Received: 24 April 2020 | Revised: 20 July 2020 | Accepted: 2 October 2020

DOI: 10.1002/widm.1393

ADVANCED REVIEW

WIREs DATA MINING AND KNOWLEDGE DISCOVERY

WILEY

Dynamical algorithms for data mining and machine learning over dynamic graphs

Mostafa Haghir Chehreghani

Department of Computer Engineering, Amirkabir University of Technology (Tehran Polytechnic), Tehran, Iran

Correspondence
Mostafa Haghir Chehreghani, Department of Computer Engineering, Amirkabir University of Technology (Tehran Polytechnic), Tehran, Iran.
Email: mostafachehreghani@gmail.com

Abstract

In many modern applications, the generated data is a *dynamic network*. These networks are graphs that change over time by a sequence of *update operations* (node addition, node deletion, edge addition, edge deletion, and edge weight change). In such networks, it is inefficient to compute from scratch the solution of a data mining/machine learning task, after any update operation. Therefore in recent years, several so-called *dynamical* algorithms have been proposed that *update* the solution, instead of computing it from scratch. In this paper, first we formulate this emerging setting and discuss its high-level algorithmic aspects. Then, we review state of the art dynamical algorithms proposed for several data mining and machine learning tasks, including frequent pattern discovery, betweenness/closeness/PageRank centralities, clustering, classification, and regression.

This article is categorized under:

Technologies > Structure Discovery and Clustering
Technologies > Machine Learning
Fundamental Concepts of Data and Knowledge > Big Data Mining

KEYWORDS

data mining, decremental algorithms, dynamic graphs, dynamical algorithms, incremental algorithms, machine learning, social network analysis, update time

1 | INTRODUCTION

In many real-world applications, the generated data are *dynamic*, that is, they frequently and quickly change over time. A well-known example is *dynamic graphs*, that is, graphs that change over time by a sequence of update operations. This type of graphs is generated in many domains such as social networks, communication and email exchange networks, collaboration networks, and information networks. An update operation (or a change) in a dynamic graph might be either a *node deletion*, or a *node addition*, or an *edge deletion*, or an *edge addition*. Moreover if the dynamic graph is weighted, the update operation might be an *edge weight change*.

There are many data mining and machine learning tasks, such as frequent pattern discovery, modeling, centrality, clustering, classification and regression, with proven applicability and usefulness in different domains. A naive approach to address these tasks over dynamic graphs is to compute the solution of the problem from scratch, after one or a batch of update operation(s) in the graph. However, this naive approach is usually time consuming and expensive. Moreover, it is useful and practically interesting to have the solution of the problem for the *new* (*updated*) network, after each single change in the dynamic graph. These observations have motivated researchers to develop several

https://doi.org/10.1002/widm.1393

dynamical algorithms that are optimized for different data mining/machine learning tasks over dynamic graphs. In such *dynamical* algorithms, first a *preprocessing* is performed wherein the solution, either exact or approximate, of the studied problem is computed for a given static graph. Then, after any update operation in the graph, the solution is also *updated* in a time much less than that required for computing it from scratch.[1] Examples of the studied problems include pattern extraction, clustering, and network centrality. In this setting, the main challenge to tackle is to efficiently update the solution, after an update operation in the network. This update should be done considerably faster than solving the problem for the updated network from scratch.

In this paper, we provide an overview of state of the art dynamical data mining/machine learning algorithms, recently proposed for dynamic graphs. Our study consists of the following sections. In Section 2, we formally introduce the setting and describe the high-level structure of dynamical algorithms proposed for solving different machine learning/data mining problems. In Section 3, we study dynamical algorithms presented for an important data mining task, called *frequent pattern discovery*. In Section 4, we review dynamical algorithms proposed for network analysis, with a particular emphasis on *network centrality*. We review most efficient exact and approximate algorithms for different centrality notions and compare their properties. In Section 5, we review state of the art dynamical algorithms presented for machine learning tasks, such as clustering, classification and regression. We also discuss some related problems, such as *online learning* and *representation learning* over dynamic graphs, and highlight the differences. Finally, Section 6 concludes this paper.

Table 1 summarizes symbols and notations widely used in the paper.

2 | PROBLEM STATEMENT

We may divide the problem of dynamical data mining/machine learning into two subproblems.

- In the first subproblem, we are given a static network G and and a learning/mining problem $\mathcal{X}$. The objective is to find a solution for $\mathcal{X}$ over G. We may refer to this subproblem as the *preprocessing problem*. This subproblem is almost the same as a standard learning/mining problem; the only difference is that we should take this into account that this subproblem will be followed by a second subproblem. Hence, we may decide to perform some extra computations and/or store some auxiliary information.
- In the second subproblem, we are given a network G, a learning or mining problem $\mathcal{X}$, a solution S already computed for $\mathcal{X}$ over G, and a sequence of update operations u_1, u_2, u_3, ... on G. We denote the network resulted by applying the update operation u_i on G by $G \odot u_i$. At each iteration i, the objective of the second subproblem is to update the solution of the problem $\mathcal{X}$ over $G \odot u_1 u_2 \cdots u_i$, based on the solution of $\mathcal{X}$ already computed for $G \odot u_1 u_2 \cdots u_{i-1}$. We may refer to this subproblem as the *update problem*.

We refer to these data mining and machine learning settings, consisting of the *preprocessing* and *update* problems, respectively as *dynamical mining* and *dynamical learning*, or `DyMine` and `DyLearn` in short. At a very high level and to solve `DyMine` and `DyLearn` problems, dynamical algorithms consist of two phases. The *preprocessing phase*, wherein it is assumed that the graph is static and a solution is found for the problem, usually using standard techniques. Also in most cases, some auxiliary results are determined and stored to help updating the solution. The *update phase*, wherein after an update operation in the network, the solution is updated to become valid for the revised network. The main challenge is to update the solution, in a time much less than computing it, for the revised network, from scratch. Along with updating the solution, auxiliary information stored by the algorithm should be updated too.

We explain the procedure by a simple example. Suppose that we want to cluster nodes of an undirected dynamic graph G. To do so, first during the pre-processing phase, we compute an embedding[2] (vector representation) for each node. The situation is depicted in Figure 1. Let n be the number of nodes of the graph. We define the embedding of a node i as a row of size n whose jth entry contains the size of a shortest path between nodes i and j. Then, we exploit a standard clustering algorithm, for example, a density-based algorithm or k-means, to group nodes using their embeddings. Now assume that during the update phase, a node (e.g., node 2) is deleted from the graph. After this update operation in the graph, the node embeddings as well as the clusters must be updated. To do so, first the embedding of the deleted node is deleted from the matrix. Then, the column corresponding to the deleted node is deleted, too. Some other entries of the matrix may need to revise. In the given example, the revised entries are highlighted with bold. In fact, after deleting the node, if the distance from i to j changes, the jth entry of the ith row (and ith entry of the jth row)

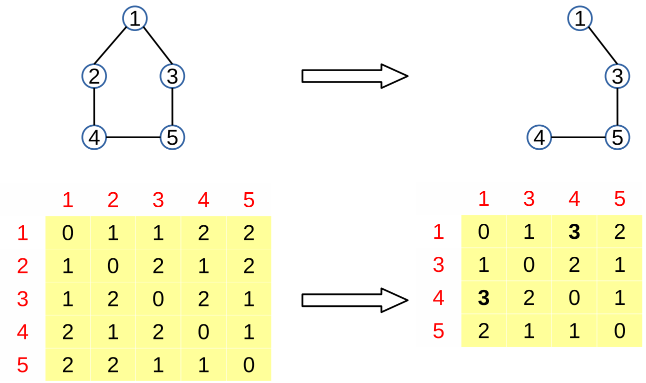
TABLE 1 Symbols and their definitions

Symbol	Definition
G	Graph (network)
$V(G)$	Set of nodes of G
$E(G)$	Set of edges of G
N	Number of nodes of G
M	Number of edges of G
P	Pattern graph
$\mathcal{X}$	Data mining/machine learning problem
$\mathrm{dis}(s, t)$	Distance (the size of a shortest path) between nodes s and t
σ_{st}	Number of shortest paths from node s to node t
$\sigma_{st}(v)$	Number of shortest paths from node s to node t that also pass through node v
$bc(v)$	Betweenness centrality of node v
$\delta_{s\bullet}(v)$	Dependency score of node s on node v
$\delta_{st}(v)$	Dependency score of a pair of nodes s, t on node v
VD	Vertex diameter (the longest shortest path) of G
c	Universal constant ($c \approx 0.5$)
$\mathcal{RF}(v)$	Set of nodes s such that there is a directed path from s to v
$\mathcal{RT}(v)$	Set of nodes t such that there is a directed path from v to t
$cc(v)$	Closeness centrality of node v
α	Teleportation parameter of PageRank
$\deg(v)$	Degree of node v
T	Transition matrix of the graph, used in PageRank
b	Preference vector used by personalized PageRank (PPR). Its ith entry is referred by b_i
x	Vector of PPR scores
Q	Modularity of the graph
A	adjacency matrix of G. A_{vu} refers to the element in row v and column u of A
C_v	Cluster of node v
$\delta(\cdot, \cdot)$	Delta function: it returns 1 iff its two parameters are the same; otherwise, it returns 0
T	Minimum cut tree
$GI(w)$	Gini-index of a word w
Y	Number of classes
γ	Graph arboricity
$P \cdot H \cdot D$	Subsampled randomized Hadamard transform
M	$n \times d$ matrix embedding (representation) of the graph
z	solution of linear regression

should be updated. To update distances, we may store and update *shortest path trees* (Hayashi, Akiba, & Yoshida, 2015). Finally and after updating the embeddings, the clusters should be updated using the new embeddings of the nodes. Updating the clusters must be done much faster than computing them from scratch, for the updated graph.

