A Scalable and Generic Framework to Mine Top-k Representative Subgraph Patterns

Dheepikaa Natarajan *
Amazon
Amazon, Bangalore, India
ndheepikaa@gmail.com

Sayan Ranu
IIT Madras
CSB 12, Dept. of CSE, Chennai 600036, India
sayan@cse.iitm.ac.in

***Abstract*—Mining subgraph patterns is an active area of research. Existing research has primarily focused on mining all subgraph patterns in the database. However, due to the exponential subgraph search space, the number of patterns mined, typically, is too large for any human mediated analysis. Consequently, deriving insights from the mined patterns is hard for domain scientists. In addition, subgraph pattern mining is posed in multiple forms: the function that models if a subgraph is a pattern varies based on the application and the database could be over multiple graphs or a single, large graph. In this paper, we ask the following question: *Given a subgraph importance function and a budget k, which are the k subgraph patterns that best represent all other patterns of interest?* We show that the problem is NP-hard, and propose a generic framework called RESLING that adapts to arbitrary subgraph importance functions and generalizable to both transactional graph databases as well as single, large graphs. Experiments show that RESLING is up to 20 times more representative of the pattern space and 2 orders of magnitude faster than the state-of-the-art techniques. The executables for the tool are available at http://www.cse.iitm.ac.in/∼sayan/software.html.**

I. INTRODUCTION

Recent technological advances have generated large volumes of data that are represented as graphs. Examples include chemical compounds [1], protein-protein interaction networks [2], social networks [3], and road networks [4]. Given a function that classifies a subgraph as either important or unimportant, the goal in subgraph mining is to identify subgraphs that are *important*. The importance of a subgraph could represent a variety of domain specific properties such as the frequent subgraphs [5]–[8], discriminative subgraphs [2], [9], statistically significant subgraphs [10]–[13], etc.

The main challenge in subgraph mining is to explore the exponential subgraph search space in an efficient manner. A graph with n nodes could contain 2^n subgraphs and evaluating each possible subgraph is not scalable. Hence, majority of the existing techniques have focused on developing strategies that are effective in pruning the search space. Despite this progress in subgraph mining, two important issues remain unsolved.

1. Answer set size: Due to the exponential subgraph search space, the number of subgraphs that are mined as important is also extremely large. Now, when the answer sets are given to domain scientists, a human-mediated analysis is not feasible due to their sheer sizes. For example, chemists want to familiarize themselves with the structures and chemical properties of the mined patterns for targeted drug discovery. However, individually going through all patterns is not possible. There is a critical need to summarize subgraph patterns using a small number of representatives without compromising on the information content. To establish the issue of answer set size empirically, we perform frequent subgraph mining on the active molecules in the DTP AIDS anti-viral screening dataset, which contains only 422 graphs. Given a threshold $\theta \in [0, 100]$, a subgraph is frequent if it is present in at least $\theta\%$ of the database graphs. Fig. 1 shows that the number of frequent subgraphs grows exponentially with decrease in θ. At 5% frequency threshold, 252331 subgraphs are mined.

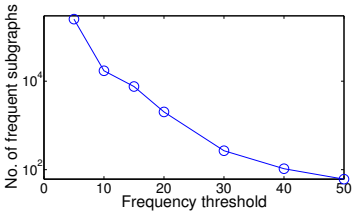


Fig. 1. The growth of the answer set size with frequency threshold in frequent subgraph mining. The y-axis is in log scale.

2. Information Redundancy: The answer sets also suffer from redundancy. Subgraphs that are structurally similar have similar importance values [14], [15]. Consequently, the answer set is overloaded with similar subgraphs.

In this paper, we resolve the above two weaknesses. We ask the following question: *Given a budget k in addition to the importance function, which are the k important subgraphs that best represent all important subgraphs?* A subgraph g *represents* another subgraph g', if they are structurally similar. There are three key challenges in this task.

1. Adapt to any importance function: As discussed earlier, a number of importance functions have been studied for subgraph mining. In our work, the goal is to develop a framework that is generalizable to all importance functions.

2. Graph datasets: There are two kinds of graph datasets that are routinely encountered. In the first kind, we have a database of objects, where each object is a graph. This setup is common while mining molecular repositories [5], [11] and

*The work was done at IIT Madras

2374-8486/16 $31.00 © 2016 IEEE
DOI 10.1109/ICDM.2016.49

IEEE computer society

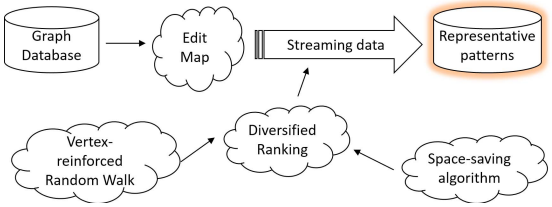


Fig. 2. Pipeline of RESLING.

is popularly known as the *transactional graph database*. The second type of scenario is where we have a single large graph. Such graphs are routinely used to model protein-protein interaction networks, social networks, and road networks [2], [7], [16]. Each dataset type brings in their own unique challenges and our goal is to develop a framework that can handle both types of graph datasets.

3. Scalability: The problem of mining top-k representative subgraphs is NP-hard and even greedy heuristics do not scale.

To resolve all of the above challenges, we develop a technique called RESLING *(REpresentative Subgraph sampLING)*. While two techniques exist to mine representative frequent subgraphs [17], [18], they do not generalize to other importance functions and dataset types. A key focus of our work is to develop a framework that works across different importance functions and graph dataset types. Such flexibility offers more freedom to end-users, enhances productivity in developments of higher-order systems that rely on representative patterns, and in general, is more future-proof.

Fig. 2 outlines the flow of RESLING. Given a graph database, we convert its exponential subgraph search space into an *edit map (EMP)*. Edit map imposes a structure on the search space where structurally similar subgraphs are naturally in close proximity. This organization of the search space negates the need to perform expensive graph clustering and lies at the core of scaling up representative subgraph mining. Since the EMP is exponential in size, we perform a diversified ranking on the EMP in a streaming manner to identify k representative patterns. This diversified ranking is powered by *vertex-reinforced random walk* [19] and the *Space-saving algorithm* [20]. A noteworthy aspect of RESLING is that it avoids a two-step approach where first the subgraph patterns are mined, and then the representative ones are identified. Instead, both operations happen in a single, integrated fashion, which allows us to scale to large graph datasets. The core contributions are as follows:

- We are the first to develop a generic framework called RESLING to mine top-k representative subgraph patterns for any given subgraph importance function.
- RESLING is scalable to large graph databases. It derives its power by structuring the search space in the form of an *edit map*, where structurally similar subgraphs are naturally in close proximity. Thus, expensive operations like subgraph clustering are avoided.
- Extensive experiments on real datasets demonstrate RESLING to be up to 2 orders of magnitude faster and 20 times more effective in representing the pattern space than state-of-the-art techniques.

