Knowl Inf Syst (2018) 54:123–149
https://doi.org/10.1007/s10115-017-1129-y

REGULAR PAPER

RESLING: a scalable and generic framework to mine top-*k* representative subgraph patterns

Dheepikaa Natarajan[1] · **Sayan Ranu**[2]

Received: 15 March 2017 / Revised: 11 July 2017 / Accepted: 23 October 2017 /
Published online: 8 November 2017

Abstract Mining subgraph patterns is an active area of research due to its wide-ranging applications. Examples include frequent subgraph mining, discriminative subgraph mining, statistically significant subgraphs. 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. RESLING derives its power by structuring the search space in the form of an *edit map*, where each subgraph is a node, and two subgraphs are connected if they have an edit distance of one. We rank nodes in the edit map through two random walk based algorithms: *vertex-reinforced random walks (*RESLING-*VR)* and *negative-reinforced random walks(*RESLING-*NR)*. Experiments show that RESLING-VR is up to 20 times more representative of the pattern space and two orders of magnitude faster than the state-of-the-art techniques. RESLING-NR further improves the running time while maintaining comparable or better performance in representative power.

Keywords Graph mining · Random walk · Diversity · Representative patterns

✉ Sayan Ranu
sayanranu@cse.iitd.ac.in

1 Machine Learning Team, Amazon, Seattle, USA

2 Department of CSE, IIT Delhi, New Delhi, India

1 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 [4,5]. Mining subgraphs of interest has received considerable interest in the graph mining community due to its wide-ranging applications. 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 [6–9], discriminative subgraphs [2,10], statistically significant subgraphs [11–14].

The applications of mining subgraphs span multiple areas. Frequent subgraph mining has been widely used for drug discovery [6,15,16]. Specifically, given a dataset of molecules that are active against a particular disease, chemists are often interested in identifying the molecular substructures that are frequent in this set. This same line of reasoning evolved into discriminative [10] and statistically significant subgraph mining [11,12,17,18], where the goal is to mine subgraphs that are "overrepresented" in the active dataset. Both discriminative subgraph mining and significant subgraph mining have shown good performance in molecular activity prediction [10,18]. Beyond drug discovery, subgraph mining has also been used for bug localization [19] and predicting disease susceptibility from gene expression data in protein–protein interaction networks [2,20].

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. Similarly, biologists are interested in identifying protein modules (important subgraphs) that can predict susceptibility to a disease [2, 20]. Monitoring thousands of modules is not a financially viable solution. Furthermore, subgraph patterns often serve as the platform for higher-order tasks like classification, structural alignment of proteins, where mining all patterns is not necessary; a small and diverse set of patterns is enough [21]. Thus, 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 antiviral 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. Further details on the dataset is provided in Sect. 7. Figure 1 shows that the number of frequent subgraphs grows exponentially with decrease in θ. At 5% frequency threshold, 252,331 subgraphs are mined, which is clearly beyond the scope of any human-mediated analysis. The scenario does not change in other form of subgraph mining [2].
2. *Information Redundancy* The answer sets also suffer from redundancy. Subgraphs that are structurally similar have similar importance values [17,22]. Consequently, the answer

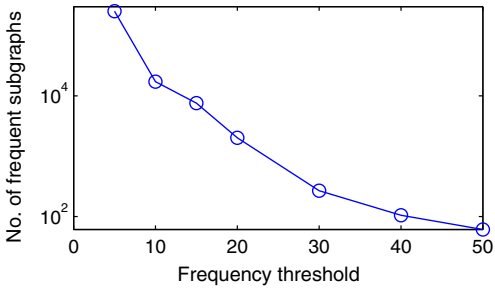


Fig. 1 The growth of the answer set size with frequency threshold in frequent subgraph mining. The *y*-axis is in log scale

set is overloaded with similar subgraphs. Providing such redundant subgraphs reduces the productivity of domain scientists since they manually need to compress the answer set into a more concise and informationally dense subset of patterns.

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 [6,12] and 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 [2,8,20]. 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. A straightforward heuristic is therefore to first mine all subgraphs that satisfy the threshold criteria and then cluster them into *k* groups. The cluster centroids would form the answer set. Unfortunately, such a strategy does not scale. Computing the *edit distance* between two graphs is NP-hard [23]. Since clustering requires us to compute at least $O(n^2)$ edit distances in a set of n subgraphs, the overall approach is not scalable.

To resolve all of the above challenges, we develop a technique called RESLING (*RE*presentative *S*ubgraph sampLING). While two techniques exist to mine representative frequent subgraphs [21,24], 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.

Figure 2 outlines the flow of RESLING. Given a graph database, which can either be the transactional setting or the single large network setup, 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

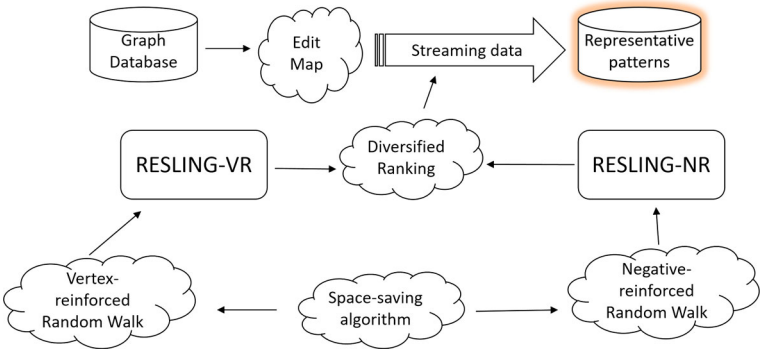


Fig. 2 Pipeline of RESLING

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 two ranking techniques *vertex-reinforced random walk (*RESLING-*VR)* and *negative-reinforced random walk (*RESLING-*NR)*. While RESLING-VR is developed by us in our earlier work [25], RESLING-NR further improves the ranking algorithm by being more efficient while maintaining comparable or better performance in representative power. Both ranking algorithms are supported by the *Space-saving algorithm* [26] to scale to exponential search spaces, which are treated as a data stream. 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. Ours is the first formulation that is flexible enough to accommodate any importance function and graph dataset.
- 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.
- RESLING-NR, based on negative-reinforced random walks, further improves RESLING-VR [25] in both running time and quality.

