{1-\gamma})$ answer time, unless the OMv conjecture fails.*

This lower bound is shown in the extended paper [13]. Theorem 3 and Proposition 4 imply that IVM$^\epsilon$ incrementally maintains the triangle count with optimal update time:

▶ **Corollary 5** (Theorem 3 and Proposition 4). *Given a database $\mathbf{D}$, IVM$^\epsilon$ incrementally maintains the result of Query (1) under single-tuple updates to $\mathbf{D}$ with worst-case optimal amortized update time $\mathcal{O}(|\mathbf{D}|^{\frac{1}{2}})$ and constant answer time, unless the OMv conjecture fails.*

IVM$^\epsilon$ also applies to triangle count queries with self-joins, such as when maintaining the count of triangles in a graph given by the edge relation. The space and time complexities are the same as in Theorem 3 [13].

IVM$^\epsilon$ defines a continuum of maintenance approaches that exhibit a space-time tradeoff based on ϵ. As depicted in Figure 1, the classical first-order IVM and the factorized IVM are specific extreme points in this continuum. To recover the former, we set $\epsilon \in \{0, 1\}$ for $\mathcal{O}(|\mathbf{D}|)$ update time and $\mathcal{O}(|\mathbf{D}|)$ space for the input relations. To recover the latter, we use a distinct parameter ϵ per relation: for example, using $\epsilon_R \in \{0, 1\}$, $\epsilon_S = 0$, and $\epsilon_T = 1$, we support updates to R in $\mathcal{O}(1)$ time and updates to S and T in $\mathcal{O}(|\mathbf{D}|)$ time; the view V_{ST} takes $\mathcal{O}(|\mathbf{D}|^2)$ space [13].

We observe that at optimality, IVM$^\epsilon$ recovers the worst-case optimal time $\mathcal{O}(|\mathbf{D}|^{\frac{3}{2}})$ of non-incremental algorithms for enumerating all triangles [18]. Whereas these algorithms are monolithic and require processing the input data in bulk and all joins at the same time, IVM$^\epsilon$ achieves the same complexity by inserting $|\mathbf{D}|$ tuples one at a time in initially empty relations R, S, and T, and by using standard join plans [13].

2 Preliminaries

Data Model. A schema $\mathbf{X}$ is a tuple of variables. Each variable X has a discrete domain $\mathsf{Dom}(X)$ of data values. A tuple $\mathbf{x}$ of data values over schema $\mathbf{X}$ is an element from $\mathsf{Dom}(\mathbf{X}) = \prod_{X \in \mathbf{X}} \mathsf{Dom}(X)$. We use uppercase letters for variables and lowercase letters for data values. Likewise, we use bold uppercase letters for schemas and bold lowercase letters for tuples of data values.

A relation K over schema $\mathbf{X}$ is a function $K : \mathsf{Dom}(\mathbf{X}) \to \mathbb{Z}$ mapping tuples over $\mathbf{X}$ to integers such that $K(\mathbf{x}) \neq 0$ for finitely many tuples $\mathbf{x}$. We say that a tuple $\mathbf{x}$ is in K, denoted by $\mathbf{x} \in K$, if $K(\mathbf{x}) \neq 0$. The value $K(\mathbf{x})$ represents the multiplicity of $\mathbf{x}$ in K. The size $|K|$ of K is the size of the set $\{\mathbf{x} \mid \mathbf{x} \in K\}$. A database $\mathbf{D}$ is a set of relations, and its size $|\mathbf{D}|$ is the sum of the sizes of the relations in $\mathbf{D}$.

Given a tuple $\mathbf{x}$ over schema $\mathbf{X}$ and a variable X in $\mathbf{X}$, we write $\mathbf{x}[X]$ to denote the value of X in $\mathbf{x}$. For a relation K over $\mathbf{X}$, a variable X in $\mathbf{X}$, and a data value $x \in \mathsf{Dom}(X)$, we use $\sigma_{X=x}K$ to denote the set of tuples in K whose X-value is x, that is, $\sigma_{X=x}K = \{\, \mathbf{x} \mid \mathbf{x} \in K \wedge \mathbf{x}[X] = x \,\}$. We write $\pi_X K$ to denote the set of X-values in K, that is, $\pi_X K = \{\, \mathbf{x}[X] \mid \mathbf{x} \in K \,\}$.

Query Language. We express queries and view definitions in the language of functional aggregate queries (FAQ) [1]. Compared to the original FAQ definition that uses several commutative semirings, we define our queries using the single commutative ring $(\mathbb{Z}, +, \cdot, 0, 1)$ of integers with the usual addition and multiplication. A query Q has one of the two forms:

1. Given a set $\{X_i\}_{i \in [n]}$ of variables and an index set $S \subseteq [n]$, let $\mathbf{X}_S$ denote a tuple $(X_i)_{i \in S}$ of variables and $\mathbf{x}_S$ denote a tuple of data values over the schema $\mathbf{X}_S$. Then,

$$Q(\mathbf{x}_{[f]}) = \sum_{x_{f+1} \in \mathsf{Dom}(X_{f+1})} \cdots \sum_{x_n \in \mathsf{Dom}(X_n)} \prod_{S \in \mathcal{M}} K_S(\mathbf{x}_S), \text{ where:}$$

- $\mathcal{M}$ is a multiset of index sets.
- For every index set $S \in \mathcal{M}$, $K_S : \mathsf{Dom}(\mathbf{X}_S) \to \mathbb{Z}$ is a relation over the schema $\mathbf{X}_S$.
- $\mathbf{X}_{[f]}$ is the tuple of free variables of Q. The variables $X_{f+1}, \ldots, X_n$ are called bound.

2. $Q(\mathbf{x}) = Q_1(\mathbf{x}) + Q_2(\mathbf{x})$, where Q_1 and Q_2 are queries over the same tuple of free variables.

In the following, we use $\sum_{x_i}$ as a shorthand for $\sum_{x_i \in \mathsf{Dom}(X_i)}$.

Updates and Delta Queries. An update δK to a relation K is a relation over the schema of K. A single-tuple update, written as $\delta K = \{\mathbf{x} \mapsto m\}$, maps the tuple $\mathbf{x}$ to the nonzero multiplicity $m \in \mathbb{Z}$ and any other tuple to 0; that is, $|\delta K| = 1$. The data model and query language make no distinction between inserts and deletes – these are updates represented as relations in which tuples have positive and negative multiplicities.

Given a query Q and an update δK, the delta query δQ defines the change in the query result after applying δK to the database. The rules for deriving delta queries follow from the associativity, commutativity, and distributivity of the ring operations.

Query $Q(\mathbf{x})$	Delta query $\delta Q(\mathbf{x})$
$Q_1(\mathbf{x}_1) \cdot Q_2(\mathbf{x}_2)$	$\delta Q_1(\mathbf{x}_1) \cdot Q_2(\mathbf{x}_2) + Q_1(\mathbf{x}_1) \cdot \delta Q_2(\mathbf{x}_2) + \delta Q_1(\mathbf{x}_1) \cdot \delta Q_2(\mathbf{x}_2)$
$\sum_x Q_1(\mathbf{x}_1)$	$\sum_x \delta Q_1(\mathbf{x}_1)$
$Q_1(\mathbf{x}) + Q_2(\mathbf{x})$	$\delta Q_1(\mathbf{x}) + \delta Q_2(\mathbf{x})$
$K'(\mathbf{x})$	$\delta K(\mathbf{x})$ when $K = K'$ and 0 otherwise

Computation Time. Our maintenance algorithm takes as input the triangle count query Q and a database $\mathbf{D}$ and maintains the result of Q under a sequence of single-tuple updates. We distinguish the following computation times: (1) *preprocessing time* is spent on initializing the algorithm using $\mathbf{D}$ before any update is received, (2) *update time* is spent on processing one single-tuple update, and (3) *answer time* is spent on obtaining the result of Q. We consider two types of bounds on the update time: *worst-case bounds*, which limit the time each individual update takes in the worst case, and *amortized worst-case bounds*, which limit the average worst-case time taken by a sequence of updates. Enumerating a set of tuples with constant delay means that the time until reporting the first tuple, the time between reporting two consecutive tuples, and the time between reporting the last tuple and the end of enumeration is constant. When referring to sublinear time, we mean $\mathcal{O}(|\mathbf{D}|^{1-\gamma})$ for some $\gamma > 0$, where $|\mathbf{D}|$ is the database size.

Computational Model. We consider the RAM model of computation. Each relation (view) K over schema $\mathbf{X}$ is implemented by a data structure that stores key-value entries $(\mathbf{x}, K(\mathbf{x}))$ for each tuple $\mathbf{x}$ over $\mathbf{X}$ with $K(\mathbf{x}) \neq 0$ and needs space linear in the number of such tuples. We assume that this data structure supports (1) looking up, inserting, and deleting entries in constant time, (2) enumerating all stored entries in K with constant delay, and (3) returning $|K|$ in constant time. For instance, a hash table with chaining, where entries are doubly linked for efficient enumeration, can support these operations in constant time on average, under the assumption of simple uniform hashing.