3 | DYNAMICAL DATA MINING ALGORITHMS

One of the key tasks in mining large networks is *frequent pattern discovery*. It has many proven applications in different domains, including the World Wide Web, bioinformatics, and social networks. It is also a fundamental problem in



	1	2	3	4	5
1	0	1	1	2	2
2	1	0	2	1	2
3	1	2	0	2	1
4	2	1	2	0	1
5	2	2	1	1	0

	1	3	4	5
1	0	1	**3**	2
3	1	0	2	1
4	**3**	2	0	1
5	2	1	1	0

FIGURE 1 An example of updating the adjacency matrix of the graph, after a node deletion

many other data mining tasks such as association rule mining. Informally speaking, *frequent patterns* are usually small subgraphs that have an interestingness score (*support/frequency*), greater than or equal to a user-defined threshold. In the following, first we briefly introduce some required notions and then, we review state of the art dynamical algorithms.

3.1 | Preliminaries

We assume the reader is familiar with basic concepts of graph theory. A good overview can be found in Diestel (2010). We use $V(G)$ and $E(G)$ to refer to the set of nodes and the set of edges of G, respectively. By n and m we, respectively, denote $|V(G)|$ and $|E(G)|$. Let λ_G be a labeling function for nodes/edges of a graph G. Two graphs P and G are isomorphic, iff there is a bijection $\mu : V(P) \rightarrow V(G)$ such that for all $u, v \in V(P)$, $(u, v) \in E(P) \Leftrightarrow (\mu(u), \mu(v)) \in E(G)$ and for all $u \in V(P) \cup E(P)$, $\lambda_P(u) = \lambda_G(\mu(u))$. The mapping μ is called an *isomorphism* from P to G. Graph P is *subgraph isomorphic* to G, iff P is isomorphic to a subgraph of G. An *embedding* of a pattern graph P in a large network G is a subgraph isomorphism mapping from P to G. An image of P under a mapping μ is the graph built by the set of images of all nodes and edges of P under μ. An *image* of P in G is an image of P, under some subgraph isomorphism from P to G (Ramon, Comendant, Chehreghani, & Wang, 2014).[3] Interestingness of pattern P in network G is defined by its *support* (*frequency*) in G. There are several support/interestingness notions in the literature (Chehreghani, Abdessalem, Bifet, & Bouzbila, 2020), including *minImage* (Bringmann & Nijssen, 2008) and *overlap-graph* based measures (Calders, Ramon, & Dyck, 2011). A pattern is *frequent*, iff its support in the network is greater than or equal to a user-defined threshold *min_sup*.

3.2 | Algorithms

In recent years, several algorithms have been proposed in the literature, to find frequent graph patterns from a large single dynamic graph. The crucial step in such algorithms is to efficiently update supports of patterns. Ray, Holder, and Choudhury (2014) studied the problem when the graph update operations are *edge addition* and *node addition*. They used a rather simple method: convert the problem of mining patterns from a large single dynamic graph into a problem of mining patterns in the *transactional setting*. In *transactional setting*, the data set consists of many relatively small graphs, called *transactions*, and support of a pattern is defined as the number of graphs in the data set that contain the pattern. Their algorithm first identifies the parts of the network that are subject to changes by the update operations, and construct transactions from these parts. Then it uses a dynamical transactional graph pattern mining algorithm, to update the set of frequent patterns.

Aslay, Nasir, Morales, and Gionis (2018) investigated approximate k-node frequent pattern mining over dynamic graphs, where the update operations are *edge addition* and *edge deletion*. The goal of this problem is to find the set of all frequent patterns that consist of exactly k-nodes. In the approximate form of this problem, supports of patterns are counted approximately with a guaranteed error (i.e., with high probability, the deviation of the estimated value from the real value is bounded by a small constant). While common strategies for estimating supports of patterns rely on sampling nodes/edges, the algorithm of Aslay et al. (2018) samples, with equal probability, subgraphs of the same size. This simplifies estimation of support of patterns. Their algorithm consists of two main components: (a) a *reservoir sampling* procedure, which detects changes to the sampled connected k-subgraphs; and (b) an *exploration* procedure, which adds new k-subgraphs to the sample set. To handle deletions, they use a variation of the *random pairing* technique (Gemulla, Lehner, & Haas, 2006).[4] Choudhury et al. (2018) developed a distributed system, called Percolator, that updates the set of frequent patterns after *edge additions* and *edge deletions*. Their system finds the set of *maximal* patterns that are relevant to user-specified keywords. A *maximal pattern* is a frequent pattern that does not have any frequent super-pattern. After an update operation in the network, first their system finds a minimal set S of patterns that are affected by these updates. Then it updates support of each pattern in S, using an incremental subgraph isomorphism algorithm.

Borgwardt, Kriegel, and Wackersreuther (2006) studied a slightly different problem. They considered a dynamic graph as a *time series* of graphs: a sequence $\{G_1, ..., G_l\}$ of l graphs, where G_i is the ith state of the dynamic graph. They developed the *Dynamic GREW* algorithm for finding patterns that are not only frequent in a large network, but also show an identical *dynamic behavior* over time. This means they have the same order of edge additions/deletions between their nodes over time. They defined support of patterns using a dynamic notion of subgraph isomorphism, defined as follows: the subgraph contains the same set of nodes and edges as its supergraph, but only a sub-interval of the states of its dynamic supergraph. A similar problem was studied in (Berlingerio, Bonchi, Bringmann, & Gionis, 2009), where the authors adapted the *minImage* support measure to dynamic graphs and proposed the GERM algorithm for finding graph-based association rules from a dynamic network. The problem becomes much more tractable (both in theory and practice),[5] when only a restricted class of patterns, for example, *triangles*, is discovered. So there are algorithms in the literature that find a restricted form of patterns (Bulteau, Froese, Kutzkov, & Pagh, 2016).

4 | DYNAMICAL NETWORK ANALYSIS ALGORITHMS

In this section, we review dynamical algorithms of analyzing large single networks. Among different tasks, we concentrate on *centrality measures* as they play a crucial role in analysis and utilization of real-world networks. There exist a number of centrality measures in the literature, including *closeness* centrality, *betweenness* centrality (and its extensions such as *group betweenness* centrality (Chehreghani, Bifet, & Abdessalem, 2018a) and *co-betweenness* centrality (Chehreghani, 2014b)), Katz centrality, PageRank and personalized PageRank. Their dynamical algorithms share several techniques and algorithmic properties in common. So in the following, we focus on the most popular measures, that is, *betweenness centrality*, *closeness centrality*, *PageRank*, and *personalized PageRank*.

4.1 | Betweenness centrality

In this section, first we briefly present preliminaries and definitions related to betweenness centrality. Then we discuss exact and approximate dynamical algorithms. Note that more than the discussed methods, there exist algorithms in the literature for streaming graphs (Green, McColl, & Bader, 2012). In streaming graphs, the problem cannot be solved for the whole graph. However, in the dynamical setting, we can run the algorithm over the entire graph, and the challenge is to effectively update the solution, after updates in the graph.

4.1.1 | Preliminaries

For two nodes s and t in a graph G, dis(s, t) denotes the distance (the size of a shortest path) from s to t. *Betweenness centrality* is used to quantify the importance of nodes or edges of a graph G. It is defined as the ratio of shortest paths passing over a node. More formally, for $s, t \in V(G)$, let σ_{st} be the number of shortest paths from s to t, and $\sigma_{st}(v)$ be the number of shortest paths from s to t that also pass through node v. *Betweenness centrality* of v is defined as follows:

$$bc(v) = \frac{1}{n(n-1)} \sum_{s,t \in V(G) \setminus \{v\}} \frac{\sigma_{st}(v)}{\sigma_{st}}. \tag{1}$$

A widely used notion is the directed acyclic graph (DAG) containing all the shortest paths starting from a node s. It is sometimes called *shortest-path-DAG*, or SPD in short, rooted at s (Chehreghani, 2014a). For each node s in G, the SPD rooted at s is unique, and it can be computed in $O(m)$ time for unweighted graphs and in $O(m + n\log n)$ time for weighted graphs with positive weights (U. Brandes, 2001). The other widely used notion is the *dependency score* of node s on another node v, which is defined as follows (U. Brandes, 2001):

$$\delta_{s\bullet}(v) = \sum_{t\in V(G)\setminus\{v,s\}} \frac{\sigma_{st}(v)}{\sigma_{st}}. \tag{2}$$

The *dependency score* of a pair of nodes s, t on node v is defined as follows: $\delta_{st}(v) = \frac{\sigma_{st}(v)}{\sigma_{st}}$. We have:

$$bc(v) = \frac{1}{n(n-1)} \sum_{s\in V(G)\setminus\{v\}} \delta_{s\bullet}(v).$$

To compute betweenness scores of all nodes, for each source node s, the well-known Brandes algorithm (U. Brandes, 2001) follows a two-phase procedure. In the first phase, it computes the SPD rooted at s (and as a result, distances and the number of shortest paths from s to other nodes of the graph). For each source node s, this phase takes $O(m)$ time for unweighted graphs and $O(m + n\log n)$ time for weighted graphs with positive weights. In the second phase, dependency scores of s on other nodes is computed using the following recursive relation (U. Brandes, 2001):

$$\delta_{s\bullet}(v) = \sum_{w:v\in P_s(w)} \frac{\sigma_{sv}}{\sigma_{sw}}(1 + \delta_{s\bullet}(w)), \tag{3}$$

where $P_s(w)$ is defined as:

$$\{u\in V(G) : \{u,w\}\in E(G) \wedge \mathrm{dis}(s,v) = \mathrm{dis}(s,u) + 1\}.$$

The second phase, for both weighted and unweighted graphs, can be computed in $O(m)$ time.

4.1.2 | Exact algorithms

Lee, Lee, Park, Choi, and Chung (2012) proposed the QUBE algorithm to efficiently update betweenness centrality of nodes after *edge additions*. QUBE first finds the set R of nodes whose betweenness scores might change by the update operation, and the sets of nodes whose betweenness scores do not change. Then, in order to compute betweenness score of a node in R, it considers the betweenness score resulted from the nodes in R, moreover, each shortest path that satisfies the following conditions: (a) it passes over a node in R, and (b) at least one of its source or target nodes are not in R. In practice, QUBE usually updates betweenness scores faster than recomputing them from scratch by Brandes algorithm (U. Brandes, 2001). However, these two have the same asymptotic time complexity. The algorithm of Nasre, Pontecorvi, and Ramachandran (2014a) updates betweenness scores after *edge additions* and *edge weight decreases*. It takes advantage of incrementally updating distances and the number of shortest paths for all pairs of nodes, and SPDs rooted at all nodes. This takes $O(mn + n^2)$ time, for all nodes. Then it utilizes Brandes' recursive relation for accumulating dependency scores (Equation (3), mentioned above). In other words, while it performs the first and more time-consuming phase of Brandes algorithm incrementally, it performs the second and less time consuming phase from scratch. This yields an $O(mn + n^2)$ algorithm, which is a $\log n$ times faster than Brandes algorithm on sparse graphs. In their following work (Nasre, Pontecorvi, & Ramachandran, 2014b), they obtained similar results for the cases of *edge deletion* and *edge weight increase*.

