LessMine: Reducing Sample Space and Data Access for Dense Pattern Mining

Tianyu Fu, Ziqian Wan, Guohao Dai, Yu Wang, Huazhong Yang
Department of Electronic Engineering, BNRist, Tsinghua University, Beijing, China
{futy18, wanzq16}@mails.tsinghua.edu.cn, {daiguohao, yu-wang, yanghz}@tsinghua.edu.cn

***Abstract*—In the era of "big data", graph has been proven to be one of the most important reflections of real-world problems. To refine the core properties of large-scale graphs, dense pattern mining plays a significant role. Because of the complexity of pattern mining problems, conventional implementations often lack scalability, consuming much time and memory space. Previous work (e.g., ASAP [1]) proposed approximate pattern mining as an efficient way to extract structural information from graphs. It demonstrates dramatic performance improvement by up to two orders of magnitude. However, we observe three main flaws of ASAP in cases of dense patterns, thus we propose LessMine, which reduces the sample space and data access for dense pattern mining. We introduce the reorganization of data structure, the method of concurrent sample, and uniform close. We also provide locality-aware partition for distributed settings. The evaluation shows that our design achieves up to 1829× speedup with 66% less error rate compared with ASAP.**

***Index Terms*—graph pattern mining, dense pattern mining**

I. Introduction

In modern society, graph is a widely used data structure for its intuitive and flexible reflection of real-world problems. Graph pattern mining is a heated field of graph data processing. Several algorithms and systems are proposed, such as Arabesque [2], RStream [3], AutoMine [4], etc [5]–[7]. They reveal the features of large-scale graphs by counting the number of certain patterns (e.g., triangles, four cliques, etc.) in them. Pattern mining has many important applications, such as drug finding [8], financial risk management [9], and accounting fraud detection [10]. Dense patterns often play more important roles because they often reflect core and cohesive relationships in many graph applications. For example, in the field of modern biology, dense patterns in protein association network suggest functionally associated proteins [11], [12]. In social networks, small, densely connected groups often represent highly specific co-interests [13], [14].

However, with the growing scale of graph data, many graph pattern mining algorithms fail to handle the problem within tolerable time and memory consumption. For example, a state of the art system called Arabesque [2] spends over 10 hours to count motifs in a 1-billion-edge graph. Luckily, it is often unnecessary to output the exact number of patterns in many real-world applications. As a result, approximate pattern mining algorithm was proposed recently and has become a wise solution to the contradiction between complexity and scalability.

ASAP proves to be the most typical and successful approximate pattern mining algorithm for our knowledge. It outperforms Arabesque with up to 77 × speedup under 5% loss of accuracy. It will have more astonishing performance if the user is able to afford larger errors (e.g., 10%). For certain tasks, ASAP even demonstrates $258\times$ improvements and is able to analyze extremely large graphs with billions of edges. However, we still observe three main defects of ASAP, which are discussed in III .

In this work, we follow the idea of approximate pattern mining and make four main contributions:

- **We orient, sort, and index the edge stream to provide fast neighbor searching with less memory consumption.** We sort the raw edge stream and store the neighboring information of each node with a dictionary. It provides higher data locality and faster neighbor searching. Meanwhile, the dictionary only uses smaller nodes to find larger ones in terms of the index value, so compared with the standard adjacency table, we consume half of the memory with no performance compromise.
- **We propose concurrent sample to narrow the sample space.** By fixing all the nodes of the embedding concurrently, we not only simplify the proposal phase but also abstract the problem to a smaller sample space, reducing the number of estimators needed by two orders of magnitude in the task of 4-clique mining.
- **We propose uniform close to speed up the closing stage.** In addition to concurrent sample, we replace conditional close with uniform close which doesn't rely on the order of the edge stream. Meanwhile, it scales down the search space of edges to a subset of conditional close.
- **We propose locality-aware partition to apply to distributed settings without bringing additional accuracy loss.** With the observation of the unbalanced degree of graphs, we design a locality-aware partition that separates the heavy nodes with light ones. Compared with random partition, it avoids edge cutting and data exchange among partitions.

II. Background and Related Work

The pattern that has at least one node connected to all other nodes is called a dense pattern (denote by DP). Some examples of dense pattern are shown in Fig. 10. Dense pattern mining tasks include triangle counting, clique mining, etc. Conventional pattern mining algorithms often suffer from massive intermediate memory consumption and poor timing performance due to the complexity of the problem. ASAP proposes the method of approximate pattern mining based on edge sampling and embedding closing. ASAP organizes undirected edges into a stream and uses the subroutine called "estimator" to calculate the number of pattern G. The workflow of every estimator can be roughly divided into two stages: the proposal stage and the closing stage.

