2018 IEEE 34th International Conference on Data Engineering

Incremental Frequent Subgraph Mining on Large Evolving Graphs

(Extended abstract)

Ehab Abdelhamid [1], Mustafa Canim [2], Mohammad Sadoghi [3]
Bishwaranjan Bhattacharjee [2], Yuan-Chi Chang [2], Panos Kalnis [4]

[1] *Imperial College London* [2] *IBM Thomas J. Watson Research Center*
[3] *University of California, Davis* [4] *King Abdullah University of Science and Technology*
[1] e.abdelhamid@imperial.ac.uk [2] mustafa, bhatta, yuanchi@us.ibm.com
[3] msadoghi@ucdavis.edu [4] panos.kalnis@kaust.edu.sa

***Abstract*—Frequent subgraph mining is a core graph operation used in many domains. Most existing techniques target static graphs. However, modern applications utilize large evolving graphs. Mining these graphs using existing techniques is infeasible because of the high computational cost. We propose *IncGM+*, a fast incremental approach for frequent subgraph mining on large evolving graphs. We adapt the notion of "fringe" to the graph context, that is, the set of subgraphs on the border between frequent and infrequent subgraphs. *IncGM+* maintains fringe subgraphs and exploits them to prune the search space. To boost efficiency, *IncGM+* stores a number of selected embeddings to avoid redundant expensive subgraph isomorphism operations. Moreover, the proposed system supports batch updates. Our results confirm that *IncGM+* outperforms existing methods, scales to larger graphs and consumes less memory.**

I. Introduction

Frequent Subgraph Mining (FSM) finds all subgraphs that have support larger than or equal to a user-defined threshold τ. FSM is crucial for graph analysis and is a basic building block of many applications in different domains. Past efforts assume graphs are static [1], [2]. However, emerging graph-based applications need to manage constantly evolving graphs, such as social networks and web graphs. A naive approach for mining frequent subgraphs in an evolving graph is to run the FSM algorithm from scratch after every graph update; we call this *FullRecomp*. A typical FSM algorithm consists of multiple iterations, each iteration generates and evaluates several candidate subgraphs. These iterations are repeated until no more frequent subgraphs are found. This process involves numerous expensive NP-complete subgraph isomorphism computations, making *FullRecomp* infeasible in practice.

Another solution would rely on ideas borrowed from the related area of frequent itemset mining. We present *MomentFSM* as a solution that adapts MOMENT [3], a well known frequent itemset mining system, to the graph domain. *MomentFSM* needs to maintain a fringe of subgraphs and process each subgraph in this fringe; for each one, it stores all of its embeddings. Our experiments reveal that *MomentFSM* is too expensive in terms of memory and computational cost.

We propose *IncGM+* to alleviate the problems associated with *FullRecomp* and *MomentFSM*. *IncGM+* is an incremental solution that uses a fringe of selected subgraphs and introduces a number of novel ideas that collectively result in superior performance. A long version of this paper appears in [4].

II. Incremental Graph Mining

IncGM+ employs three novel techniques to improve the efficiency of incremental FSM. First, it prunes the search space by focusing on a set of carefully selected subgraphs (fringe subgraphs). These subgraphs represent the boundary between frequent and infrequent subgraphs. They are the most sensitive to updates. Any changes to the result shall affect the fringe first. After each graph update, these subgraphs are evaluated first followed by other elements of the search space if needed.

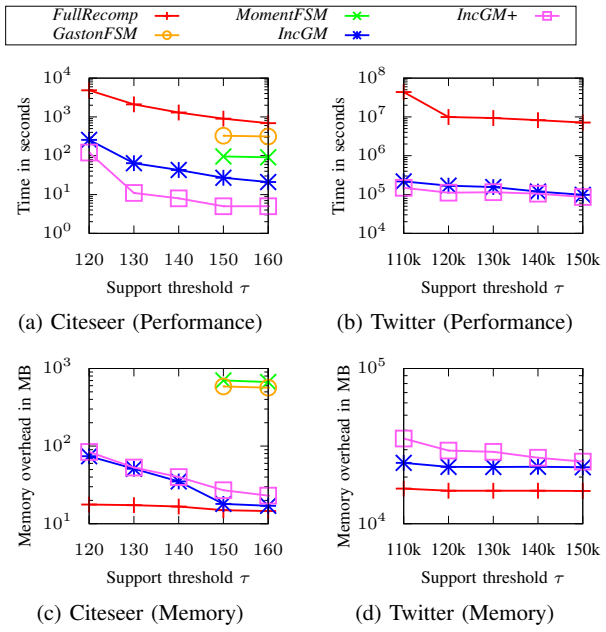
Second, *IncGM+* materializes a minimal number of embeddings for each fringe subgraph. Maintaining these embeddings minimizes the overhead of searching for them repeatedly from scratch after each graph update. Hence, significant optimization is achieved in evaluating fringe subgraphs.

Finally, *IncGM+* collects information from past iterations to improve the efficiency of future iterations. *IncGM+* uses this information to decide in which order it shall conduct evaluations. Based on past knowledge, *IncGM+* prioritizes fringe subgraph nodes that are expected to lead to faster evaluation over other nodes. Furthermore, input graph nodes that were previously evaluated and did not contribute to the result are postponed in favor of other graph nodes.