The algorithm of Kourtellis, Morales, and Bonchi (2015) updates betweenness centrality of nodes and edges after *edges additions* and *edge deletions*. They used a compact representation for the shortest path DAGs rooted at source nodes. More precisely, for each source node s, their auxiliary data structure stores the following information: (a) the distance of each node t from s; (b) the number of shortest paths starting from s and ending at t; and (c) the dependency score of node s on t. Kas, Carley, and Carley (2014)] presented a *fully dynamic* algorithm, that is, an algorithm that supports all the graph update operations. Their algorithm first identifies the set of nodes affected by the update operation and then, determines the new distances and shortest paths. To perform these operations effectively, it uses three hash maps: (a) D_{old} which maps each pair of nodes s, t to the set of shortest paths from s to t before the update operation; (b) σ_{old} which maps each pair of nodes s, t to the value of σ_{st} before the update operation; and (c) *trackLost* which maps each pair of nodes s, t to the set of nodes that were on a shortest path from s to t before the update operation, but are not on such a path after updating the graph.

Table 2 compares update time and supported update operations of state of the art exact dynamical algorithms for betweenness centrality.

4.1.3 | Approximate algorithms

Updating betweenness scores in large graphs consisting of millions of nodes is not tractable in practice. Therefore, in the literature, a number of dynamical approximate algorithms have been proposed. After completing the update operations in the graph, these algorithms try to update the approximate betweenness scores, so that they deviate from the exact scores by at most ϵ, with a probability at least $1 - \delta$. At a high level, we may divide these algorithms into three main categories:

- (*Source*) *node samplers*, where each sample is a single node. These methods estimate betweenness scores by computing dependency scores of the sampled node on the other nodes and scaling the results.
- (*Node*) *pair samplers*, where each sample is a pair of nodes. These methods estimate betweenness scores by computing dependency scores of the sampled pair on the other nodes and scaling the results.
- *Shortest path samplers*, where each sample is a shortest path. In these methods, first a pair of nodes is randomly chosen and then, one of the shortest paths between them is sampled. The number of times each node becomes an internal node of the sampled paths, is counted. In the end, the obtained counts are scaled, to give unbiased estimations of the betweenness scores.

In theory, all the three types of sampling methods spend the same time to pick up and process a single sample, which is $O(m)$ for unweighted graphs and $O(m + n\log n)$ for weighted graphs with positive weights. In the following, we review state of the art dynamical approximate algorithms for betweenness centrality. Table 3 summarizes our discussion and provides a concise comparison of the algorithms.

Bergamini and Meyerhenke (2016) proposed one of the first dynamical approximate algorithms for estimating betweenness centrality. Their algorithm updates the scores after one or a batch of *edge update operations* (i.e., edge addition, edge deletion, edge weight change). They use the RK algorithm (Riondato & Kornaropoulos, 2016) as the base. For given $\epsilon, \delta \in (0, 1)$, RK (Riondato & Kornaropoulos, 2016) samples

TABLE 2 Comparison of exact dynamical algorithms for betweenness centrality

Algorithm	Supported update operations	Update time
Lee et al. (2012)	Edge addition	$O(nm + n^2\log n)$
Nasre et al. (2014a)	Edge/node addition and edge weight decrease	$O(m_1 n + n^2)$
Nasre et al. (2014b)	Edge/node deletion and edge weight increase	$O(m_1 n + n^2)$
Kas et al. (2014)	Fully dynamic	$O(nm + n^2\log n)$
Kourtellis et al. (2015)	Edge addition/deletion	$O(nm + n^2\log n)$

Note: The value of m_1 is bounded by the maximum number of edges that lie on a shortest path in the graph (and hence, by n). *Fully dynamic* means the algorithm support all the update operations.

TABLE 3 Comparison of different approximate dynamical algorithms for betweenness centrality ($\epsilon, \delta \in (0, 1)$)

Algorithm	Sampling type	Supported update operations	#Samples
Bergamini and Meyerhenke (2016)	Shortest path sampling	Edge addition/deletion, edge weight change	Equation (4)
Hayashi et al. (2015)	Pair sampling	Fully dynamic	$\Omega\left(\frac{1}{\epsilon^2}\log\frac{1}{\delta}\right)$
Chehreghani, Bifet, and Abdessalem (2018b)	Pair sampling	Fully dynamic	$\Omega\left(\frac{1}{\epsilon^2}\log\frac{2}{\delta}\eta(v)^2\right)$

Note: Unlike the other algorithms, DyBED (Chehreghani et al., 2018b) estimate betweenness of a single node v; and its efficiency depends on $\eta(v)$ defined in Equation (5).

$$r = \Omega\left(\frac{c}{\epsilon^2}\left(\lfloor \log_2(\mathrm{VD}-2)\rfloor + 1 + \ln\left(\frac{1}{\delta}\right)\right)\right) \tag{4}$$

shortest paths, where VD is the vertex diameter (the longest shortest path) of the network and c is an universal constant ($c \approx 0.5$). Then RK estimates betweenness of each node v as the fraction of the sampled shortest paths that pass over v, multiplied by $\frac{1}{r}$. Since exact computation of VD is computationally heavy, RK uses upper bounds that can be computed in $O(n+m)$ time. After one or a batch of graph update operations, the algorithm of Bergamini and Meyerhenke (2016) updates the used upper bound for VD and the set S of sampled shortest paths stored to estimate betweenness scores. Let $\mathcal{S}'_{st}$ be the set of all shortest paths from s to t in the updated graph. To update S, the algorithm of Bergamini and Meyerhenke (2016) acts as follows: it keeps each path $p_{st} \in S$ (which starts from s and ends to t) if the distance and the number of shortest paths between s and t do not change by the update operations. Otherwise, p_{st} is replaced by another shortest path p'_{st}, chosen uniformly at random from $\mathcal{S}'_{st}$.

The algorithm of Hayashi et al. (2015) is a pair sampler, which means it uses all the shortest paths between a sampled pair s, t. It uses *hypergraph sketch* to keep track of sampled shortest paths. In this hypergraph, for each sampled pair s, t, a hyperedge e_{st} with auxiliary information on the nodes of the shortest paths from s to t is built. After a graph update operation and in order to efficiently detect the parts of hypergraph sketches that require modification, it uses the *two-ball index* data structure. For each sample s, t, *two-ball index* stores subtrees of shortest path trees in two directions, one from s and the other from t. The size of these balls is tuned by a parameter, which allows trade-offing between the index size and the update time. In order to improve the efficiency of the algorithm over graphs that are not (strongly) connected, a *reachability* index is kept. In this index, for each sample s, t, the set R_s of all nodes that can be reached from s is stored. When the graph changes, these data structures are updated using the algorithms of updating *shortest path trees* (Akiba, Iwata, & Yoshida, 2014).

Chehreghani et al. (2018b) presented the DyBED algorithm to estimate betweenness score of a single node v (or a small set of nodes). DyBED first computes two sets $\mathcal{RF}(v)$ and $\mathcal{RT}(v)$. $\mathcal{RF}(v)$ is defined as the set of nodes s such that there is a directed path from s to v. $\mathcal{RT}(v)$ is defined as the set of nodes t such that there is a directed path from v to t (Chehreghani, Bifet, & Abdessalem, 2019). Then, DyBED samples a set S of pairs s, t from $\mathcal{RF}(v) \times \mathcal{RT}(v)$. Then, for each pair s, t, a *hyperedge* (Hayashi et al., 2015) is constructed. Finally, betweenness score of v is estimated as:

$$\frac{|\mathcal{RF}(v)| \cdot |\mathcal{RT}(v)|}{n \cdot (n-1) \cdot |S|} \sum_{(s,t)\in S} \frac{\sigma_{st}(v)}{\sigma_{st}}.$$

When the graph is updated, DyBED updates $\mathcal{RF}(v)$, $\mathcal{RT}(v)$, and the set of samples. The number of samples required by DyBED is a function of $\eta(v)$, defined as follows:

$$\eta(v) = \frac{|\mathcal{RF}(v)| \cdot |\mathcal{RT}(v)|}{n \cdot (n-1)}. \tag{5}$$

Since η is always less than 1 (and in many real-world networks, it is less than, e.g., 0.04), DyBED requires less samples than the algorithm of Hayashi et al. (2015) (see Table 3). Note that computing betweenness of a single node faster than betweenness of all nodes is not always feasible or straightforward. For static graphs, whether exact betweenness score of a single node can be computed faster than exact betweenness scores of all nodes is a well-known conjecture.

4.2 | Closeness centrality

In this section, we review the state of the art dynamical algorithms for updating *closeness centrality*. Closeness centrality of node v in a (strongly) connected graph is defined as follows (Bavelas, 1950):

$$cc(v) = \frac{n-1}{\sum_{u \in V(G)} \text{dis}(v,u)}, \tag{6}$$

where n is the number of nodes of the graph.

One of the first algorithms in the literature is the algorithm of Kas, Carley, and Carley (2013), which supports all the update operations. It first uses the dynamical algorithm of Ramalingam and Reps (1996) to update pairwise distances between nodes. Then, it increases or decreases closeness scores of nodes whose distances have changed. However, a weakness of this algorithm is that it needs to store information for all pairs (connected/unconnected) of nodes. Sariyuce, Kaya, Saule, and Catalyurek (2013) studied updating closeness centrality after *edge insertion/deletion* in an unweighted graph. They introduced three pruning heuristics that help to improve the efficiency of updating the scores. Let v_1 and v_2 be the endpoints of the newly inserted or deleted edge. First, they showed that for all nodes s such that $|\text{dis}(s, v_1) - \text{dis}(s, v_2)| = 1$, there is no need to recompute closeness score. Second, they partitioned the network into biconnected components and showed that for nodes outside the biconnected component of (v_1, v_2), the recomputation is not required. Third, they noticed that nodes with the same neighborhood have the same closeness, therefore, it is sufficient to recompute the closeness score for only one of such nodes (Sariyuce et al., 2013).