For each variable X in the schema $\mathbf{X}$ of relation K, we further assume there is an index structure on X that allows: (4) enumerating all entries in K matching $\sigma_{X=x}K$ with constant delay, (5) checking $x \in \pi_X K$ in constant time, and (6) returning $|\sigma_{X=x}K|$ in constant time, for any $x \in \mathsf{Dom}(X)$, and (7) inserting and deleting index entries in constant time. Such an index structure can be realized, for instance, as a hash table with chaining where each key-value entry stores an X-value x and a doubly-linked list of pointers to the entries in K having the X-value x. Looking up an index entry given x takes constant time on average, and its doubly-linked list enables enumeration of the matching entries in K with constant delay. Inserting an index entry into the hash table additionally prepends a new pointer to

the doubly-linked list for a given x; overall, this operation takes constant time on average. For efficient deletion of index entries, each entry in K also stores back-pointers to its index entries (as many back-pointers as there are index structures for K). When an entry is deleted from K, locating and deleting its index entries takes constant time per index.

Data Partitioning. We partition each input relation into two parts based on the degrees of its values. Similar to common techniques used in databases to deal with data skew, our IVM approach employs different maintenance strategies for values of high and low frequency.

▶ **Definition 6** (Relation Partition). *Given a relation K over schema $\mathbf{X}$, a variable X from the schema $\mathbf{X}$, and a threshold θ, a partition of K on X with threshold θ is a set $\{K_h, K_l\}$ satisfying the following conditions:*

$$\begin{aligned}
&\textit{(union)} && K(\mathbf{x}) = K_h(\mathbf{x}) + K_l(\mathbf{x}) \textit{ for } \mathbf{x} \in \mathsf{Dom}(\mathbf{X}) \\
&\textit{(domain partition)} && (\pi_X K_h) \cap (\pi_X K_l) = \emptyset \\
&\textit{(heavy part)} && \textit{for all } x \in \pi_X K_h : \ |\sigma_{X=x} K_h| \geq \tfrac{1}{2}\theta \\
&\textit{(light part)} && \textit{for all } x \in \pi_X K_l : \ |\sigma_{X=x} K_l| < \tfrac{3}{2}\theta
\end{aligned}$$

The set $\{K_h, K_l\}$ is called a strict partition of K on X with threshold θ if it satisfies the union and domain partition conditions and the following strict versions of the heavy part and light part conditions:

$$\begin{aligned}
&\textit{(strict heavy part)} && \textit{for all } x \in \pi_X K_h : \ |\sigma_{X=x} K_h| \geq \theta \\
&\textit{(strict light part)} && \textit{for all } x \in \pi_X K_l : \ |\sigma_{X=x} K_l| < \theta
\end{aligned}$$

The relations K_h and K_l are called the heavy and light parts of K.

Definition 6 admits multiple ways to (non-strictly) partition a relation K on variable X with threshold θ. For instance, assume that $|\sigma_{X=x} K| = \theta$ for some X-value x in K. Then, all tuples in K with X-value x can be in either the heavy or light part of K; but they cannot be in both parts because of the domain partition condition. If the partition is strict, then all such tuples are in the heavy part of K.

The strict partition of a relation K is unique for a given threshold and can be computed in time linear in the size of K.

3 IVM$^\epsilon$: Adaptive Maintenance of the Triangle Count

We present IVM$^\epsilon$, our algorithm for the incremental maintenance of the result of Query (1). We start with a high-level overview. Consider a database $\mathbf{D}$ consisting of three relations R, S, and T with schemas (A, B), (B, C), and (C, A), respectively. We partition R, S, and T on variables A, B, and C, respectively, for a given threshold. We then decompose Query (1) into eight skew-aware views expressed over these relation parts:

$$Q_{rst}() = \sum_{a,b,c} R_r(a, b) \cdot S_s(b, c) \cdot T_t(c, a), \qquad \text{for } r, s, t \in \{h, l\}.$$

Query (1) is then the sum of these skew-aware views: $Q() = \sum_{r,s,t \in \{h,l\}} Q_{rst}()$.

IVM$^\epsilon$ adapts its maintenance strategy to each skew-aware view Q_{rst} to ensure amortized update time that is sublinear in the database size. While most of these views admit sublinear delta computation over the relation parts, few exceptions require linear-time maintenance. For these exceptions, IVM$^\epsilon$ precomputes the update-independent parts of the delta queries as *auxiliary materialized views* and then exploits these views to speed up the delta evaluation.

Materialized View Definition	Space Complexity
$Q() = \sum_{r,s,t \in \{h,l\}} \sum_{a,b,c} R_r(a,b) \cdot S_s(b,c) \cdot T_t(c,a)$	$\mathcal{O}(1)$
$V_{RS}(a,c) = \sum_b R_h(a,b) \cdot S_l(b,c)$	$\mathcal{O}(\lvert\mathbf{D}\rvert^{1+\min\{\epsilon,1-\epsilon\}})$
$V_{ST}(b,a) = \sum_c S_h(b,c) \cdot T_l(c,a)$	$\mathcal{O}(\lvert\mathbf{D}\rvert^{1+\min\{\epsilon,1-\epsilon\}})$
$V_{TR}(c,b) = \sum_a T_h(c,a) \cdot R_l(a,b)$	$\mathcal{O}(\lvert\mathbf{D}\rvert^{1+\min\{\epsilon,1-\epsilon\}})$

Figure 2 The definition and space complexity of the materialized views in $\mathbf{V} = \{Q, V_{RS}, V_{ST}, V_{TR}\}$ as part of an IVM$^\epsilon$ state of a database $\mathbf{D}$ partitioned for $\epsilon \in [0,1]$.

One such exception is the view Q_{hhl}. Consider a single-tuple update $\delta R_h = \{(\alpha,\beta) \mapsto m\}$ to the heavy part R_h of relation R, where α and β are fixed data values. Computing the delta view $\delta Q_{hhl}() = \delta R_h(\alpha,\beta) \cdot \sum_c S_h(\beta,c) \cdot T_l(c,\alpha)$ requires iterating over all the C-values c paired with β in S_h and with α in T_l; the number of such C-values can be linear in the size of the database. To avoid this iteration, IVM$^\epsilon$ precomputes the view $V_{ST}(b,a) = \sum_c S_h(b,c) \cdot T_l(c,a)$ and uses this view to evaluate $\delta Q_{hhl}() = \delta R_h(\alpha,\beta) \cdot V_{ST}(\beta,\alpha)$ in constant time.

Such auxiliary views, however, also require maintenance. All such views created by IVM$^\epsilon$ can be maintained in sublinear time under single-tuple updates to the input relations. Figure 2 summarizes these views used by IVM$^\epsilon$ to maintain Query (1): V_{RS}, V_{ST} and V_{TR}. They serve to avoid linear-time delta computation for updates to T, R, and S, respectively. IVM$^\epsilon$ also materializes the result of Query (1), which ensures constant answer time.

We now describe our strategy in detail. We start by defining the state that IVM$^\epsilon$ initially creates and maintains upon each update. Then, we specify the procedure for processing a single-tuple update to any input relation, followed by the space complexity analysis of IVM$^\epsilon$. Section 4 gives the procedure for processing a sequence of such updates.

▶ **Definition 7** (IVM$^\epsilon$ State). *Given a database $\mathbf{D} = \{R,S,T\}$ and $\epsilon \in [0,1]$, an IVM$^\epsilon$ state of $\mathbf{D}$ is a tuple $\mathcal{Z} = (\epsilon, N, \mathbf{P}, \mathbf{V})$, where:*

- *N is a natural number such that the size invariant $\lfloor \frac{1}{4}N \rfloor \leq |\mathbf{D}| < N$ holds. N is called the threshold base.*
- *$\mathbf{P} = \{R_h, R_l, S_h, S_l, T_h, T_l\}$ consists of the partitions of R, S, and T on variables A, B, and C, respectively, with threshold $\theta = N^\epsilon$.*
- *$\mathbf{V}$ is the set of materialized views $\{Q, V_{RS}, V_{ST}, V_{TR}\}$ as defined in Figure 2.*

The initial state $\mathcal{Z}$ of $\mathbf{D}$ has $N = 2 \cdot |\mathbf{D}| + 1$ and the three partitions in $\mathbf{P}$ are strict.