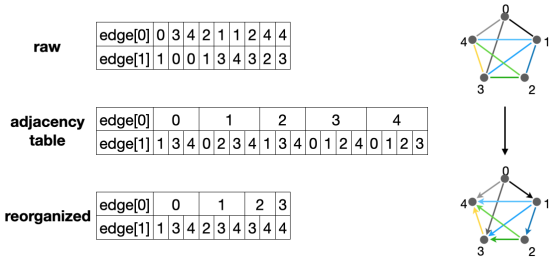

Fig. 1. Data Strcture of Edges.

In the proposal stage, ASAP first randomly samples an edge $e_0 = [n_0, n_1]$ from edge stream to form the initial embedding containing two nodes and one edge. The edge stream that appears before e_0 is ignored in the following process of this estimator to ensure a bijection between embeddings and the pattern. To mine a pattern with k nodes, ASAP then execute k-2 times of conditional samples. In each conditional sample, it searches the remaining edge stream to find all the adjacent edges of current embedding and randomly samples one to add one new edge e_i and one new node n_{i+1} to the embedding. After each conditional sample, it deletes the edge stream before e_i. The proposal stage ends with an embedding containing k nodes and k-1 edges. The probability of proposing such embedding P_i is also calculated using conditional probability. If the edge stream is reduced to none before the embedding is formed, the estimator early returns with the value $X_i = 0$; otherwise, it enters the closing stage. ASAP executes conditional close, which tries to complete the pattern by adding edges to the embedding from the remaining edge stream. If succeeded, $X_i = 1/P_i$; otherwise, $X_i = 0$. Let p^* be the pattern that the embedding maps to, and $p(G)$ be the set of the given pattern in the graph, ASAP proves that

$$E[X_i] = \sum_{p \in p(G)} P_i X_i = \sum_{p \in p(G)} P_i \cdot \frac{1}{P_i} = G \qquad (1)$$

We also illustrate the proof of (1) by showing an equivalent cube-picking experiment in Section III.B .

III. Motivation

Though ASAP demonstrates outstanding performance, we reveal three main flaws of ASAP which may lead to performance loss in this section.

A. Data Structure

ASAP stores all the edges following the order of the raw edge stream. It causes two main problems. First, it has to iterate through all the remaining edge stream in every estimator to determine the candidate edges of the conditional sample step and the search space of the conditional close step, which poses massive overhead. Second, the adjacent nodes are stored separately so that it lacks data locality, making the edge access and partition more time-consuming.

B. Sample Space Abstraction

We manage to transfer the process of approximate graph pattern mining into an equivalent "cube-picking experiment" so that one can get a more straightforward and intuitive understanding of the problem. In the experiment, you have a black box containing different sizes of red and white cubes

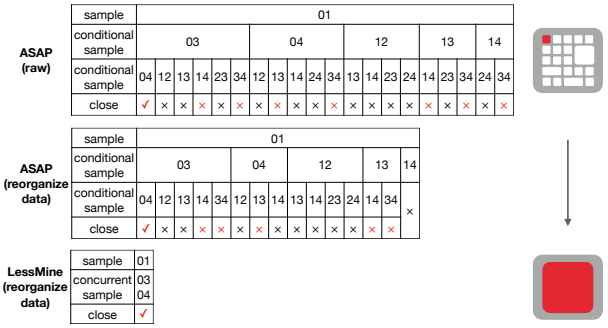

Fig. 2. Part of Sample Space of 4-clique.

like the right upper one in Fig. 2. The volume of the box is 1. The probability P_i of picking up one specific cube is fixed, and equals to its size. Let G be the number of red cubes in the box. To estimate G, you explore the black box by randomly picking one cube at a time with replacement; if it is red, let $X_i = 1/P_i$; if it is white, let $X_i = 0$. If the cubes share the same size, $E[X_i]$ means the number of all cubes $N = 1/P_i$ times the proportion of red ones, which equals to G. In fact, using the same idea as in (1), the result $E[X_i] = G$ still holds if the cubes have different sizes. Here, cubes represent proposed embeddings, and red color means successful conditional close. Naturally, we desire fewer cubes because we can only pick up one cube at a time to explore the box. Since the number of red cubes (the pattern) is fixed, the real trick lies in how to define fewer white ones.

However, ASAP proposes unnecessary white cubes. For ASAP, the search space of the closing stage doesn't include the edges before the last sample. As is shown in Fig.2, if we execute 4-clique mining on the graph of Fig.1 and sample the edge 0,1 for the first edge, ASAP can give 20 different proposals. However, 12 of them can't form 4-cliques (black cross mark), 7 of them find the pattern but not in the acceptable order of ASAP's closing phase, only 1 of them successfully finds the 4-clique (red check mark). The evaluation in V.A also shows that over 98% estimators of ASAP return the value 0.