2 Problem formulation

In this section, we define the concepts central to our paper.

A graph $G = \{V, E\}$ is composed of a set of vertices $V = \{v_1, \ldots, v_n\}$ 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 $\mathbb{D} = \{G_1, \ldots, G_m\}$. 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*. Transactional graph datasets are commonly used to characterize chemical compounds and function-call graphs. Single large networks are more popular in modeling protein–protein interaction networks, social networks, etc. As evident from its name, single large networks typically are much larger in sizes than graphs encountered in the transactional setting.

Definition 2 (*Subgraph of a database*) 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 ref [2,6,7], 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 [6], discriminative potential [2,10,20], or statistical significance of g [11,12].

As illustrated earlier, subgraph mining is performed with various importance functions. We therefore introduce the generic definition of a *important subgraph*.

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. Toward that goal, we define the δ-neighborhood of a subgraph.

Definition 3 (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* [23]. In other words, g is a structural representative of all subgraphs in its δ-neighborhood. In edit distance, the distance between two graphs g, g' is the minimum number of "edits" required to convert g to g'. An edit is either deletion or addition of an edge, or deletion or addition of a node. 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 $\mathbb{S}$.

Definition 4 (*Representative power*) The representative power $\pi(\mathbb{S})$ of a set of graphs $\mathbb{S}$ with respect to the set of important graphs $\mathbb{T}$ and a distance threshold δ, is the proportion of $\mathbb{T}$ that is represented by $\mathbb{S}$. More specifically,

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

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

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

In essence, the proposed model captures as much of the various groups of important subgraphs as possible within the budget k. Similar models to capture representative power have been studied before in similarity queries [27,28], but not in graph mining.

3 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 [27].

Fortunately, $\pi(\mathbb{S})$ is a *submodular* function. A submodular function f is a set function that satisfies the following condition. The marginal gain from adding an element to a set S is at least as high as the marginal gain obtained by adding the same element to a superset of S. Mathematically,

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

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

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

PROOF BY CONTRADICTION: Assume,

$$\pi(T \cup \{g\}) - \pi(T) > \pi(S \cup \{g\}) - \pi(S) \tag{5}$$

where S is a subset of T and both S and T are sets of graphs and $g \in \mathbb{T}$ is the graph being added.

$$\begin{aligned} &|N(\{g\}) \setminus N(T)| > |N(\{g\}) \setminus N(S)| \\ &\text{or, } g \nsubseteq T \end{aligned} \tag{6}$$

which is a contradiction to the assumption that $S \subseteq T$. □

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 $(1 - \frac{1}{e})$ [29].

3.1 The greedy approach

Based on Theorem 2, the following two-step approach can be adopted. Algorithm. 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

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

```
1: Compute 𝕋 based on importance function φ(.)
2: ℝ ← ∅
3: while |ℝ| < k do
4:    g* ← arg max_g {π(ℝ ∪ {g}) − π(ℝ) | g ∈ 𝕋}
5:    ℝ ← ℝ ∪ g*
6:    for g ∈ 𝕋 \ ℝ do
7:       N(g) ← N(g) \ N(g*)
```

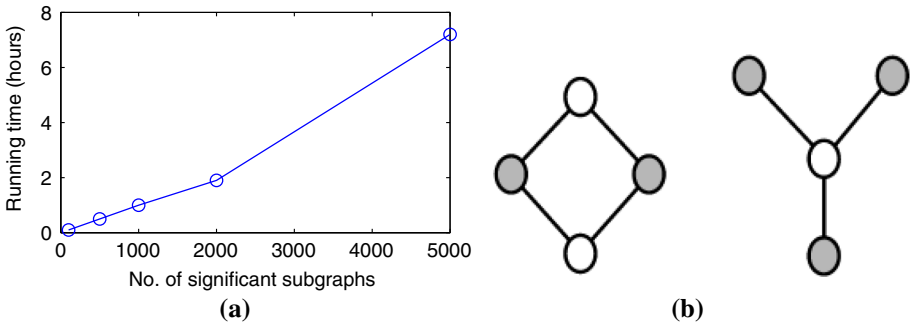


Fig. 3 **a** The growth of the running time with the size of $\mathbb{T}$. **b** Example of a graph database containing two graphs. The node color denotes their labels

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 [23]. Hence, the algorithm is not scalable when $\mathbb{T}$ is large, which is the case for most subgraph mining problems.

To empirically establish the non-scalability of the baseline-greedy approach, we plot the growth of the running time with the size of $\mathbb{T}$. Figure 3a presents the results. As shown, it takes more than 7 h 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. Clearly, we need to devise a more efficient technique than the greedy algorithm.

An obvious approach to scale the baseline-greedy algorithm is to index the subgraphs in $\mathbb{T}$ so that the neighborhood update step can be performed faster [27,28,30]. However, indexing $\mathbb{T}$ is not feasible. $\mathbb{T}$ is computed at runtime when the importance function and threshold is provided, and thus violates the assumption of a static database in existing graph index structures. Alternatively, one can build the index structure at runtime after $\mathbb{T}$ is computed. However, building the index structure is an even more expensive procedure and the computation time can run into days [27].

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.*

4 RESLING

To address the weaknesses of the baseline-greedy algorithm, we develop a sampling-based framework called RESLING to directly mine the top-k representative patterns.

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*

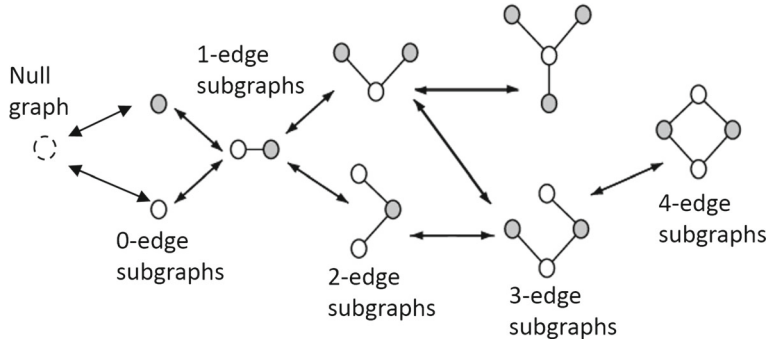


Fig. 4 The edit map of the example database in Fig. 3b. Each subgraph is a node in the EMP

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)*.