II. PROBLEM FORMULATION

A graph $G = \{V, E\}$ is composed of a set of vertices $V = \{v_1, \cdots, v_n\}$ and a set of edges $E = \{(v_i, v_j) \mid v_i, v_j \in V\}$ modeling the relationships between vertices. The *size* of a graph is defined as $|\mathbb{E}|$. Generally, there are two kinds of graph datasets and our goal is to develop a technique that can handle both.

***Definition** 1:* GRAPH DATABASE: *A graph database consists of a collection of graphs $\mathbb{D} = \{G_1, \cdots, G_m\}$. Each graph has its own set of vertices and edges.*

When $|\mathbb{D}| > 1$, we call it the *transactional* setting; otherwise, it corresponds to the *single-large-network setting*. As evident from its name, single-large networks typically are much larger in sizes than graphs encountered in the transactional setting. A graph $g = \{V_g, E_g\}$ is a subgraph of a graph database $\mathbb{D}$, denoted by $g \subseteq \mathbb{D}$, if there exists a graph $G \in \mathbb{D}$, such that $V_g \subseteq V_G$, $E_g \subseteq E_G$. $g = \{V_g, E_g\}$ is connected if there exists a path from u to v, $\forall u, v \in V_g$. As in majority of the subgraph mining techniques [2], [5], [6], [16], we consider only connected subgraphs.

We assume there is an existing algorithm $\mathcal{A}$ to mine subgraph patterns. For our purposes, $\mathcal{A}$ is a black box and supports two operations.

1) Given a graph database, $\mathcal{A}$ can mine all important subgraph patterns $\mathbb{T}$.
2) For any given subgraph pattern $g \subseteq \mathbb{D}$, $\mathcal{A}$ can quantify the importance of g. We denote $g's$ importance using the notation $\phi(g)$. The importance function can model any graph property such as subgraph frequency [5], discriminative potential [2], [9], [16], or statistical significance of g [10], [11].

Given a budget k indicating the desired size of the answer set, our goal is therefore to "represent" the spectrum of important subgraphs in $\mathbb{T}$ using k representatives. Towards that goal, we define the δ-neighborhood of a subgraph.

***Definition** 2:* δ-NEIGHBORHOOD. *The δ-neighborhood of a subgraph g, denoted as $N(g)$, contains all important subgraphs within a distance threshold δ from g.*

$$N(g) = \{g' \in \mathbb{T} \mid d(g, g') \leq \delta\} \tag{1}$$

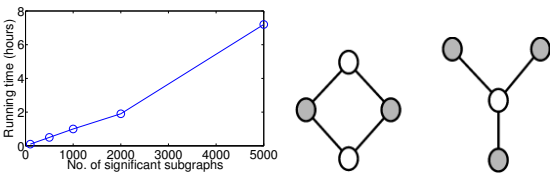
where $d(g, g')$ is the classical *graph edit distance* [21]. In other words, g is a structural representative of all subgraphs in its δ-neighborhood. Due to the simplicity of this definition, δ is intuitive to set. Now, extending the same formulation, we define the representative power $\pi(\mathbb{S})$ of a set of graphs $\mathbb{S}$.

$$\pi(\mathbb{S}) = \frac{|N(\mathbb{S})|}{|\mathbb{T}|} \tag{2}$$

where $N(\mathbb{S}) = \bigcup_{g \in \mathbb{S}} N(g)$. Hereon, we denote the representative power $\pi(\{g\})$ of a graph g as $\pi(g)$.

***Problem** 1:* TOP-k REPRESENTATIVE SUBGRAPHS: Given a budget k, compute the representative set $\mathbb{R}$ such that

$$\mathbb{R} = \arg\max_{\mathbb{S}} \{\pi(\mathbb{S}) \mid \mathbb{S} \subseteq \mathbb{T},\ |\mathbb{S}| = k\} \tag{3}$$



(a) (b)

Fig. 3. (a) The growth of the running time with the size of $\mathbb{T}$. (b) Example of a graph database containing two graphs. The node color denotes their labels.

In essence, the proposed model captures as much of the various groups of important subgraphs as possible within the budget k.

III. Challenges of mining Top-k Representative Subgraphs

In this section, we analyze the problem formulation and lay out the challenges.

***Theorem** 1: The problem of mining top-k representative subgraph patterns is NP-hard.*

PROOF: The problem reduces to the set-cover problem. The proof follows in the same manner as in [22].

Fortunately, $\pi(\mathbb{S})$ is a *submodular* function. A submodular function f is a set function that satisfies the following condition.

$$f(S \cup \{e\}) - f(S) \geq f(T \cup \{e\}) - f(T) \tag{4}$$

for any element e, any set S and any of its superset T.

***Theorem** 2: Representative power (Eq. 2) is monotone and submodular.*

The proof follows in the same manner as in [22].

If the function is submodular and monotone, the greedy hill-climbing algorithm of iteratively choosing the element with maximal marginal gain approximates the optimal solution within a factor of $(1 - \frac{1}{e})$ [23].

A. The greedy approach

Based on Theorem 2, the following two-step approach can be adopted. Alg. 1 presents the pseudocode. First, we utilize existing algorithms to mine the important subgraphs $\mathbb{T}$ (line 1). Then, we iteratively choose the subgraph providing the highest marginal gain in the representative power to populate $\mathbb{R}$ till it attains the size of k (lines 2-7). The following bound can be provided on the quality of $\mathbb{R}$ with respect to the optimal answer set $\mathbb{R}^*$.

***Corollary** 1:* $\pi(\mathbb{R}) \geq (1 - \frac{1}{e})\pi(\mathbb{R}^*)$

Unfortunately, the greedy approach does not scale. The primary bottleneck lies in the neighborhood update step (lines 6-7), which is performed at each iteration. The neighborhood update set requires $O(n^2)$ edit distance computations. Moreover, computing edit distance between two graphs is NP-hard [21]. To empirically establish the non-scalability of the baseline greedy approach, we plot the growth of the running time with the size of $\mathbb{T}$. Fig. 3(a) presents the results. As shown, it takes more than 7 hours to complete even when cardinality of $\mathbb{T}$ is 5000. As was shown earlier in Fig. 1, even in a small graph database of 400 graphs, the number of important subgraphs can reach 200,000. Note that $\mathbb{T}$ can only be computed at runtime and thus cannot be indexed. Hence any post-processing based heuristic, such as k-means clustering of $\mathbb{T}$ is also not scalable.

Algorithm 1 Baseline-Greedy($\phi(.)$, k)

1: Compute $\mathbb{T}$ based on importance function $\phi(.)$
2: $\mathbb{R} \leftarrow \emptyset$
3: **while** $|\mathbb{R}| < k$ **do**
4: $\quad g^* \leftarrow \arg\max_g \{\pi(\mathbb{R} \cup \{g\}) - \pi(\mathbb{R}) \mid g \in \mathbb{T}\}$
5: $\quad \mathbb{R} \leftarrow \mathbb{R} \cup g^*$
6: $\quad$ **for** $g \in \mathbb{T} \setminus \mathbb{R}$ **do**
7: $\quad\quad N(g) \leftarrow N(g) \setminus N(g^*)$

The key challenge is therefore the following: *A two-step, post-processing based procedure is not scalable. We need to directly mine the representative patterns by integrating both the importance computation and the representative power computation in a single framework.*

IV. RESLING

The most expensive step in the greedy approach is grouping graphs based on their structural similarity. *Can we organize subgraphs in a manner such that structurally similar subgraphs are naturally in the same group without the explicit need to compute edit distances between them?* We solve this question by organizing the search space in the form of an *Edit Map (EMP)*.