C. Lack of Graph Feature Analysis

Let r be the estimator needed to get an ϵ - approximation to the count of k-clique, with probability at least $1-\delta$. According to ASAP's theoretical error-latency profile

$$r \geq \frac{1}{P_{min}} \frac{3\ln(\frac{2}{\delta})}{G\epsilon^2} \qquad (2)$$

$$= m(k-1)! \cdot \Delta^{k-2} \cdot \frac{3\ln(\frac{2}{\delta})}{G\epsilon^2} \qquad (3)$$

P_{min} is the smallest non-zero value that estimators can possibly return, m is the number of edges of the graph, G is the exact number of k-clique in the graph, and Δ is the maximum degree of the graph. Note that most graphs in real world follow the Power-Law [15], which means that they're often unbalanced in terms of degree distribution. For example, a graph with the average degree of 32.4 can have a maximum degree of 1469 [16]. However, ASAP takes no effort in scaling down Δ, resulting in unnecessarily large r. The influence of Δ can also be intuitively observed using the cube analogy mentioned before. The k-clique requires one sample

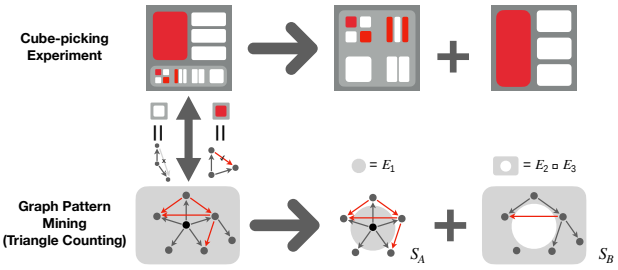

Fig. 3. Graph Feature Analysis. Embeddings with heavy node(s) are like small cubes in the box. Locality-aware partition separates the heaviest node with light ones.

and $k-2$ times of conditional sample, according to conditional probability and (2)(3), the minimum probability of proposing a specific embedding would be $P_{min} = \frac{1}{m(k-1)!^{k-2}}$. As the size of the cube equals P, $\frac{1}{m^{k-2}}$ defines how small the smallest cube can be. It is obvious that we do not like tiny cubes in the box. Because when we explore the box by sampling, tiny cubes are unlikely to be sampled, thus stay unknown. However, little space of tiny cubes can influence the number of red cubes greatly. This also leads to the idea that we should scale down Δ or m to get a better estimation.

IV. LessMine Algorithm

In this section, we'll introduce LessMine and explain how it works on the task of k-clique mining. Later we'll extend it to other dense patterns in VI .

A. LessMine Overview

The workflow of LessMine is shown in Fig. 4 . It first reorganizes the data of the graph as the input of the following steps. Then it may perform locality-aware partition or not based on user's settings. If the partition is done, the pattern numbers of different parts of the graph are counted respectively and added together as described in V.E . If the partition is not required, LessMine uses multiple parallel estimators (3 in Fig.4) to count the number of the given pattern on reorganized data. Each estimator uses sample, concurrent sample and uniform close to give X_i. Estimators may execute an early return without doing uniform close like the second estimator in Fig.4. LessMine outputs the average value of X_i as the number of the pattern in the graph.

B. Reorganize Data Structure

To lessen the access of the whole edge stream and eliminate non-injective mapping from embeddings to patterns, we reorganize the data structure. For an undirected graph with m edges, consider its raw edge stream as a $2 \times m$ matrix, each column represents an edge, and each element represents a node index. The node index is randomly assigned, so the raw matrix is often unordered. By doing compare and switch iterations through columns and rows, we make sure that for every edge e_i, node $e_i[0] < e_i[1]$ and that the edge stream is sorted in ascending order. Then we make it into a dictionary that uses $e_i[0]$ as keys (Fig. 1). Compared with the raw matrix, it has better data locality and provides faster indexing.

By doing so, the neighboring information of an undirected graph is stored in a directed manner, so it only takes half of the space compared to the standard adjacency table. It also scales down the maximum out-degree of nodes, which lessens the estimator needed according to (3)(5). The example in Fig. 2 also shows how it helps to scale down the sample space of ASAP.

Note that we want to construct a bijection between embeddings and patterns. ASAP relays on the order of the raw edge stream to eliminate non-injective mapping, so it has to iterate through the remaining stream repetitively. The directed data manner of ours ensures that each embedding "grows" only from smaller index nodes to larger ones, thus natively an injective function from embeddings to patterns.

C. Concurrent Sample

As introduced in III.B , ASAP proposes embeddings in a large sample space, which is not effective for dense patterns, so we introduce the method of concurrent sample. We will first introduce the method and then proves the correctness of the algorithm. Finally, in V.A , we will compare LessMine with ASAP to show the efficiency improvement of it.