4.1 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

$$w(g, g') = \phi(g') - \phi(g) \tag{7}$$

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 5 (*Edit Map*) Edit map of a graph database $\mathbb{D}$ is an edge-weighted graph $M = (V_M, E_M)$, where $V_M = \{g \mid g \subseteq \mathbb{D}\}$, $E_M = \{(g = (V, E), g' = (V', E')) \mid \text{either } g' \supseteq g, E' = E \cup \{e\}, \text{ or } g' \subseteq g, E' = E \backslash \{e\}, e \in E\}$.

Example 1 Figure 4 shows the edit map corresponding to the database in Fig. 3b. 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 gray and the white. These 0-edge subgraphs are connected to the null graph and the 1-edge subgraphs. The 1-edge

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

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 [17,22]. 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.

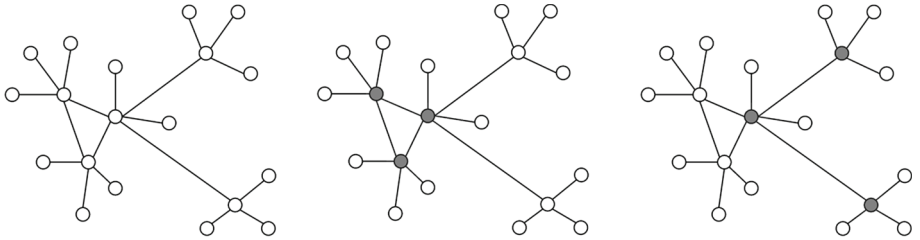
Connection of EMP to past work EMP-like structures have been studied in past work [2,6,11]. For example, the lattice of the depth-first traversal of the search space in frequent subgraph mining, first proposed by gSpan [6], is similar to EMP with the exceptions of being a tree and not having edge weights. The novel contribution of our work is therefore not much in proposing the EMP. Rather, we innovate in identifying and exploiting the above properties of the EMP, which lie at the core of our ability to scale to exponential search spaces and is the key contribution of our paper. Specifically, we design a ranking algorithm that exploits the "Proximity" property of EMP. Furthermore, RESLING takes local decisions to converge toward a global solution, and hence, the exponential size of the EMP is not a bottleneck.

4.2 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 [31] 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. Furthermore, structurally similar subgraphs are likely to have similar importance values [2,6,17]. Thus, communities of important subgraphs would strongly attract the random walker within it receiving high PageRank scores. However, PageRank does not reward representativeness.

Example 2 Consider the example network shown in Fig. 5a. Let the task be to summarize information of the whole network by the top three nodes. For simplicity, assume that all edge weights are equal. Figure 5b shows the three shaded nodes that would receive the highest PageRank. This results from the fact that the shaded nodes are closely connected to each other and are present in the largest community. The result that we would rather like to achieve is shown in Fig. 5c. Figure 5c is preferable since the shaded nodes capture all three communities in the network.

To address this weakness of PageRank, we employ two ranking algorithms on the edit map. In the first algorithm, we perform *vertex-reinforced random walks (VRRW)* [32] on the EMP. Our second sampling algorithm is based on *negative-reinforced random walks (NRRW)* [33].



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

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

In both these ranking algorithms the goal is not only to reward high centrality (or subgraph importance), but also representative power. It is to be noted that the second ranking algorithm based on negative-reinforced random walks is the new contribution from our earlier work on vertex-reinforced random walks [25].

5 RESLING-VR

RESLING-VR ranks nodes in the EMP through vertex-reinforced random walks.

5.1 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.

Before explaining the mathematical foundation behind VRRW, we first explain the general intuition. In a real-world scenario, it is reasonable to consider that in random walk, the transition probabilities change over time. Indeed, a visitor is more likely to visit a restaurant that has already been visited by many other people; people tend to read books that has already been read by many; an actor accumulates prestige when acting in various roles and the accumulated prestige in turn helps him to get even more opportunities. One particular family of time-variant random walk processes is known as the vertex-reinforced random walks (VRRW). The basic assumption is that the transition probability to one state from its neighbors is reinforced by the number of previous visits to that state which is true intuitively.

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 $i = 1, \ldots, n$. 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.

$$p_T(u, v) = (1 - \lambda)\frac{1}{|V_M|} + \lambda\frac{p_0(u, v)N_T(v)}{D_T(u)} \tag{8}$$

where $D_{\mathrm{T}}(u) = \sum_v p_0(u, v)N_{\mathrm{T}}(v)$ 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 [32] after a large T, i.e.,

$$\tau(v) = \sum_{u \in V} p_{\mathrm{T}}(u, v)\tau(u) \tag{9}$$

Furthermore, $\sum_{\forall v \in V} \tau(v) = 1.$, where V is the set of all nodes. Also note that "Rich gets richer" phenomena holds only when there exists a self-edge. Self-edge to a node ensures that even if all its neighbors shrink, its score will still be large as long as its number of visits is large.

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 ($N_{\mathrm{T}}(v)$). 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 [34].

5.2 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 [6]. Thus, if we simply use change in frequency as the edge weight, the random walker would always be biased toward 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 toward 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 $X_{\mathrm{T}} = g = (V_g, E_g)$ 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.

$$p_{\mathrm{T}}(g, h) = 0.5 \frac{w(g, h)N_{\mathrm{T}}(h)}{\sum_{g' \in g_{sup}} w(g, g')N_{\mathrm{T}}(g')} \tag{10}$$

where $w(g, h)$ (Eq. 7) 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 toward "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 toward

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

$$p_T(g, h) = 0.5 \frac{N_T(h)}{\sum_{g' \in g_{sub} \cup \{g\}} N_T(g')} \tag{11}$$

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.

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.

5.3 Managing the exponential search space

Section 5.2 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 toward a global solution. However, a scalability issue remains. Specifically, to compute the transition probability $p_T(g, h)$ to any neighboring supergraph h, we not only need to compute the importance of g and h, but also all other supergraph neighbors of g. This is necessary due to the $w(g, g')$ term in the denominator of Eq. 10, which is essentially the normalization factor. Computing the importance of a subgraph is often time consuming. For example, in frequent subgraph mining, we need to perform subgraph isomorphism tests on all graphs in the database [6]. 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) [26].

5.3.1 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.