A. Structuring the subgraph search space

Our search space consists of all possible subgraphs of the database. The EMP organizes this search space into an edge-weighted undirected graph where each node is a subgraph of the database and the edges correspond to *edits* that can be performed on a subgraph to construct other subgraphs of the database. An edit is either a deletion of an edge or an addition of an edge. The impact of the edit, which is the change in the importance score due to the edit, is captured as the edge weight. Suppose, g and g' are two subgraphs of the database. Hence, both these graphs are part of the search space and therefore nodes in the EMP. Furthermore, let $g' \supseteq g$, and g' contains exactly one more edge than g. Thus, in the EMP g will be connected to g' since by adding an edge to g, we can construct g'. The weight of this edge from g to g' is

$$w(g, g') = \phi(g') - \phi(g) \tag{5}$$

where $\phi(g)$ denotes the importance of subgraph g. [1] In summary, any subgraph $g \subseteq \mathbb{D}$ is a node in the EMP. g is connected to those subgraphs $g' \subseteq \mathbb{D}$, where either $g' \subseteq g$ or $g' \supseteq g$, and g' contains either one additional edge or one less edge than g. An edit is performed on g by either deleting an edge or adding an edge to construct g'. The formal definition is as follows.

***Definition** 3:* EDIT MAP. *Edit map of a graph database $\mathbb{D}$ is an edge-weighted graph $M = (V_M, E_M)$, where $V_M = \{g \mid$*

[1] We slightly abuse the term "undirected graph" here. Although the edges are undirected, the sign of the edge weight depends on the direction.

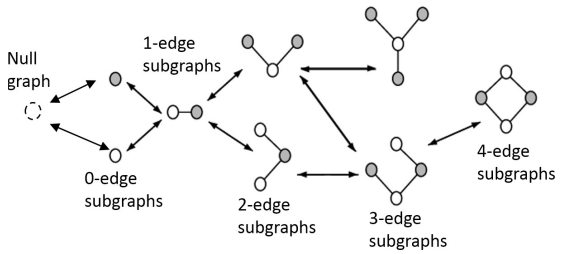


Fig. 4. The edit map of the example database in Fig.3(b). Each subgraph is a node in the EMP.

$g \subseteq \mathbb{D}\}$, $E_M = \{(g = (V, E), g' = (V', E')) \mid$ *either* $g' \supseteq g, E' = E \cup \{e\},$ *or* $g' \subseteq g, E' = E \backslash \{e\}, e \in E\}$.

***Example** 1: Fig. 4 shows the edit map corresponding to the database in Fig. 3(b). For simplicity, the edge weights are not shown. The smallest subgraph of the database is the null graph. The null graph is connected to 0-edge subgraphs of the database, which are essentially all possible nodes. In our case, there are two types of nodes: the grey and the white. These 0-edge subgraphs are connected to the null graph and the 1-edge subgraphs. The 1-edge subgraphs are all possible edge-types in the database. The EMP is extended in this manner till it reaches the largest subgraph of the database, which is of size 4 and is present only in the first graph of the database. Any subgraph of the database, is a node in the EMP and connected to its immediate subgraphs and supergraphs.*

Properties of the edit map:

• *Connectivity:* The EMP is connected. From any node in the EMP, one can reach the null graph by repeatedly deleting edges. From the null graph, a path exists to all other nodes.

• *Proximity:* If the edit distance between two subgraphs is small, then they are in close proximity in the EMP as well due to our construction strategy. Thus, with a high likelihood, there would be zones in the EMP that contain a cluster of important subgraphs due to the correlation between structural similarity and importance [14], [15]. As we will see next, our strategy is to reach these zones and identify the best representatives.

• *Size:* Since each subgraph of the database corresponds to a node in the EMP, the size of the EMP is exponential. As a result, it is neither possible to compute it entirely, nor store it in memory. However, given any particular node in the EMP, we can easily compute its one-hop neighborhood by performing all possible edits.

B. Ranking Subgraphs in Edit Map

Since the EMP is a connected network, and a high edge weight indicates transition to a more important subgraph, if we perform PageRank [24] on the EMP, with a high likelihood, the random walk will be concentrated among the important subgraphs. Thus, important subgraphs are likely to have a high PageRank and non-important subgraphs are likely to have low PageRanks. However, PageRank does not reward representativeness.

***Example** 2: Consider the example network shown in Fig:5(a). Let the task be to summarize information of the whole network by the top-3 nodes. For simplicity, assume that all edge weights are equal. Fig. 5(b) shows the 3 shaded nodes that would receive the highest PageRank. The result that we would rather like to achieve is shown in Fig. 5(c). Fig. 5(c) is preferable since the shaded nodes capture all three communities in the network.*

To address this weakness of PageRank, we perform *vertex-reinforced random walks (VRRW)* [19] on the EMP. VRRW rewards both the importance of the node and its representative power.

1) Vertex-Reinforced Random Walk (VRRW): VRRW is similar to PageRank, but it is a time-variant *random walk* process. A random walk on a network defines a *Markov chain*, where each node represents a state and a walk transits from node u to node v proportional to the transition probability, denoted as $p(u, v)$. Transitions happen only through edges in the network and the transition probabilities are proportional to the edge weights. While in PageRank $p(u, v)$ is static, in VRRW the transition probability to a node u from other nodes is reinforced by the number of previous visits to u.

To formalize VRRW, let $p_0(u, v)$ be the transition probability from u to v at timestamp 0, which is the start of the random walk. Furthermore, let $N_T(v)$ be the number of times the walk has visited v up to time T . Then, VRRW is defined sequentially as follows. Let there be n states. Initially, $N_0(i) = 1$ for $i = 1, \cdots, n$. Suppose the random walker is at node u at the current time T. Then, at time $T + 1$, the random walk moves to some node v with probability $p_T(u, v) \propto p_0(u, v)N_T(v)$. In other words, $p_T(u, v)$ is reinforced by $N_T(v)$. For our problem, VRRW is generalized as follows.

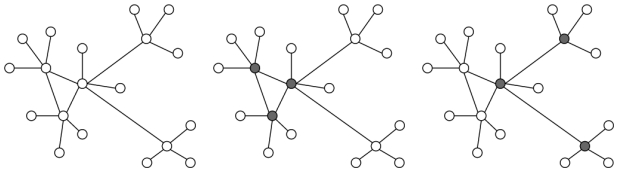
$$p_T(u, v) = (1 - \lambda)\frac{1}{|V_M|} + \lambda\frac{p_0(u, v)N_T(v)}{D_T(u)} \tag{6}$$

where $D_T(u) = \sum_v p_0(u, v)N_T(v)$ is the normalizing term. Here, $1-\lambda$ is the teleportation probability, which is also present in PageRank. λ represents the probability of choosing one of the neighboring nodes based on the reinforced transition probability. However, with probability $(1-\lambda)$ the walk jumps to a random node in the EMP. In VRRW, we also add a self-edge to all nodes. If the network is ergodic, VRRW converges to some stationary distribution τ [19].

$$\tau(v) = \sum_{u \in V} p_T(u, v)\tau(u) \tag{7}$$

Furthermore, $\sum_{\forall v \in V} \tau(v) = 1$.