By construction, $|\mathbf{P}| = |\mathbf{D}|$. The size invariant implies $|\mathbf{D}| = \Theta(N)$ and, together with the heavy and light part conditions, facilitates the amortized analysis of IVM$^\epsilon$ in Section 4. Definition 6 provides two essential upper bounds for each relation partition in an IVM$^\epsilon$ state: The number of distinct A-values in R_h is at most $\frac{N}{\frac{1}{2}N^\epsilon} = 2N^{1-\epsilon}$, i.e., $|\pi_A R_h| \leq 2N^{1-\epsilon}$, and the number of tuples in R_l with an A-value a is less than $\frac{3}{2}N^\epsilon$, i.e., $|\sigma_{A=a} R_l| < \frac{3}{2}N^\epsilon$, for any $a \in \mathsf{Dom}(A)$. The same bounds hold for B-values in $\{S_h, S_l\}$ and C-values in $\{T_h, T_l\}$.

3.1 Preprocessing Stage

The preprocessing stage constructs the initial IVM$^\epsilon$ state given a database $\mathbf{D}$ and $\epsilon \in [0,1]$.

▶ **Proposition 8.** *Given a database $\mathbf{D}$ and $\epsilon \in [0,1]$, constructing the initial IVM$^\epsilon$ state of $\mathbf{D}$ takes $\mathcal{O}(|\mathbf{D}|^{\frac{3}{2}})$ time.*

Proof. We analyze the time to construct the initial state $\mathcal{Z} = (\epsilon, N, \mathbf{P}, \mathbf{V})$ of $\mathbf{D}$. Retrieving the size $|\mathbf{D}|$ and computing $N = 2 \cdot |\mathbf{D}| + 1$ take constant time. Strictly partitioning the input relations from $\mathbf{D}$ using the threshold N^ϵ, as described in Definition 6, takes $\mathcal{O}(|\mathbf{D}|)$ time. Computing the result of the triangle count query on $\mathbf{D}$ (or $\mathbf{P}$) using a worst-case optimal join algorithm [18] takes $\mathcal{O}(|\mathbf{D}|^{\frac{3}{2}})$ time. Computing the auxiliary views V_{RS}, V_{ST}, and V_{TR} takes $\mathcal{O}(|\mathbf{D}|^{1+\min\{\epsilon,1-\epsilon\}})$ time, as shown next. Consider the view $V_{RS}(a,c) = \sum_b R_h(a,b) \cdot S_l(b,c)$. To compute V_{RS}, one can iterate over all (a,b) pairs in R_h and then find the C-values in S_l for each b. The light part S_l contains at most N^ϵ distinct C-values for any B-value, which gives an upper bound of $|R_h| \cdot N^\epsilon$ on the size of V_{RS}. Alternatively, one can iterate over all (b,c) pairs in S_l and then find the A-values in R_h for each b. The heavy part R_h contains at most $N^{1-\epsilon}$ distinct A-values, which gives an upper bound of $|S_l| \cdot N^{1-\epsilon}$ on the size of V_{RS}. The number of steps needed to compute this result is upper-bounded by $\min\{\, |R_h| \cdot N^\epsilon,\, |S_l| \cdot N^{1-\epsilon} \,\} < \min\{\, N \cdot N^\epsilon,\, N \cdot N^{1-\epsilon} \,\} = N^{1+\min\{\epsilon,1-\epsilon\}}$. From $|\mathbf{D}| = \Theta(N)$ follows that computing V_{RS} on the database partition $\mathbf{P}$ takes $\mathcal{O}(|\mathbf{D}|^{1+\min\{\epsilon,1-\epsilon\}})$ time; the analysis for V_{ST} and V_{TR} is analogous. Note that $\max_{\epsilon\in[0,1]}\{1+\min\{\epsilon,1-\epsilon\}\} = \frac{3}{2}$. Overall, the initial state $\mathcal{Z}$ of $\mathbf{D}$ can be constructed in $\mathcal{O}(|\mathbf{D}|^{\frac{3}{2}})$ time. ◀

The preprocessing stage of IVM$^\epsilon$ happens *before* any update is received. In case we start from an empty database, the preprocessing cost of IVM$^\epsilon$ is $\mathcal{O}(1)$.

3.2 Processing a Single-Tuple Update

We describe the IVM$^\epsilon$ strategy for maintaining the result of Query (1) under a single-tuple update to the relation R. This update can affect either the heavy or light part of R, hence we write δR_r, where r stands for h or l. We assume that checking whether the update affects the heavy or light part of R takes constant time. The update is represented as a relation $\delta R_r = \{\, (\alpha,\beta) \mapsto m \,\}$, where α and β are data values and $m \in \mathbb{Z}$. Due to the symmetry of the triangle query and auxiliary views, updates to S and T are handled similarly.

Figure 3 shows the procedure ApplyUpdate that takes as input a current IVM$^\epsilon$ state $\mathcal{Z}$ and the update δR_r, and returns a new state that results from applying δR_r to $\mathcal{Z}$. The procedure computes the deltas of the skew-aware views referencing R_r, which are δQ_{rhh} (Line 3), δQ_{rhl} (Line 4), δQ_{rlh} (Line 5), and δQ_{rll} (Line 6), and uses these deltas to maintain the triangle count (Line 7). These skew-aware views are not materialized, but their deltas facilitate the maintenance of the triangle count. If the update affects the heavy part R_h of R, the procedure maintains V_{RS} (Line 9) and R_h (Line 12); otherwise, it maintains V_{TR} (Line 11) and R_l (Line 12). The view V_{ST} remains unchanged as it has no reference to R_h or R_l.

Figure 3 also gives the time complexity of computing these deltas and applying them to $\mathcal{Z}$. This complexity is either constant or dependent on the number of C-values for which matching tuples in the parts of S and T have nonzero multiplicities.

▶ **Proposition 9.** *Given a state $\mathcal{Z}$ constructed from a database $\mathbf{D}$ for $\epsilon \in [0,1]$, IVM$^\epsilon$ maintains $\mathcal{Z}$ under a single-tuple update to any input relation in $\mathcal{O}(|\mathbf{D}|^{\max\{\epsilon,1-\epsilon\}})$ time.*

Proof. We analyze the running time of the procedure from Figure 3 given a single-tuple update $\delta R_r = \{(\alpha,\beta) \mapsto m\}$ and a state $\mathcal{Z} = (\epsilon, N, \mathbf{P}, \mathbf{V})$ of $\mathbf{D}$. Since the query and auxiliary views are symmetric, the analysis for updates to S and T is similar.

We first analyze the evaluation strategies for the deltas of the skew-aware views Q_{rst}:

- (Line 3) Computing δQ_{rhh} requires summing over C-values (α and β are fixed). The minimum degree of each C-value in T_h is $\frac{1}{2}N^\epsilon$, which means the number of distinct C-values in T_h is at most $\frac{N}{\frac{1}{2}N^\epsilon} = 2N^{1-\epsilon}$. Thus, this delta evaluation takes $\mathcal{O}(N^{1-\epsilon})$ time.

ApplyUpdate($\delta R_r, \mathcal{Z}$)		Time
1	**let** $\delta R_r = \{(\alpha, \beta) \mapsto m\}$	
2	**let** $\mathcal{Z} = (\epsilon, N, \{R_h, R_l, S_h, S_l, T_h, T_l\}, \{Q, V_{RS}, V_{ST}, V_{TR}\})$	
3	$\delta Q_{rhh}() = \delta R_r(\alpha, \beta) \cdot \sum_c S_h(\beta, c) \cdot T_h(c, \alpha)$	$\mathcal{O}(\lvert\mathbf{D}\rvert^{1-\epsilon})$
4	$\delta Q_{rhl}() = \delta R_r(\alpha, \beta) \cdot V_{ST}(\beta, \alpha)$	$\mathcal{O}(1)$
5	$\delta Q_{rlh}() = \delta R_r(\alpha, \beta) \cdot \sum_c S_l(\beta, c) \cdot T_h(c, \alpha)$	$\mathcal{O}(\lvert\mathbf{D}\rvert^{\min\{\epsilon, 1-\epsilon\}})$
6	$\delta Q_{rll}() = \delta R_r(\alpha, \beta) \cdot \sum_c S_l(\beta, c) \cdot T_l(c, \alpha)$	$\mathcal{O}(\lvert\mathbf{D}\rvert^{\epsilon})$
7	$Q() = Q() + \delta Q_{rhh}() + \delta Q_{rhl}() + \delta Q_{rlh}() + \delta Q_{rll}()$	$\mathcal{O}(1)$
8	**if** (r is h)	
9	$\quad V_{RS}(\alpha, c) = V_{RS}(\alpha, c) + \delta R_h(\alpha, \beta) \cdot S_l(\beta, c)$	$\mathcal{O}(\lvert\mathbf{D}\rvert^{\epsilon})$
10	**else**	
11	$\quad V_{TR}(c, \beta) = V_{TR}(c, \beta) + T_h(c, \alpha) \cdot \delta R_l(\alpha, \beta)$	$\mathcal{O}(\lvert\mathbf{D}\rvert^{1-\epsilon})$
12	$R_r(\alpha, \beta) = R_r(\alpha, \beta) + \delta R_r(\alpha, \beta)$	$\mathcal{O}(1)$
13	**return** $\mathcal{Z}$	
	Total update time:	$\mathcal{O}(\lvert\mathbf{D}\rvert^{\max\{\epsilon, 1-\epsilon\}})$

Figure 3 (left) Counting triangles under a single-tuple update. ApplyUpdate takes as input an update δR_r to the heavy or light part of R, hence $r \in \{h, l\}$, and the current IVM$^\epsilon$ state $\mathcal{Z}$ of a database $\mathbf{D}$ partitioned using $\epsilon \in [0, 1]$. It returns a new state that results from applying δR_r to $\mathcal{Z}$. Lines 3-6 compute the deltas of the affected skew-aware views, and Line 7 maintains Q. Lines 9 and 11 maintain the auxiliary views V_{RS} and V_{TR}, respectively. Line 12 maintains the affected part R_r. (right) The time complexity of computing and applying deltas. The evaluation strategy for computing δQ_{rlh} in Line 5 may choose either S_l or T_h to bound C-values, depending on ϵ. The total time is the maximum of all individual times. The maintenance procedures for S and T are similar.

