Learning-Based Dynamic Graph Stream Sketch

Ding Li, Wenzhong Li(✉), Yizhou Chen, Mingkai Lin, and Sanglu Lu

State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210023, China
liding@smail.nju.edu.cn, lwz@nju.edu.cn

Abstract. A graph stream is a kind of dynamic graph representation that consists of a consecutive sequence of edges where each edge is represented by two endpoints and a weight. Graph stream is widely applied in many application scenarios to describe the relationships in social networks, communication networks, academic collaboration networks, etc. Graph sketch mechanisms were proposed to summarize large-scale graphs by compact data structures with hash functions to support fast queries in a graph stream. However, the existing graph sketches use fixed-size memory and inevitably suffer from dramatic performance drops after a massive number of edge updates. In this paper, we propose a novel Dynamic Graph Sketch (DGS) mechanism, which is able to adaptively extend graph sketch size to mitigate the performance degradation caused by memory overload. The proposed DGS mechanism incorporates deep neural network structures with graph sketch to actively detect the query errors, and dynamically expand the memory size and hash space of a graph sketch to keep the error below a pre-defined threshold. We conducted extensive experiments on three real-world graph stream datasets, which show that DGS outperforms the state-of-the-arts with regard to the accuracy of different kinds of graph queries.

Keywords: Sketch · Data stream · Graph stream

1 Introduction

A *graph stream* [4,13] is a consecutive sequence of items, where each item represents a graph edge. Each edge is usually denoted by a tuple consisting of two endpoints and a weight. Nowadays graph stream is ubiquitously applied to describe the relationships in social networks, communication networks, academic

This work was partially supported by the National Key R&D Program of China (Grant No. 2018YFB1004704), the National Natural Science Foundation of China (Grant Nos. 61972196, 61832008, 61832005), the Key R&D Program of Jiangsu Province, China (Grant No. BE2018116), the science and technology project from State Grid Corporation of China (Contract No. SGJSXT00XTJS2100049), and the Collaborative Innovation Center of Novel Software Technology and Industrialization.

K. Karlapalem et al. (Eds.): PAKDD 2021, LNAI 12712, pp. 383–394, 2021.
https://doi.org/10.1007/978-3-030-75762-5_31

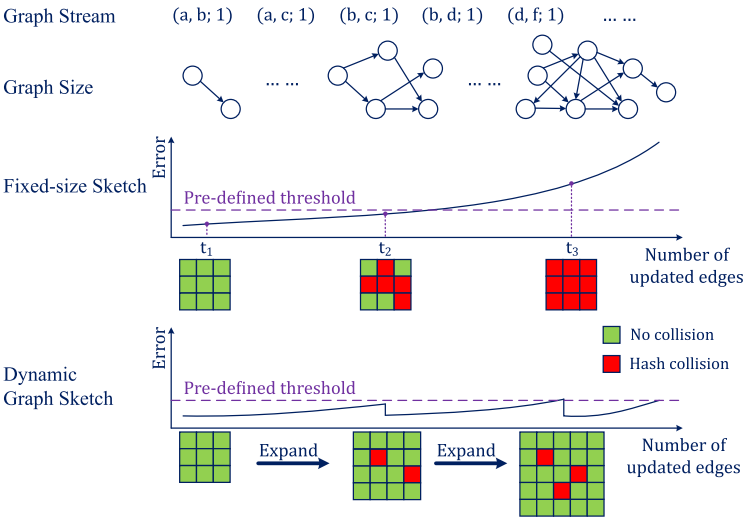


Fig. 1. This figure shows the change of query error with the increasing number of nodes and edge updates using fixed-size sketch and dynamic graph sketch, respectively.

collaboration networks, etc. For example, a graph stream can be used to describe temporal-spatial varying network traffics, or represent a dynamic social network with increasing number of users and their social interactions. It plays an important role since the graph topology is indispensable to many data analysis tasks in these applications. Apparently, traditional data structures such as adjacency matrix and the adjacency list cannot be directly adopted to store a graph stream due to its large volume and high dynamicity.

