Counting Triangles under Updates*

Ahmet Kara[1], Hung Q. Ngo[2], Milos Nikolic[1], Dan Olteanu[1], Haozhe Zhang[1]

[1] University of Oxford [2] RelationalAI, Inc.

1 Problem Setting and Contributions

We consider the problem of maintaining the result of the triangle count query $Q(\,) = \mathit{\Gamma}_{;sum}\, R(A,B) \bowtie S(B,C) \bowtie T(C,A)$ under single-tuple updates to the input relations R, S, and T. The relations are given as key-payload maps whose keys are tuples over relation schemas, payloads are tuple multiplicities, and key lookups are (amortized) $\mathcal{O}(1)$-time operations. A single-tuple update $\delta R(a,b) = \{\,(a,b) \mapsto p\,\}$ to relation R maps a key (a,b) to a nonzero payload p (positive for inserts and negative for deletes); updates to S and T are analogous.

The naïve maintenance approach recomputes the triangle count from scratch after each update. Computing this query using worst-case optimal join algorithms [5] takes $\mathcal{O}(N^{1.5})$ time, where N is the current size of the input database.

To *incrementally* maintain the triangle count under single-tuple updates, existing incremental view maintenance (IVM) approaches need linear time. For instance, under the update δR to R, the classical IVM [2] computes the delta query $\mathit{\Gamma}_{;sum}\, \delta R(a,b) \bowtie S(b,C) \bowtie T(C,a)$ in $\mathcal{O}(N)$ time because it needs to intersect two lists of possibly linearly many C-values that are paired with b in S and with a in T. The factorized IVM [6] materializes the view $V_{ST}(B,A) = \mathit{\Gamma}_{B,A;sum}\, S(B,C) \bowtie T(C,A)$ using $\mathcal{O}(N^2)$ space. It then computes the delta query $\mathit{\Gamma}_{;sum}\, \delta R(a,b) \bowtie V_{ST}(b,a)$ in $\mathcal{O}(1)$ time; however, updates to S and T still require $\mathcal{O}(N)$ time to maintain the triangle count Q and view V_{ST}.

This raises the question of whether the triangle count can be maintained in sublinear time. Recent work proves that no algorithm can maintain Q in time $\mathcal{O}(N^{0.5-\gamma})$ for any $\gamma > 0$, under reasonable complexity-theoretic assumptions [1]. An algorithm with sublinear maintenance time for Q is not yet known.

This work introduces IVM$^\epsilon$, an IVM approach that maintains the triangle count in *amortized sublinear* time. IVM$^\epsilon$ partitions each input relation into two parts, heavy and light, based on the degrees of data values, the database size, and a parameter ϵ. It then adapts the maintenance strategy to different heavy-light combinations of parts of the input relations to achieve worst-case sublinear maintenance. As the database evolves under updates, IVM$^\epsilon$ rebalances the 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 update.

Given a database of size N and $\epsilon \in [0,1]$, IVM$^\epsilon$ maintains the triangle count in $\mathcal{O}(N^{\max\{\epsilon,1-\epsilon\}})$ amortized time while using $\mathcal{O}(N^{1+\min\{\epsilon,1-\epsilon\}})$ space. It thus defines a continuum of approaches exhibiting a space-time tradeoff based on ϵ.

* An extended version of this work is available online [3].

Materialized View Definition	Space Complexity
$Q(\,) = \bigcup_{u,v,w\in\{h,l\}} \Gamma_{;sum}\ R_u(A,B) \bowtie S_v(B,C) \bowtie T_w(C,A)$	$\mathcal{O}(1)$
$V_{RS}(A,C) = \Gamma_{A,C;sum}\ R_h(A,B) \bowtie S_l(B,C)$	$\mathcal{O}(N^{1+\min\{\epsilon,1-\epsilon\}})$
$V_{ST}(B,A) = \Gamma_{B,A;sum}\ S_h(B,C) \bowtie T_l(C,A)$	$\mathcal{O}(N^{1+\min\{\epsilon,1-\epsilon\}})$
$V_{TR}(C,B) = \Gamma_{C,B;sum}\ T_h(C,A) \bowtie R_l(A,B)$	$\mathcal{O}(N^{1+\min\{\epsilon,1-\epsilon\}})$

Fig. 1. The materialized views used by IVM$^\epsilon$ for a database of size N and $\epsilon \in [0,1]$.

Setting $\epsilon = 0.5$ gives $\mathcal{O}(N^{0.5})$ amortized *worst-case optimal* time and $\mathcal{O}(N^{1.5})$ space utilization. Existing IVM approaches are extreme points in this continuum of approaches defined by IVM$^\epsilon$. For instance, to recover classical IVM, we set $\epsilon \in \{0,1\}$ to achieve $\mathcal{O}(N)$ update time and $\mathcal{O}(N)$ space utilization; to recover factorized IVM, we set distinct parameters ϵ for each relation (cf. [3] for details). IVM$^\epsilon$ can also count all triangles in a static database in worst-case optimal time $\mathcal{O}(N^{1.5})$ by inserting N tuples, one at a time, into initially empty relations.