The problem studied by Bisenius, Bergamini, Angriman, and Meyerhenke (2018) has three differences with the problem studied by Kas et al. (2013) and Sariyuce et al. (2013). First, it aims to compute approximate closeness scores, rather than exact scores. Second, it considers top-k closeness centrality computation, that is, finding a list of k nodes that have the highest closeness scores. Third, it considers the following variation of the standard definition of closeness centrality presented in Equation (6), which is more suitable for disconnected or directed graphs:

$$cc(v) = \sum_{u \in V(G), u \neq v} \frac{1}{\text{dis}(v,u)}. \tag{7}$$

Let v_1 and v_2 be the endpoints of the newly inserted or deleted edge. Their algorithm is based on finding the set of reachable nodes, that is, those nodes that have a distance less than ∞ from v_1 or v_2. Then, they use this set to find a set of nodes whose closeness scores are affected by the update operation.

Table 4 provides a concise comparison of the discussed dynamical closeness centrality algorithms.

4.3 | PageRank and Personalized PageRank

Two other notions that are widely used to quantify the importance of nodes of a graph are *PageRank* and *Personalized PageRank* (PPR). PageRank is the stationary distribution of a random walk, defined as follows. Let $\alpha \in (0, 1)$ be a real constant, called the *teleport* probability. The surfer starts from a random initial node. Then at each step,

- with probability $1 - \alpha$, it goes to a randomly chosen out-neighbor of the current node, and
- with probability α, it jumps (teleports) to some random node, chosen uniformly.

TABLE 4 Comparison of dynamical closeness algorithms

Algorithm	Algorithm type	Supported update operations	Space complexity
Kas et al. (2013)	Exact	Fully dynamic	$\Theta(n^2)$
Sariyuce et al. (2013)	Exact	Edge insertion/deletion	$\Theta(n)$
Bisenius et al. (2018)	Approximate	Edge insertion/deletion	$\Theta(n)$

Note: n is the number of nodes of the graph.

For an illustrative numerical example, the interested reader can refer to (Tabassum, Pereira, da Silva Fernandes, & Gama, 2018). Let b be a *preference* vector satisfying: (a) $b_i \geq 0$ for all $i \in V(G)$, and (b) $\sum_{i \in V(G)} b_i = 1$. In PPR, the random walk is defined as follows [Ohsaka, Maehara, & Kawarabayashi, 2015):

- with probability $1 - \alpha$, it goes to a randomly chosen out-neighbor of the current node, and
- with probability α, it jumps (teleports) to some random node j, chosen with probability b_j.

It is also common to define PageRank and PPR in the matrix form. Let T be the *transition* matrix of the graph defined as follows:

$$T_{ij} = \left(\begin{array}{cc} 1/\deg(j) & \text{if}(j,i) \in E(G) \\ 0 & \text{otherwise} \end{array} \right. .$$

The vector x of PPR scores is the unique solution to the following linear equation:

$$x = (1-\alpha)Tx + \alpha b. \tag{8}$$

Note that if for all $i \in V(G)$, $b_i = 1/n$, x will give the PageRank scores. Sometimes, PPR is defined in a restricted form and with respect to a specific source node s, as follows: we initially start from s. Then at each step, with probability 1 $- \alpha$ we go to a uniformly chosen out-neighbor of the current node, and with probability α we jump (teleport) to the source node s. This definition can be obtained from Equation (8) by setting b_s to 1 and the b values of the other nodes to 0. PPR has been used in many applications such as community detection (Yang & Leskovec, 2015) and recommendation (Backstrom & Leskovec, 2011).

The initial algorithms for computing PageRank and PPR were *power iteration* methods. In the simple form of power iteration, the vector x of PageRank scores is computed as follows (McSherry, 2005):

```
    while no convergence do
set $x$ to $Tx$
end while.
```

However, these algorithms are not practically efficient for large networks. In the following, we review most efficient dynamical algorithms proposed for PageRank and PPR.

Chakrabarti (2007) studied updating PPR scores in entity–relation graphs, wherein nodes are entities (paper, author, journal, etc.) and edges are relations (wrote, cited, etc.). He developed the HubRank system that during preprocessing, computes random walks for a small fraction of carefully chosen nodes. Then after update operations, it forms a small subgraph by the nodes in the random walks, and exploits it along with the random walks, to approximate PPR scores. Bahmani, Chowdhury, and Goel (2010) presented another algorithm for updating PageRank and PPR over dynamic graphs. Their algorithm works as follows: it conducts r random walks starting from each node of the network. Each of these random walks, called *walk segments*, is continued until its first teleport occurs. For each node v, let Y_v be the number of times that the generated walk segments visit v. Then, the algorithm approximates PageRank score of v as: $\frac{\alpha Y_v}{nr}$. They showed that in a network with n nodes and m (adversarially chosen) edges arriving in random order, accurate estimates of the PageRank scores of all nodes at all times can be computed with $O\left(\frac{n \ln m}{\alpha^2}\right)$ expected work[6] (Bahmani et al., 2010).

To update PageRank scores, Bahmani, Kumar, Mahdian, and Upfal (2012) suggested periodically probing nodes of the graph, to obtain their current set of outgoing neighbors. They argued that nodes with higher PageRank scores must be probed more often since they have more impact on the PageRank scores of the other nodes. Ohsaka et al. (2015) studied estimating PPR scores of nodes of a dynamic graph, with a guaranteed accuracy: the l_∞ norm of the deviations is always less than a user-defined threshold ϵ.[7] Their algorithm is an adaptation of the Gauss–Southwell method (Southwell, 1947), proposed to solve linear equations. If $m^{'}$ edges are randomly and sequentially inserted, the total number of iterations needed by their algorithm is $O(\log m^{'}/\epsilon)$.

Two techniques introduced to improve the efficiency of PPR computation are *forward push* and *reverse push*. In the *forward push* technique, for the source node s we maintain an estimation of the probability that a random walk from

s stops at any other node $t \in V(G)$. To compute these probabilities, we start from s and push the probabilities forwards along the edges (Andersen et al., 2008; Andersen, Chung, & Lang, 2006). In the *reverse push* technique, we start from each target node t and push the values backwards along the edges (Andersen et al., 2006; Andersen et al., 2008). Zhang, Lofgren, and Goel (2016) analyzed time complexity of these techniques over dynamic graphs. In particular, they presented the following results:

- Zhang et al. (2016, Theorem 4): Suppose that for $0 \le i \le k$, $G_i = (V_i, E_i)$ is a sequence of $k + 1$ undirected graphs such that each graph is obtained from the previous graph with an arbitrary edge insertion/deletion. Let $\bar{d}$ be the average degree of G_0, t be a random node of G_0 and $\epsilon \in (0, 1)$. Then, time complexity of maintaining a *reverse push* solution for each graph G_i and any $s \in V_i$ with additive error guarantee ϵ is at most $O\left(k/\alpha + k/(n\epsilon\alpha^2) + \bar{d}/(\alpha\epsilon)\right)$.
- Zhang et al. (2016, Theorem 9): Suppose that for $0 \le i \le k$, $G_i = (V_i, E_i)$ is a sequence of $k + 1$ undirected graphs, such that each graph is obtained from the previous graph by an edge insertion/deletion. Let s be a random vertex of V_0 and $\epsilon \in (0, 1)$. Then, time complexity of maintaining a *forward push* solution for each graph G_i and any $t \in V_i$ with additive error guarantee ϵ is at most $O(k/(\alpha^2) + k/(n\alpha^2\epsilon) + 1/(\alpha\epsilon))$.

In the literature, several variations of the classic setting of PageRank/PPR have been investigated, too. For example, Rossi and Gleich (2012) formulated a dynamic PageRank model for a static graph wherein teleportation is time dependent, that is, its distribution changes over time. Guo, Li, Sha, and Tan (2017) extended a *parallel* algorithm for updating PPR scores after a batch of edge insertions/deletions. They presented parallel push algorithms and showed that they can produce estimations of PPR scores, with the same error bound as the sequential push algorithms. *Top-k PPR* is another variation of the PPR computation problem, which was studied in (Wei et al., 2018). In this problem, given a graph G, a source node s and a parameter k, the objective is to find a set of k nodes with the highest PPR scores with respect to s. Their algorithm, called TopPPR, utilizes the three above-mentioned techniques (*forward push*, *reverse push*, *random walk sampling*) and combines the results.

5 | DYNAMICAL MACHINE LEARNING ALGORITHMS

In this section, we review dynamical algorithms proposed for different machine learning tasks. We also discuss some related problems and explain the differences.

5.1 | Clustering

A graph *cluster* is a group of densely connected nodes that are sparsely connected to the nodes outside the group. In the graph *clustering* problem, the objective is to detect all the clusters in a given graph. There are different types of graph clustering algorithms in the literature. Four well-known types of algorithms that are also extended to dynamic graphs, are: *spectral* algorithms, *modularity*-based algorithms, algorithms based on *minimum cut tree*, and algorithms based on *structural similarity*. In the following, we review dynamical extensions of these algorithms.

5.1.1 | Spectral clustering

Dhanjal, Gaudel, and Clemencon (2014) proposed one of the state of the art dynamical clustering algorithms. Their method is a spectral clustering algorithm for nodes of a dynamic graph: first, using rank-k singular value decomposition (SVD) of the Laplacian matrix of the graph, it builds a new eigenspace for the nodes of the graph. Then, it groups nodes in the new space. To extend this algorithm to dynamic graphs, Dhanjal et al. studied approximately updating the rank-k SVD of the Laplacian matrix of the graph, when the graph changes. After dynamically updating the eigenspace, their algorithm accordingly updates the clusters of the graph.