Sketch is a compact data structure for data stream summarization. Traditional sketches such as CM-sketch [2] and CU-sketch [3] were designed to summarize a stream of isolated data items, which are not suitable for graph stream summarization due to the lack of ability to capture the connections between items and to answer graph topology queries. To address the problem, Tang et al. proposed a *graph sketch* called TCM [13]. It summarized a graph stream by a matrix where each edge was mapped to a bucket of the matrix using a hash function, and the edge weight was recorded in the corresponding bucket. Since the graph sketch recorded not only the edge weight, but also the connections of a graph, it was able to answer graph topology queries such as reachability queries and subgraph queries. Several variants of graph sketch were proposed to improve the query accuracy and efficiency [4,8].

However, the major drawback of the existing graph sketches is that they construct a graph summarization based on a pre-defined fixed-size matrix which are not expandable. Many real-world applications are dealing with dynamic graph streams, i.e., the population (the nodes) of a social network may increase rapidly and the interaction between users (the edges) may change dynamically. Applying the fixed-size graph sketch on the dynamic graph streams inevitably suffers from poor accuracy of query results after a large number of edge updates. We

illustrate this problem in Fig. 1. The figure shows the change of the average relative error of edge queries with the number of edge updates in TCM. As can be seen, TCM suffers from serious performance degradation during the edge updating process. At the beginning, hash collisions are rare and the query error is relatively low. With more and more edge updates, hash collision occurs in every buckets of TCM and the memory is overloaded. This leads to the fact that the average relative error of edge queries becomes very high. Thus, it is desirable for a graph sketch to have the ability of incremental expansion to avoid rapid performance degradation and keep the relative error below a threshold.

In this paper, we propose a novel Dynamic Graph Sketch (DGS), which is able to adaptively extend its size and redistribute hash collisions to mitigate the performance degradation caused by overloaded memory. In contrast to the existing graph sketches that use fixed-size memory, DGS is able to keep the query accuracy always at a high level, no matter how many edges have been updated. Figure 1 presents an example how DGS works and how the performance of DGS changes with the number of edge updates. As shown in the figure, DGS periodically predict its current query error. If the error exceeds a pre-defined threshold, it will expand its size and simultaneously redistribute hash collisions to reduce the query error. Then, the expanded sketch continues to record the subsequent edges. In this way, the query error can be always limited under the pre-defined threshold.

The main results and contributions of this paper are summarized as follows:

- We propose Dynamic Graph Sketch (DGS), a novel mechanism for graph stream summarization. It is able to adaptively expand its size to mitigate the performance degradation during the edge updating process, keeping the query accuracy always at a high level. To the best of our knowledge, we are the first to propose the dynamic graph sketch which uses adaptively incremental memory.
- We integrate deep learning techniques into graph sketch design. Specifically, we introduce a deep neural network (DNN) to predict a graph sketch's query error, and based on which we design a convolutional neural network (CNN) to aid expanding a small-size graph sketch to a large-size one and simultaneously mitigate hash collisions to improve graph query accuracy.
- We conduct extensive experiments on three real-world graph streams to evaluate the effectiveness of our proposed algorithm. The experimental results show that DGS outperforms state-of-the-art graph sketches in terms of different kinds of graph queries.

2 Related Work

Data stream sketches are designed to summarize the data streams that are modeled as isolated items. Thus, they cannot answer graph topology queries. C-sketch [1] utilized multiple hash tables to store a data stream and was able to estimate the frequencies of all the items. However, it suffered from both overestimation and underestimation. Cormode et al. improved the work of [1] and

proposed CM-sketch [2] which only suffered from overestimation. Estan et al. designed CU-sketch [3] to improve CM-sketch's query accuracy at the cost of not supporting item deletions. More recently, Tang et al. proposed MV-sketch [12] which tracked candidate heavy items inside the sketch data structure via the idea of majority voting. Liu et al. proposed SF-sketch [9] which consisted of both a large sketch and a small sketch to upgrade the query accuracy. Besides designing novel data structures, some researchers proposed to utilize the power of machine learning to optimize the performance of sketch. Later, Hsu et al. [6] and Zhang et al. [14] proposed Learned Count-Min algorithm which combined the traditional CM-sketch with a heavy hitter oracle to improve efficiency.

