CommonGraph: Graph Analytics on Evolving Data (Abstract)

Mahbod Afarin*
mafar001@ucr.edu
CSE Department, UC Riverside
USA

Chao Gao*
cgao037@ucr.edu
CSE Department, UC Riverside
USA

Shafiur Rahman
mrahm008@ucr.edu
CSE Department, UC Riverside
USA

Nael Abu-Ghazaleh
nael@cs.ucr.edu
CSE Department, UC Riverside
USA

Rajiv Gupta
rajivg@ucr.edu
CSE Department, UC Riverside
USA

ABSTRACT

We consider the problem of graph analytics on evolving graphs. In this scenario, a query typically needs to be applied to different snapshots of the graph over an extended time window. We propose *CommonGraph*, an approach for efficient processing of queries on evolving graphs. We first observe that edge deletions are significantly more expensive than addition operations. *CommonGraph* converts all deletions to additions by finding a common graph that exists across all snapshots. After computing the query on this graph, to reach any snapshot, we simply need to add the missing edges and incrementally update the query results. *CommonGraph* also allows sharing of common additions among snapshots that require them, and breaks the sequential dependency inherent in the traditional streaming approach where snapshots are processed in sequence, enabling additional opportunities for parallelism. We incorporate the *CommonGraph* approach by extending the KickStarter streaming framework. *CommonGraph* achieves 1.38×-8.17× improvement in performance over Kickstarter across multiple benchmarks.

CCS CONCEPTS

• **Computing methodologies** → **Parallel computing methodologies**; • **Information systems** → **Computing platforms**.

KEYWORDS

evolving graphs, iterative graph algorithms, work sharing

ACM Reference Format:
Mahbod Afarin, Chao Gao, Shafiur Rahman, Nael Abu-Ghazaleh, and Rajiv Gupta. 2023. CommonGraph: Graph Analytics on Evolving Data (Abstract). In *Proceedings of the 2023 ACM Workshop on Highlights of Parallel Computing (HOPC '23), June 16, 2023, Orlando, FL, USA.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3597635.3598022

1 MOTIVATION

Analyses on large graphs are an increasingly important computational workload as graph analytics is employed in many domains to uncover insights by mining high volumes of connected data. Graphs are often dynamic, with edges and vertices being added or removed over time. There are two broad classes of analyses on dynamic graphs: (1) *Streaming graph analytics*: where results of a query are continuously updated as the graph continues to change because updates to it stream in over time. Typically incremental algorithms are employed to update query results in response to graph changes [2–5]; and (2) *Evolving graph analytics*: where multiple snapshots of the graph are available and an evolving graph query seeks to *track a graph property over a time window* by computing its value for different snapshots within the time window. Such evaluation is computationally expensive. A typical approach for the evolving scenario is to start with query evaluation the earliest snapshot and then use streaming to incrementally evaluate the query on snapshots in sequence. This approach has a number of drawbacks. First, for many algorithms the cost of edge deletions is very high. Second, the solution moves between snapshots in sequence which limits opportunities for sharing query evaluation work among them.

When incremental algorithms are used, as we move from one snapshot to the next, the graph for the snapshot is first mutated to obtain the one for the next snapshot and then the incremental algorithm is used to update query results. A primary observation that motivates *CommonGraph* is that when the system has to handle both deletions and additions, the cost of incremental computation that handles deletions is significantly higher than that for additions. Moreover, the cost of graph mutations is also significant. The incremental cost of processing a batch of deletions is nearly 3× the cost of processing an equal number of additions. Handling deletions is more expensive because the algorithm is more complex for monotonic algorithms and impacts on query results are more widespread across the graph, necessitating significantly more processing. Finally, Kickstarter's cost of graph mutation is greater for deletions than additions. This work is an abstracted version of the CommonGraph paper [1].

2 SOLUTION

To address the above problems (i.e., the high incremental cost of deletions and the significant cost of mutation), our system shown in Figure 1 based on *CommonGraph* introduces these three complementary techniques. The graph is represented in form of the shared *CommonGraph* and additional batches of edges (Δ batches) that can be used in conjunction with the *CommonGraph* to realize different snapshots. This representation of the graph and its updates allows

*Both authors contributed equally to this research.

HOPC '23, June 16, 2023, Orlando, FL, USA

ACM ISBN 979-8-4007-0218-1/23/06.
https://doi.org/10.1145/3597635.3598022

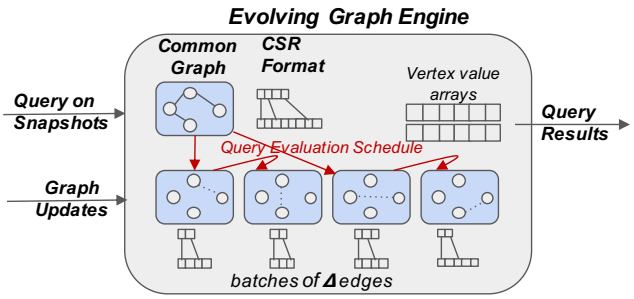

Figure 1: *CommonGraph* Approach.

different query evaluation schedules (shown as red arrows) to be realized that do not require deletions, incorporate work sharing, and do not require explicit graph mutation. Next, we provide an overview of these three features.