Why does VRRW favor representativeness? As in PageRank, nodes with higher centralities get higher weights due to the flow arriving at these nodes. This, in turn results in larger visit counts $(N_T(v))$. When the random walk proceeds, the nodes that already have high visit counts tend to get an even higher weight. In other words, a high-weighted node starts dominating all other nodes in its neighborhood and the "Rich gets Richer". Note that the self-edge to a node ensures that even if all of its neighbors shrink, its score will still be large as long as its number of visits is large. For further details on the mathematical basis and the theoretical correctness of this phenomenon, we point readers to [25].



(a) Network (b) PageRank (c) Goal

Fig. 5. Illustration of why PageRank is not good enough for our problem.

C. VRRW on Edit map

To perform VRRW on the EMP, we just need to formalize the transition matrix. The rest of the components have already been formalized. Intuitively, better the impact of the edit on the subgraph importance, the higher is the edge weight. An edit is either addition of an edge or a deletion. However, for a broad range of subgraph mining tasks, the importance score behaves monotonically with the subgraph size. For example, in frequent subgraph mining, the frequency decreases with addition of an edge [5]. Thus, if we simply use change in frequency as the edge weight, the random walker would always be biased towards deleting edges and never explore larger frequent subgraphs. On the other hand, in discriminative subgraph mining [2], the discriminative potential increases with addition of an edge since more information becomes available to discriminate. In this scenario, the random walker would be biased towards larger subgraphs. This behavior is problematic since all important subgraphs are not explored adequately. Thus, with 0.5 probability we perform a delete, and with 0.5 probability we add an edge to the current subgraph.

Addition of an edge: Let the current state in VRRW be $X_T = g = (V_g, E_g)$ and $\phi(g)$ be the importance of g. For each edge e that can be added to g to transition to a supergraph h, the transition probability is defined as follows.

$$p_T(g,h) = 0.5\frac{w(g,h)N_T(h)}{\sum_{g' \in g_{sup}} w(g,g')N_T(g')} \tag{8}$$

where $w(g,h)$ (Eq. 5) is the edge weight and g_{sup} is the set of all supergraphs of g containing one more edge.

Deletion of an edge and self-edges: Edge deletions can be modeled just like edge additions. However, when both deletions and additions are biased towards "good edits", then a neighborhood of important subgraphs would be like a black hole that the random walker would never be able to escape. It is necessary to explore "bad edits" that may lead towards other neighborhoods of important subgraphs. To incorporate this property, edge deletions or staying at the same node through self edges are weighted uniformly. Mathematically, let $X_T = g = (V_g, E_g)$ be the current state. The probability of an edge delete or self-edge is

$$p_T(g,h) = 0.5\frac{N_T(h)}{\sum_{g' \in g_{sub} \cup \{g\}} N_T(g')} \tag{9}$$

where g_{sub} is the set of all subgraphs of g containing one less edge and $h \subseteq g$ is the resulting subgraph following the edit. In other words, we either stay in the current node g, or transition to a subgraph of g proportional to their visit counts.

Algorithm 2 SSA.add(g)

Input: g is the current node (subgraph) to be updated
Input: $\vec{V}$ is the count vector that is being maintained in descending order

```
if V⃗ contains less than M nodes then
    Add g to V⃗ with count 2.
else
    if g ∈ V⃗ then
        Count(g) ← Count(g) + 1
    else
        g_M ← least frequent subgraph in V⃗
        Replace L with g
        Count(g) ← Count(g_M) + 1
        Compute and store neighborhood of g
```

Key property: The key advantage in the proposed framework is that we do not need to compute any edit distances. In the EMP, subgraphs that are structurally similar would naturally be in close proximity. Furthermore, VRRW ensures diversity. Thus, two subgraphs from the same cluster would not receive high scores. Overall, the process of identifying important subgraphs, computing their neighborhoods, and identifying representatives are all integrated in a single, coherent ranking process.

D. Managing the exponential search space

Sec. IV-C formalizes VRRW on the EMP. At this juncture, recall that the EMP is exponential in size and therefore, it can never be computed in its entirety. However, a closer analysis of the VRRW reveals that we never need to load the entire EMP. Given the subgraph of the database that is the current state, we only need to construct its neighbors and perform a state transition according to the VRRW principles. Specifically,

1) Construct the neighborhood of g in the EMP by enumerating all possible edge additions and edge deletions.
2) Choose a neighbor based on the transition probability.

In other words, we take local decisions to converge towards a global solution. However, a scalability issue remains. Specifically, in step 1, we need to compute the neighborhood of g and the importance of each graph in the neighborhood (due to the denominator of Eq. 8). The number of neighbors is typically large (> 700) and thus this step is prohibitively expensive. Furthermore, this operation needs to be repeated at each timestamp till the VRRW converges. Since the number of iterations in VRRW can be extremely large, we need a mechanism to avoid computing the neighborhood of a subgraph g at every step. We overcome this bottleneck using the *Space Saving Algorithm* (SSA) [20].

E. Space Saving Algorithm

Before we discuss SSA, we first outline the intuition behind our idea. Recall, that in VRRW, the rich gets richer. In other words, although initially all nodes (subgraphs in EMP) start with similar ranks, slowly few diverse, as well as central, nodes emerge that receive more visits than the rest. Owing to higher visits, their likelihoods of being visited again get further reinforced, and eventually, the VRRW is concentrated

on a minority of diverse nodes. We exploit this property. More specifically, if we store the neighborhoods of the frequently visited, then VRRW can be performed much more efficiently. Towards that goal, we employ the following strategy. At any timestamp T, we store the neighborhoods of the top-M most frequent nodes visited till now. M is selected based on the main-memory budget of the system. We assume that $M \gg k$. When we transition to a new node, its neighborhood is computed only if it is not among the top-M most frequent nodes; otherwise, it is a simple look-up. Note that the top-M list continuously changes since the frequency of nodes change with every transition in the VRRW.

If N is the total number of nodes in the EMP, and I is the total number of iterations till VRRW converges, then the storage complexity of computing the top-M frequent nodes is $O(min\{N, I\})$. Since, both N and I are extremely large, storing the frequency of every node visited is not feasible. We thus model our problem as that of computing the top-M most frequent items from a data stream. Specifically, the stream of items are the nodes that we visit in the EMP during the VRRW and we have a budget to track and store only M nodes.