In contrast to data stream sketches, graph sketches are able to summarize graph streams, keeping the topology of a graph and thus supporting more kinds of queries including 1-hop precursor/successor query, reachability query, etc. Zhao et al. designed gSketch [15], which utilized CM-sketch to support edge query and aggregate subgraph query for a graph stream. Tang et al. proposed TCM [13], which adopted an adjacency matrix to store the compressed graph stream. gMatrix [8] was similar to TCM, and it used reversible hash functions to generate graph sketches. More recently, Gou et al. proposed GSS [4], which was the state-of-the-art for graph stream summarization. To improve query accuracy, GSS consisted of not only an adjacency matrix, but also an adjacency list buffer, which can avoid edges collisions to some extent and improve the query accuracy.

In summary, all the existing sketches use fixed-size memory and thus suffer from serious performance degradation after a certain number of edge updates. To the best of our knowledge, the dynamic graph sketch expansion problem has not been well addressed in the past.

3 Preliminaries

In this section, we introduce some preliminaries about graph stream summarization.

Definition 1 (Graph stream). *A graph stream is a consecutive sequence of items* $S = \{e_1, e_2, \ldots, e_n\}$*, where each item* $e_i = (s, d, t, \omega)$ *denotes a directed edge from node* s *to node* d *arriving at timestamp* t *with weight* ω*. Note that an edge* e_i *can be updated multiple times at different timestamps.*

A graph stream S forms a dynamic directed graph $G = (V, E)$ that changes with every arrival of an edge update, where V denotes the node set and E denotes the edge set of the graph. Since an edge e_i may appear multiple times in the graph stream S, the weight of e_i is computed by an *aggregation function* based on all the edge weights that share the same endpoints. Common aggregation functions include $min(\cdot)$, $max(\cdot)$, $average(\cdot)$, $sum(\cdot)$, etc. In the rest of this paper, we adopt $sum(\cdot)$ as the default aggregation function to introduce our method.

Definition 2 (Graph sketch [13]). *A graph sketch is a graph* $K = (V_K, E_K)$ *whose size is smaller than the original graph* $G = (V, E)$ *formed by a given graph stream. Specifically,* $|V_K| \leq |V|$ *and* $|E_K| \leq |E|$*. A hash function* $H(\cdot)$ *is used to map each node in* V *to a node in* V_K*, and edge* (s, d) *in* E *is mapped to edge* $(H(s), H(d))$ *in* E_K*.*

The graph sketch is usually implemented by an adjacency matrix M, and each element $M[i][j]$ in the adjacency matrix is usually called a *counter*.

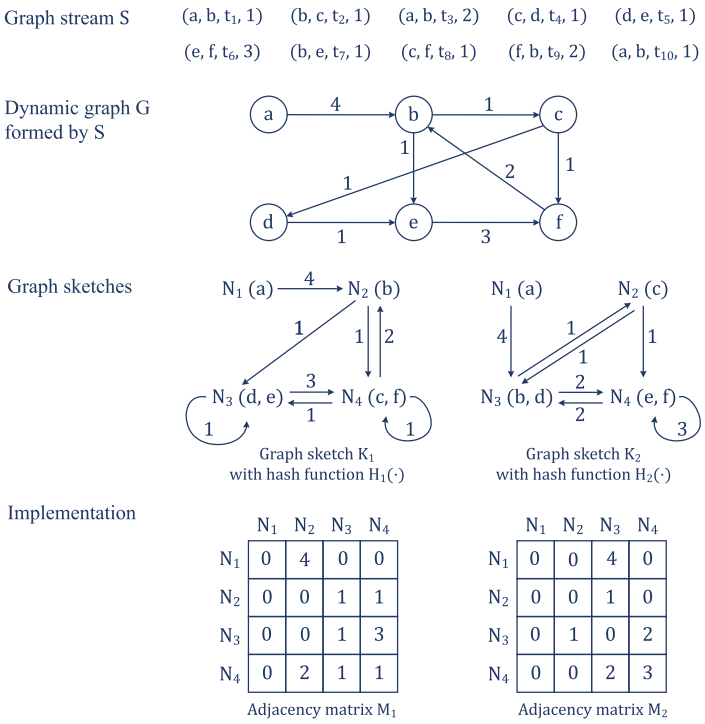


Fig. 2. An example of using two graph sketches to summarize a graph stream.