Algorithm 2 SSA.add(g)

Require: g is the current node (subgraph) to be updated
Require: **V** is the count vector that is being maintained in descending order
1: **if** **V** contains less than M nodes **then**
2: Add g to **V** with count 2.
3: **else**
4: **if** $g \in$ **V then**
5: $Count(g) \leftarrow Count(g) + 1$
6: **else**
7: $g_M \leftarrow$ least frequent subgraph in **V**
8: Replace L with g
9: $Count(g) \leftarrow Count(g_M) + 1$
10: Compute and store neighborhood of g

Toward 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 [6], 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, $M = \frac{8\times10^9}{128\times10001+8} = 6.92 \times 10^4$.

As known in the stream processing literature [26], it is not possible to compute the top-M frequent items in a stream optimally. However, the space-saving algorithm (SSA) [26] provides good approximations along with some guarantees on the reported counts. Here we explain how the SSA is adopted for RESLING. Algorithm 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

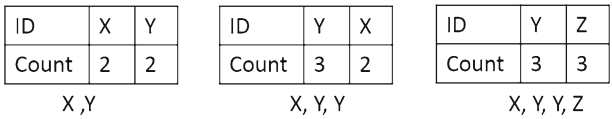


Fig. 6 Example of updates to count vector at $M = 2$

to 2. Recall, that in VRRW, $N_0(v) = 1$ for all nodes. 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. Specifically, if there are few nodes that are highly frequent, they will always remain in the count vector. The replacements would affect only the infrequent nodes that reside on the tail of the count vector. This intuition is correct, and it has been shown that SSA performs best if the frequency distribution follows a power law [26]. This is indeed the case with node visits in VRRW. We show empirical evidence in Sect. 7. Consequently, the approximation is both accurate and scalable.

5.4 The RESLING-VR algorithm

With the formalization of the SSA, the RESLING algorithm based on vertex-reinforced random walks, which we refer to by the name RESLING-VR, is complete. The VRRW starts from a randomly selected subgraph in the database (Algorithm 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 in the count vector are returned as the representative set (lines 18 and 19).

Space complexity During runtime, RESLING stores the SSA count vector $\mathbf{V}$ 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.

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

Ensure: $\mathbb{R}$ is the representative set
Initialize SSA count vector
$T := 0$
$X_T \leftarrow$ A randomly selected subgraph of $\mathbb{D}$
Compute $\phi(X_T)$
repeat
  if $uniform(0, 1) > \lambda$ **then**
    $h \leftarrow$ randomly selected subgraph of $\mathbb{D}$
  else
    if $uniform(0, 1) \leq 0.5$ **then**
      $h \leftarrow$ selected subgraph through an edge delete or self-edge (Eq. 11)
      SSA.add(X_T)
    else
      $h \leftarrow$ randomly selected proposed edge addition
      SSA.add(X_T)
  $T \leftarrow T + 1$
  $X_T \leftarrow h$
until convergence of count vector
$\mathbb{R} \leftarrow k$ most visited important subgraphs in $\mathbf{V}$
return $\mathbb{R}$

1. First, we compute the importance of the subgraph. The cost of computing the subgraph importance depends on the importance function. For example, in frequent subgraph mining [6], the cost is linear to the graph database size. 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 substeps 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 $O(I(S + \text{NR} + M))$, where I is the total number of iterations.

6 RESLING-NR

In this section, we develop the second diversified node ranking mechanism based on negative reinforcement.

6.1 Negative-reinforced random walk (NRRW)

RESLING-NR is based on negative reinforcement [33] and is similar to VRRW. In VRRW, transition probabilities are reinforced based on number of visits which results in "rich gets richer" phenomenon. In NRRW, transition probabilities to the nodes that have been already selected as representatives during the course of the random walk are *negatively* reinforced.

To illustrate the mathematical formulation, let the initial answer set be $\mathbb{R} = \phi$. To find the first ranked node, we run standard personalized PageRank algorithm and pick the top scoring

node, i.e., the node that has been visited maximum number of times and add to answer set $\mathbb{R}$. It is crucial that the next item we select is far from the first ranked item. Hence, the transition probabilities are altered such that the nodes in answer set $\mathbb{R}$ are not reachable during the random walk, which in turn reduces the scores of their neighbors as well. However, nodes that are far from the ranked items remain unaffected.

Let the current node in the random walk be u. Probability of choosing a node v from u is given below.

$$p(u, v) = \begin{cases} (1-\lambda)\frac{1}{|V_M \setminus \mathbb{R}|} + \lambda \frac{p_0(u,v)}{D_T(u)}, & \forall v \notin \mathbb{R} \\ 0, & \forall v \in \mathbb{R} \end{cases} \tag{12}$$

where V_M is the set of nodes in the network, $D_T(u) = \sum_{v \in V_M \setminus \mathbb{R}} p_0(u, v)$ is the normalizing term and $p_0(u, v)$ is the initial transition probability from u to v.

Following the selection of the node with the highest PageRank, the answer set $\mathbb{R}$ is updated, and the process is repeated till $\mathbb{R}$ reaches the size of k.

6.2 NRRW on edit map

For the same reasons as mentioned in VRRW on Edit Map, with 0.5 probability we perform a delete or stay in the current subgraph and with 0.5 probability we add an edge to the current subgraph.

Addition of an edge Edge additions are similar to that of VRRW, the main difference being that the nodes (or subgraphs) that are already chosen as representative elements have 0 probability of being visited again. Let the current state in NRRW 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.

$$p(g, h) = \begin{cases} 0.5\frac{w(g,h)}{\sum_{g' \in g_{sup} \setminus \mathbb{R}} w(g,g')}, & \forall h \notin \mathbb{R} \\ 0, & \forall h \in \mathbb{R} \end{cases} \tag{13}$$

where $w(g, h)$ (Eq. 7) is the edge weight and g_{sup} is the set of all supergraphs of g containing one more edge and $\mathbb{R}$ is the representative set. Note that if the supergraph h of g has already been selected as a ranked item, the transition probability to all those supergraphs is set to 0 and those probabilities are distributed among the nodes that have not been selected.