Concurrent sample works as the replacement of $k-2$ times of conditional samples of ASAP. If we're counting the number of k-cliques in the graph using LessMine, in every estimators, it first uniformly samples an edge $e = [e[0], e[1]]$ from a m-edge graph, just like ASAP. Because of the data reorganization, we know that node $e[0] < e[1]$. Then, it goes into the process of concurrent sample. It proposes C candidate nodes $\{n_0, n_1, \ldots, n_{C-1}\}$ that share the key $e[0]$ and have greater index value than $e[1]$. After that, it compares $C+2$ with threshold value k: if $C+2 < k$, this suggests that there are not enough nodes to form the pattern for this estimator, so it early returns with value 0; if $C+2 \geq k$, it ramdomly chooses $k-2$ nodes from $[n_0, n_1, \ldots, n_{C-1}]$. Together with $e[0], e[1]$, it fixes all of the k nodes and enters the closing stage with the reciprocal probability $1/P_i = m \times \binom{C}{k-2}$. Compared with $k-2$ times of conditional sample for k-clique, concurrent sample only needs to be executed 1 time every estimator. It also helps to scale down the sample space of the proposal stage as shown in Fig. 2.

Now we're going to prove that concurrent sample establishes a bijection between the embedding and k-clique. We name the node that connects to all the other nodes in the pattern a densely connected node (denote by DCN). For dense patterns, if the smallest node is a DCN, then it is the first kind of dense pattern (denote by DP1); otherwise, it is the second kind of dense pattern (denote by DP2). Examples of DP are shown in Fig. 10. Equation (4) shows the adjacency matrix of DP1.

For any DP1(e.g. clique) in the graph, we sort its nodes in ascending order: $\{n_0, n_1, \ldots, n_{k-1}\}$. According to the definition, n_0 is connected to $n_1, n_2, n_3, \ldots, n_{k-1}$, so the embedding can always be proposed by first sampling $[n_0, n_1]$ and then concurrent sampling $\{n_2, \ldots, n_{k-1}\}$ from adjacent nodes of n_0. The process is also exclusive because the nodes fixed by concurrent sample are larger than those fixed by the first sampled edge. So the mapping between the embedding and DP1, k-clique included, is bijective.

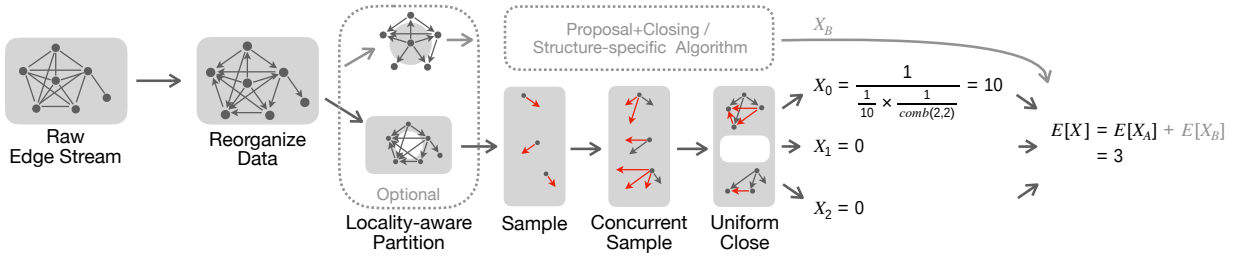

Fig. 4. The overall workflow of LessMine. Reorganize data: organizing edges in the graph. Locality-aware partition(optional): separating the heavy nodes with light ones. Sample/Concurrent sample/Uniform close: constructing the pattern based on sampling and closing operations.

It is also interesting to point out that if we also reorganize data for ASAP, concurrent sample automatically meets the sample order constrain of ASAP's conditional sample, but it doesn't require accessing the whole edge stream every time to get the order information. Meanwhile, it eliminates many missed embeddings and fully preserves the hit ones, as is shown in Fig. 2 .

D. Uniform Close

ASAP uses conditional close to examine whether an embedding from the proposal stage maps to the pattern we are looking for, which needs to iterate through the remaining edge stream with the worst complexity of $O(m)$. The iteration is essential for ASAP, because it needs the proposal and closing stages to work together to guarantee the injective mapping from the pattern set to the embedding set, otherwise there will be more than one embedding mapping to the same pattern.

As is proven in the last section, LessMine uses concurrent sample, which already guarantees the bijection between the embedding set and the pattern set. So we introduce uniform close to speed up the process.

For a k-clique whose nodes $\{n_0, n_1, \ldots, n_{k-1}\}$ are fixed in proposal stage, assume the index $n_0 < n_1 < \cdots < n_{k-1}$, we know from the proposal stage that n_0 is adjacent to $n_1, n_2, \ldots, n_{k-1}$, so we only need to check the connectivity between other nodes to give the adjacency matrix of the embedding:

$$\{a(n_i,n_j)\} = \begin{bmatrix} 0 & 1 & 1 & \cdots & 1 \\ \times & 0 & a(n_1,n_2) & \cdots & a(n_1,n_{k-1}) \\ \times & \times & 0 & \cdots & a(n_2,n_{k-1}) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \times & \times & \times & \cdots & 0 \end{bmatrix} \quad (4)$$

Using the reorganized data, $a(n_i, n_j)$ means whether we can find n_j using the key n_i, whose worst complexity is $O(klog(\Delta))$. If the bool value above the diagonal are all 1, the estimator returns the reciprocal for the probability $1/P$ given by concurrent sample; otherwise, it returns 0.