Usually, several graph sketches will be simultaneously used to record a graph stream to reduce the query error. The hash functions of these sketches are different and mutually independent. In order to better illustrate how to use a set of graph sketches to summarize a graph stream, we give an example which is presented in Fig. 2. As shown in the figure, two graph sketches with different hash functions are used to summarize graph stream S. For each edge in the graph stream, the graph sketches conduct an *edge update* as follows.

Edge Update: To record an edge $e_i = (s, d, t, \omega)$ of the graph stream, the graph sketch first calculates the hash values $(H(s), H(d))$. Then, it locates the corresponding position $M[H(s)][H(d)]$ in the adjacency matrix, and adds the

value in that position by ω. For example, to record the edge $(b, c, t_2, 1)$, graph sketch K_1 first calculates the hash values $(H_1(b), H_1(c)) = (N_2, N_4)$, and then the value in $M_1[N_2][N_4]$ is added by 1. Similarly, for graph sketch K_2, the value in $M_2[H_2(b)][H_2(c)]$ is added by 1.

Graph sketches usually support two basic queries: *edge query* and *node query*.

Edge Query: Given an edge e_i, edge query is to return the weight of e_i. To answer this query, we can first query the weight of the edge that e_i is mapped to in all the graph sketches, obtaining a set of weights $\{\omega_1, \omega_2, \ldots, \omega_m\}$. Then, we return the minimum of $\{\omega_1, \omega_2, \ldots, \omega_m\}$. For example, to query the weight of edge (b, e), we can first map it to edge (N_2, N_3) of graph sketch K_1, whose weight is 1. Similarly, we can query the weight of edge (N_3, N_4) of graph sketch K_2, which is 2. Finally, we return $min\{1, 2\}$ as the answer.

Node Query: Given a node n, node query is to return the aggregated edge weight *from* node n. To answer this query, for each adjacency matrix, we can first locate the row corresponding to node n, and then sum up the values in that row, obtaining a set of sums $\{sum_1, sum_2, \ldots, sum_m\}$. Then, we return the minimum of $\{sum_1, sum_2, \ldots, sum_m\}$. For example, to query the aggregated edge weight *from* node e, we can first locate the row N_3 of adjacency matrix M_1, and sum up the values in that row, which is $0 + 0 + 1 + 3 = 4$. Similarly, we can sum up the values in row N_4 of adjacency matrix M_2, which is $0 + 0 + 2 + 3 = 5$. Finally, we return $min\{4, 5\}$ as the answer.

An important application of this query is to find *top-k heavy nodes*, i.e. the top-k nodes with the highest aggregated weight. Together with the graph sketches, a *min-heap* is usually used to maintain the top-k nodes [13].

4 Learning-Based Dynamic Graph Sketch Mechanism

In this section, we propose a learning-based dynamic graph sketch mechanism called DGS to mitigate the performance problem of fixed-sized sketch. The proposed framework is illustrated in Fig. 3. During edge updates, DGS actively detects the query error of the graph sketch. If the error exceeds a pre-defined threshold, it adaptively expands the sketch size and hash space to reduce hash collisions and improve query accuracy. The framework consists of two major components: a *learning-based error detection* module and a *CNN-based sketch expansion* module, which are introduced in detail as follows.

4.1 Learning-Based Error Detection

Given a set of graph sketches formed after a certain number of edge updates, we periodically detect its current query error to decide whether to expand it. Since it is infeasible to calculate the precise error by exhausting all possible node queries and edge queries, we construct a learning-based prediction model to estimate the average relative error of edge queries.