2 Adaptive Maintenance Strategy

We split each input relation into two disjoint parts, called heavy and light parts. Given $\epsilon_R \in [0,1]$, an A-value a is heavy in R if $|\sigma_{A=a}R| \geq N^{\epsilon_R}$, where N is the database size; otherwise, it is light. We partition R into R_h and R_l such that $R_h = \{\, t \in R \mid t.A \text{ is heavy} \,\}$ and $R_l = R \setminus R_h$; similarly, we partition S on B, and T on C. In the following, we assume that $\epsilon = \epsilon_R = \epsilon_S = \epsilon_T$ is fixed.

We decompose the query Q into skew-aware views expressed over the relation parts: $Q_{uvw}(\,) = \Gamma_{;sum}\, R_u(A,B) \bowtie S_v(B,C) \bowtie T_w(C,A)$, where $u,v,w \in \{h,l\}$. The query Q is thus a union (sum) of partial counts: $Q(\,) = \bigcup_{u,v,w\in\{h,l\}} Q_{uvw}(\,)$.

We adapt the maintenance strategy to each skew-aware view to ensure sublinear update time. While most of these views admit sublinear delta computation, few exceptions require linear-time maintenance. For these exceptions, IVM$^\epsilon$ precomputes the update-independent parts of delta queries as *materialized views* and uses them to speed up the delta evaluation. Such auxiliary views also require maintenance, yet their maintenance cost is sublinear for single-tuple updates.

Figure 1 shows the materialized views used by IVM$^\epsilon$ to maintain the triangle count query. The size of the view $V_{RS}(A,C)$ is upper-bounded by the size of the result of the join of $R_h(A,B)$ and $S_l(B,C)$ in two distinct ways. One can iterate over all (a,b) pairs in R_h and then find the C-values in S_l for each b. Since S_l contains only tuples with light B-values, there are at most N^ϵ distinct C-values for each B-value. This gives an upper bound of $\mathcal{O}(|R_h| \cdot N^\epsilon) = \mathcal{O}(N^{1+\epsilon})$. Alternatively, one can iterate over all (b,c) pairs in S_l and then find the A-values in R_h for each b. Since R_h contains only tuples with heavy A-values, there are at most $\frac{N}{N^\epsilon} = N^{1-\epsilon}$ distinct A-values. This gives an upper bound of $\mathcal{O}(|S_l| \cdot N^{1-\epsilon}) = \mathcal{O}(N^{2-\epsilon})$. The overall space complexity is the minimum of the bounds. The space analysis for V_{ST} and V_{TR} is analogous.

We explain our adaptive strategy on a single-tuple update $\delta R_*(a,b)$ to relation R. This update can affect either the heavy or light part of R, hence the $*$

Delta Evaluation Strategy	Time Complexity
$\delta Q_{*hh}(\,) = \delta R_*(a,b) \cdot \sum_C T_h(C,a) \cdot S_h(b,C)$	$\mathcal{O}(N^{1-\epsilon})$
$\delta Q_{*hl}(\,) = \delta R_*(a,b) \cdot V_{ST}(b,a)$	$\mathcal{O}(1)$
$\delta Q_{*lh}(\,) = \delta R_*(a,b) \cdot \sum_C T_h(C,a) \cdot S_l(b,C)$ or $= \delta R_*(a,b) \cdot \sum_C S_l(b,C) \cdot T_h(C,a)$	$\mathcal{O}(N^{\min\{\epsilon,1-\epsilon\}})$
$\delta Q_{*ll}(\,) = \delta R_*(a,b) \cdot \sum_C S_l(b,C) \cdot T_l(C,a)$	$\mathcal{O}(N^{\epsilon})$
$\delta Q(\,) = \delta Q_{*hh}(\,) + \delta Q_{*hl}(\,) + \delta Q_{*lh}(\,) + \delta Q_{*ll}(\,)$	$\mathcal{O}(1)$
$\delta V_{RS}(a,C) = \delta R_h(a,b) \cdot S_l(b,C)$	$\mathcal{O}(N^{\epsilon})$
$\delta V_{TR}(C,b) = \delta R_l(a,b) \cdot T_h(C,a)$	$\mathcal{O}(N^{1-\epsilon})$

Fig. 2. Computing the deltas of the views from Figure 1 for an update $\delta R_*(a,b)$ to the heavy or light part of R. The symbol $*$ stands for h or l. The delta δV_{ST} is empty since V_{ST} does not refer to R. The evaluation order of deltas is from left to right.

symbol; we assume that checking whether a is heavy or not in R is a constant-time operation. Updates to the other two relations are handled similarly.

Figure 2 shows the deltas of the views affected by the update $\delta R_*(a,b)$ and their time complexity when evaluated from left to right. In all but one case, the complexity is determined by the number of C-values that need to be iterated over. Computing the deltas involves multiplying the payloads of matching tuples and, if C is not in the target view schema, summing them over C-values.