Selecting M: M is selected based on the main-memory capacity of the processing system. We need to store three pieces of information for each of the M cells: the node being monitored, the visit count of the node, and its neighboring nodes. Each node in the EMP corresponds to a subgraph of a database. A graph can be uniquely represented through its canonical label [5], which is a "string". A reasonable assumption is to allocate 128 bytes for each canonical label. The visit counts can be stored as a "long" data type requiring 8 bytes. Assuming that the number of neighbors of a node, on average, is 1000, if we have 8 GB of main memory available, $M = \frac{8\times10^9}{128\times1000+8} = 6.92 \times 10^4$.

As known in the stream processing literature [20], it is not possible to compute the top-M frequent items in a stream optimally. However, the space saving algorithm (SSA) [20] provides good approximations along with some guarantees on the reported counts. Here we explain how the SSA is adopted for RESLING. Alg. 2 presents the pseudocode. At the start of VRRW, we initialize a count vector of size M. This count vector stores the visit counts and the neighborhood of each of the M nodes being monitored. Now, as nodes are processed from the EMP, till M unique nodes arrive, all are stored in the count vector (lines 1-2). Additionally, the count vector maintains all nodes in descending order of their counts. When the $M + 1^{th}$ unique node, g, arrives, we replace the least frequent node, g_M in the count vector with the current node. In addition, the count, $Count(g)$ is stored as $Count(g_M)+1$, the neighborhood of g is computed and stored (lines 7-10). On the other hand, if a node arrives that is already being monitored, its visit count is updated and the neighborhood is extracted. This procedure continues till the end of VRRW.

***Example** 3: Consider the example shown in Fig. 6. M is set to 2. For simplicity, we only show the counts stored. In the stream, first nodes X and Y are seen, and their counts are updated to 2. Recall, that in VRRW, $N_0(v) = 1$ for all nodes.*

Fig. 6. Example of updates to count-vector at M=2.

Algorithm 3 RESLING($\phi(\cdot), k$)

Output: $\mathbb{R}$ is the representative set
1: Initialize SSA count vector
2: $T := 0$
3: $X_T \leftarrow$ A randomly selected subgraph of $\mathbb{D}$
4: Compute $\phi(X_T)$
5: **repeat**
6: **if** $uniform(0, 1) > \lambda$ **then**
7: $h \leftarrow$ randomly selected subgraph of $\mathbb{D}$
8: **else**
9: **if** $uniform(0, 1) \leq 0.5$ **then**
10: $h \leftarrow$ selected subgraph through an edge delete or self-edge (Eqs. 9)
11: SSA.add($X_T, null$)
12: **else**
13: $h \leftarrow$ randomly selected proposed edge addition
14: SSA.add(X_T, h)
15: $T \leftarrow T + 1$
16: $X_T \leftarrow h$
17: **until** convergence of count vector
18: $\mathbb{R} \leftarrow k$ most frequent important subgraphs in $\vec{V}$
19: **return** $\mathbb{R}$

Thus, the first visit to a node sets the count to 2. Now, when the next node observed is Y again, Y's count is updated to 3 and it moves to the first position in the count vector. Next, Z is observed. Since all the counters are already occupied and Z is an unmonitored node, the node X with the least count is replaced by Z and its count is set to 3.

***Theorem** 3: Let $N_T(v)$ be the actual count and $\hat{N}_T(v)$ be the stored count in the count vector. $\hat{N}_T(v) \geq N_T(v)$.*

PROOF: When an unmonitored node arrives, there are two possibilities. **Case 1:** This is the first time the node has been visited in the VRRW. Although the actual count is 2, it is stored as one more than the count of the least frequent node in the count vector.

Case 2: The unmonitored node was earlier being monitored, but became the least frequent at some stage and then got replaced. In this case, the highest possible count for this node is the count of the least frequent node in the count vector.

Thus, overall, it is guaranteed that the stored count of any monitored node is at least as large as its actual count. □

Accuracy analysis: Inaccuracies creep in when replacements occur in the count vector. If a node is monitored throughout, then its actual count will be reported. Thus, lower the number of replacements, better is the accuracy. The chances of replacements are low if the frequency distribution is skewed. and it has been shown that SSA performs best if the frequency distribution follows a power law [20]. This is indeed the case with node visits in VRRW. We show empirical evidence in Sec. V. Consequently, the approximation is both accurate and scalable.

F. The RESLING algorithm

With the formalization of the SSA, the RESLING algorithm is complete. The VRRW starts from a randomly selected

subgraph in the database (Alg. 3, line 3). Next, a transition is chosen according to the VRRW principles (lines 9-18). After the transition takes place, the current subgraph is added to the count vector (lines 11 and 14). Following the transition, the new subgraph becomes the current state and the process is repeated till convergence of the count vector (line 17). Finally, the top-k most frequently visited important subgraphs are returned as the representative set (lines 18-19).

Space Complexity: During runtime, RESLING stores the SSA count vector $\vec{V}$ of size M and the database of graphs $\mathbb{D}$. Thus, the total space complexity is $O(M + |\mathbb{D}|)$.

Time Complexity: The worst-case scenario occurs when the VRRW transitions to a subgraph that is not in the count vector. In such a case, we take three steps.

1. First, we compute the importance of the subgraph. The cost of computing the subgraph importance depends on the importance function. Let us denote this cost of computing subgraph importance as S.

2. The second step is to compute its neighbors in the EMP. This cost is linear to the number of neighbors NR.

3. The visit count of the subgraph is updated in the SSA count vector. There are two sub-steps in this update procedure. First, one is added to the count of the current subgraph and then the count vector is re-arranged to keep it sorted in descending order. To maintain the sorted order, once the count of a subgraph is updated, we need to compare its count with all counts that were higher in the previous iteration. In the worst case, $O(M)$ comparisons are made.

Since the above steps are repeated in each iteration till convergence, the total time complexity is $O(I(S+NR+M))$, where I is the total number of iterations.

V. EXPERIMENTS

In this section, we demonstrate that RESLING is the **most scalable technique** to mine representative subgraph patterns. Furthermore, it is **the only technique** that can adapt to any importance function and graph dataset type. Finally, it achieves quality that is better than the state-of-the-art techniques.

A. Datasets

For the transactional graph database, we use the following two datasets.

1. ZINC chemical compound dataset: The ZINC dataset contains 179,197 graphs. This is the **largest** publicly available transaction graph database. The ZINC dataset can be downloaded from http://zinc.docking.org/.

2. DTP-AIDS Antiviral Screen chemical compound dataset: The dataset consists of 43,905 classified chemical molecules, and a total of 1.09 million atoms. On average, each molecule contains 25.4 atoms (vertices) and 27.3 bonds (edges). Each molecule in the AIDS dataset is labeled. There are 422 molecules that are *active* against the HIV virus, 1084 *moderately active* molecules, and 41176 *inactive* molecules. This dataset has been extremely popular in previous subgraph mining works [5], [10], [11]. The dataset can be downloaded from http://dtp.nci.nih.gov/.

Single-Network dataset: We choose one of the largest publicly available protein-protein interaction networks (PPI) [2], [16]. The PPI contains 11203 vertices and 57235 edges. Each vertex represents a protein and two vertices are connected by an edge if the corresponding proteins co-regulate a biological process. The PPI contains data on breast cancer. 371 human beings were studied. Each human either has breast cancer or does not.