- (Line 4) Computing δQ_{rhl} requires constant-time lookups in δR_r and V_{ST}.
- (Line 5) Computing δQ_{rlh} can be done in two ways, depending on ϵ: either sum over at most $2N^{1-\epsilon}$ C-values in T_h for the given α or sum over at most $\frac{3}{2}N^\epsilon$ C-values in S_l for the given β. This delta computation takes at most $\min\{2N^{1-\epsilon}, \frac{3}{2}N^\epsilon\}$ constant-time operations, thus $\mathcal{O}(N^{\min\{\epsilon, 1-\epsilon\}})$ time.
- (Line 6) Computing δQ_{rll} requires summing over at most $\frac{3}{2}N^\epsilon$ C-values in S_l for the given β. This delta computation takes $\mathcal{O}(N^\epsilon)$ time.

Maintaining the result of Query (1) using these deltas takes constant time (Line 7). The views V_{RS} and V_{TR} are maintained for updates to distinct parts of R. Maintaining V_{RS} requires iterating over at most $\frac{3}{2}N^\epsilon$ C-values in S_l for the given β (Line 9); similarly, maintaining V_{TR} requires iterating over at most $2N^{1-\epsilon}$ C-values in T_h for the given α (Line 11). Finally, maintaining the (heavy or light) part of R affected by δR_r takes constant time (Line 12). The total update time is $\mathcal{O}(\max\{1, N^\epsilon, N^{1-\epsilon}, N^{\min\{\epsilon, 1-\epsilon\}}\}) = \mathcal{O}(N^{\max\{\epsilon, 1-\epsilon\}})$. From the invariant $|\mathbf{D}| = \Theta(N)$ follows the claimed time complexity $\mathcal{O}(|\mathbf{D}|^{\max\{\epsilon, 1-\epsilon\}})$. ◀

3.3 Space Complexity

We next analyze the space complexity of the IVM$^\epsilon$ maintenance strategy.

▶ **Proposition 10.** *Given a database* $\mathbf{D}$ *and* $\epsilon \in [0, 1]$*, the IVM*$^\epsilon$ *state constructed from* $\mathbf{D}$ *to support the maintenance of the result of Query* (1) *takes* $\mathcal{O}(|\mathbf{D}|^{1+\min\{\epsilon, 1-\epsilon\}})$ *space.*

Proof. We consider a state $\mathcal{Z} = (\epsilon, N, \mathbf{P}, \mathbf{V})$ of database $\mathbf{D}$. N and ϵ take constant space and $|\mathbf{P}| = |\mathbf{D}|$. Figure 2 summarizes the space complexity of the materialized views Q, V_{RS}, V_{ST}, and V_{TR} from $\mathbf{V}$. The result of Q takes constant space. As discussed in the proof of Proposition 8, to compute the auxiliary view $V_{RS}(a,c) = \sum_b R_h(a,b) \cdot S_l(b,c)$, we can use either R_h or S_l as the outer relation:

$$|V_{RS}| \leq \min\{\, |R_h| \cdot \max_{b \in \pi_B S_l} |\sigma_{B=b} S_l|,\ |S_l| \cdot \max_{b \in \pi_B R_h} |\sigma_{B=b} R_h| \,\} < \min\{\, N \cdot \frac{3}{2} N^\epsilon,\ N \cdot 2N^{1-\epsilon} \,\}$$

The size of V_{RS} is thus $\mathcal{O}(N^{1+\min\{\epsilon,1-\epsilon\}})$. From $|\mathbf{D}| = \Theta(N)$ follows that V_{RS} takes $\mathcal{O}(|\mathbf{D}|^{1+\min\{\epsilon,1-\epsilon\}})$ space; the space analysis for V_{ST} and V_{TR} is analogous. Overall, the state $\mathcal{Z}$ of $\mathbf{D}$ takes $\mathcal{O}(|\mathbf{D}|^{1+\min\{\epsilon,1-\epsilon\}})$ space. ◀

4 Rebalancing Partitions

The partition of a relation may change after updates. For instance, an insert $\delta R_l = \{(\alpha, \beta) \mapsto 1\}$ may violate the size invariant $\lfloor \frac{1}{4} N \rfloor \leq |\mathbf{D}| < N$ or may violate the light part condition $|\sigma_{A=\alpha} R_l| < \frac{3}{2} N^\epsilon$ and require moving all tuples with the A-value α from R_l to R_h. As the database evolves under updates, IVM$^\epsilon$ performs *major* and *minor* rebalancing steps to ensure the size invariant and the conditions for heavy and light parts of each partition always hold. This rebalancing also ensures that the upper bounds on the number of data values, such as the number of B-values paired with α in R_l and the number of distinct A-values in R_h, are valid. The rebalancing cost is amortized over multiple updates.

Major Rebalancing

If an update causes the database size to fall below $\lfloor \frac{1}{4} N \rfloor$ or reach N, IVM$^\epsilon$ halves or, respectively, doubles N, followed by strictly repartitioning the database with the new threshold N^ϵ and recomputing the materialized views, as shown in Figure 4.

▶ **Proposition 11.** *Given $\epsilon \in [0,1]$, major rebalancing of an IVM$^\epsilon$ state constructed from a database $\mathbf{D}$ takes $\mathcal{O}(|\mathbf{D}|^{1+\min\{\epsilon,1-\epsilon\}})$ time.*

Proof. We consider the major rebalancing procedure from Figure 4. Strictly partitioning the input relations takes $\mathcal{O}(|\mathbf{D}|)$ time. From the proof of Proposition 8 and $|\mathbf{D}| = \Theta(N)$ follow that recomputing V_{RS}, V_{ST}, and V_{TR} takes $\mathcal{O}(|\mathbf{D}|^{1+\min\{\epsilon,1-\epsilon\}})$ time. ◀

The (super)linear time of major rebalancing is amortized over $\Omega(N)$ updates. After a major rebalancing step, it holds that $|\mathbf{D}| = \frac{1}{2} N$ (after doubling), or $|\mathbf{D}| = \frac{1}{2} N - \frac{1}{2}$ or $|\mathbf{D}| = \frac{1}{2} N - 1$ (after halving, i.e., setting N to $\lfloor \frac{1}{2} N \rfloor - 1$; the two options are due to the floor functions in the size invariant and halving expression). To violate the size invariant $\lfloor \frac{1}{4} N \rfloor \leq |\mathbf{D}| < N$ and trigger another major rebalancing, the number of required updates is at least $\frac{1}{4} N$. Section 4.1 proves the amortized $\mathcal{O}(|\mathbf{D}|^{\min\{\epsilon,1-\epsilon\}})$ time of major rebalancing.

Minor Rebalancing

After each update $\delta R = \{(\alpha, \beta) \mapsto m\}$, IVM$^\epsilon$ checks whether the two conditions $|\sigma_{A=\alpha} R_h| \geq \frac{1}{2} N^\epsilon$ and $|\sigma_{A=\alpha} R_l| < \frac{3}{2} N^\epsilon$ still hold. If the first condition is violated, all tuples in R_h with the A-value α are moved to R_l and the affected views are updated; similarly, if the second condition is violated, all tuples with the A-value α are moved from R_l to R_h, followed by updating the affected views. Figure 4 shows the procedure for minor rebalancing, which deletes affected tuples from one part and inserts them into the other part.