We should also point out that a concurrent sample guarantees that uniform close automatically meets the requirement

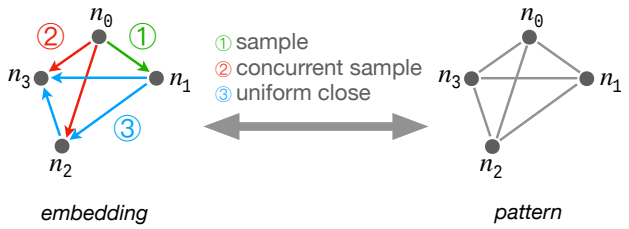

Fig. 5. LessMine guarantees the bijection between embedding and pattern.

of conditional close. Because in reorganized data structure, for any $i, j > 0$, edge $[n_i, n_j]$ appears after $[n_0, n_j]$. So all the edges in the search space of the uniform close appear after the last sampled edge in proposal stage. This suggests that the search space of uniform close is a subset of that of conditional close, so it is usually faster.

E. Locality-aware Partition

In this section, we start by analyzing how graph features influence the number of estimators. Then we'll propose a locality-aware partition in replacement of random partition in distributed settings and explain why it helps to scale down the number of estimators with no accuracy loss.

ASAP and LessMine both construct a bijection between the embedding set and the pattern set. The error profile is the analysis of return values of all the estimators, so it has the similar form and proving process for two algorithms. Equation (5) can be proven using Chernoff bound, but with the analysis, we can directly substitude P_{min} in (2). With concurrent sample of LessMine, $1/P_{min} = m \cdot$ $_{k-2}$, so the estimator needed to the count of k-clique is:

$$\begin{aligned} r &\geq m \cdot \frac{\Delta!}{(k-2)!(\Delta-k+2)!} \cdot \frac{3\ln(\frac{2}{})}{G\epsilon^2} \\ &\approx \frac{m}{(k-2)!} \cdot \Delta^{k-2} \cdot \frac{3\ln(\frac{2}{})}{G\epsilon^2} \end{aligned} \quad (5)$$

Δ here is the maximum out-degree of the equivalent directed graph because LessMine uses the reordered data structure.

Conventional random partition cuts the edges between different parts, leading to inevitable accuracy loss. Or it may require frequent data transactions between parts to avoid the cut. Besides, it also pays no attention to scaling down the influence of Δ. Our idea is to separate the parts without damaging the accuracy. Meanwhile, we try to decrease the influence of Δ .

The locality-aware partition first finds the node n_p that has the greatest out-degree Δ. We change the index of n_p to -1 so the in-degree of n_p adds to the out-degree to make n_p the smallest node with the largest out-degree Δ'. The adjacent nodes of n_p forms the set $N = \{n_0, n_1 \ldots, n_{\ 0-1}\}$, whose elements share the key n_p in our data structure, so they can be found with little cost. Then the whole edge stream E is divided into 3 disjoint parts:

$$\begin{cases} E_1 = \{[n_p, n_j] \mid n_j \in N\} \\ E_2 = \{[n_i, n_j] \mid n_i, n_j \in N\} \\ E_3 = \{[n_i, n_j] \mid n_j \notin N\} = E \setminus (E_1 \cup E_2) \end{cases} \quad (6)$$

We make the set $S_A = E_1 \cup E_2$ the first partition, $S_B = E_2 \cup E_3 = E \setminus E_1$ the second as shown in Fig. 3 .

For S_B, we directly execute LessMine. The theoretical bound of estimators needed is

$$r_B \geq \Delta_s^{k-2} \cdot \frac{m - \Delta'}{(k-2)!} \cdot \frac{3\ln(\frac{2}{})}{G_B \epsilon^2} \tag{7}$$

where Δ_s is the second largest out-degree.

For S_A , we also execute LessMine but only sample in E_1, which substitutes m with Δ' in (5). So the theoretical bound is

$$r_A \geq \Delta'^{k-2} \cdot \frac{\Delta'}{(k-2)!} \cdot \frac{3\ln(\frac{2}{})}{G_A \epsilon^2} \tag{8}$$

Now, the graph with a larger maximum degree has fewer edges, while the graph with most of the edges has a smaller maximum degree. Though $\Delta' > \Delta$, it is still bound by the degree of undirected graph Δ_{ASAP}. For imbalanced graphs whose $\Delta - \Delta_s >> 1$, it's often true that $r_A + r_B < r$.

Moreover, the partition is done with no accuracy loss. For any estimator of LessMine, if the first edge is sampled in E_1, then we only need E_1 for concurrent sample and E_2 for uniform close (see Fig. 5); if the first edge $[n_0, n_1]$ is not sampled in E_1, then we won't use E_1 in the process, because $n_0 > n_p$ and LessMine finds nodes in ascending order. So the locality-aware partition equals to the classification of the first sampled node, thus doesn't harm accuracy.