5.1.2 | Modularity-based clustering

A criteria used to quantify the quality of a graph clustering algorithm and also to find high quality clusters is *modularity*. For a graph G with a given set of clusters or communities, its *modularity* is defined as follows (Newman & Girvan, 2004):

$$Q = \frac{1}{2m}\sum_{vu}\left[A_{vu} - \frac{\deg(v)\deg(u)}{2m}\right]\delta(C_v, C_u), \tag{9}$$

where A is the adjacency matrix of G, A_{vu} is the element in row v and column u of A, C_v and C_u are, respectively, the clusters of nodes v and u, and $\delta(\cdot,\cdot)$ is the delta function which returns 1 iff its two parameters are the same; otherwise, it returns 0. Since it is NP-hard to find clusters with maximum modularity (Brandes et al., 2008), several heuristics/approximations have been proposed to alleviate the problem. Gorke, Maillard, Staudt, and Wagner (2010) studied modularity optimization over dynamic graphs. Let $G^{'}$ be a graph resulted by applying an update operation to another graph G. Gorke et al. showed that given a modularity-optimal clustering C of G, it is NP-hard to find a modularity-optimal clustering $C^{'}$ of $G^{'}$. By revising a number of static modularity-based graph clustering algorithms, they developed heuristics that update clusters of a dynamic graph, by the changes in the graph.

5.1.3 | Clustering based on minimum cut tree

Gorke, Hartmann, and Wagner (2012), extended the idea of graph clustering using a partial *minimum cut tree*, to dynamic graphs. A minimum cut tree T of a graph G is a tree on $V(G)$ and represents for any pair of nodes u, v a minimum-u-v-cut in G by the cheapest edge on the unique path between u and v in T (Gorke et al., 2012). Given a parameter β set by the user, clustering using minimum cut tree works as follows (Flake, Tarjan, & Tsioutsiouliklis, 2003): an artificial node t is added to the graph G, which is connected to all other nodes by weight β. Then, a minimum cut tree T of this augmented graph is computed. In the end, t is deleted and the resulting connected components of T are used to define the clustering. The algorithm of (Gorke et al., 2012) first tries to dynamically update the (partial) minimum cut tree, after an *edge insertion* or an *edge deletion* in the graph. Then, it uses the updated tree to update the clustering.

5.1.4 | Clustering based on structural similarity

A well-known algorithm proposed for static graph clustering is SCAN (X. Xu, Yuruk, Feng, & Schweiger, 2007). Wen, Qin, Zhang, Chang, and Lin (2019) improved its time complexity and extended it to dynamic graphs. SCAN uses *core* nodes to process those nodes that are not assigned to a cluster. A *core* is a node that shares structural similarity with at least μ neighbors, where μ is a parameter set by the user. Cores are regarded as the seeds of the clusters. Each cluster is constructed by expanding a core through all the nodes that are *structurally similar* to the core. For a node u, let $N^{'}(u)$ be a set consisting of u and all the neighbors of u. Structural similarity of two nodes v and u is defined as:

$$\frac{|N'(v) \cap N'(u)|}{\sqrt{|N'(v)| \cdot |N'(u)|}}.$$

A weakness of SCAN is that it needs to compute the structural similarity for every pair of connected nodes. Wen et al. (2019) improved SCAN by presenting the GS^{*} – Index data structure, that consists of two components: (a) *core-orders*, which facilitates finding all the core nodes, and (b) *neighbor-orders*, which helps to effectively group core nodes and assign noncore nodes to the clusters. Space complexity of GS^{*} – Index is $O(m)$, and time complexity of its construction is $O((m+\gamma)\log n)$, where n and m are the number of nodes and the number of edges of the graph, and γ is *graph arboricity* defined as the minimum number of edge-disjoint spanning forests into which the graph can be decomposed and $\gamma \leq \sqrt{m}$ (Chiba & Nishizeki, 1985). Wen et al. (2019) also proposed a method to update the clustering, after an *edge insertion* or an *edge deletion*. Their method is based on heuristics used to dynamically update *core-orders* and *neighbor-orders*, instead of computing them from scratch. It first determines for which nodes *core-orders* and *neighbor-orders* need to be updated and then, dynamically updates the required sets.

5.2 | Classification

The next learning problem studied in this paper is *node classification*. In a *partially labeled graph*, the node set is divided into two subsets V_L and V_U, where V_L is the set of labeled nodes and V_U is the set of unlabeled nodes. Furthermore, we are given: (a) a matrix of attributes/features wherein each row corresponds to one node, and each column corresponds to an attribute; and (b) a mapping function that assigns to each labeled node a label. In the *node classification* problem, given a partially labeled graph, the objective is to predict/learn the labels of unlabeled nodes. In *dynamical node classification*, we want to efficiently update the predicted labels of unlabeled nodes, by the changes in the graph. In the following, we discuss state of the art node classification algorithms proposed for dynamic graphs.

Aggarwal and Li (2011) presented the first algorithm for the classification problem over dynamic graphs. They proposed DYCOS, a random walk-based approach which utilizes and combines both the structure (links) and content (text) of the network. In the first step, DYCOS picks a set M_t of the top words which have the highest values of *Gini-index*. The Gini-index of a word w is defined as follows:

$$GI(w)=\sum_{j=1}^{y}\left(p_j(w)\right)^2, \tag{10}$$

where y is the number of classes and $p_1(w)\cdot\cdot\cdot p_y(w)$ are the fractional presence of w in the y classes. These *discriminative* words are used to create a *semi-bipartite* graph representation of the network. In this semi-bipartite graph, nodes of one partition have edges either within the partition, or to the nodes in the other partition. The nodes of the other partition may only have edges to the nodes in the first partition, and they not have edges among themselves. The first partition, includes the original nodes and edges of the network. The second partition, contains each word in M_t as a node. An undirected edge exists between a node i in the first partition and a word node j in the second partition, if the word is contained in the content/text of node i. An example of this *semi-bipartite* graph is presented in Figure 2. When the algorithm wants to determine the label of an unlabeled node s, it conducts l random walks, each of length h, that start from s. When constructing a random walk, at each iteration, with probability p_s a link of the original network, that is, a *structural* hop, is chosen and with the remaining probability, a *content* multihop is chosen. By changing the value of p_s, the user can control the relative importance of structure and content. In total, $l \cdot h$ nodes are visited by the random walks. The most frequently seen class label among these visited nodes, is reported as the class label of node s. For dynamic graphs, the algorithm takes advantage of carefully designed summary data structures, to efficiently update the random walks and the model, after an update operation (*node*/*edge insertion*/ *deletion*) in the network. Then it calls the updated model, to update the predicted labels of unlabeled nodes.

Yao and Holder (2014) presented an SVM-based algorithm for classifying nodes of a dynamic graph, where update operations are limited to *node addition* and *edge addition*. Their model is updated after a *set* of node/edge additions, called *batch*. The classifier is constructed from the current batch of data. The support vectors are retrained from the previous batch; and the learned model is used to predict the class labels in the future batch. D. Xu et al. (2019) studied the following classification problem: given a sequence of networks formed over the same set of nodes, where nodes are with attributes and each node has an unique label across difference time steps, the objective is to classify the unlabeled nodes based on the labeled nodes. They presented the AdaNN algorithm that learns node representations for classification by considering the evolution of both node attributes and network structure. At each time step, AdaNN learns the node

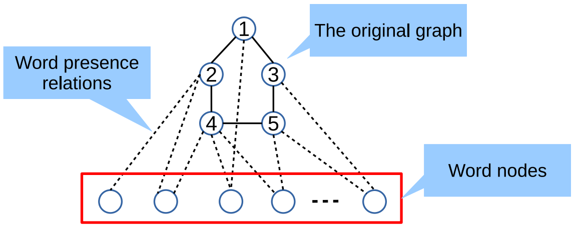


FIGURE 2 An example of *semi-bipartite* graph representation of the network. A dotted line shows the presence of a discriminative word in a graph node

attribute information by aggregating the representation of the node and its neighbors. To exploit the network topology information, AdaNN applies a random walk method to obtain the structural neighborhood of each node.

A slightly different problem, called *time-variant graph classification*, was studied by Wang, Wu, Zhu, Chen, and Zhang (2020). In this problem, the authors looked at a time-variant graph as a sequence of graphs. They used a class of graph patterns, called *graph-shapelet patterns*, as the features for the classification. *Graph-shapelet patterns* are graphical extensions of *shapelets* (Ye & Keogh, 2009), a family of discriminative features designed for vector based temporal data classification. In their algorithm, first the graph sequence is converted into a time series. Then, the shapelets are extracted from the time series. Then, the graph subsequences that match the shapelets are discovered from the original graph sequence. Finally, the most discriminative subsequences are extracted and used for classification.

5.3 | Regression

There are a number of algorithms in the literature for regression over static and temporal graphs. For example, Arne and Smith (2011) presented a model for nonparametric regression of nodes of a static graph, where distance between estimate and observation is measured at nodes by L_2 norm, and roughness is penalized on edges in the L_1 norm. However, developing dynamical algorithms for regression is not studied yet. An exception is the recent work of Chehreghani (2020), who studied updating the solution of *graph linear regression* in dynamic graphs.