Deletion of an edge and self-edges Edge deletions are also similar to that of VRRW (Eq. 11) with minor modifications. To incorporate exploration, edge deletions or staying at the same node through self-edges are weighted uniformly as in the case of VRRW. Mathematically, let $X_T = g = (V_g, E_g)$ be the current state. The probability of an edge delete or self-edge is

$$p(g, h) = \begin{cases} 0.5\frac{1}{\sum_{g' \in [g_{sub} \cup \{g\}] \setminus \mathbb{R}} 1}, & \forall h \notin \mathbb{R} \\ 0, & \forall h \in \mathbb{R} \end{cases} \tag{14}$$

where g_{sub} is the set of all subgraphs of g containing one less edge, $h \subseteq g$ is the resulting subgraph following the edit and $\mathbb{R}$ is the representative set of subgraphs.

6.3 The RESLING-NR algorithm

Like RESLING-VR, RESLING-NR also uses SSA as it involves selecting the most visited node during PageRank. First we initialize the count vector. RESLING-NR starts with empty set $\mathbb{R}$ from a randomly selected subgraph in the database (Algorithm 4, line 5). Next transition is chosen based on NRRW principles (lines 8–14). After the transition takes place the subgraph is

Algorithm 4 RESLING-NR($\phi(\cdot), k$)

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

added to the count vector. Following the transition, the new subgraph becomes the current state and the process is repeated until the convergence of count vector (PageRank convergence). Next, the node with the highest count is chosen and added to the answer set $\mathbb{R}$. This process (lines 3–19) is repeated until there are k chosen items in the answer set.

Space complexity This is similar to that of RESLING-VR. During runtime, RESLING-NR stores the SSA count vector $\mathbf{V}$ of size M and the database of graphs $\mathbb{D}$. Thus, the total space complexity is $O(M + |\mathbb{D}|)$. Since we are limiting the number of iterations for PageRank to P, count vector size could be much lesser than that of RESLING-VR.

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

1. First, we compute the importance of the subgraph. The cost of computing the subgraph importance depends on the importance function. For example, in frequent subgraph mining [6], the cost is linear to the graph database size. 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 substeps 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 when doing standard random walk, and the whole process is repeated till we get top-k nodes the total time complexity is $O(kP(S + NR + M))$, where P is the number of iterations run for pageRank.

7 Experiments

In this section, we demonstrate that

- *Adaptability* RESLING is generic enough to tackle multiple forms of graph datasets and importance functions.
- *Quality* RESLING produces results that are up to 20 times more representative of the pattern space than selecting the top-k most important subgraphs.
- *Scalability* RESLING is up to two orders of magnitude faster than baseline techniques.
- RESLING-*VR versus* RESLING-*NR* We show the RESLING-NR is faster than RESLING-VR while providing comparable or better performance than RESLING-VR in terms of representative power.

7.1 Datasets

As discussed in Sect. 2, there are two kinds of graph datasets: the transactional setting and the single-network setting. Note that in the transactional setting, although the sizes of the graphs are small, the complexity arises from the number of graphs that reside in the database. In contrast, in the single network setting, the size of the graph is the source of complexity. For either scenario, one cannot be called to be harder than the other since each brings in their unique challenges. For a thorough evaluation of RESLING, we use both.

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 41,176 *inactive* molecules. This dataset has been extremely popular in previous subgraph mining works [6,11,12]. The dataset can be downloaded from http://dtp.nci.nih.gov/.

We use this dataset to evaluate the performance of RESLING in frequent subgraph mining (FSM). Given a frequency threshold θ, a subgraph g is frequent, if more than $\theta\%$ of graphs in the database contain g.

Single-Network dataset We choose one of the largest publicly available protein–protein interaction networks (PPI) [2,20]. We choose PPI since motif or subgraph mining in PPI has been an active area of research. The PPI contains 11,203 vertices and 57,235 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. Three hundred and seventy-one human beings were studied. Each human either has breast cancer or does not. Furthermore, for each human being, each protein in the PPI is tagged with a binary class label: normal activity or abnormal activity. Thus, in this dataset, we have 371 different snapshots of the network corresponding to each human being. All snapshots have the same structure. However, each snapshot has a global class label denoting the presence of breast cancer, and local vertex labels denoting the functioning of the corresponding proteins.

7.2 Experimental setup

All our experiments are performed on a 3.4 GHz quad-core i7 machine running Ubuntu 14.04 with 16 GB 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. 8) is set to 0.95.

To demonstrate the adaptability of RESLING-VR and RESLING-NR, 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 θ% 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. As discussed in Sect. 1, no technique exists to mine representative subgraphs pattern in a generic manner across any given importance function and dataset type. In the specific domain of frequent subgraph mining (FSM), Origami [21] and RING [24] 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 [24], 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.

It is beyond the scope of this paper to present the mathematical formulations of these importance functions. We direct readers to gSpan [6] and MINDS [2] for the exact definitions of FSM and DSM, respectively.