OnUpdate($\delta R, \mathcal{Z}$)

let $\delta R = \{(\alpha, \beta) \mapsto m\}$
let $\mathcal{Z} = (\epsilon, N, \{R_h, R_l\} \cup \mathbf{P}, \mathbf{V})$
if ($\alpha \in \pi_A R_h$ **or** $\epsilon = 0$)
  $\mathcal{Z} =$ ApplyUpdate($\delta R_h = \{(\alpha, \beta) \mapsto m\}, \mathcal{Z}$)
else
  $\mathcal{Z} =$ ApplyUpdate($\delta R_l = \{(\alpha, \beta) \mapsto m\}, \mathcal{Z}$)
if ($|\mathbf{D}| = N$)
  $N = 2N$
  $\mathcal{Z} =$ MajorRebalancing($\mathcal{Z}$)
else if ($|\mathbf{D}| < \lfloor \frac{1}{4} N \rfloor$)
  $N = \lfloor \frac{1}{2} N \rfloor - 1$
  $\mathcal{Z} =$ MajorRebalancing($\mathcal{Z}$)
else if ($\alpha \in \pi_A R_l$ **and** $|\sigma_{A=\alpha} R_l| \geq \frac{3}{2} N^\epsilon$)
  $\mathcal{Z} =$ MinorRebalancing($R_l, R_h, A, \alpha, \mathcal{Z}$)
else if ($\alpha \in \pi_A R_h$ **and** $|\sigma_{A=\alpha} R_h| < \frac{1}{2} N^\epsilon$)
  $\mathcal{Z} =$ MinorRebalancing($R_h, R_l, A, \alpha, \mathcal{Z}$)
return $\mathcal{Z}$

MajorRebalancing($\mathcal{Z}$)

let $\mathcal{Z} = (\epsilon, N, \{R_h, R_l, S_h, S_l, T_h, T_l\},$
  $\{Q, V_{RS}, V_{ST}, V_{TR}\})$
$\{R_h, R_l\} =$ StrictPartition(R_h, R_l, A, N^ϵ)
$\{S_h, S_l\} =$ StrictPartition(S_h, S_l, B, N^ϵ)
$\{T_h, T_l\} =$ StrictPartition(T_h, T_l, C, N^ϵ)
$V_{RS}(a, c) = \sum_b R_h(a, b) \cdot S_l(b, c)$
$V_{ST}(b, a) = \sum_c S_h(b, c) \cdot T_l(c, a)$
$V_{TR}(c, b) = \sum_a T_h(c, a) \cdot R_l(a, b)$
return $\mathcal{Z}$

MinorRebalancing($K_{src}, K_{dst}, X, x, \mathcal{Z}$)

foreach $\mathbf{t} \in \sigma_{X=x} K_{src}$ **do**
  $m = K_{src}(\mathbf{t})$
  $\mathcal{Z} =$ ApplyUpdate($\delta K_{src} = \{\, \mathbf{t} \mapsto -m \,\}, \mathcal{Z}$)
  $\mathcal{Z} =$ ApplyUpdate($\delta K_{dst} = \{\, \mathbf{t} \mapsto m \,\}, \mathcal{Z}$)
return $\mathcal{Z}$

Figure 4 Counting triangles under a single-tuple update with rebalancing. OnUpdate takes as input an update δR and the current IVM$^\epsilon$ state $\mathcal{Z}$ of a database $\mathbf{D}$. It returns a new state that results from applying δR to $\mathcal{Z}$ and, if necessary, rebalancing partitions. The condition $\epsilon = 0$ in the third line ensures that all tuples are in R_h when $\epsilon = 0$. ApplyUpdate is given in Figure 3. MinorRebalancing($K_{src}, K_{dst}, X, x, \mathcal{Z}$) moves all tuples with the X-value x from K_{src} to K_{dst}. MajorRebalancing($\mathcal{Z}$) recomputes the relation partitions and views in $\mathcal{Z}$. StrictPartition(K_h, K_l, X, θ) constructs a strict partition of relation K on variable X with threshold θ (see Definition 6). The OnUpdate procedures for updates to relations S and T are analogous.

▶ **Proposition 12.** *Given $\epsilon \in [0, 1]$, minor rebalancing of an IVM$^\epsilon$ state constructed from a database $\mathbf{D}$ takes $\mathcal{O}(|\mathbf{D}|^{\epsilon + \max\{\epsilon, 1-\epsilon\}})$ time.*

Proof. Consider a state $\mathcal{Z} = (\epsilon, N, \mathbf{P}, \mathbf{V})$. Minor rebalancing moves fewer than $\frac{1}{2} N^\epsilon$ tuples (from heavy to light) or fewer than $\frac{3}{2} N^\epsilon + 1$ tuples (from light to heavy). Each tuple move performs one delete and one insert and costs $\mathcal{O}(|\mathbf{D}|^{\max\{\epsilon, 1-\epsilon\}})$ by Proposition 9. Since there are $\mathcal{O}(N^\epsilon)$ such operations and $|\mathbf{D}| = \Theta(N)$, the total time is $\mathcal{O}(|\mathbf{D}|^{\epsilon + \max\{\epsilon, 1-\epsilon\}})$. ◀

The (super)linear time of minor rebalancing is amortized over $\Omega(N^\epsilon)$ updates. This lower bound on the number of updates comes from the heavy and light part conditions (cf. Definition 6), namely from the gap between the two thresholds in these conditions. Section 4.1 proves the amortized $\mathcal{O}(|\mathbf{D}|^{\max\{\epsilon, 1-\epsilon\}})$ time of minor rebalancing.

Figure 4 gives the trigger procedure OnUpdate that maintains Query (1) under a single-tuple update to relation R and, if necessary, rebalances partitions; the procedures for updates to S and T are analogous. Given an update $\delta R = \{(\alpha, \beta) \mapsto m\}$ and an IVM$^\epsilon$ state of a database $\mathbf{D}$, the procedure first checks in constant time whether the update affects the heavy or light part of R. The update targets R_h if there exists a tuple with the same A-value α already in R_h, or ϵ is set to 0; otherwise, the update targets R_l. When $\epsilon = 0$, all tuples are in R_h, while R_l remains empty. Although this behavior is not required by IVM$^\epsilon$ (without

the $\epsilon = 0$ condition, R_l would contain only tuples whose A-values have the degree of 1, and R_h would contain all other tuples), it allows us to recover existing IVM approaches, such as classical IVM and factorized IVM, which do not partition relations; by setting ϵ to 0 or 1, IVM$^\epsilon$ ensures that all tuples are in R_h or respectively R_l. The procedure OnUpdate then invokes ApplyUpdate from Figure 3. If the update causes a violation of the size invariant $\lfloor \frac{1}{4}N \rfloor \leq |\mathbf{D}| < N$, the procedure invokes MajorRebalancing to recompute the relation partitions and auxiliary views (note that major rebalancing has no effect on the triangle count). Otherwise, if the heavy or light part condition is violated, MinorRebalancing moves all tuples with the given A-value α from the source part to the destination part of R.

4.1 Proof of Theorem 3

We are now ready to prove Theorem 3 that states the complexity of IVM$^\epsilon$.

Proof. The preprocessing stage constructs the initial IVM$^\epsilon$ state from a database $\mathbf{D}$ in $\mathcal{O}(|\mathbf{D}|^{\frac{3}{2}})$ time, as shown in Proposition 8. Materializing the query result ensures constant answer time. The space complexity $\mathcal{O}(|\mathbf{D}|^{1+\min\{\epsilon,1-\epsilon\}})$ follows from Proposition 10.

We next analyze the amortized update time complexity. Let $\mathcal{Z}_0 = (\epsilon, N_0, \mathbf{P}_0, \mathbf{V}_0)$ be the initial IVM$^\epsilon$ state of a database $\mathbf{D}_0$ and $u_0, u_1, \ldots, u_{n-1}$ a sequence of arbitrary single-tuple updates. The application of this update sequence to $\mathcal{Z}_0$ yields a sequence $\mathcal{Z}_0 \xrightarrow{u_0} \mathcal{Z}_1 \xrightarrow{u_1} \ldots \xrightarrow{u_{n-1}} \mathcal{Z}_n$ of IVM$^\epsilon$ states, where $\mathcal{Z}_{i+1}$ is the result of executing the procedure OnUpdate$(u_i, \mathcal{Z}_i)$ from Figure 4, for $0 \leq i < n$. Let c_i denote the actual execution cost of OnUpdate$(u_i, \mathcal{Z}_i)$. For some $\Gamma > 0$, we can decompose each c_i as:

$$c_i = c_i^{apply} + c_i^{major} + c_i^{minor} + \Gamma, \qquad \text{for } 0 \leq i < n,$$

where c_i^{apply}, c_i^{major}, and c_i^{minor} are the actual costs of the subprocedures ApplyUpdate, MajorRebalancing, and MinorRebalancing, respectively, in OnUpdate. If update u_i causes no major rebalancing, then $c_i^{major} = 0$; similarly, if u_i causes no minor rebalancing, then $c_i^{minor} = 0$. These actual costs admit the following worst-case upper bounds:

$$\begin{aligned}
c_i^{apply} &\leq \gamma N_i^{\max\{\epsilon,1-\epsilon\}} && \text{(by Proposition 9),}\\
c_i^{major} &\leq \gamma N_i^{1+\min\{\epsilon,1-\epsilon\}} && \text{(by Proposition 11), and}\\
c_i^{minor} &\leq \gamma N_i^{\epsilon+\max\{\epsilon,1-\epsilon\}} && \text{(by Proposition 12),}
\end{aligned}$$

where γ is a constant derived from their asymptotic bounds, and N_i is the threshold base of $\mathcal{Z}_i$. The actual costs of major and minor rebalancing can be superlinear in the database size.