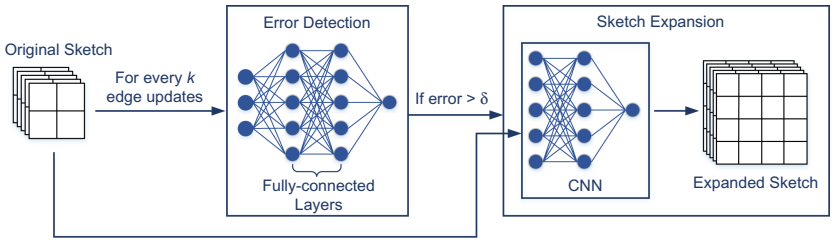


Fig. 3. The framework of dynamic graph sketch (DGS).

The proposed prediction model is illustrated in Fig. 3. It is a neural network consisting of two fully-connected layers. It takes a set of graph sketches as input, flattens all the counters of the graph sketches to a real-value vector, and feeds the vector into the neural network, which outputs the predicted relative error.

The error detection model can be trained based on historical data. For example, we can use a small percentage of the graph stream edge updates to form several graph sketches at different timestamps, and test the edge query errors of those graph sketches to form a labeled dataset, which can be used to train the prediction model.

4.2 CNN-Based Graph Sketch Expansion

Given a set of graph sketches whose current query error is detected to be higher than a pre-defined threshold, we introduce a DNN-based graph sketch expansion module to expand its size and reduce the hash collision and query error.

Convolutional Neural Network (CNN) Architecture. Inspired by the fact that CNN is expert in image super-resolution [5,11] and that images are usually represented by matrices, we design a deep convolutional neural network for graph sketch expansion. The basic idea is to design a CNN model to map low-resolution matrices (smaller-size graph sketches) to high-resolution ones (larger-size graph sketches) while keeping the similarities between them.

The structure of the proposed deep convolutional neural network is presented in Fig. 4. It includes three convolutional layers and a reshaping layer. A graph sketch with size $w \times w$ first goes through three convolutional layers, and the last convolutional layer outputs 4 matrices with size $w \times w$. To form a larger graph sketch, the reshaping layer reshapes these 4 matrices, obtaining a larger matrix with size $2w \times 2w$.

Similar to the training of the error prediction model in Sect. 4.1, this network can be trained using historical data as well. We use a small percentage of edge updates to form several small-size sketches and corresponding large-size sketches at different timestamps. A small-size sketch together with its corresponding large-size sketch at the same timestamp can be used as one training sample to train the network.

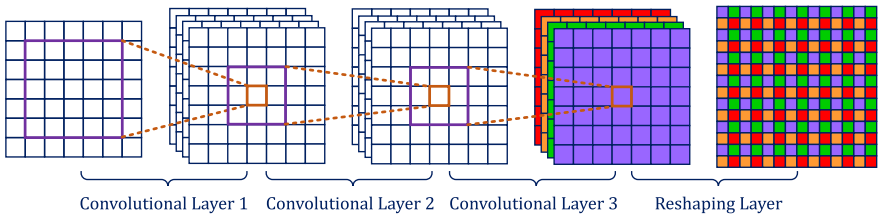


Fig. 4. The proposed CNN structure for graph sketch expansion.

Reshape of Counter Values. After graph sketch expansion, we can obtain a set of large-size graph sketches. However, although the expanded graph sketches have the similar shape as the original ones, their weights (counters) maybe not conform with the definition of graph sketch. To better illustrate this problem, we give an example illustrated in Fig. 5. The original graph sketch is a 2×2 matrix with hash function $H(x) = x \ mod \ 2$. After expansion, the matrix size is 4×4 with hash function $H'(x) = x \ mod \ 4$ (the hash space is expanded both horizontally and vertically). Therefore an edge originally mapped to cell (0, 0) can be now mapped to four cells (0, 0), (0, 2), (2, 0), (2, 2) in the expanded graph sketch. The sum of the weights in the four cells are 20 + 30 + 30 + 40 = 120 which is unequal to the original weight 12. According to the definition of graph sketch, the weights of the matrix should be the counter of the graph stream edges, so we need to reshape the weight of the expanded graph sketch with a normalization trick.