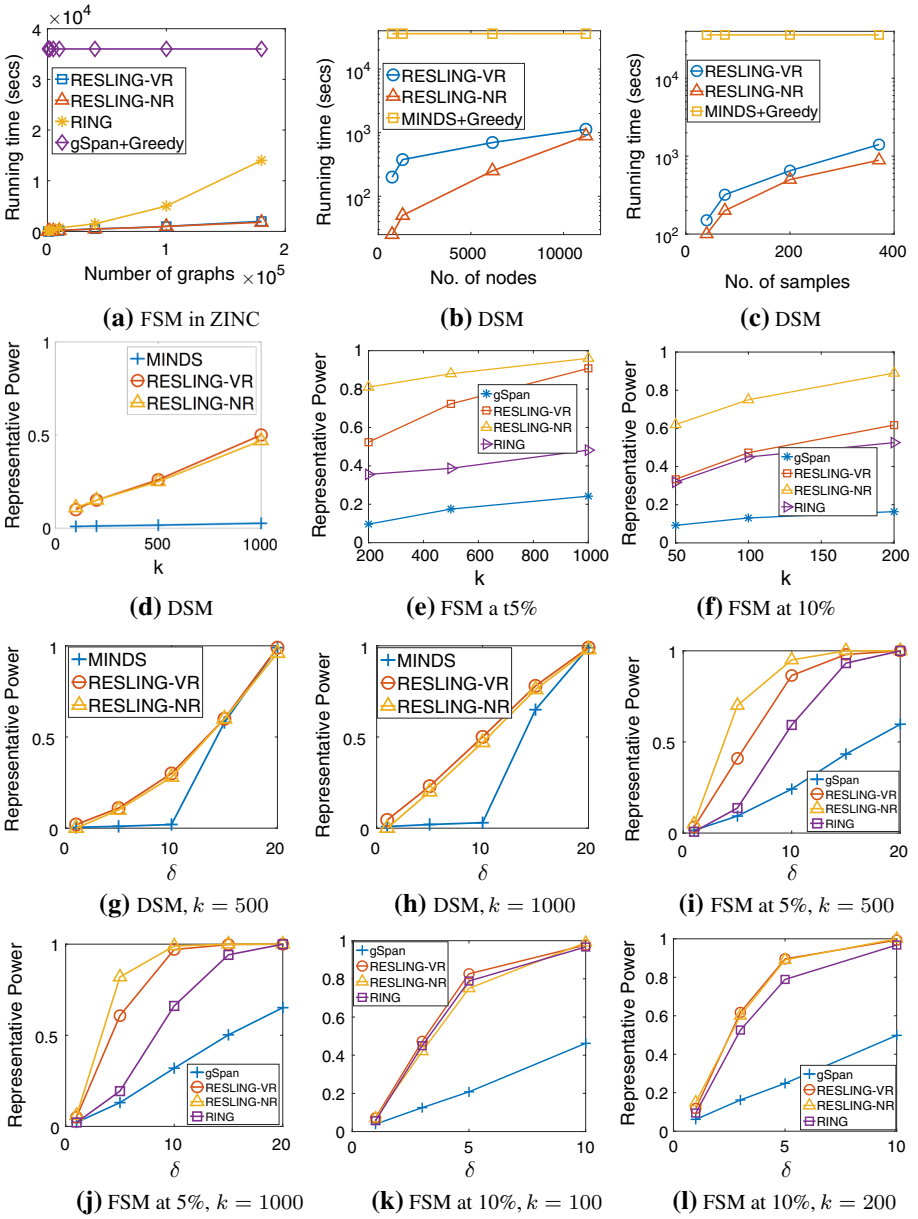
7.3 Scalability

Not only is the problem of mining representative subgraphs NP-hard, even the greedy algorithm discussed in Sect. 3.1 does not scale. Given this context, 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. No other subgraph mining technique has been evaluated on a transactional dataset as large as ZINC. As visible in Fig. 7a, both RESLING-VR and RESLING-NR are 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, both ranking algorithms in the RESLING framework do 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 Algorithm 1 after mining the frequent subgraphs using gSpan. The greedy approach fails to complete even after 10 h, 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 Sect. 5.4.

Although RESLING-NR is marginally faster than RESLING-VR, this difference in efficiency is not visible in the plot. In discriminative subgraph mining (DSM), however, this gap is more prominently visible, which we discuss next.



(a) FSM in ZINC **(b)** DSM **(c)** DSM

(d) DSM **(e)** FSM a t5% **(f)** FSM at 10%

(g) DSM, $k = 500$ **(h)** DSM, $k = 1000$ **(i)** FSM at 5%, $k = 500$

(j) FSM at 5%, $k = 1000$ **(k)** FSM at 10%, $k = 100$ **(l)** FSM at 10%, $k = 200$

Fig. 7 **a** Growth rate of running time in frequent subgraph mining against database size. 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 **d** DSM and **e**, **f** FSM. **g**–**l** Representative power across various values of δ

Figure 7b 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 ranking converges in RESLING-VR and RESLING-NR, which in turn depends on the number of clusters of discriminative subgraphs. Overall, both techniques in the RESLING framework requires less than 20 min with RESLING-NR being significantly faster in smaller networks.

RESLING-NR is faster since it consumes less number of iterations to converge. The faster convergence of RESLING-NR is a direct consequence of negative reinforcement being more effective than vertex reinforcement in ensuring diversity in the random walk. More specifically, in VRRW, diversity propagates in the random walk once a node gets "rich" enough to draw the ranking of all its neighborhood as its own. On the other hand in NRRW, as soon as a node is selected using Pagerank, its ranking is turned negative and this negative ranking flows to its neighborhood. When the network is not too large, the flow of negative ranking to neighborhood is faster than a node consuming the ranking of its neighborhood in VRRW. Consequently, RESLING-NR is faster. As in FSM, the greedy post-processing approach fails to complete in 10 h.

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. Figure 7d 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, even on the entire dataset, both RESLING-VR and RESLING-NR finish within 20 min. As in the previous experiment, RESLING-NR is faster since it takes less number of iterations to converge. We attempted running the baseline-greedy algorithm (Algorithm 1). However, it did not finish even after 10 h due to the large number of discriminative patterns mined.

7.4 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.

Figure 7d presents the representative power (Eq. 2) at various values of k in DSM. We compare the performance of RESLING-VR and RESLING-NR with the top-k most discriminative subgraphs returned by MINDS [2]. As visible in the plot, both ranking algorithms in the RESLING framework produce similar representative power, which is up to 20 times better than MINDS. 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 the RESLING framework with RING [24] 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 AIDS dataset since RING fails to scale in ZINC. Figure 7e, f presents the results at 5 and 10% frequency thresholds, respectively. Two observations stand out in this plot. First, RESLING-NR is significantly better than RESLING-VR in FSM, particularly at low values of k. Drilling down further into the results, we observe that in RESLING-NR, the

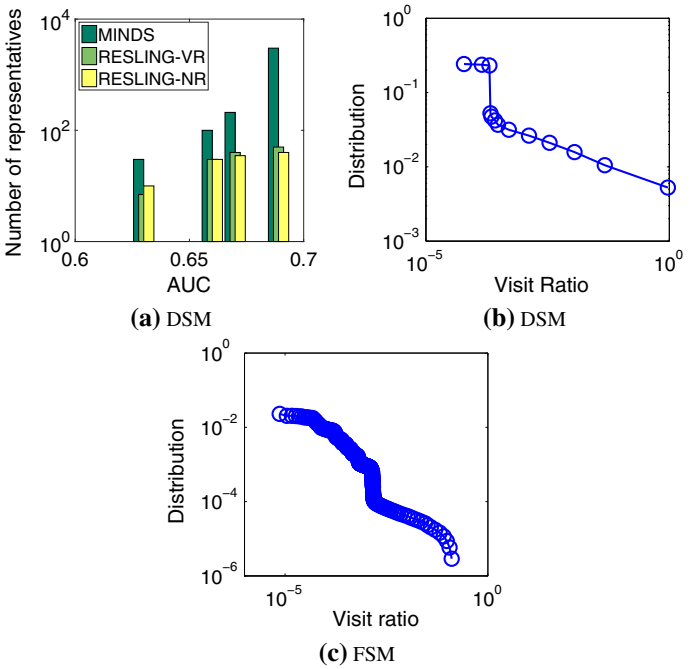
initial selection of subgraphs (or nodes in EMP), which are purely based on Pagerank, is of much higher representative power than the nodes selected through RESLING-VR. However, as more and more subgraphs are selected, the need for diversity among the selected subgraphs become more necessary and the gap in the performance of the two ranking algorithms starts to diminish.