Converting Deletions to Additions. To overcome the high computational cost of processing edge deletions, we make a key observation: all deletions can be converted to additions by computing a *CommonGraph* that includes only the edges that are common to all the snapshots under consideration. Once query results have been computed on this graph, then results for any snapshot can be computed by *adding the batch of missing edges* to the *CommonGraph* and employing the incremental algorithm to update query results. That is, we can avoid the use of the more expensive incremental algorithm for deletions since deletions become additions if we reverse the order in which the snapshots are processed.

Work Sharing for a Large Number of Snapshots. Although the *CommonGraph* achieves work sharing among all the snapshots to process the common graph itself, as the time window grows and the number of snapshots increases, additional opportunities for work sharing of the graph updates among subsets of snapshots arise. Specifically, any subset of the snapshots may share additional edges in common, and can share work if we stream the additional edges to reach this larger common graph together instead of each query adding them separately to each snapshot. Assume that we have n snapshots to consider. Our second contribution, the *Triangular Grid* representation, allows systematic exploration and discovery of a n incremental computations that result in query results for all n snapshots while at the same time maximizing the *work sharing*.

3 KEY RESULTS

We evaluate *CommonGraph* on five benchmarks and five graphs on 50 snapshots. Each snapshot is separated from the next by a batch of 75K edge changes split *evenly* between additions and deletions. The first row for each benchmark in the Table 1 is baseline *KickStarter*: we start from the initial snapshot and stream in the batches to reach the next snapshot repeatedly. In the second row for each benchmark, we can see the speedup for *CommonGraph* with Direct-Hop; it outperforms the baseline KickStarter 1.02×-7.91×; even though it processes a higher number of edges compared to KickStarter, all these edges are additions and benefit also from parallelism among additions since they are processed in a single batch. Moreover, some of the benefits come from avoiding the cost of graph mutation through our graph representation. Additional speedup is achieved using work sharing, for an overall speedup of 1.38x-8.17x over baseline KickStarter.

Table 1: Average Execution Times in Seconds for KickStarter (KS), and the speedup of CommonGraph Direct Hop (DH) and Work-Sharing (WS) over KickStarter for 50 Snapshots.

G	Alg.	BFS	SSSP	SSWP	SSNP	VT
LJ	KS Time	3.43s	3.88s	3.69s	3.75s	5.17s
	DH Spe.	1.58×	1.07×	1.23×	1.18×	1.02×
	WS Spe.	1.86×	1.43×	1.38×	1.43×	1.62×
DL	KS Time	27.22s	27.64s	27.91s	27.51s	31.87s
	DH Spe.	7.09×	7.45×	7.3×	6.7×	7.91×
	WS Spe.	7.17×	8.17×	7.64×	7.21×	8.17×
Wen	KS Time	4.65s	4.59s	4.72s	4.20s	2.03s
	DH Spe.	4.53×	1.32×	2.73×	2.08×	3.24×
	WS Spe.	4.68×	2.42×	3.31×	2.40×	3.8×
TTW	KS Time	10.91s	11.73s	11.32s	11.31s	15.30s
	DH Spe.	3.09×	2.36×	2.52×	1.85×	2.85×
	WS Spe.	3.35×	2.94×	3.14×	2.62×	3.42×

4 KEY CONTRIBUTION

The key contributions of our work are as follows:

- A new approach to evolving graphs analysis that converts all graph updates to additions over the *CommonGraph*, replaces expensive deletions by additions, and removes dependencies that enable work sharing among the snapshots.
- A structure called *Triangular Grid* (TG) that exposes work sharing possibilities among the snapshots.
- A graph representation that avoids the need to mutate graphs and enables reuse of edges by snapshots that share them.
- We build the *CommonGraph* that exploits the three ideas above to achieve speedups ranging between 1.38× and 8.17× over Kickstarter.

REFERENCES

[1] Mahbod Afarin, Chao Gao, Shafiur Rahman, Nael Abu-Ghazaleh, and Rajiv Gupta. 2023. CommonGraph: Graph Analytics on Evolving Data. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'23), Vancouver, BC, Canada, March 25–29, 2023.* ACM, 133–145. https://doi.org/10.1145/3575693.3575713

[2] Xiaolin Jiang, Chengshuo Xu, Xizhe Yin, Zhijia Zhao, and Rajiv Gupta. 2021. Tripoline: generalized incremental graph processing via graph triangle inequality. In *EuroSys '21: Sixteenth European Conference on Computer Systems, Online Event, United Kingdom, April 26-28, 2021.* ACM, 17–32.

[3] Shafiur Rahman, Mahbod Afarin, Nael Abu-Ghazaleh, and Rajiv Gupta. 2021. JetStream: Graph Analytics on Streaming Data with Event-Driven Hardware Accelerator. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture* (Virtual Event, Greece) *(MICRO '21).* Association for Computing Machinery, New York, NY, USA, 1091–1105. https://doi.org/10.1145/3466752.3480126

[4] Keval Vora, Rajiv Gupta, and Guoqing Xu. 2017. Kickstarter: Fast and accurate computations on streaming graphs via trimmed approximations. In *Proceedings of the twenty-second international conference on architectural support for programming languages and operating systems.* 237–251.

[5] Xizhe Yin, Zhijia Zhao, and Rajiv Gupta. 2022. Glign: Taming Misaligned Graph Traversals in Concurrent Graph Processing. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1.* 78–92.