B. Experimental setup

All our experiments are performed on a 3.4 GHz quad-core i7 machine running Ubuntu 14.04 with 16GB of main memory. Unless specifically mentioned, the default value of k is 1000. The default value of δ is 5 (Eq. 1), which means that if a graph g can be converted to g' by performing 5 edits, then g represents g'. However, for both k and δ, we show that we do better across all values. The restart probability λ in VRRW (Eq. 6) is set to 0.95.

To demonstrate the adaptability of RESLING, we evaluate against two importance functions: *frequent subgraph mining (FSM)* and *discriminative subgraph mining (DSM)*.

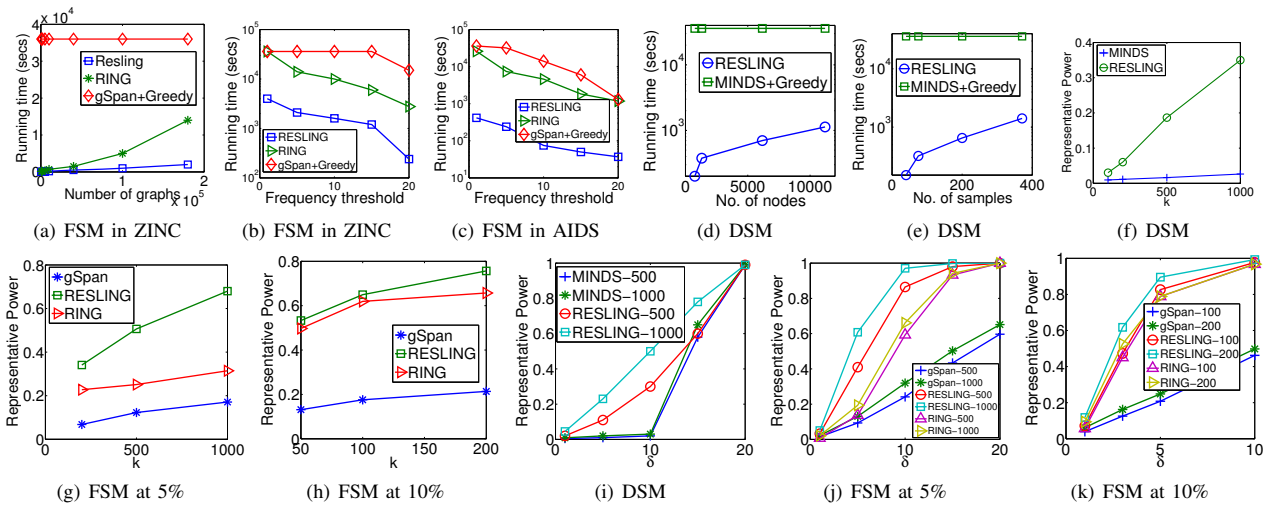
FSM: A subgraph is frequent if it occurs in more than $\theta\%$ graphs in the database, where θ is provided by the user. Our default value of θ is 5%. We use the AIDS and ZINC datasets for FSM. In the specific domain of frequent subgraph mining (FSM), Origami [17] and RING [18] mine representative frequent subgraph patterns from transactional graph databases. Since RING is the more recent technique and offer better performance than Origami, we compare the performance of RESLING with RING in FSM. For RING [18], all parameters are set to the values as recommended by the authors.

DSM: A subgraph is discriminative if a classifier can be learned to predict the class label with an accuracy above a user provided threshold. We perform DSM on the PPI dataset, where the goal is to predict the likelihood of breast cancer. No technique exists to mine *representative* discriminative subgraphs. Hence our baseline for this evaluation is to select the top-k most discriminative subgraphs.

Due to space limitations, it is beyond the scope of this paper to present the mathematical formulations of these importance functions. We direct readers to gSpan [5] and MINDS [2] for the exact definitions of FSM and DSM respectively.

C. Scalability

We first analyze the scalability of RESLING in FSM in the transactional graph setting. We benchmark the scalability of RESLING against dataset size on the ZINC database. As visible in Fig. 7(a), RESLING is more than ten times faster than RING. The high running time of RING results from a clustering step, where it groups a huge number of patterns based on structural similarity. In contrast, due to our design of the EMP, RESLING does not need to perform any clustering, which results in the stark difference between the running times of the two techniques. The gSpan+greedy approach denotes the running time of Alg. 1 after mining the frequent subgraphs



(a) FSM in ZINC (b) FSM in ZINC (c) FSM in AIDS (d) DSM (e) DSM (f) DSM

(g) FSM at 5% (h) FSM at 10% (i) DSM (j) FSM at 5% (k) FSM at 10%

Fig. 7. Growth rate of running time in frequent subgraph mining against (a) database size and (b-c) frequency threshold. Growth rate of running time in discriminative subgraph mining against the (d) network size and (e) the number of samples. Growth of representative powers against k in (f) DSM and (g-h) FSM. (i-k) Representative power across various values of δ.

using gSpan. The greedy approach fails to complete even after 10 hours and hence we denote it as a straight line indicating the time we stopped its execution. The growth rate of RESLING is linear. This is consistent with our theoretical analysis of the running time in Sec. IV-F.

While the dataset size is one aspect that affects the running time, the other aspect with a profound impact is the number of important subgraphs that actually exist in the dataset. To verify this dependency, we measure the running time against the frequency threshold in FSM. As the frequency threshold is lowered, the number of frequent (important) subgraphs increases exponentially. Figs. 7(b)-7(c) present the results. In ZINC, gSpan+greedy fails to complete within 10 hours except at 20% threshold. Compared to RING, RESLING is more than 10 times faster. Even at 1% frequency threshold, where the number of frequent subgraphs exceeds 1 million, RESLING finishes in 8 minutes. The results follow a similar trend on the active set of AIDS molecules. In AIDS, RESLING is up to 100 times faster that gSpan+greedy. These results clearly establish than RESLING is dramatically more efficient than existing techniques including RING.

We next focus on analyzing the scalability of RESLING in the single-network setting by performing DSM. Fig. 7(d) shows the growth of the running time against the size of the network in terms of number of nodes. To construct a network of a desired size, we start a depth-first traversal from a random node in the original, full dataset, and keep adding the traversed nodes to the new dataset till the desired size is reached. Next, all edges between any of the selected nodes are added. The growth rate is slightly larger than linear. Essentially, the running time depends on how quickly the VRRW converges, which in turn depends on the number of clusters of discriminative subgraphs. Overall, RESLING requires 20 minutes to converge. As in FSM, the greedy post-processing approach fails to complete in 10 hours.