The second prominent observation from Fig. 7e, f is that 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. While it is hard to pinpoint why RING's growth rate is slower than RESLING, we suspect this stems from the fact that RING performs the analysis in the feature space. More specifically, RING converts graphs into feature vectors and then performs a clustering based selection. This feature space conversion is necessary since, like in RESLING, computing a large number of edit distance computations between graphs is not scalable. While converting graphs to feature space, loss of information is inevitable. In contrast, RESLING negates the need to compute edit distances by organizing graphs in the form of an edit map, where similar graphs are naturally in close proximity. Thus, the entire mining process remains within the graph space. 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? Figure 7g–l answers this question. Figure 7g, h presents the results in DSM at $k = 500$ and $k = 1000$, respectively. 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. Figure 7i–l analyzes the representative powers in FSM at frequency thresholds of 5 and 10%. As can be seen, the ranking algorithms within the RESLING framework represent up to three 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. Consistent with previous experiments, RESLING-NR displays better quality than RESLING-VR in FSM.

An interesting observation from the above experiments is that RESLING-NR shows significant improvement over RESLING-VR in representative power only in FSM. While it is hard to pinpoint the exact reason, the behavior is likely to stem from the size of the search spaces in the two problem domains. IN DSM, the subgraph space is much larger. Hence, the need to ensure diversity among subgraphs of high representative powers is of less importance as two randomly picked representative subgraphs are naturally far apart from each other. In contrast, owing to smaller search spaces in FSM, there are less number of clusters. Since each representative subgraph is essentially like cluster centers, the need to ensure that we do not select multiple representatives of the same clusters is of high importance. In this aspect, NRRW performs better than VRRW to ensure diversity.

Applications In the next experiment, we highlight a prominent application of mining representative subgraphs. Discriminative subgraphs are often mined from PPI to identify protein modules that are critical for the functioning of a biological process [2,20]. In case of abnormalities in these modules, disease set in. 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



(a) DSM **(b)** DSM

(c) 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

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 [2,20], 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. Figure 8a 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. 8a also shows the growth rate of number of subgraphs required against various accuracy points. At all accuracy levels, RESLING-VR and RESLING-NR require 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. We however note that even in RESLING, we could not go beyond an accuracy of 0.69. This probably results from the fact that representative set do not have any additional information over entire set of patterns. Nevertheless, due to rewarding representativeness, there is less information redundancy in the mined subgraphs, which results in the shown plot.

7.5 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, where few nodes are visited frequently, but the vast majority of nodes are visited rarely. In the next set of experiments, we study if this property is indeed true in RESLING. Figure 8b, c presents 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 [17,22]. 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. Consequently, SSA also produces good approximations. On the whole, these plots reveal an important reason behind the good performance of the RESLING framework.

7.6 Summary

Overall, the experimental results clearly establish that RESLING is scalable, effective in representing the pattern space using a small number of exemplars and can be applied for other higher-order tasks such as network classification with good results. Among the two random-based ranking algorithms proposed in this paper, it would be safe to say that RESLING-NR produces a stronger performance. In addition to being more efficient than RESLING-VR in running time, RESLING-NR produces representative powers that are often better.

8 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 [6–8]. Frequent subgraph mining generated significant interest in the research community due to their applications in a large number of areas. The main computational challenge in frequent subgraph mining is to analyze the exponential subgraph search space. To scale frequent subgraph mining, the mining community exploited techniques from frequent itemset mining since they both share the *apriori* property. As in itemset mining, there are two different approaches to frequent subgraph mining: the depth-first approach adopted by gSpan [6] and Gaston [16], and the breadth-first approach adopted by FSG [15].

As the area matured, the community realized that frequent subgraph mining produces too many subgraph patterns. In fact, the number of patterns is often larger than the size of the graph database itself. This motivated the line of work in mining *closed* frequent subgraphs [22] and *maximally* frequent subgraphs [35,36]. 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 [21] and RING [24]. Origami and RING use a two-step, post-processing approach like the greedy algorithm (Algorithm 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 [11,12], and discriminative patterns [2,10].

Subsequently, the interest shifted toward mining patterns that better classify labeled graphs. Toward that goal, the idea of discriminative subgraphs [2,10,20] and significant subgraphs [11,12] 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, and recently, this problem has been studied [7,8] on single large networks, with GRAMI [8] being the state of the art in this space.

In terms of the techniques proposed in this work, sampling-based approaches have been employed to mine subgraph patterns [2,11]. However, none of these techniques mine representative patterns. VRRW has been used on networks before [34,37], but they do not mine subgraph patterns and hence cannot be applied directly to our problem.

9 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 diversified ranking through two random walk based algorithms: vertex-reinforced random walks and negative-reinforced random walks. Finally, the top-k most representative subgraph patterns are returned, where k is the budget provided by the user.

Scalability is achieved through a combination of two strategies. First, the overhead of clustering subgraphs is avoided due to the organization of subgraphs in the edit map. Second, the space-saving algorithm is employed to tackle the heavy memory requirements imposed by random walk procedures. 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 two orders of magnitude faster. Among the two random walk algorithms studied, negative-reinforced random walks outperform vertex-reinforced random walks on running time, while maintaining comparable or better performance in representative power. Overall, RESLING allows us to do more with less.

As a future work, it would be an interesting study to further improve scalability of RESLING through parallelization.