It is also necessary to point out that S_A looks like a concentric circle with a densely connected node in the middle. This special structure resembles a breadth-first search, so by executing other structure-specific algorithms on S_A, it's possible that one can get a faster count.

V. Evaluation

In this section, we evaluate the performance of LessMine and ASAP using the Python implementation of both algorithms. There's no open-source code of ASAP to our knowledge, so we implement the algorithm by following the description of the article [1]. We use three real-world graphs from the Stanford Large Network Dataset Collection [17] to evaluate the algorithms, as shown in TABLE I. The exact numbers of 4-cliques are counted using RStream [3].

TABLE I
Properties of Graphs

Graph	# Edges	# Vertices	# 4-cliques
ego-Facebook(e-F) [18]	88,234	4,039	30,004,668
gemsec-Deezer-HR(g-D) [16]	498,202	54,573	633,995
gemsec-Facebook-Artist(g-F) [16]	819,306	50,515	4,479,206

A. Sample Space Reduction

LessMine constructs a smaller sample space. As illustrated before, the number of "red cube" is fixed, so the number of instances of the whole sample space is inversely proportional to the proportion of non-zero-estimators. By using 100,000 estimators of ASAP and LessMine to find 4-cliques on real-world graph ego-Facebook [18], we find that only 0.85% of the estimators of ASAP return with non-zero value while LessMine has 25.36% of them (29 times larger than ASAP). It means that the number of cubes in the "black box" of LessMine is only 3.3% of that in ASAP, making it much easier to explore.

Besides, LessMine has 7.80% estimators that return zero before the closing stage, while ASAP only has 0.96% early return estimators. This also speeds up LessMine by sparing the closing time.

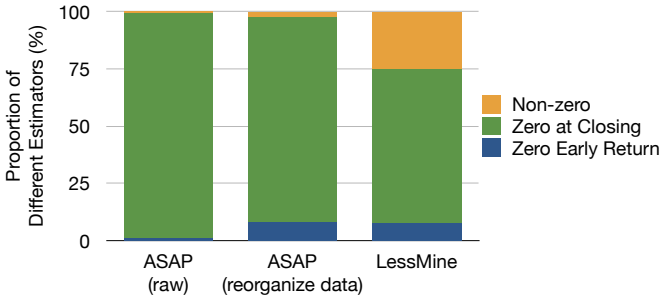

Fig. 6. Proportion of Non-Zero Estimators

By using the reorganized data, the number of non-zero estimators of ASAP goes up to 2.24%, which confirms the analysis of Section III.B and the example of Fig. 2.

B. Performance Improvement

We execute ASAP and LessMine with 100,000 estimators on three real-world graphs [16], [18]. The timing performance of both algorithms are shown in Fig. 7. The time of data reorganization of LessMine is also calculated in the evaluation, which typically takes less than 15% run time of LessMine and continues to drop when graphs get larger and need more estimators.

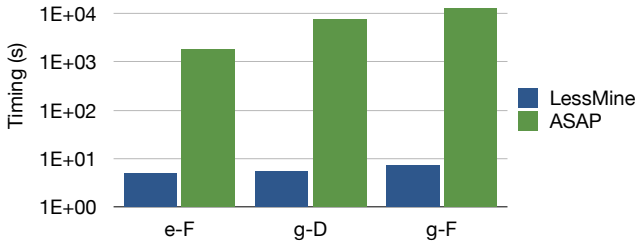

Fig. 7. Speedup Using 100,000 Estimators on Different Graphs

LessMine demonstrates two to three orders of magnitude of speed up with the increase of graph size. On Facebook-Artist (819306 edges), LessMine is $1829\times$ faster against ASAP.

C. Faster Convergence

LessMine not only greatly speeds up the process of each estimator, but also makes it more effective so that one can use fewer estimators to get better error bound.

Fig. 8 shows the theoretical bound of estimators needed to count the number of a k-clique within 5% error with 95% confidence on ego-Facebook [18]. Though it's a relatively loose bound, it reflects the complexity of algorithms, especially in worst cases. With the growth of k, the number of estimators decreased by orders of magnitudes, as shown in the log-space graph Fig. 8. With (3) and (5), we know that LessMine only use $1/[(k-1)!(k-2)!\alpha^{k-2}]$ estimators of ASAP where $\alpha = \Delta_{ASAP}/\Delta_{LessMine} > 1$, so LessMine gets

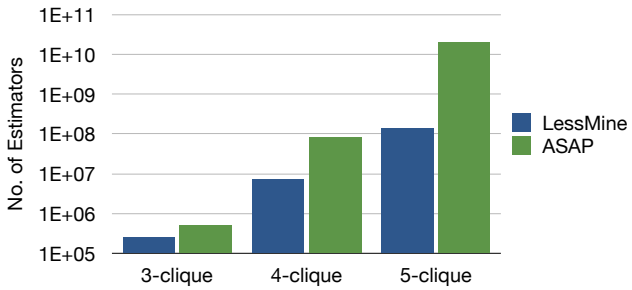

Fig. 8. Theoretical Bound of Number of Estimators Needed.

much more effective with little increase of k. Fig. 9 shows the error-estimator profile of 4-clique counting with 10^5 estimators for 10 times.