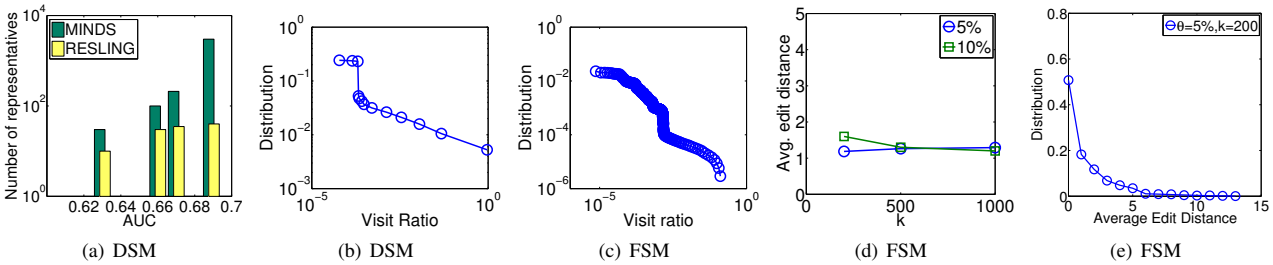
Another factor that affects the running time of RESLING is the cost of computing the importance of a subgraph. In DSM, the importance of a subgraph corresponds to its discriminative potential. The cost of the computing the discriminative potential of a subgraph depends on the number of samples (human being tested for breast cancer) that we need to classify. Hence, we investigate the growth of running time with the number of samples. Fig. 7(e) presents the results. As can be seen, the growth rate is almost linear. This is consistent with the theory since to compute the discriminative potential of a subgraph, a scan is made across all samples in the dataset. Overall, RESLING finishes within 20 minutes, whereas MINDS+greedy fails to complete even in 10 hours.

D. Quality

Scalability is of no use if the quality is poor. Hence, we next focus on evaluating the quality of the answer sets returned. In our problem, the higher the representative power of the answer set, the better is its quality.

Fig. 7(f) presents the representative power (Eq. 2) at various values of k in DSM. We compare the performance of RESLING with the top-k most discriminative subgraphs returned by MINDS [2]. As visible in the plot, RESLING is up to 20 times better. The number of discriminative subgraphs exceeds 100,000. Yet, with just 1000 representatives, RESLING is able to represent 35% of the pattern space. This result concretely brings out the presence of information redundancy in discriminative subgraph mining.

We proceed with same line of experiments in FSM. In this study, we compare the performance of RESLING with RING [18] and the top-k most frequent subgraphs mined by gSpan. RING mines representative subgraphs just like RESLING. However, it is not generalizable to other importance functions. All experiments for FSM in this section is performed on the



(a) DSM (b) DSM (c) FSM (d) FSM (e) FSM

Fig. 8. (a) The number of subgraphs required to reach a particular accuracy level in breast cancer prediction. (b-c) The distribution of node visit counts in VRRW on the EMP. (d) Stability of the mined patterns in terms of average distance between subgraphs of two independent runs. (e) The distribution of subgraph distances between two independent runs.

AIDS dataset since RING fails to scale in ZINC. Figs. 7(g) at 7(h) present the results at 5% and 10% frequency thresholds respectively. As can be seen, not only does RESLING perform better than RING, the rate at which the representative power of RESLING grows is significantly higher than that of RING. The difference between RING and RESLING is more drastic at 5% threshold since the number of frequent subgraphs is larger and hence representing them within a short budget is more difficult. As expected, simply selecting the top-k most frequent subgraphs does not produce good results since they generally belong to a single cluster of highly frequent subgraphs.

How does the quality vary for other values of δ? Figs. 7(i)-7(k) answer this question. Fig. 7(i) presents the results in DSM at $k = 500$ and $k = 1000$. As clearly visible in the plot, if we simply select the top-k discriminative subgraphs as representatives, then the answer set has extremely low representative power unless $\delta \geq 15$. Note that as δ approaches a high value, any subgraph represents the entire pattern space. The task of representing is more difficult at smaller values of δ and easier at larger values. Overall, RESLING is up to 20 times better than selecting the top-k most discriminative subgraphs. Figs. 7(j) and 7(k) analyze the representative powers in FSM at frequency thresholds of 5% and 10% respectively. As can be seen, RESLING represents up to 3 times more frequent subgraphs than RING. The performance gap with RING decreases at 10% frequency threshold since less number of subgraphs are classified as frequent.

Applications: Discriminative subgraphs are often mined from PPI to identify protein modules that are critical for the functioning of a biological process [2], [16]. For example, improper functioning of a module may result in the onset of breast cancer. Therefore, doctors and biological scientists monitor the protein expression levels in these modules to predict diseases. To identify the modules to monitor, discriminative subgraphs are fed as features to a classifier. If they achieve an acceptable level of accuracy, then real life studies on actual biological samples are carried out. Naturally, one would like to minimize the number of discriminative subgraphs (protein modules) since the higher the number, the more is the cost of monitoring them in biological samples. In existing works [2], [16], the focus has solely been on classification quality. Here, we analyze whether our focus on representativeness allows us to reduce the number of subgraphs required to predict breast cancer. Fig. 8(a) presents the results.

In MINDS, the classification accuracy saturates at 0.69 after feeding 3000 discriminative subgraphs to a random forest classifier. The accuracy is measured in terms of the area under the Receiver Operating Characteristic curve (AUC). We repeat the same classification procedure using representative discriminative subgraphs. As can be seen, using only 40 subgraphs, we reach the same accuracy level of 0.69. Furthermore, Fig. 8(a) also shows the growth rate of number of subgraphs required against various accuracy points. At all accuracy levels, RESLING requires far less number of subgraphs. This clearly shows the need to reduce information redundancy in subgraph pattern mining. More importantly, RESLING allows us to do more with less.

E. Other aspects of RESLING

Recall that in RESLING we are approximating the node visit counts, which are required in VRRW, using the space saving algorithm (SSA). In the analysis of SSA, we reasoned that the approximation of the counts would be accurate if the count distribution follows a power law. In the next set of experiments, we study if this property is indeed true in RESLING. Figs. 8(b) and 8(c) present the distribution of node visit counts in DSM and FSM. While the distributions are not exactly power law, they follow a similar trend. More specifically, only a small minority of subgraphs are visited very frequently, while the majority do not receive much repeated visits. This behavior is expected since from existing subgraph mining literature, it is well known that structurally similar subgraphs have similar importance values [14], [15]. The random walker in VRRW thus gets concentrated on these regions, which in turn generates the skewed distributions of visit counts visible in the plots.

Stability: Next, we examine one more dimension of mining representative subgraphs. Since RESLING is a randomized algorithm, we study whether the representative subgraphs returned in two different runs of the algorithm are similar. For that purpose, we run 10 different instances of RESLING on FSM, while keeping all other factors constant. Then, for each subgraph in the i^{th} instance, we find the closest subgraph in the $i{+}1^{th}$ instance. The distance between the i^{th} and the $i{+}1^{th}$ instance is summarized as the average of all the subgraph

distances. Fig. 8(d) presents the average distance across all instances against k. We show the performance at frequency thresholds of 5% and 10%. It is clear from the results that RESLING produces similar answer sets. The average distance is always below 2 and is generally around 1.2 on average. To further establish the stability of RESLING, we also look at the distribution of subgraph distances between two different instances in Fig. 8(e). As can be seen, 50% of the subgraph distances are 0. This means that between two independent runs of RESLING, 50% of the subgraphs are common. Even when the exact same subgraph is not found, 80% of the subgraphs have some subgraph in the other instance within an edit distance of 2.