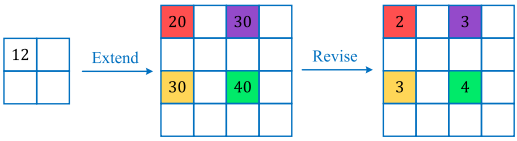


Fig. 5. Reshape of counter values.

Generally speaking, denoted by $K^{original}$ the original graph sketch of size $w \times w$ and $K^{expanded}$ the expanded graph sketch of size $2w \times 2w$, we adopt the following normalized formula to reshape the counter values in the expanded graph sketch:

$$K^{expanded}_{i,j} \leftarrow K^{original}_{H(i),H(j)} \times \frac{K^{expanded}_{i,j}}{sum} \tag{1}$$

where sum represents sum of the four counters expanded from the original one which can be calculated by:

$$sum = K^{expanded}_{H(i),H(j)} + K^{expanded}_{H(i),H(j)+w} + K^{expanded}_{H(i)+w,H(j)} + K^{expanded}_{H(i)+w,H(j)+w} \tag{2}$$

5 Performance Evaluation

5.1 Experimental Environment

We conduct extensive experiments to validate the effectiveness of the proposed dynamic graph sketch (DGS). We compare our method with two state-of-the-art graph sketches: TCM [13] and GSS [4]. All experiments were performed on a desktop with Intel Core i7-7700 processors (4 cores, 8 threads), 8 GB of memory, and NVIDIA GeForce GTX 1050 GPU. All sketches except GSS were implemented in Java. For GSS, we used the C++ source code provided on the Github[1]. For fair comparison, we disabled the buffer list of GSS since it does not limit the memory usage. We used the PyTorch library [10] to implement our proposed learning-based prediction and expansion model.

The experiments are based on three real-world graph stream datasets.

- **lkml-reply**[2]**:** The first dataset is a collection of communication records in the network of the Linux kernel mailing list. It contains 63,399 email addresses (nodes) and 1,096,440 communication records (edges).
- **prosper-loans**[3]**:** The second dataset is loans between members of the peer-to-peer lending network at Prosper.com. Nodes represent members; edges represent loans and are directed from lenders to borrowers. The dataset contains 89,269 nodes and 3,394,979 edges.
- **facebook-wosn-wall**[4]**:** The third data set is the directed network of a small subset of posts to other user's wall on Facebook. The nodes of the network are Facebook users, and each directed edge represents one post, linking the users writing a post to the users whose wall the post is written on. The dataset contains 46,952 nodes and 876,993 edges.

We adopt the following performance metrics in our experiments.

- **Average relative error (ARE):** measures the accuracy of the reported weights in edge queries. Given a query q, the *relative error* $RE(q)$ is defined as: $RE(q) = \frac{|\hat{f}(q) - f(q)|}{f(q)}$ where $\hat{f}(q)$ denotes the estimated answer of query q, and $f(q)$ denotes the real answer of query q. Given a set of queries $\mathcal{Q} = \{q_1, q_2, \ldots, q_n\}$, the ARE is calculated by $ARE(\mathcal{Q}) = \frac{\sum_{i=1}^{n} RE(q_i)}{n}$.
- **Intersection accuracy (IA)** [13]**:** measures the accuracy of the top-k heavy nodes reported by a sketch. Let X be the set of reported top-k heavy nodes, and Y be the set of ground truth top-k heavy nodes. The IA is formulated as $IA = \frac{|X \cap Y|}{k}$.
- **Normalized discounted cumulative gain (NDCG)** [7]**:** measures the quality of a ranking. Given a ranking list of heavy nodes reported by a sketch, *discounted cumulative gain* ($DCG@k$) is defined as $DCG@k =$

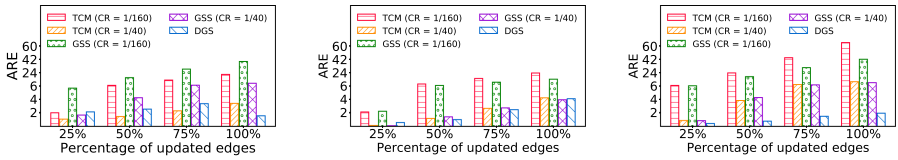
[1] https://github.com/Puppy95/Graph-Stream-Sketch.
[2] http://konect.cc/networks/lkml-reply.
[3] http://konect.cc/networks/prosper-loans.
[4] http://konect.cc/networks/facebook-wosn-wall.