III. Batching

For practical applications with heavy workloads, batching can be used to speed-up processing. It allows expensive support computations to be aggregated for improved efficiency. Moreover, a simple yet effective technique can be applied for bypassing the evaluation for some candidate subgraphs. The aggregated candidate subgraphs have relationships among each other. A subgraph S_1 is a child of a subgraph S_2 if S_1 is infrequent and it is a supergraph of S_2. Also, a subgraph S_2 is a parent of S_1 if S_2 is a frequent subgraph and it is a subgraph of S_1. Based on these relationships and the anti-monotonicity requirement [5], there is no need to evaluate a frequent subgraph if one of its children is found to be frequent. Also, there is no need to evaluate an infrequent subgraph if one of its parents is found to be infrequent. Finally, evaluating

2375-026X/18/$31.00 ©2018 IEEE
DOI 10.1109/ICDE.2018.00241



(a) Citeseer (Performance) (b) Twitter (Performance)

(c) Citeseer (Memory) (d) Twitter (Memory)

Fig. 1. Efficiency and memory consumption evaluation

candidate subgraphs is prioritized such that maximum pruning is achieved (i.e., subgraphs that are predicted to change their status are evaluated before other subgraphs).

IV. Experimental Evaluation

This section compares the proposed incremental approaches; *IncGM* (the approach based on fringe pruning) and *IncGM+* (the extension of *IncGM* that maintains embeddings and utilizes ordering optimizations) against the competitors: *FullRecomp*, *MomentFSM* and *GastonFSM*, which extends *MomentFSM* and borrows ideas from the Gaston system [6] to efficiently maintain the list of embeddings. All experiments are conducted on a machine with 2.67GHz Intel Xeon processor and 192GB RAM. All systems are implemented in Java.

Two graphs are used in the experiments: 1- CiteSeer[1]: a citation network with 3.3K nodes, 4.7K edges and 6 distinct node labels, and 2- Twitter [2]: a social graph with 11.3M nodes and 85.3M edges. Nodes in Twitter are randomly labeled using 25 distinct labels. Random workloads of 2500 edge additions are applied on each graph to represent dynamic changes.

Figures 1.a and 1.b compare the efficiency of the different systems. Efficiency is measured as the time needed for graph loading and processing the whole workload. For Twitter, the results of *FullRecomp* are extrapolated since it cannot process the entire workload within a reasonable time. The results show that *IncGM+* is at least two orders of magnitude faster than *FullRecomp*. This improvement is the result of using the fringe subgraphs to prune the search space. Both *MomentFSM* and *GastonFSM* show good performance for smaller graphs and large τ values. But for larger graphs and lower τ, they both fail to complete the task since they consume the available memory to store an enormous number of embeddings.

Figures 1.c and 1.d show the memory consumption of each system. *FullRecomp* does not store intermediate results, thus its memory consumption is the least. *MomentFSM* and *GastonFSM* consume excessive memory to store intermediate embeddings. Although *IncGM+* uses a fringe and maintains a number of embeddings, its extra memory usage is insignificant compared to *IncGM*. This is due to our approach that carefully maintains a limited number of embeddings.

Figures 2 shows how batching improves the performance on the Twitter dataset. Three batching sizes are used; edge by edge update (Inc1), batch of 100 updates (Inc100) and batch of 500 updates (Inc500). Each bar in Figures 2 represents the overall processing time for each corresponding batch size. As shown in the figure, batching improves the overall performance as the batch size increases (e.g., Inc500 is an order of magnitude faster than Inc1).

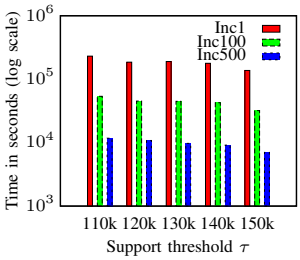


Fig. 2. Batching evaluation.

V. Conclusions

This paper presents a novel solution for mining frequent subgraphs from evolving graphs. This solution exploits some information collected during previous iterations to enhance the efficiency of future iterations. Such information includes which parts of the search space to maintain, which nodes of the input graph to postpone, and which subgraph nodes to prioritize in order to obtain quicker results. Furthermore, we discuss how batching can be utilized to further improve the performance. Our experiments show that *IncGM+* outperforms existing solutions and consumes significantly less memory.

[1] http://linqs.cs.umd.edu/projects/projects/lbc

[2] http://socialcomputing.asu.edu/datasets/Twitter

References

[1] M. Kuramochi and G. Karypis, "Finding frequent patterns in a large sparse graph," *Data Mining and Knowledge Discovery*, 2005.

[2] E. Abdelhamid, I. Abdelaziz, P. Kalnis, Z. Khayyat, and F. Jamour, "Scalemine: Scalable parallel frequent subgraph mining in a single large graph," in *High Performance Computing, Networking, Storage and Analysis, SC16: International Conference for*. IEEE, 2016.

[3] Y. Chi, H. Wang, P. S. Yu, and R. R. Muntz, "Moment: Maintaining closed frequent itemsets over a stream sliding window," in *Proc. of ICDM*, 2004.

[4] E. Abdelhamid, M. Canim, M. Sadoghi, B. Bhattacharjee, Y.-C. Chang, and P. Kalnis, "Incremental frequent subgraph mining on large evolving graphs," *IEEE Transactions on Knowledge and Data Engineering*, 2017.

[5] B. Bringmann and S. Nijssen, "What is frequent in a single graph?" in *Proc. of PAKDD*, 2008.

[6] S. Nijssen and J. N. Kok, "A quickstart in frequent structure mining can make a difference," in *Proc. of SIGKDD*, 2004.