In *graph linear regression*, we are given an $n \times d$ matrix embedding M of the graph and an $n \times 1$ vector f. So for each node i, its associated data consist of the ith row of M and the ith element of f. Matrix M is called the *predictor values* and f is called the *measured values*. The objective of the problem is to find a vector z such that $M \cdot z$ is the closest point to f in the column span of M, under some distance measure, for example, the Euclidean distance or the $L2$ norm. In other words, we want to solve the following problem:

$$\mathrm{argmin}_z \|M \cdot z - f\|_2^2. \tag{11}$$

A technique used to improve the efficiency of linear regression over static data is *subsampled randomized Hadamard transform* (Ailon & Liberty, 2008). For $n = 2^k$, the $n \times n$ Hadamard matrix H is defined as follows: $H_{i,j} = \frac{(-1)^{\langle i,j \rangle}}{\sqrt{n}}$, where $\langle i, j \rangle$ is the dot product of the binary representations of i and j over the field $\mathbb{F}_2$. A subsampled randomized Hadamard transform for matrix M and error bound $\epsilon \in (0, 1)$ is defined as $P \cdot H \cdot D$, where:

- matrix D is an $n \times n$ diagonal matrix with ± 1 on the diagonal (each one with the same probability),
- matrix H is an $n \times n$ Hadamard matrix, and
- matrix P is an $r \times n$ sampling matrix that samples r rows of $P \cdot H$ uniformly with replacement, where (Drineas, Mahoney, Muthukrishnan, & Sarlos, 2011):

$$r = \max\left\{48^2 d \ln(40nd) \ln\left(100^2 d \ln(40nd)\right), 40 d \ln(40nd)/\epsilon\right\}. \tag{12}$$

If in the jth sample row i is sampled, $P_{j,i} = \frac{\sqrt{n}}{\sqrt{r}}$; otherwise, it is 0. In other words, matrix P has one nonzero per row, where the column of each nonzero indicates a row of $P \cdot H$ which is to be selected.

The high-level algorithm of solving linear regression using subsampled randomized Hadamard transform consists of the following steps: (a) compute a $P \cdot H \cdot D$ matrix S, (b) compute matrices $S \cdot M$ and $S \cdot f$, and (c) output the solution of the equation,

$$\mathrm{argmin} x' \|(S \cdot A) \cdot x' - S \cdot f\|_2^2. \tag{13}$$

It is known that this procedure considerably reduces time complexity of solving linear regression, at the expense of having an approximate result (Drineas et al., 2011).

Chehreghani (2020) suggested using *subsampled randomized Hadamard transform* for *dynamic graph linear regression*. He discussed that if the matrix embedding of the graph satisfies certain properties, updating the solution of linear regression can be done efficiently. A possible matrix embedding is as follows. For each node v in graph G, its vector embedding is defined as a vector consisting of d nodes of G that have the smallest distances to v, where d is a constant. This vector embedding is called the d-nearest neighborhood of v. Matrix embedding M of G is defined as an $n \times d$ matrix whose ith row is the vector embedding of the ith node. He showed that after an *edge insertion* or an *edge deletion*, the approximate solution of linear regression can be updated in $O(rm)$ time, where r is defined in Equation 12.

5.4 | Representation (embedding) learning

Even though dynamical algorithms considerably outperform traditional methods that work with static data, several machine learning problems/settings, such as nonlinear regression and semi-supervised learning, are not studied under DyLearn. This could be due to more technical difficulty involved in developing dynamical algorithms. To partially alleviate this issue and due to good performance of learning *representations* for complex objects, in recent years several dynamical representation learning algorithms have been proposed for dynamic graphs (Goyal, Chhetri, & Canedo, 2020; Kazemi et al., 2020). A recent and comprehensive review on this can be found in (Kazemi et al., 2020). However, for the sake of completeness, in the following we briefly review a few well-known algorithms.

In machine learning, vector representations are particularly important, as they can be used as the input of many algorithms. Hence, a significant attempt has made to develop methods that learn a *representation*, that is, a continuous vector space, for a dynamic graph and assign each node or edge or subgraph to a specific point in that vector space. For example, Li, Hu, Jian, and Liu (2016) assumed that features of nodes of a graph evolve over time and studied dynamic feature selection in an unsupervised setting. They used gated graph neural networks (GGNNs) to preserve the topology of the dynamic graph at each time step, and long short-term memory neural networks to capture the dynamics by propagating temporal information between consecutive time steps. A slightly different task was studied in (Goyal et al., 2020), wherein given temporal snapshots of a graph and while the dynamics is captured, a representation of nodes at each time step is learned. A common shortcoming of these methods is that the dynamicity of data is only considered when a representation (embedding) is learned. After that and when these *dynamic representations* are used to solve a learning problem (classification, prediction, learning dependencies/associations, etc.), the algorithms do not take into account the dynamicity of data, that is, they do not update the solution of the learning problem by the changes in data.

5.5 | Online learning

A problem that might seem similar to dynamical learning is *online learning*. The key difference between *online learning* and *dynamical learning* is that *online learning* is used whenever it is computationally infeasible to solve the learning problem over the entire data set. However, in *dynamical setting* the learning problem can be solved over the entire data set and the challenge is to efficiently update the solution after changes in the graph. Examples of online learning algorithms for graphs can be found in Herbster, Lever, and Pontil (2008); Herbster, Pasteris, and Pontil (2015); Herbster, Pontil, and Wainer (2005). As an instance, Herbster and Pontil (2006) studied the problem of online label prediction of a graph with the perceptron. Herbster et al. (2005) presented a family of online learning algorithms and their application to graphs. Their algorithms learn a function defined on a graph from a set of labeled nodes. They represented the defined function by a Hilbert space associated with the graph Laplacian. In the consequent work, Herbster et al. (2008) showed that if the graph has a large diameter, the number of mistakes made by the Laplacian-based online graph learning algorithms can be proportional to the square root of the number of nodes. They tried to address this problem by using a *path graph* that yields a linear embedding of the original graph.

6 | CONCLUSIONS

In this paper, we formulated the DyMine and DyLearn settings that are used for mining and learning with dynamic networks. In these settings, the main objective is to efficiently update the solution of a mining/learning task, after an update operation in the network. We discussed high level algorithmic aspects of these settings. Furthermore, we reviewed state of the art dynamical algorithms proposed for different tasks, including frequent pattern discovery, betweenness/closeness/PageRank centrality, clustering, classification, and regression. Finally, we highlighted that, despite their potential usefulness and interest, a number of machine learning problems have not been studied under this setting yet.

CONFLICT OF INTEREST

The author has declared no conflicts of interest for this article.

ORCID

Mostafa Haghir Chehreghani https://orcid.org/0000-0003-3436-0541

ENDNOTES

[1] In the literature, sometimes those algorithms that update the solution of a problem after node/edge addition or edge weight decrease are called *incremental*; and those algorithms that update the solution after node/edge deletion or edge weight increase are called *decremental*.

[2] Note that this notion of *embedding* refers to a vector, representing structural/contextual properties of a node/edge/subgraph. There exists another notion of *embedding*, presented in Section 3, which is used to define frequent patterns.

[3] For general graphs, *isomorphism* is the most common mapping. However, for restricted classes of graphs such as *trees*, more than isomorphism (Chehreghani et al., 2011), *homeomorphism* is also common (Chehreghani & Bruynooghe, 2016).

[4] In *random pairing*, every deletion from the dataset is compensated by a subsequent addition. The details can be found in Gemulla et al. (2006).

[5] While *subgraph isomorphism*, which is a key task in pattern discovery, is NP-hard for general graphs, it can be solved in polynomial time for several restricted forms of graphs.

[6] In *expected work complexity*, the expected total number of basic operations performed by the algorithm is counted.

[7] The l_∞ norm of a vector is defined as the maximum of the absolute values of its components.

AUTHOR CONTRIBUTION

Mostafa Haghir Chehreghani: Conceptualization; formal analysis; investigation; writing-original draft; writing-review and editing.

RELATED WIREs ARTICLE

Social network analysis: An overview

FURTHER READING

Chehreghani, M. H. (2020). Subsampled randomized hadamard transform for regression of dynamic graphs. In *Proceedings of the 29th ACM International Conference on Information and Knowledge Management, CIKM 2020, Virtual Event, Ireland, October 19-23, 2020*, New York, NY: ACM.

Chehreghani, M. H., Chehreghani, M. H., Lucas, C., & Rahgozar, M. (2011). OInduced: An efficient algorithm for mining induced patterns from rooted ordered trees. *IEEE Transactions on Systems, Man, and Cybernetics—Part A: Systems and Humans*, *41*(5), 1013–1025. https://doi.org/10.1109/TSMCA.2010.2096808

REFERENCES

Aggarwal, C. C., & Li, N. (2011). On node classification in dynamic content-based networks. In *Proceedings of the Eleventh SIAM International Conference on Data Mining, SDM 2011*, April 28–30, 2011, Mesa, AZ (pp. 355–366). Philadelphia, PA: SIAM/Omnipress.

Ailon, N., Liberty, & E. (2008). Fast dimension reduction using rademacher series on dual BCH codes. In S.-H. Teng (Ed.), *Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2008*, San Francisco, CA, January 20–22, 2008 (pp. 1–9). Philadelphia, PA: SIAM. Retrieved from. http://dl.acm.org/citation.cfm?id=1347082.1347083

Akiba, T., Iwata, Y., & Yoshida, Y. (2014). Dynamic and historical shortest-path distance queries on large evolving networks by pruned landmark labeling. In C.-W. Chung, A. Z. Broder, K. Shim, & T. Suel (Eds.), *23rd International World Wide Web Conference, WWW '14*, Seoul, Republic of Korea, April 7–11, 2014 (pp. 237–248). New York, NY: ACM, https://doi.org/10.1145/2566486.2568007

Andersen, R., Borgs, C., Chayes, J. T., Hopcroft, J. E., Mirrokni, V. S., & Teng, S. (2008). Local computation of pagerank contributions. *Internet Mathematics*, *5*(1), 23–45. https://doi.org/10.1080/15427951.2008.10129302

Andersen, R., Chung, F. R. K., & Lang, K. J. (2006). Local graph partitioning using pagerank vectors. In *Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2006)*, 21–24 October 2006, Berkeley, CA (pp. 475–486). Washington, D.C.: IEEE. https://doi.org/10.1109/FOCS.2006.44

Arne, K., & Smith, A. D. (2011). Nonparametric regression on a graph. *Journal of Computational and Graphical Statistics*, *20*(2), 432–447. https://doi.org/10.1198/jcgs.2011.09203

Aslay, C., Nasir, M. A. U., Morales, G. D. F., & Gionis, A. (2018). Mining frequent patterns in evolving graphs. In A. Cuzzocrea, J. Allan, N. W. Paton, D. Srivastava, R. Agrawal, A. Z. Broder, et al. (Eds.), *Proceedings of the 27th ACM International Conference on Information and Knowledge Management, CIKM 2018*, Torino, Italy, October 22–26, 2018 (pp. 923–932). New York, NY: ACM. https://doi.org/10.1145/3269206.3271772

Backstrom, L., & Leskovec, J. (2011). Supervised random walks: Predicting and recommending links in social networks. In *Proceedings of the Forth International Conference on Web Search and Web Data Mining, WSDM 2011*, Hong Kong, China, February September 12, 2011 (pp. 635–644). New York, NY: ACM.