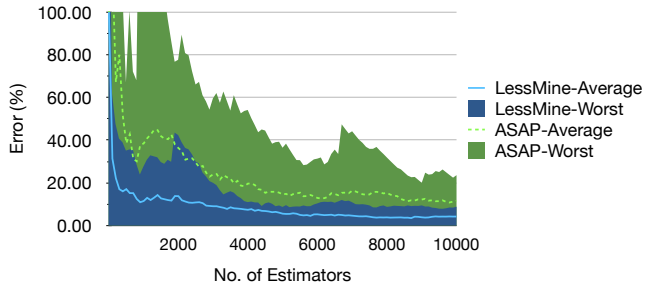

Fig. 9. Error VS. Number of Estimators for 4-clique Counting.

LessMine demonstrates faster and more consistent convergence with approximately 34% of error rate of that of ASAP in average and worst cases.

VI. Discussion

In this section, we'll explain how to apply LessMine to other dense patterns. Some examples of dense patterns are shown in Fig. 10 .

Given a dense pattern, for example, the adjacency matrix of a 4-near-clique, we start by listing out all of the isomorphic patterns of the given one. Note that graph isomorphism is considered an NP-complete problem and beyond the scope of this paper. Thankfully, despite very large graphs, the patterns to be counted are typically small ones within dozens of nodes for pattern mining problems. So we just iterate through all the permutations of the pattern's node indexes to generate all possible adjacency matrix of isomorphic patterns and merge the same matrices to get the final list of isomorphic patterns $[\{p_0\}, \{p_2\}, \dots]$. In the 4-near-clique example, it would be merging $4! = 24$ matrices into 6 different adjacency matrices. The 6 isomorphic patterns are shown in Fig. 10. The shape of the matrix is shown in (4).

Based on whether the smallest node is a DCN, we divide the patterns into two categories: DP1 and DP2, as defined in IV.C .

For DP1, we use sample and concurrent sample to get the proposal and probability P just as we do with k-cliques. As proven in IV.C , the proposal and DP1 form a bijection, so we won't miss or double count any DP1 in the graph. Then we execute uniform close to fulfill the adjacency matrix $\{a(n_i, n_j)\}$ of current embedding as shown in (4). If $\{a(n_i, n_j)\}$ equals to one of the isomorphic patterns $[\{p_0\}, \{p_2\}, \dots]$, the embedding closes and the estimator returns the reciprocal for

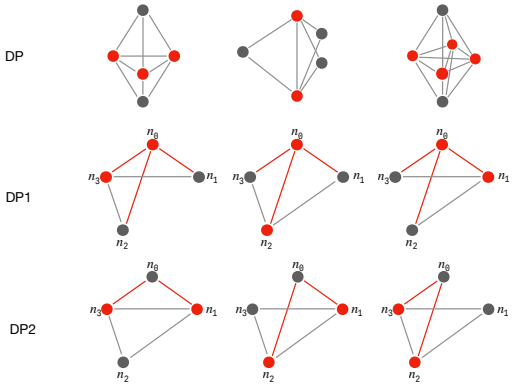

Fig. 10. Examples of Dense Pattern (DP).

the probability $X_i = 1/P$ given by concurrent sample; If $\{a(n_i, n_j)\}$ is not equal to any of the isomorphic patterns, it returns 0. The number of DP1 equals to $E[X_i]$.

The same approach doesn't work on DP2 since the smallest node can't find all the other nodes concurrently. For near-clique, we may first sample a non-existent edge and then concurrent sample and uniform close like DP1, but it can only apply on near-clique. So We introduce another solution based on the relationship between DP1 and DP2. Because the indexes of nodes are randomly assigned (if not, we can reassign the index randomly), any node has the same probability of becoming the smallest in the pattern. So we calculate the number of DCN n_{DCN} of a k-node pattern. Then we know that in any pattern, the smallest node may be a DCN with the probability of n_{DCN}/k. This gives the equation:

$$n_{DP} : n_{DP1} : n_{DP2} = k : n_{DCN} : k - n_{DCN} \tag{9}$$

which extends LessMine to any dense pattern. For example, in Fig.10 we first use LessMine to count the number of DP1. Because 4-near-clique has 2 DCNs, so the number of 4-near-clique would be $n_{DP} = n_{DP1} \times 4 \div 2 = 2n_{DP1}$

VII. Conclusion

In this paper, we propose LessMine, an algorithm that reduces the sample space and data access of approximate dense pattern mining with excellent timing and accuracy performance. To tackle the problem such as heavy edge searching, inefficient sample space abstraction, and lack of graph feature analysis faced by previous approximate pattern mining systems, we proposed four optimizations. First, we orient, sort, and index the edge stream to provide faster searching. Second, we propose concurrent sample to narrow the sample space of the approximation. Third, we use uniform close to avoid iterating on the whole remaining edge stream. Finally, we introduce locality-aware partition that avoids edge cutting and data exchanging between partitions. With all these designs, LessMine has up to 1829× faster estimators with much fewer estimators needed for certain error bound against ASAP.