References

1. Ranu S, Singh AK (2012) Indexing and mining topological patterns for drug discovery. In: EDBT, pp 562–565
2. Ranu S, Hoang M, Singh A (2013) Mining discriminative subgraphs from global-state networks. In: KDD, pp 509–517
3. Chaoji V, Ranu S, Rastogi R, Bhatt R (2012) Recommendations to boost content spread in social networks. In: WWW, pp 529–538
4. Banerjee P, Ranu S, Raghavan S (2014) Inferring uncertain trajectories from partial observations. In: ICDM, pp 30–39
5. Banerjee P, Yawalkar P, Ranu S (2016) Mantra: a scalable approach to mining temporally anomalous sub-trajectories. In: KDD, pp 1415–1424

6. Yan X, Han J (2002) Gspan: graph-based substructure pattern mining. In: ICDM, p 721. ISBN: 0-7695-1754-4
7. Kuramochi M, Karypis G (2005) Finding frequent patterns in a large sparse graph*. Data Min Knowl Discov 11(3):243–271
8. Elseidy M, Abdelhamid E, Skiadopoulos S, Kalnis P (2014) Grami: frequent subgraph and pattern mining in a single large graph. PVLDB 7(7):517–528
9. Gurukar S, Ranu S, Ravindran B (2015) Commit: A scalable approach to mining communication motifs from dynamic networks. In: SIGMOD, pp 475–489
10. Thoma M, Cheng H, Gretton A, Han J, Kriegel H-P, Smola A, Song L, Yu PS, Yan X, Borgwardt K (2009) Near-optimal supervised feature selection among frequent subgraphs. In: SDM 2009, pp 1076–1087
11. Hasan MA, Zaki MJ (2009) Output space sampling for graph patterns. PVLDB 2(1):730–741
12. Ranu S, Singh AK (2009) Graphsig: a scalable approach to mining significant subgraphs in large graph databases. In: ICDE
13. Ranu S, Calhoun BT, Singh AK, Swamidass SJ (2011) Probabilistic substructure mining from small-molecule screens. Mol Inf 30(9):809–815
14. Ranu S, Singh AK (2009) Mining statistically significant molecular substructures for efficient molecular classification. J Chem Inf Model 49:2537–2550
15. Kuramochi M, Karypis G (2001) Frequent subgraph discovery. In: ICDM
16. Nijssen S, Kok JN (2004) The Gaston tool for frequent subgraph mining. In: Proceedings of the international workshop on graph-based tools
17. Yan X, Cheng H, Han J, Yu PS (2008) Mining significant graph patterns by scalable leap search. In: SIGMOD
18. Jin N, Young C, Wang W (2010) Gaia: graph classification using evolutionary computation. In: SIGMOD
19. Cheng H, Lo D, Zhou Y, Wang X, Yan X (2009) Identifying bug signatures using discriminative graph mining, In: Proceedings of the eighteenth international symposium on software testing and analysis, pp 141–152
20. Dutkowski J, Ideker T (2011) Protein networks as logic functions in development and cancer. PLoS Comput Biol 7:e1002180
21. Hasan MA, Chaoji V, Salem S, Besson J, Zaki MJ (2007) Origami: mining representative orthogonal graph patterns. In: ICDM, pp 153–162
22. Yan X, Han J (2003) Closegraph: mining closed frequent graph patterns. In: KDD, pp 286–295
23. Zeng Z, Tung AKH, Wang J, Feng J, Zhou L (2009) Comparing stars: on approximating graph edit distance. PVLDB 2(1):25–36
24. Zhang S, Yang J, Li S (2009) Ring: an integrated method for frequent representative subgraph mining, In: ICDM, pp 1082–1087
25. Natarajan D, Ranu S (2016) A scalable and generic framework to mine top-k representative subgraph patterns. In: 2016 IEEE 16th international conference on data mining (ICDM). IEEE, pp 370–379
26. Metwally A, Agrawal D, El Abbadi A (2005) Efficient computation of frequent and top-k elements in data streams. In: ICDT, pp 398–412
27. Ranu S, Hoang M, Singh A (2014) Answering top-k representative queries on graph databases. In: SIGMOD, pp 1163–1174
28. Drosou M, Pitoura E (2012) Disc diversity: result diversification based on dissimilarity and coverage. PVLDB 6(1):13–24
29. Cornuejols G, Fisher ML, Nemhauser GL (1977) Location of bank accounts to optimize float: an analytic study of exact and approximate algorithms. Manag Sci 23(8):789–810
30. He H, Singh AK (2006) Closure-tree: an index structure for graph queries. In: ICDE
31. Page L, Brin S, Motwani R, Winograd T (1998) The pagerank citation ranking: bringing order to the web. In: WWW, pp 161–172
32. Pemantle R (1992) Vertex-reinforced random walk. Probab Theory Relat Fields 92(1):117–136
33. Badrinath R, Madhavan CEV (2012) Diversity in ranking using negative reinforcement. In: Proceedings of the ACM SIGKDD workshop on mining data semantics, vol 11, no 1–11, p 6
34. Mei Q, Guo J, Radev D (2010) Divrank: the interplay of prestige and diversity in information networks. In: KDD
35. Huan J, Wang W, Prins J, Yang J (2004) Spin: mining maximal frequent subgraphs from graph databases. In: KDD, pp 581–586
36. Thomas L, Valluri S, Karlapalem K (2006) Margin: maximal frequent subgraph mining. In: ICDM, pp 1097–1101
37. Krishnan A, Padmanabhan D, Ranu S, Mehta S (2016) Select, link and rank: diversified query expansion and entity ranking using wikipedia. In: International conference on web information systems engineering, pp 157–173

Dheepikaa Natarajan received a B.Tech. and a M.Tech. degree in Computer Science from Indian Institute of Technology Madras, Chennai, India, in 2015. She is now working as a Software Development Engineer II in Amazon Machine Learning team in Seattle, USA.

Sayan Ranu is an assistant professor in the Department of Computer Science and Engineering at IIT Delhi. His research interests include spatiotemporal data analytics, graph indexing and mining, and bioinformatics. Prior to joining IIT Delhi, he spent close to 3 years as an Assistant Professor at IIT Madras and a year and half in the role of a Research Scientist at IBM Research. He obtained his PhD from the Department of Computer Science, University of California, Santa Barbara (UCSB), in March 2012.