VI. Related Work

Subgraph mining has been an active area of research for more than 15 years. One of the most popular subareas in this domain is frequent subgraph mining [5]–[7]. As the area matured, the community realized that frequent subgraph mining produces too many subgraph patterns. This motivated the line of work in mining *closed* frequent subgraphs [14] and *maximally* frequent subgraphs [26], [27]. While they are somewhat effective in reducing the number of patterns mined, the number continue to be large. This limitation motivated the development of Origami [17] and RING [18]. Origami and RING use a two-step, post-processing approach like the greedy algorithm (Alg. 1). As already analyzed, this two-step approach does not scale. In addition, Origami and RING cannot be applied for subgraph mining with other popular importance functions such as statistically significant patterns [10], [11], and discriminative patterns [2], [9]. Subsequently, the interest shifted towards mining patterns that better classify labeled graphs. Towards that goal, the idea of discriminative subgraphs [2], [9], [16] and significant subgraphs [10], [11] were proposed. For both these lines of work, no technique exists to mine representative subgraph patterns. While the initial works on frequent subgraph mining focused on transactional graph databases, recently, this problem has been studied [6], [7] on single-large networks, with GRAMI [7] being the state of the art in this space.

VII. Conclusion

In this paper, we formulated the problem of mining representative subgraph patterns from graph databases. The key challenges in this problem were dealing with the exponential subgraph search space and being generic enough to accommodate any importance function and graph dataset type. To overcome them, we developed a generic framework called RESLING *(REpresentative Subgraph sampLING)*, which carefully organizes the exponential subgraph search space in the form of an edit map, where structurally similar subgraphs are naturally in close proximity. RESLING evaluates subgraphs from the edit map in a streaming manner and performs vertex-reinforced random walks to mine k representative subgraph patterns, where k is the budget provided by the user. Extensive experiments on real graph datasets showed that RESLING is indeed able to mine subgraphs that are representative of the pattern space. Compared to the state-of-the-art techniques, RESLING is up to 20 times better in its representative power, and 2 orders of magnitude faster. Overall, RESLING allows us to do more with less.

References

[1] S. Ranu and A. K. Singh, "Indexing and mining topological patterns for drug discovery," in *EDBT*, 2012, pp. 562–565.

[2] S. Ranu, M. Hoang, and A. Singh, "Mining discriminative subgraphs from global-state networks," in *KDD*, 2013, pp. 509–517.

[3] V. Chaoji, S. Ranu, R. Rastogi, and R. Bhatt, "Recommendations to boost content spread in social networks," in *WWW*, 2012, pp. 529–538.

[4] P. Banerjee, S. Ranu, and S. Raghavan, "Inferring uncertain trajectories from partial observations," in *ICDM*, 2014, pp. 30–39.

[5] X. Yan and J. Han, "gspan: Graph-based substructure pattern mining," in *ICDM*, 2002.

[6] M. Kuramochi and G. Karypis, "Finding frequent patterns in a large sparse graph*," *Data Min. Knowl. Discov.*, vol. 11, no. 3, pp. 243–271, 2005.

[7] M. Elseidy, E. Abdelhamid, S. Skiadopoulos, and P. Kalnis, "Grami: Frequent subgraph and pattern mining in a single large graph," *PVLDB*, vol. 7, no. 7, 2014.

[8] S. Gurukar, S. Ranu, and B. Ravindran, "Commit: A scalable approach to mining communication motifs from dynamic networks," in *SIGMOD*, 2015, pp. 475–489.

[9] M. Thoma, H. Cheng, A. Gretton, J. Han, H.-P. Kriegel, A. Smola, L. Song, P. S. Yu, X. Yan, and K. Borgwardt, "Near-optimal supervised feature selection among frequent subgraphs," in *SDM*, 2009, pp. 1076–1087.

[10] M. A. Hasan and M. J. Zaki, "Output space sampling for graph patterns," *PVLDB*, vol. 2, no. 1, pp. 730–741, 2009.

[11] S. Ranu and A. K. Singh, "Graphsig: A scalable approach to mining significant subgraphs in large graph databases," in *ICDE*, 2009.

[12] S. Ranu, B. T. Calhoun, A. K. Singh, and S. J. Swamidass, "Probabilistic substructure mining from small-molecule screens," *Molecular Informatics*, vol. 30, no. 9, pp. 809–815, 2011.

[13] S. Ranu and A. K. Singh, "Mining statistically significant molecular substructures for efficient molecular classification," *J. Chem. Inf. Model.*, vol. 49, pp. 2537–2550, 2009.

[14] X. Yan and J. Han, "Closegraph: Mining closed frequent graph patterns," in *KDD*, 2003, pp. 286–295.

[15] X. Yan, H. Cheng, J. Han, and P. S. Yu, "Mining Significant Graph Patterns by Scalable Leap Search," in *SIGMOD*, 2008.

[16] J. Dutkowski and T. Ideker, "Protein networks as logic functions in development and cancer," *PLoS Comput Biol*, vol. 7, 09 2011.

[17] M. A. Hasan, V. Chaoji, S. Salem, J. Besson, and M. J. Zaki, "Origami: Mining representative orthogonal graph patterns." in *ICDM*, 2007, pp. 153–162.

[18] S. Zhang, J. Yang, and S. Li, "Ring: An integrated method for frequent representative subgraph mining," in *ICDM*, 2009, pp. 1082–1087.

[19] R. Pemantle, "Vertex-reinforced random walk," *Probability Theory and Related Fields*, vol. 92, no. 1, pp. 117–136, 1992.

[20] A. Metwally, D. Agrawal, and A. El Abbadi, "Efficient computation of frequent and top-k elements in data streams," in *ICDT*, 2005, pp. 398–412.

[21] Z. Zeng, A. K. H. Tung, J. Wang, J. Feng, and L. Zhou, "Comparing stars: On approximating graph edit distance," *PVLDB*, vol. 2, no. 1, 2009.

[22] S. Ranu, M. Hoang, and A. Singh, "Answering top-k representative queries on graph databases," in *SIGMOD*, 2014, pp. 1163–1174.

[23] G. Cornuejols, M. L. Fisher, and G. L. Nemhauser, "Location of bank accounts to optimize float: An analytic study of exact and approximate algorithms," *Management Science*, vol. 23, no. 8, pp. 789–810, 1977.

[24] L. Page, S. Brin, R. Motwani, and T. Winograd, "The pagerank citation ranking: Bringing order to the web," in *WWW*, 1998, pp. 161–172.

[25] Q. Mei, J. Guo, and D. Radev, "Divrank: the interplay of prestige and diversity in information networks," in *KDD*, 2010.

[26] J. Huan, W. Wang, J. Prins, and J. Yang, "Spin: Mining maximal frequent subgraphs from graph databases," in *KDD*, 2004, pp. 581–586.

[27] L. Thomas, S. Valluri, and K. Karlapalem, "Margin: Maximal frequent subgraph mining," in *ICDM*, 2006, pp. 1097–1101.