We first analyze the access patterns of the skew-aware delta views: (1) For δQ_{*hh}, we iterate over at most $N^{1-\epsilon}$ C-values in T_h for the given a and then look up in S_h for each (b,c); (2) For δQ_{*hl}, we look up in the materialized view V_{ST} for the given (a,b); (3) For δQ_{*lh}, we either iterate over at most $N^{1-\epsilon}$ C-values in T_h for the given a and look up in S_l for each (b,c), or we iterate over at most N^{ϵ} C-values in S_l for the given b and look up in T_h for each (c,a); (4) For δQ_{*ll}, we iterate over at most N^{ϵ} C-values in S_l for the given b and then look up in T_l for each (c,a). Then, summing these partial deltas and updating Q take constant time. The views V_{RS} and V_{TR}, which facilitate updates to T and respectively to S, are maintained for updates to distinct parts of R. Computing δV_{RS} and updating V_{RS} requires iterating over at most N^{ϵ} C-values in S_l for the given b; similarly, computing δV_{TR} and updating V_{TR} involves at most $N^{1-\epsilon}$ heavy C-values in T_h. The final step of IVM$^{\epsilon}$ updates the (heavy or light) part of R that corresponds to δR_* in (amortized) $\mathcal{O}(1)$ time. Overall, IVM$^{\epsilon}$ maintains the views from Figure 1 under single-tuple updates to any of the input relations in $\mathcal{O}(N^{\max\{\epsilon,1-\epsilon\}})$ time using $\mathcal{O}(N^{1+\min\{\epsilon,1-\epsilon\}})$ space.

An insert (a,b) into R may promote a from light to heavy in R or may increase the heavy-light threshold such that some A-values change from heavy to light. Without rebalancing the partitions, our assumptions on the number of B-values paired with a or the number of heavy A-values may become invalid.

IVM$^{\epsilon}$ loosens the partition threshold to amortize the cost of rebalancing over multiple updates. Instead of the actual database size N, the threshold now depends on a variable M for which the invariant $\lfloor \frac{1}{4}M \rfloor \leq N < M$ always holds. If the database size violates one of the limits, we perform *major rebalancing* where we double or halve M to satisfy the invariant again, repartition the input relations using the new threshold M^ϵ, and recompute the auxiliary views. The time complexity of this operation is $\mathcal{O}(M^{1+\min\{\epsilon,1-\epsilon\}})$, which is amortized over at least $\lceil \frac{1}{4}M \rceil$ updates between two major rebalancing steps.

IVM$^\epsilon$ also enforces the following two invariants: The number of tuples with the same value of the partitioning attribute is less than $\frac{3}{2}M^\epsilon$ in each light part and at least $\frac{1}{2}M^\epsilon$ in each heavy part. If any of the two invariants is violated, we perform *minor rebalancing* where we move at most $\lceil \frac{3}{2}M^\epsilon \rceil$ tuples from one part to another and update the affected views. The time complexity of this operation is $\mathcal{O}(M^{\epsilon+\max\{\epsilon,1-\epsilon\}})$, which is amortized over at least $\lceil \frac{1}{2}M^\epsilon \rceil$ updates between two minor rebalancing steps for the same value of the partitioning attribute.

In conclusion, both rebalancing steps together take $\mathcal{O}(M^{\max\{\epsilon,1-\epsilon\}})$ amortized time. Since each single-tuple update can be realized in time $\mathcal{O}(M^{\max\{\epsilon,1-\epsilon\}})$ and $M = \mathcal{O}(N)$, IVM$^\epsilon$ needs $\mathcal{O}(N^{\max\{\epsilon,1-\epsilon\}})$ overall amortized time. The extended version of this work presents a detailed complexity analysis of IVM$^\epsilon$ [3].

3 Beyond the Triangle Query

IVM$^\epsilon$ can be applied to any query but may not always yield asymptotic improvements over existing approaches. It can achieve sublinear maintenance for the counting variants of acyclic queries, e.g., 3-path and 4-path, and cyclic queries, e.g., Loomis-Whitney and 4-cycle. Different semirings can be used to specifiy operations on the payloads [6]; we used here $(\mathbb{Z}, +, *, 0, 1)$ to express counting. An early prototype implementation of IVM$^\epsilon$ on top of DBToaster [4] shows several factors performance improvement over classical and factorized IVM.

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

References

1. Christoph Berkholz, Jens Keppeler, and Nicole Schweikardt. Answering Conjunctive Queries under Updates. In *PODS*, pages 303–318, 2017.
2. Rada Chirkova and Jun Yang. Materialized Views. *Found. & Trends in DB*, 4(4):295–405, 2012.
3. 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.
4. Christoph Koch, Yanif Ahmad, et al. DBToaster: Higher-order Delta Processing for Dynamic, Frequently Fresh Views. *VLDB J.*, 23(2):253–278, 2014.
5. 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.
6. Milos Nikolic and Dan Olteanu. Incremental View Maintenance with Triple Lock Factorization Benefits. In *SIGMOD*, 2018. (to appear).