The crux of this proof is to show that assigning a *sublinear amortized cost* $\hat{c}_i$ to each update u_i accumulates enough budget to pay for such expensive but less frequent rebalancing procedures. For any sequence of n updates, our goal is to show that the accumulated amortized cost is no smaller than the accumulated actual cost:

$$\sum_{i=0}^{n-1} \hat{c}_i \geq \sum_{i=0}^{n-1} c_i. \tag{2}$$

The amortized cost assigned to an update u_i is $\hat{c}_i = \hat{c}_i^{apply} + \hat{c}_i^{major} + \hat{c}_i^{minor} + \Gamma$, where

$$\hat{c}_i^{apply} = \gamma N_i^{\max\{\epsilon,1-\epsilon\}}, \quad \hat{c}_i^{major} = 4\gamma N_i^{\min\{\epsilon,1-\epsilon\}}, \quad \hat{c}_i^{minor} = 2\gamma N_i^{\max\{\epsilon,1-\epsilon\}}, \quad \text{and}$$

Γ and γ are the constants used to upper bound the actual cost of OnUpdate. In contrast to the actual costs c_i^{major} and c_i^{minor}, the amortized costs $\hat{c}_i^{major}$ and $\hat{c}_i^{minor}$ are always nonzero.

We prove that such amortized costs satisfy Inequality (2). Since $\hat{c}_i^{apply} \geq c_i^{apply}$ for $0 \leq i < n$, it suffices to show that the following inequalities hold:

$$\textit{(amortizing major rebalancing)} \quad \sum_{i=0}^{n-1} \hat{c}_i^{major} \geq \sum_{i=0}^{n-1} c_i^{major} \qquad \text{and} \tag{3}$$

$$\textit{(amortizing minor rebalancing)} \quad \sum_{i=0}^{n-1} \hat{c}_i^{minor} \geq \sum_{i=0}^{n-1} c_i^{minor}. \tag{4}$$

We prove Inequalities (3) and (4) by induction on the length n of the update sequence.

Major rebalancing.

- *Base case*: We show that Inequality (3) holds for $n = 1$. The preprocessing stage sets $N_0 = 2 \cdot |\mathbf{D}_0| + 1$. If the initial database $\mathbf{D}_0$ is empty, then $N_0 = 1$ and u_0 triggers major rebalancing (and no minor rebalancing). The amortized cost $\hat{c}_0^{major} = 4\gamma N_0^{\min\{\epsilon,1-\epsilon\}} = 4\gamma$ suffices to cover the actual cost $c_0^{major} \leq \gamma N_0^{1+\min\{\epsilon,1-\epsilon\}} = \gamma$. If the initial database is nonempty, u_0 cannot trigger major rebalancing (i.e., violate the size invariant) because $\lfloor \frac{1}{4} N_0 \rfloor = \lfloor \frac{1}{2} |\mathbf{D}_0| \rfloor \leq |\mathbf{D}_0| - 1$ (lower threshold) and $|\mathbf{D}_0| + 1 < N_0 = 2 \cdot |\mathbf{D}_0| + 1$ (upper threshold); then, $\hat{c}_0^{major} \geq c_0^{major} = 0$. Thus, Inequality (3) holds for $n = 1$.
- *Inductive step*: Assumed that Inequality (3) holds for all update sequences of length up to $n-1$, we show it holds for update sequences of length n. If update u_{n-1} causes no major rebalancing, then $\hat{c}_{n-1}^{major} = 4\gamma N_{n-1}^{\min\{\epsilon,1-\epsilon\}} \geq 0$ and $c_{n-1}^{major} = 0$, thus Inequality (3) holds for n. Otherwise, if applying u_{n-1} violates the size invariant, the database size $|\mathbf{D}_n|$ is either $\lfloor \frac{1}{4} N_{n-1} \rfloor - 1$ or N_{n-1}. Let $\mathcal{Z}_j$ be the state created after the previous major rebalancing or, if there is no such step, the initial state. For the former ($j > 0$), the major rebalancing step ensures $|\mathbf{D}_j| = \frac{1}{2} N_j$ after doubling and $|\mathbf{D}_j| = \frac{1}{2} N_j - \frac{1}{2}$ or $|\mathbf{D}_j| = \frac{1}{2} N_j - 1$ after halving the threshold base N_j; for the latter ($j = 0$), the preprocessing stage ensures $|\mathbf{D}_j| = \frac{1}{2} N_j - \frac{1}{2}$. The threshold base N_j changes only with major rebalancing, thus $N_j = N_{j+1} = \ldots = N_{n-1}$. The number of updates needed to change the database size from $|\mathbf{D}_j|$ to $|\mathbf{D}_n|$ (i.e., between two major rebalancing) is at least $\frac{1}{4} N_{n-1}$ since $\min\{\frac{1}{2} N_j - 1 - (\lfloor \frac{1}{4} N_{n-1} \rfloor - 1), N_{n-1} - \frac{1}{2} N_j\} \geq \frac{1}{4} N_{n-1}$. Then,

$$\begin{aligned}
\sum_{i=0}^{n-1} \hat{c}_i^{major} &\geq \sum_{i=0}^{j-1} c_i^{major} + \sum_{i=j}^{n-1} \hat{c}_i^{major} && \textit{(by induction hypothesis)} \\
&= \sum_{i=0}^{j-1} c_i^{major} + \sum_{i=j}^{n-1} 4\gamma N_{n-1}^{\min\{\epsilon,1-\epsilon\}} && \textit{($N_j = \ldots = N_{n-1}$)} \\
&\geq \sum_{i=0}^{j-1} c_i^{major} + \frac{1}{4} N_{n-1}\, 4\gamma N_{n-1}^{\min\{\epsilon,1-\epsilon\}} && \textit{(at least } \frac{1}{4} N_{n-1} \textit{ updates)} \\
&= \sum_{i=0}^{j-1} c_i^{major} + \gamma N_{n-1}^{1+\min\{\epsilon,1-\epsilon\}} \\
&\geq \sum_{i=0}^{j-1} c_i^{major} + c_{n-1}^{major} = \sum_{i=0}^{n-1} c_i^{major} && \textit{($c_j^{major} = \ldots = c_{n-2}^{major} = 0$).}
\end{aligned}$$

Thus, Inequality (3) holds for update sequences of length n.

Minor rebalancing. When the degree of a value in a partition changes such that the heavy or light part condition no longer holds, minor rebalancing moves the affected tuples between

the heavy and light parts of the partition. To prove Inequality (4), we decompose the cost of minor rebalancing per relation and data value of its partitioning variable.

$$c_i^{minor} = \sum_{a \in \mathsf{Dom}(A)} c_i^{R,a} + \sum_{b \in \mathsf{Dom}(B)} c_i^{S,b} + \sum_{c \in \mathsf{Dom}(C)} c_i^{T,c} \quad \text{and}$$

$$\hat{c}_i^{minor} = \sum_{a \in \mathsf{Dom}(A)} \hat{c}_i^{R,a} + \sum_{b \in \mathsf{Dom}(B)} \hat{c}_i^{S,b} + \sum_{c \in \mathsf{Dom}(C)} \hat{c}_i^{T,c}$$

We write $c_i^{R,a}$ and $\hat{c}_i^{R,a}$ to denote the actual and respectively amortized costs of minor rebalancing caused by update u_i, for relation R and an A-value a. If update u_i is of the form $\delta R = \{(\alpha, \beta) \mapsto m\}$ and causes minor rebalancing, then $c_i^{R,\alpha} = c_i^{minor}$; otherwise, $c_i^{R,\alpha} = 0$. If update u_i is of the form $\delta R = \{(\alpha, \beta) \mapsto m\}$, then $\hat{c}_i^{R,\alpha} = \hat{c}_i^{minor}$ regardless of whether u_i causes minor rebalancing or not; otherwise, $\hat{c}_i^{R,\alpha} = 0$. The actual costs $c_i^{S,b}$ and $c_i^{T,c}$ and the amortized costs $\hat{c}_i^{S,b}$ and $\hat{c}_i^{T,c}$ are defined similarly.

We prove that for the partition of R and any $\alpha \in \mathsf{Dom}(A)$ the following inequality holds:

$$\sum_{i=0}^{n-1} \hat{c}_i^{R,\alpha} \geq \sum_{i=0}^{n-1} c_i^{R,\alpha}. \tag{5}$$

Due to the symmetry of the triangle query, Inequality (4) follows directly from Inequality (5).

We prove Inequality (5) for an arbitrary $\alpha \in \mathsf{Dom}(A)$ by induction on the length n of the update sequence.