$\sum_{i=1}^{k} \frac{r(i)}{log_2(i+1)}$, where $r(i)$ denotes the relative score of the ith node in the ranking list and it belongs to $\{0, 1\}$. If the ith node in the ranking list is indeed a heavy node, $r(i)$ will be 1; otherwise, it will be 0. Using the definition above, $NDCG@k$ is formulated as $NDCG@k = \frac{DCG@k}{IDCG@k}$, where $IDCG@k$ represents the $DCG@k$ of an ideal ranking list obtained by sorting the nodes in the ranking list in descending order with respect to their relative scores. Thus, NDCG ranges in $[0, 1]$, which can be used to evaluate the ability to find top-k heavy nodes of graph sketches (the higher the better).

5.2 Numerical Results

We analyze the performance for edge query and node query of different sketches.

Edge Query. Figure 6 shows the change of edge query's ARE in TCM, GSS, and our proposed DGS. As can be seen, with the increase of updated edges, the ARE of edge queries in fixed-size sketches (TCM and GSS) grows drastically. For high compression ratio (compression ratio = 1/160) in TCM and GSS, the ARE of edge queries in dataset *lkml-reply* is 2.032 and 5.691 after 25% edges updates, and it grows to 14.022 and 28.714 after 75% edge updates, respectively. The performance of TCM (compression ratio = 1/40) and GSS (compression ratio = 1/40) performs better, but they also suffer from high ARE from 75% to 100% edge updates. In contrast, the ARE of the proposed DGS is not sensitive to the number of edge updates, and it outperforms the other algorithms significantly when the percentage of edge updates is larger than 50%. It verifies the effectiveness of variable-size graph sketch for edge query.



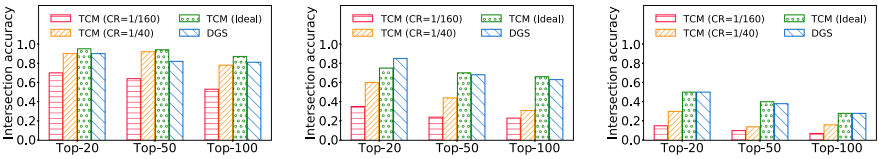
(a) lkml-reply (b) prosper-loans (c) facebook-wosn-wall

Fig. 6. The ARE of edge queries.

Node Query. We evaluate the ability to find top-k heavy nodes of DGS as well as TCM. We do not conduct this experiment on GSS since GSS does not support heavy node query. The results are shown in Fig. 7 and Fig. 8. As shown in Fig. 7, when the size of TCM is small, i.e., with compression ratio 1/160, the intersection accuracy falls down to 24% in some cases. For larger sketch size, i.e., TCM(1/40), the performance improved. In the figures, TCM(ideal) means we set a sufficient large memory size for TCM, which is as large as the final size of DGS. Although

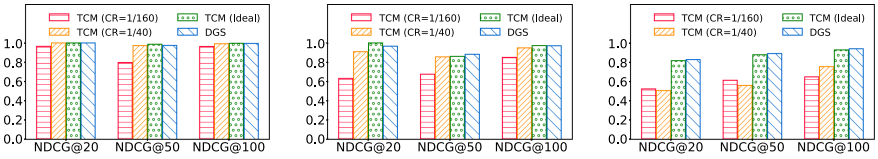
TCM(ideal) achieves the best accuracy in finding top-k nodes, we emphasize that it is hard to set a proper size for TCM at initialization since the size of the incoming graph stream is unknown. As shown in Fig. 7, DGS achieves an intersection accuracy of 85%, 85%, and 80% in the task of finding top-20 heavy nodes on data set *lkml-reply*, *prosper-loans*, and *facebook-wosn-wall* respectively, which outperforms other situations significantly, and it performs very close to the TCM(ideal) situation. This illustrates that our proposed dynamic graph sketch expansion algorithm is effective and able to keep the performance always at a high level.