VIII. Acknowledgements

This work was supported by National Key Research and Development Program of China (No. 2018YFB0105000, 2017YFA0207600), National Natural Science Foundation of China (No. U19B2019, 61832007, 61622403, 61621091),

China Postdoctoral Science Foundation (No. 2019M660641), Beijing National Research Center for Information Science and Technology (BNRist), and Beijing Innovation Center for Future Chips.

REFERENCES

[1] A. P. Iyer, Z. Liu, X. Jin, S. Venkataraman, V. Braverman, and I. Stoica, "ASAP: Fast, approximate graph pattern mining at scale," in *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, 2018, pp. 745–761.

[2] C. H. C. Teixeira, A. J. Fonseca, M. Serafini, G. Siganos, M. J. Zaki, and A. Aboulnaga, "Arabesque: a system for distributed graph mining," in *Proceedings of the 25th Symposium on Operating Systems Principles*, 2015, pp. 425–440.

[3] K. Wang, Z. Zuo, J. Thorpe, T. Q. Nguyen, and G. H. Xu, "Rstream: marrying relational algebra with streaming for efficient graph mining on a single machine," in *OSDI'18 Proceedings of the 12th USENIX conference on Operating Systems Design and Implementation*, 2018, pp. 763–782.

[4] D. Mawhirter and B. Wu, "Automine: harmonizing high-level abstraction and high performance for graph mining," in *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, 2019, pp. 509–523.

[5] W. Wang, C. Wang, Y. Zhu, B. Shi, J. Pei, X. Yan, and J. Han, "Graphminer: a structural pattern-mining system for large disk-based graph databases and its applications," in *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, 2005, pp. 879–881.

[6] U. Kang, C. E. Tsourakakis, and C. Faloutsos, "Pegasus: A peta-scale graph mining system implementation and observations," in *2009 Ninth IEEE International Conference on Data Mining*, 2009, pp. 229–238.

[7] K. Wang, G. Xu, Z. Su, and Y. D. Liu, "Graphq: graph query processing with abstraction refinement: scalable and programmable analytics over very large graphs on a single pc," in *USENIX ATC '15 Proceedings of the 2015 USENIX Conference on Usenix Annual Technical Conference*, 2015, pp. 387–401.

[8] I. Takigawa and H. Mamitsuka, "Graph mining: procedure, application to drug discovery and recent advances," *Drug discovery today*, vol. 18, no. 1-2, pp. 50–57, 2013.

[9] B. Ribeiro, N. Chen, and A. Kovacec, "Shaping graph pattern mining for financial risk," *Neurocomputingshapng*, 2017.

[10] L. Akoglu and C. Faloutsos, "Anomaly, event, and fraud detection in large network datasets," in *Proceedings of the sixth ACM international conference on Web search and data mining*. ACM, 2013, pp. 773–774.

[11] B. Adamcsek, G. Palla, I. J. Farkas, I. Derényi, and T. Vicsek, "Cfinder: locating cliques and overlapping modules in biological networks," *Bioinformatics*, vol. 22, no. 8, pp. 1021–1023, 2006.

[12] G. D. Bader and C. W. V. Hogue, "An automated method for finding molecular complexes in large protein interaction networks." *BMC Bioinformatics*, vol. 4, no. 1, pp. 2–2, 2003.

[13] J. Kim and M. Hastak, "Social network analysis: Characteristics of online social networks after a disaster," *International Journal of Information Management*, vol. 38, no. 1, pp. 86 – 96, 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S026840121730525X

[14] N. Pržulj *et al.*, "Modeling interactome: scale-free or geometric?" *Bioinformatics*, vol. 20, no. 18, pp. 3508–3515, 2004.

[15] J. E. Gonzalez, Y. Low, H. Gu, D. Bickson, and C. Guestrin, "Powergraph: distributed graph-parallel computation on natural graphs," in *OSDI'12 Proceedings of the 10th USENIX conference on Operating Systems Design and Implementation*, 2012, pp. 17–30.

[16] B. Rozemberczki, R. Davies, R. Sarkar, and C. Sutton, "Gemsec: Graph embedding with self clustering," in *Proceedings of the 2019 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining 2019*. ACM, 2019, pp. 65–72.

[17] J. Leskovec and A. Krevl, "SNAP Datasets: Stanford large network dataset collection," http://snap.stanford.edu/data, Jun. 2014.

[18] J. Leskovec and J. J. Mcauley, "Learning to discover social circles in ego networks," in *NIPS*, 2012, pp. 539–547.