- *Base case*: We show that Inequality (5) holds for $n = 1$. Assume that update u_0 is of the form $\delta R = \{(\alpha, \beta) \mapsto m\}$; otherwise, $\hat{c}_0^{R,\alpha} = c_0^{R,\alpha} = 0$, and Inequality (5) follows trivially for $n = 1$. If the initial database is empty, u_0 triggers major rebalancing but no minor rebalancing, thus $\hat{c}_0^{R,\alpha} = 2\gamma N_0^{\max\{\epsilon, 1-\epsilon\}} \geq c_0^{R,\alpha} = 0$. If the initial database is nonempty, each relation is partitioned using the threshold N_0^ϵ. For update u_0 to trigger minor rebalancing, the degree of the A-value α in R_h or R_l has to either decrease from $\lceil N_0^\epsilon \rceil$ to $\lceil \frac{1}{2} N_0^\epsilon \rceil - 1$ (heavy to light) or increase from $\lceil N_0^\epsilon \rceil - 1$ to $\lceil \frac{3}{2} N_0^\epsilon \rceil$ (light to heavy). The former happens only if $\lceil N_0^\epsilon \rceil = 1$ and update u_0 removes the last tuple with the A-value α from R_h, thus no minor rebalancing is needed; the latter cannot happen since update u_0 can increase $|\sigma_{A=\alpha} R_l|$ to at most $\lceil N_0^\epsilon \rceil$, and $\lceil N_0^\epsilon \rceil < \lceil \frac{3}{2} N_0^\epsilon \rceil$. In any case, $\hat{c}_0^{R,\alpha} \geq c_0^{R,\alpha}$, which implies that Inequality (5) holds for $n = 1$.
- *Inductive step*: Assumed that Inequality (5) holds for all update sequences of length up to $n - 1$, we show it holds for update sequences of length n. Consider that update u_{n-1} is of the form $\delta R = \{(\alpha, \beta) \mapsto m\}$ and causes minor rebalancing; otherwise, $\hat{c}_{n-1}^{R,\alpha} \geq 0$ and $c_{n-1}^{R,\alpha} = 0$, and Inequality (5) follows trivially for n. Let $\mathcal{Z}_j$ be the state created after the previous major rebalancing or, if there is no such step, the initial state. The threshold changes only with major rebalancing, thus $N_j = N_{j+1} = \ldots = N_{n-1}$. Depending on whether there exist minor rebalancing steps since state $\mathcal{Z}_j$, we distinguish two cases:

Case 1: There is no minor rebalancing caused by an update of the form $\delta R = \{(\alpha, \beta') \mapsto m'\}$ since state $\mathcal{Z}_j$; thus, $c_j^{R,\alpha} = \ldots = c_{n-2}^{R,\alpha} = 0$. From state $\mathcal{Z}_j$ to state $\mathcal{Z}_n$, the number of tuples with the A-value α either decreases from at least $\lceil N_j^\epsilon \rceil$ to $\lceil \frac{1}{2} N_{n-1}^\epsilon \rceil - 1$ (heavy to light) or increases from at most $\lceil N_j^\epsilon \rceil - 1$ to $\lceil \frac{3}{2} N_{n-1}^\epsilon \rceil$ (light to heavy). For this change to happen, the number of updates needs to be greater than $\frac{1}{2} N_{n-1}^\epsilon$ since $N_j = N_{n-1}$ and $\min\{\lceil N_j^\epsilon \rceil - (\lceil \frac{1}{2} N_{n-1}^\epsilon \rceil - 1), \lceil \frac{3}{2} N_{n-1}^\epsilon \rceil - (\lceil N_j^\epsilon \rceil - 1)\} > \frac{1}{2} N_{n-1}^\epsilon$. Then,

$$\begin{aligned}
\sum_{i=0}^{n-1} \hat{c}_i^{R,\alpha} &\geq \sum_{i=0}^{j-1} c_i^{R,\alpha} + \sum_{i=j}^{n-1} \hat{c}_i^{R,\alpha} && \textit{(by induction hypothesis)} \\
&= \sum_{i=0}^{j-1} c_i^{R,\alpha} + \sum_{i=j}^{n-1} 2\gamma N_{n-1}^{\max\{\epsilon,1-\epsilon\}} && (N_j = \ldots = N_{n-1}) \\
&> \sum_{i=0}^{j-1} c_i^{R,\alpha} + \frac{1}{2} N_{n-1}^{\epsilon} 2\gamma N_{n-1}^{\max\{\epsilon,1-\epsilon\}} && \textit{(more than } \frac{1}{2} N_{n-1}^{\epsilon} \textit{ updates)} \\
&\geq \sum_{i=0}^{j-1} c_i^{R,\alpha} + c_{n-1}^{R,\alpha} = \sum_{i=0}^{n-1} c_i^{R,\alpha} && (c_j^{R,\alpha} = \ldots = c_{n-2}^{R,\alpha} = 0).
\end{aligned}$$

Case 2: There is at least one minor rebalancing step caused by an update of the form $\delta R = \{(\alpha, \beta') \mapsto m'\}$ since state $\mathcal{Z}_j$. Let $\mathcal{Z}_\ell$ denote the state created after the previous minor rebalancing caused by an update of this form; thus, $c_\ell^{R,\alpha} = \ldots = c_{n-2}^{R,\alpha} = 0$. The minor rebalancing steps creating $\mathcal{Z}_\ell$ and $\mathcal{Z}_n$ move tuples with the A-value α between R_h and R_l in opposite directions. From state $\mathcal{Z}_\ell$ to state $\mathcal{Z}_n$, the number of such tuples either decreases from $\lceil \frac{3}{2} N_l^{\epsilon} \rceil$ to $\lceil \frac{1}{2} N_{n-1}^{\epsilon} \rceil - 1$ (heavy to light) or increases from $\lceil \frac{1}{2} N_l^{\epsilon} \rceil - 1$ to $\lceil \frac{3}{2} N_{n-1}^{\epsilon} \rceil$ (light to heavy). For this change to happen, the number of updates needs to be greater than N_{n-1}^{ϵ} since $N_l = N_{n-1}$ and $\min\{\lceil \frac{3}{2} N_l^{\epsilon} \rceil - (\lceil \frac{1}{2} N_{n-1}^{\epsilon} \rceil - 1), \lceil \frac{3}{2} N_{n-1}^{\epsilon} \rceil - (\lceil \frac{1}{2} N_l^{\epsilon} \rceil - 1)\} > N_{n-1}^{\epsilon}$. Then,

$$\begin{aligned}
\sum_{i=0}^{n-1} \hat{c}_i^{R,\alpha} &\geq \sum_{i=0}^{\ell-1} c_i^{R,\alpha} + \sum_{i=\ell}^{n-1} \hat{c}_i^{R,\alpha} && \textit{(by induction hypothesis)} \\
&= \sum_{i=0}^{\ell-1} c_i^{R,\alpha} + \sum_{i=\ell}^{n-1} 2\gamma N_{n-1}^{\max\{\epsilon,1-\epsilon\}} && (N_j = \ldots = N_{n-1}) \\
&> \sum_{i=0}^{\ell-1} c_i^{R,\alpha} + N_{n-1}^{\epsilon} 2\gamma N_{n-1}^{\max\{\epsilon,1-\epsilon\}} && \textit{(more than } N_{n-1}^{\epsilon} \textit{ updates)} \\
&> \sum_{i=0}^{\ell-1} c_i^{R,\alpha} + c_{n-1}^{R,\alpha} = \sum_{i=0}^{n-1} c_i^{R,\alpha} && (c_\ell^{R,\alpha} = \ldots = c_{n-2}^{R,\alpha} = 0).
\end{aligned}$$

Cases 1 and 2 imply that Inequality (5) holds for update sequences of length n.