Bahmani, B., Chowdhury, A., & Goel, A. (2010). Fast incremental and personalized pagerank. *Proceedings of the VLDB Endowment*, *4*(3), 173–184.

Bahmani, B., Kumar, R., Mahdian, M., & Upfal, E. (2012). Pagerank on an evolving graph. In Q. Yang, D. Agarwal, & J. Pei (Eds.), *The 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '12*, Beijing, China, August 12–16, 2012 (pp. 24–32). New York, NY: ACM.

Bavelas, A. (1950). Communication patterns in task-oriented groups. *The Journal of the Acoustical Society of America*, *22*(6), 725–730. https://doi.org/10.1121/1.1906679

Bergamini, E., & Meyerhenke, H. (2016). Approximating betweenness centrality in fully dynamic networks. *Internet Mathematics*, *12*(5), 281–314. https://doi.org/10.1080/15427951.2016.1177802

Berlingerio, M., Bonchi, F., Bringmann, B., & Gionis, A. (2009). Mining graph evolution rules. In W. L. Buntine, M. Grobelnik, D. Mladenic, & J. Shawe-Taylor (Eds.), *Machine learning and knowledge discovery in databases, European Conference, ECML PKDD 2009, Proceedings, Part I*, Bled, Slovenia, September July 11, 2009, (Vol. 5781, pp. 115–130). Germany: Springer. https://doi.org/10.1007/978-3-642-04180-825

Bisenius, P., Bergamini, E., Angriman, E., & Meyerhenke, H. (2018). Computing top-k closeness centrality in fully-dynamic graphs. In R. Pagh & S. Venkatasubramanian (Eds.), *Proceedings of the Twentieth Workshop on Algorithm Engineering and Experiments, ALENEX 2018*, New Orleans, LA, January 7–8, 2018 (pp. 21–35). Philadelphia, PA: SIAM. https://doi.org/10.1137/1.9781611975055.3

Borgwardt, K. M., Kriegel, H.-P., & Wackersreuther, P. (2006). Pattern mining in frequent dynamic sub-graphs. In *Proceedings of the 6th IEEE International Conference on Data Mining (ICDM 2006)*, December 18–22, 2006, Hong Kong, China (pp. 818–822). Washington, D.C.: IEEE. https://doi.org/10.1109/ICDM.2006.124

Brandes, U. U. (2001). A faster algorithm for betweenness centrality. *Journal of Mathematical Sociology*, *25*(2), 163–177.

Brandes, U. U., Delling, D., Gaertler, M., Gorke, R., Hoefer, M., Nikoloski, Z., & Wagner, D. (2008). On modularity clustering. *IEEE Transactions on Knowledge and Data Engineering*, *20*(2), 172–188. https://doi.org/10.1109/TKDE.2007.190689

Bringmann, B., & Nijssen, S. (2008). What is frequent in a single graph? In *Advances in knowledge discovery and data mining, 12th Pacific–Asia Conference, PAKDD 2008* Osaka, Japan, May 20–23 (pp. 858–863). Germany: Springer.

Bulteau, L., Froese, V., Kutzkov, K., & Pagh, R. (2016). Triangle counting in dynamic graph streams. *Algorithmica*, *76*(1), 259–278. https://doi.org/10.1007/s00453-015-0036-4

Calders, T., Ramon, J., & Dyck, D. V. (2011). All normalized anti-monotonic overlap graph measures are bounded. *Data Mining and Knowledge Discovery*, *23*(3), 503–548.

Chakrabarti, S. (2007). Dynamic personalized pagerank in entity-relation graphs. In C. L. Williamson, M. E. Zurko, P. F. Patel-Schneider, & P. J. Shenoy (Eds.), *Proceedings of the 16th International Conference on World Wide Web, WWW 2007*, Banff, AL, Canada, May 8–12, 2007 (pp. 571–580). New York, NY: ACM.

Chehreghani, M. H. (2014a). An efficient algorithm for approximate betweenness centrality computation. *Comput. J.*, *57*(9), 1371–1382. https://doi.org/10.1093/comjnl/bxu003

Chehreghani, M. H. (2014b). Effective co-betweenness centrality computation. In B. Carterette, F. Diaz, C. Castillo, & D. Metzler (Eds.), *Seventh ACM International Conference on Web Search and Data Mining, WSDM 2014*, New York, NY, February 24–28, 2014 (pp. 423–432). New York, NY: ACM.

Chehreghani, M. H., Abdessalem, T., Bifet, A., & Bouzbila, M. (2020). Sampling informative patterns from large single networks. *Future Generation Computer Systems*, *106*, 653–658. https://doi.org/10.1016/j.future.2020.01.042

Chehreghani, M. H., Bifet, A., & Abdessalem, T. (2018a). An in-depth comparison of group betweenness centrality estimation algorithms. In N. Abe, H. Liu, C. Pu, X. Hu, N. K. Ahmed, M. Qiao, et al. (Eds.), *IEEE International Conference on Big Data, Big Data 2018*, Seattle, WA, December 10–13, 2018 (pp. 2104–2113). Washington, D.C.: IEEE.

Chehreghani, M. H., Bifet, A., & Abdessalem, T. (2018b). DyBED: An efficient algorithm for updating betweenness centrality in directed dynamic graphs. In N. Abe, H. Liu, C. Pu, X. Hu, N. Ahmed, M. Qiao, et al. (Eds.), *IEEE International Conference on Big Data, Big Data 2018*, Seattle, WA, December 10–13, 2018 (pp. 2114–2123). Washington, D.C.: IEEE.

Chehreghani, M. H., Bifet, A., & Abdessalem, T. (2019). Adaptive algorithms for estimating betweenness and k-path centralities. In W. Zhu, D. Tao, X. Cheng, P. Cui, E. A. Rundensteiner, D. Carmel, et al. (Eds.), *Proceedings of the 28th ACM International Conference on Information and Knowledge management, CIKM 2019*, Beijing, China, November 3–7, 2019 (pp. 1231–1240). New York, NY: ACM. https://doi.org/10.1145/3357384.3358064

Chehreghani, M. H., & Bruynooghe, M. (2016). Mining rooted ordered trees under subtree homeomorphism. *Data Mining and Knowledge Discovery*, *30*(5), 1249–1272. https://doi.org/10.1007/s10618-015-0439-5

Chiba, N., & Nishizeki, T. (1985). Arboricity and subgraph listing algorithms. *SIAM Journal on Computing*, *14*(1), 210–223.

Choudhury, S., Purohit, S., Lin, P., Wu, Y., Holder, L. B., & Agarwal, K. (2018). Percolator: Scalable pattern discovery in dynamic graphs. In Y. Chang, C. Zhai, Y. Liu, & Y. Maarek (Eds.), *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining, WSDM 2018*, Marina Del Rey, CA, February 5–9, 2018 (pp. 759–762). New York, NY: ACM. https://doi.org/10.1145/3159652.3160589

Dhanjal, C., Gaudel, R., & Clemencon, S. (2014). Efficient eigen-updating for spectral graph clustering. *Neurocomputing*, *131*, 440–452. https://doi.org/10.1016/j.neucom.2013.11.015

Diestel, R. (2010). *Graph theory* (4th ed.). Heidelberg: Springer-Verlag.

Drineas, P., Mahoney, M. W., Muthukrishnan, S., & Sarlos, T. (2011). Faster least squares approximation. *Numerische Mathematik*, *117*(2), 219–249. https://doi.org/10.1007/s00211-010-0331-6

Flake, G. W., Tarjan, R. E., & Tsioutsiouliklis, K. (2003). Graph clustering and minimum cut trees. *Internet Mathematics*, *1*(4), 385–408.

Gemulla, R., Lehner, W., & Haas, P. J. (2006). A dip in the reservoir: Maintaining sample synopses of evolving datasets. In U. Dayal, K. Whang, D. B. Lomet, G. Alonso, G. M. Lohman, M. L. Kersten, et al. (Eds.), *Proceedings of the 32nd International Conference on Very Large Data Bases*, Seoul, Korea, September 12–15, 2006 (pp. 595–606). New York, NY: ACM. Retrieved from. http://dl.acm.org/citation.cfm?id=1164179

Gorke, R., Hartmann, T., & Wagner, D. (2012). Dynamic graph clustering using minimum-cut trees. *Journal of Graph Algorithms and Applications*, *16*(2), 411–446.

Gorke, R., Maillard, P., Staudt, C., & Wagner, D. (2010). Modularity-driven clustering of dynamic graphs. In *Experimental algorithms, 9th International Symposium, SEA 2010, proceedings*, Ischia Island, Naples, Italy, May 20–22, 2010 (pp. 436–448). Germany: Springer.

Goyal, P., Chhetri, S. R., & Canedo, A. (2020). Dyngraph2vec: Capturing network dynamics using dynamic graph representation learning. *Knowledge-based Systems*, *187*, 104816. https://doi.org/10.1016/j.knosys.2019.06.024

Green, O., McColl, R., & Bader, D. A. (2012). A fast algorithm for streaming betweenness centrality. In *2012 International Conference on Privacy, Security, Risk and Trust, PASSAT 2012, and 2012 International Conference on Social Computing, SOCIALCOM 2012*, Amsterdam, The Netherlands, September 3–5, 2012 (pp. 11–20). Washington, DC: IEEE. https://doi.org/10.1109/SocialCom-PASSAT.2012.37

Guo, W., Li, Y., Sha, M., & Tan, K.-L. (2017). Parallel personalized pagerank on dynamic graphs. *Proceedings of the VLDB Endowment*, *11*(1), 93–106.