We also calculate the NDCG based on the result list of top-k heavy node query. The results are shown in Fig. 8. Similarly, the NDCG is worse for pre-defined fixed-size TCM(1/160) and TCM(1/40). Again, DGS outperforms other situations and it performs very close to TCM(ideal).



(a) lkml-reply (b) prosper-loans (c) facebook-wosn-wall

Fig. 7. Heavy node query (intersection accuracy)



(a) lkml-reply (b) prosper-loans (c) facebook-wosn-wall

Fig. 8. Heavy node query (NDCG)

6 Conclusion

In this paper, we proposed *Dynamic Graph Sketch (DGS)*, a novel framework for large-scale graph stream summarization. Unlike conventional graph sketches that used fixed-sized memory, DGS adopted an expandable-size design to avoid performance drop after a large number of edge updates. DGS introduced a deep neural network to actively predict the query error of the graph sketch. If the predicted error exceeded a pre-defined threshold, it used a convolutional neural

network to learn to expand its memory size and hash space. Extensive experiments based on three real-world graph streams showed that the proposed method is able to achieve high accuracy for different kinds of graph queries compared to the state-of-the-arts.

References

1. Charikar, M., Chen, K., Farach-Colton, M.: Finding frequent items in data streams. In: Widmayer, P., Eidenbenz, S., Triguero, F., Morales, R., Conejo, R., Hennessy, M. (eds.) ICALP 2002. LNCS, vol. 2380, pp. 693–703. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-45465-9_59
2. Cormode, G., Muthukrishnan, S.: An improved data stream summary: the count-min sketch and its applications. J. Algorithms **55**(1), 58–75 (2005)
3. Estan, C., Varghese, G.: New directions in traffic measurement and accounting: focusing on the elephants, ignoring the mice. ACM Trans. Comput. Syst. **21**(3), 270–313 (2003)
4. Gou, X., Zou, L., Zhao, C., Yang, T.: Fast and accurate graph stream summarization. In: 35th IEEE International Conference on Data Engineering (ICDE 2019), pp. 1118–1129 (2019)
5. Guo, Y., et al.: Closed-loop matters: dual regression networks for single image super-resolution. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 5406–5415 (2020)
6. Hsu, C., Indyk, P., Katabi, D., Vakilian, A.: Learning-based frequency estimation algorithms. In: 7th International Conference on Learning Representations (ICLR 2019) (2019)
7. Järvelin, K., Kekäläinen, J.: Cumulated gain-based evaluation of IR techniques. ACM Trans. Inf. Syst. **20**(4), 422–446 (2002)
8. Khan, A., Aggarwal, C.C.: Query-friendly compression of graph streams. In: Kumar, R., Caverlee, J., Tong, H. (eds.) 2016 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM 2016), pp. 130–137 (2016)
9. Liu, L., et al.: Sf-sketch: a two-stage sketch for data streams. IEEE Trans. Parallel Distrib. Syst. **31**(10), 2263–2276 (2020)
10. Paszke, A., et al.: Pytorch: an imperative style, high-performance deep learning library. In: Annual Conference on Neural Information Processing Systems (NeurIPS 2019), pp. 8024–8035 (2019)
11. Shi, W., et al.: Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 1874–1883 (2016)
12. Tang, L., Huang, Q., Lee, P.P.C.: Mv-sketch: a fast and compact invertible sketch for heavy flow detection in network data streams. In: IEEE Conference on Computer Communications (INFOCOM 2019), pp. 2026–2034 (2019)
13. Tang, N., Chen, Q., Mitra, P.: Graph stream summarization: from big bang to big crunch. In: Proceedings of the 2016 International Conference on Management of Data (SIGMOD 2016), pp. 1481–1496 (2016)
14. Zhang, M., Wang, H., Li, J., Gao, H.: Learned sketches for frequency estimation. Inf. Sci. **507**, 365–385 (2020)
15. Zhao, P., Aggarwal, C.C., Wang, M.: gSketch: on query estimation in graph streams. Proc. VLDB Endow. **5**(3), 193–204 (2011)