This shows that Inequality (2) holds when the amortized cost of OnUpdate$(u_i, \mathcal{Z}_i)$ is

$$\hat{c}_i = \gamma N_i^{\max\{\epsilon,1-\epsilon\}} + 4\gamma N_i^{\min\{\epsilon,1-\epsilon\}} + 2\gamma N_i^{\max\{\epsilon,1-\epsilon\}} + \Gamma, \quad \text{for } 0 \leq i < n,$$

where Γ and γ are constants. The amortized cost $\hat{c}_i^{major}$ of major rebalancing is $4\gamma N_i^{\min\{\epsilon,1-\epsilon\}}$, and the amortized cost $\hat{c}_i^{minor}$ of minor rebalancing is $2\gamma N_i^{\max\{\epsilon,1-\epsilon\}}$. From the size invariant $\lfloor \frac{1}{4} N_i \rfloor \leq |\mathbf{D}_i| < N_i$ follows that $|\mathbf{D}_i| < N_i < 4(|\mathbf{D}_i| + 1)$ for $0 \leq i < n$, where $|\mathbf{D}_i|$ is the database size before update u_i. This implies that for any database $\mathbf{D}$, the amortized major rebalancing time is $\mathcal{O}(|\mathbf{D}|^{\min\{\epsilon,1-\epsilon\}})$, the amortized minor rebalancing time is $\mathcal{O}(|\mathbf{D}|^{\max\{\epsilon,1-\epsilon\}})$, and the overall amortized update time of IVM$^\epsilon$ is $\mathcal{O}(|\mathbf{D}|^{\max\{\epsilon,1-\epsilon\}})$. ◀

Given $\epsilon \in [0, 1]$, IVM$^\epsilon$ maintains the triangle count query in $\mathcal{O}(|\mathbf{D}|^{\max\{\epsilon,1-\epsilon\}})$ amortized update time while using $\mathcal{O}(|\mathbf{D}|^{1+\min\{\epsilon,1-\epsilon\}})$ space. It thus defines a tradeoff between time and space parameterized by ϵ, as shown in Figure 1. IVM$^\epsilon$ achieves the optimal amortized update time $\mathcal{O}(|\mathbf{D}|^{\frac{1}{2}})$ at $\epsilon = \frac{1}{2}$, for which the space is $\mathcal{O}(|\mathbf{D}|^{\frac{3}{2}})$.

5 Conclusion and Future Work

This paper introduces IVM$^\epsilon$, an incremental maintenance approach to counting triangles under updates that exhibits a space-time tradeoff such that the space-time product is quadratic in the size of the database. IVM$^\epsilon$ can trade space for update time. The amortized update time can be as low as the square root of the database size, which is worst-case optimal conditioned on the Online Matrix-Vector Multiplication (OMv) conjecture. The space requirements of IVM$^\epsilon$ can be improved to linear while keeping the amortized update time optimal by using a refined partitioning that takes into account the degrees of data values for both variables (instead of one variable only) in each relation [13]. IVM$^\epsilon$ captures classical and factorized IVM as special cases with suboptimal, linear update time [13].

There are worst-case optimal algorithms for *join* queries in the *static* setting [17]. In contrast, IVM$^\epsilon$ is worst-case optimal for the *count* aggregate over the triangle join query in the *dynamic* setting. The latter setting poses challenges beyond the former. First, the optimality argument for static join algorithms follows from their runtime being linear(ithmic) in their output size; this argument does not apply to our triangle count query, since its output is a scalar and hence of constant size. Second, optimality in the dynamic setting requires a more fine-grained argument that exploits the skew in the data for different evaluation strategies, view materialization, and delta computation; in contrast, there are static worst-case optimal join algorithms that do not need to exploit skew, materialize views, nor delta computation.

This paper opens up a line of work on *dynamic worst-case optimal query evaluation algorithms*. The goal is a complete characterization of the complexity of incremental maintenance for arbitrary functional aggregate queries over various rings [1]. Different rings can be used as the domain of tuple multiplicities (or payloads). We used here the ring $(\mathbb{Z}, +, \cdot, 0, 1)$ of integers to support counting. The relational data ring supports payloads with listing and factorized representations of relations, and the degree-m matrix ring supports payloads with gradients used for learning linear regression models [19].

Towards the aforementioned goal, we would first like to find a syntactical characterization of all queries that admit incremental maintenance in (amortized) sublinear time. Using known (first-order, fully recursive, or factorized) incremental maintenance techniques, cyclic and even acyclic joins require at least linear update time. Our intuition is that this characterization is given by a notion of diameter of the query hypergraph. This class strictly contains the q-hierarchical queries, which admit constant-time updates [4].

Minor variants of IVM$^\epsilon$ can be used to maintain the counting versions of any query built using three relations, the 4-path query, and the Loomis-Whitney queries in worst-case optimal time [13]. The same conditional lower bound on the update time shown for the triangle count applies for most of the mentioned queries, too. This leads to the striking realization that, while in the static setting the counting versions of the cyclic query computing triangles and the acyclic query computing paths of length 3 have different complexities, $\mathcal{O}(|\mathbf{D}|^{\frac{3}{2}})$ and $\mathcal{O}(|\mathbf{D}|)$, and pose distinct computational challenges, they share the same complexity and can use a very similar approach in the dynamic setting. A further IVM$^\epsilon$ variant allows the constant-delay enumeration of all triangles after each update, while preserving the same optimal amortized update time as for counting triangles [13]. These variants exploit the fact that our amortization technique is agnostic to the query to maintain and the update mechanism. It relies on two prerequisites. First, rebalancing is performed by moving tuples between relation parts. Second, the number of moved tuples per rebalancing is asymptotically no more than the number of updates performed since the previous rebalancing.

References

1 Mahmoud Abo Khamis, Hung Q. Ngo, and Atri Rudra. FAQ: Questions Asked Frequently. In *PODS*, pages 13–28, 2016.

2 N. Alon, R. Yuster, and U. Zwick. Finding and Counting Given Length Cycles. *Algorithmica*, 17(3):209–223, 1997.

3 Ziv Bar-Yossef, Ravi Kumar, and D Sivakumar. Reductions in Streaming Algorithms, with an Application to Counting Triangles in Graphs. In *SODA*, pages 623–632, 2002.

4 Christoph Berkholz, Jens Keppeler, and Nicole Schweikardt. Answering Conjunctive Queries Under Updates. In *PODS*, pages 303–318, 2017.

5 Christoph Berkholz, Jens Keppeler, and Nicole Schweikardt. Answering UCQs Under Updates and in the Presence of Integrity Constraints. In *ICDT*, pages 8:1–8:19, 2018.

6 Luciana S Buriol, Gereon Frahling, Stefano Leonardi, Alberto Marchetti-Spaccamela, and Christian Sohler. Counting Triangles in Data Streams. In *PODS*, pages 253–262, 2006.

7 Rada Chirkova and Jun Yang. Materialized Views. *Found. & Trends DB*, 4(4):295–405, 2012.

8 Graham Cormode and Hossein Jowhari. A Second Look at Counting Triangles in Graph Streams (Corrected). *Theor. Comput. Sci.*, 683:22–30, 2017.

9 Talya Eden, Amit Levi, Dana Ron, and C Seshadhri. Approximately Counting Triangles in Sublinear Time. In *FOCS*, pages 614–633, 2015.

10 Monika Henzinger, Sebastian Krinninger, Danupon Nanongkai, and Thatchaphol Saranurak. Unifying and Strengthening Hardness for Dynamic Problems via the Online Matrix-Vector Multiplication Conjecture. In *STOC*, pages 21–30, 2015.

11 Muhammad Idris, Martín Ugarte, and Stijn Vansummeren. The Dynamic Yannakakis Algorithm: Compact and Efficient Query Processing Under Updates. In *SIGMOD*, pages 1259–1274, 2017.

12 Hossein Jowhari and Mohammad Ghodsi. New Streaming Algorithms for Counting Triangles in Graphs. In *COCOON*, pages 710–716, 2005.

13 Ahmet Kara, Hung Q. Ngo, Milos Nikolic, Dan Olteanu, and Haozhe Zhang. Counting Triangles under Updates in Worst-Case Optimal Time. *CoRR*, abs/1804.02780, 2018. URL: `http://arxiv.org/abs/1804.02780`.

14 Christoph Koch, Yanif Ahmad, Oliver Kennedy, Milos Nikolic, Andres Nötzli, Daniel Lupei, and Amir Shaikhha. DBToaster: Higher-Order Delta Processing for Dynamic, Frequently Fresh Views. *VLDB J.*, 23(2):253–278, 2014.

15 Paraschos Koutris, Semih Salihoglu, and Dan Suciu. Algorithmic Aspects of Parallel Data Processing. *Found. & Trends DB*, 8(4):239–370, 2018.

16 Andrew McGregor, Sofya Vorotnikova, and Hoa T Vu. Better Algorithms for Counting Triangles in Data Streams. In *PODS*, pages 401–411, 2016.

17 Hung Q. Ngo. Worst-Case Optimal Join Algorithms: Techniques, Results, and Open Problems. In *PODS*, pages 111–124, 2018.

18 Hung Q. Ngo, Christopher Ré, and Atri Rudra. Skew Strikes Back: New Developments in the Theory of Join Algorithms. *SIGMOD Record*, 42(4):5–16, 2013.

19 Milos Nikolic and Dan Olteanu. Incremental View Maintenance with Triple Lock Factorization Benefits. In *SIGMOD*, pages 365–380, 2018.

20 Thomas Schwentick and Thomas Zeume. Dynamic Complexity: Recent Updates. *SIGLOG News*, 3(2):30–52, 2016.

21 Virginia Vassilevska Williams. On Some Fine-Grained Questions in Algorithms and Complexity. In *ICM*, volume 3, pages 3431–3472, 2018.

22 Thomas Zeume. The Dynamic Descriptive Complexity of k-Clique. *Inf. Comput.*, 256:9–22, 2017.