Hayashi, T., Akiba, T., & Yoshida, Y. (2015). Fully dynamic betweenness centrality maintenance on massive networks. *Proceedings of the VLDB Endowment*, *9*(2), 48–59. https://doi.org/10.14778/2850578.2850580

Herbster, M., Lever, G., & Pontil, M. (2008). Online prediction on large diameter graphs. In *Advances in neural information processing systems 21, Proceedings of the Twenty-Second Annual Conference on Neural Information Processing Systems*, Vancouver, British Columbia, Canada, December 08–11, 2008 (pp. 649–656). Red Hook, NY: Curran Associates, Inc.

Herbster, M., Pasteris, S., & Pontil, M. (2015). Predicting a switching sequence of graph labelings. *Journal of Machine Learning Research*, *16*, 2003–2022.

Herbster, M., & Pontil, M. (2006). Prediction on a graph with a perceptron. In *Proceedings of the Twentieth Annual Conference on Neural Information Processing Systems* Vancouver, British Columbia, Canada, December 4–7, 2006 (pp. 577–584). Cambridge, MA: MIT Press.

Herbster, M., Pontil, M., & Wainer, L. (2005). Online learning over graphs. In *Machine learning, Proceedings of the Twenty-Second International Conference (ICML 2005)*, Bonn, Germany, August 7–11, 2005 (pp. 305–312). New York, NY: ACM.

Kas, M., Carley, K. M., & Carley, L. R. (2013). Incremental closeness centrality for dynamically changing social networks. In J. G. Rokne & C. Faloutsos (Eds.), *Advances in social networks analysis and mining 2013, ASONAM '13*, Niagara, ON, Canada, August 25–29, 2013 (pp. 1250–1258). New York, NY: ACM. https://doi.org/10.1145/2492517.2500270

Kas, M., Carley, K. M., & Carley, L. R. (2014). An incremental algorithm for updating betweenness centrality and k-betweenness centrality and its performance on realistic dynamic social network data. *Social Network Analysis and Mining*, *4*(1), 235. https://doi.org/10.1007/s13278-014-0235-z

Kazemi, S. M., Goel, R., Jain, K., Kobyzev, I., Sethi, A., Forsyth, P., & Poupart, P. (2020). Representation learning for dynamic graphs: A survey. *Journal of Machine Learning Research*, *21*, 70:1–70:73.

Kourtellis, N., Morales, G. D. F., & Bonchi, F. (2015). Scalable online betweenness centrality in evolving graphs. *IEEE Transactions on Knowledge and Data Engineering*, *27*(9), 2494–2506. https://doi.org/10.1109/TKDE.2015.2419666

Lee, M.-J., Lee, J., Park, J. Y., Choi, R. H., & Chung, C.-W. (2012). Qube: A quick algorithm for updating betweenness centrality. In *Proceedings of the 21st World Wide Web Conference (WWW)* (pp. 351–360). New York, NY: ACM.

Li, J., Hu, X., Jian, L., & Liu, H. (2016). Toward time-evolving feature selection on dynamic networks. In *IEEE 16th International Conference on Data Mining, ICDM 2016*, December 12–15, 2016, Barcelona, Spain (pp. 1003–1008). Washington, D.C.: IEEE.

McSherry, F. (2005). A uniform approach to accelerated pagerank computation. In *Proceedings of the 14th International Conference on World Wide Web, WWW 2005*, Chiba, Japan, May 10–14, 2005 (pp. 575–582). New York, NY: ACM.

Nasre, M., Pontecorvi, M., & Ramachandran, V. (2014a). Betweenness centrality—incremental and faster. In E. Csuhaj-Varju, M. Dietzfelbinger, & Z. Esik (Eds.), *Mathematical foundations of computer science 2014—39th International Symposium, MFCS 2014, Proceedings, part II, Lecture Notes in Computer Science*, Budapest, Hungary, August 25–29, 2014 (Vol. 8635, pp. 577–588). Germany: Springer. https://doi.org/10.1007/978-3-662-44465-8_49

Nasre, M., Pontecorvi, M., & Ramachandran, V. (2014b). Decremental all-pairs ALL shortest paths and betweenness centrality. In H.-K. Ahn & C.-S. Shin (Eds.), *Algorithms and computation—25th International Symposium, ISAAC 2014, Proceedings, Lecture Notes in*

Computer Science, Jeonju, Korea, December 15–17, 2014 (Vol. 8889, pp. 766–778). Germany: Springer. https://doi.org/10.1007/978-3-319-13075-060

Newman, M. E. J., & Girvan, M. (2004). Finding and evaluating community structure in networks. *Physical Review E*, *69*(2), 026113. https://doi.org/10.1103/PhysRevE.69.026113

Ohsaka, N., Maehara, T., & Kawarabayashi, K. (2015). Efficient pagerank tracking in evolving networks. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Sydney, NSW, Australia, August 10–13, 2015 (pp. 875–884). New York, NY: ACM.

Ramalingam, G., & Reps, T. W. (1996). On the computational complexity of dynamic graph problems. *Theoretical Computer Science*, *158* (1&2), 233–277. https://doi.org/10.1016/0304-3975(95)00079-8

Ramon, J., Comendant, C., Chehreghani, M. H., & Wang, Y. (2014). Graph and network pattern mining. In *Mining user generated content* (pp. 97–126). London, UK: Chapman and Hall/CRC. https://doi.org/10.1201/b16413-8

Ray, A., Holder, L., & Choudhury, S. (2014). Frequent subgraph discovery in large attributed streaming graphs. In W. Fan, A. Bifet, Q. Yang, & P. S. Yu (Eds.), *Proceedings of the 3rd International Workshop on big data, streams and heterogeneous source mining: algorithms, systems, programming models and applications*, Bigmine 2014, New York City, August 24, 2014 (Vol. 36, pp. 166–181). Brookline, MA: JMLR Workshop and Conference Proceedings. Retrieved from. http://proceedings.mlr.press/v36/, ray14.html

Riondato, M., & Kornaropoulos, E. M. (2016). Fast approximation of betweenness centrality through sampling. *Data Mining and Knowledge Discovery*, *30*(2), 438–475. https://doi.org/10.1007/s10618-015-0423-0

Rossi, R. A., & Gleich, D. F. (2012). Dynamic pagerank using evolving teleportation. In *Algorithms and models for the web graph—9th International Workshop, WAW 2012, proceedings*, Halifax, NS, Canada, June 22-23, 2012 (pp. 126–137). Germany: Springer.

Sariyuce, A. E., Kaya, K., Saule, E., & Catalyurek, U. V. (2013). Incremental algorithms for closeness centrality. In *Proceedings of the 2013 IEEE International Conference on big data*, October 6–9, 2013, Santa Clara, CA (pp. 487–492). Washington, D.C.: IEEE.

Southwell, R. V. (1947). Relaxation methods in theoretical physics. *The Journal of the Royal Aeronautical Society*, *51*(434), 270–270. https://doi.org/10.1017/S0368393100111915

Tabassum, S., Pereira, F. S. F., da Silva Fernandes, S., & Gama, J. (2018). Social network analysis: An overview. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, *8*(5), e1256.

Wang, H., Wu, J., Zhu, X., Chen, Y., & Zhang, C. (2020). Time-variant graph classification. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, *50*(8), 2883–2896.

Wei, Z., He, X., Xiao, X., Wang, S., Shang, S., & Wen, J.-R. (2018). TopPPR: Top-k personalized pagerank queries with precision guarantees on large graphs. In G. Das, C. M. Jermaine, & P. A. Bernstein (Eds.), *Proceedings of the 2018 International Conference on management of data, SIGMOD conference 2018*, Houston, TX, June 10–15, 2018 (pp. 441–456). New York, NY: ACM.

Wen, D., Qin, L., Zhang, Y., Chang, L., & Lin, X. (2019). Efficient structural graph clustering: An index- based approach. *VLDB J.*, *28*(3), 377–399.

Xu, D., Cheng, W., Luo, D., Gu, Y., Liu, X., Ni, J., … Zhang, X. (2019). Adaptive neural network for node classification in dynamic networks. In J. Wang, K. Shim, & X. Wu (Eds.), *2019 IEEE international conference on data mining, ICDM 2019*, Beijing, China, November 8–11, 2019 (pp. 1402–1407). Washington, D.C.: IEEE.

Xu, X., Yuruk, N., Feng, Z., & Schweiger, T. A. J., & (2007). SCAN: a structural clustering algorithm for networks. In P. Berkhin, R. Caruana & X. Wu (Eds.), *Proceedings of the 13th ACM SIGKDD International Conference on knowledge discovery and data mining*, San Jose, CA, August 12–15, 2007 (pp. 824–833). New York, NY: ACM.

Yang, J., & Leskovec, J. (2015). Defining and evaluating network communities based on ground-truth. *Knowledge and Information Systems*, *42*(1), 181–213. https://doi.org/10.1007/s10115-013-0693-z

Yao, Y., & Holder, L. B. (2014). Scalable svm-based classification in dynamic graphs. In R. Kumar, H. Toivonen, J. Pei, J. Z. Huang, & X. Wu (Eds.), *2014 IEEE international conference on data mining, ICDM 2014*, Shenzhen, China, December 14–17, 2014 (pp. 650–659). Washington, D.C.: IEEE. https://doi.org/10.1109/ICDM.2014.69

Ye, L., & Keogh, E. J. (2009). Time series shapelets: A new primitive for data mining. In J. Elder, F. S. Fogelman, P. Flach, & M. Zaki (Eds.), *Proceedings of the 15th ACM SIGKDD International Conference on knowledge discovery and data mining*, Paris, France, June 28–July 1, 2009 (pp. 947–956). New York, NY: ACM.

Zhang, H., Lofgren, P., & Goel, A. (2016). Approximate personalized pagerank on dynamic graphs. In B. Krishnapuram, M. Shah, A. J. Smola, C. C. Aggarwal, D. Shen, & R. Rastogi (Eds.), *Proceedings of the 22nd ACM SIGKDD International Conference on knowledge discovery and data mining*, San Francisco, CA, August 13–17, 2016 (pp. 1315–1324). New York, NY: ACM. https://doi.org/10.1145/2939672.2939804

How to cite this article: Haghir Chehreghani M. Dynamical algorithms for data mining and machine learning over dynamic graphs. *WIREs Data Mining Knowl Discov*. 2021;11:e1393. https://doi.org/10.1002/widm.1393