A Stochastic Approach to Finding Densest Temporal Subgraphs in Dynamic Graphs

Xuanming Liu, *Student Member, IEEE,* Tingjian Ge, *Member, IEEE,* and Yinghui Wu, *Member, IEEE*

Abstract—One important problem that is insufficiently studied is finding densest *lasting*-subgraphs in large dynamic graphs, which considers the time duration of the subgraph pattern. We propose a framework called Expectation-Maximization with Utility functions (EMU), a novel stochastic approach that nontrivially extends the conventional EM approach. EMU has the flexibility of optimizing any user-defined utility functions. We validate our EMU approach by showing that it converges to the optimum—by proving that it is a specification of the general Minorization-Maximization (MM) framework with convergence guarantees. We devise EMU algorithms for the densest lasting subgraph problem, as well as several variants by varying the utility function. Using real-world data, we evaluate the effectiveness and efficiency of our techniques, and compare them with two prior approaches on dense subgraph detection.

Index Terms—Data mining, Graph algorithms, Stochastic methods

1 Introduction

Big Data is often represented by large dynamic graphs. Discovering dense subgraphs is especially of interest and has been studied for static graphs, but little has been done on detecting dense subgraphs that last for a long time interval. The need for detecting such dense lasting subgraphs is especially evident in telecommunication, traffic networks, and social network analysis.

Communication hotspots. In a mobile phone network, each user is a vertex, and a phone call corresponds to one or more edges with a time duration. Upon a significant event or breaking news (e.g., a natural disaster or a social spotlight event), dense, long-lasting phone calls among users pose a challenge to the quality of mobile services, and should be detected in a timely fashion for fast response [1]. The service provider may want to identify the densest subgraph region having edges that last for a long time, and allocate more resources there. A similar need arises in Internet service providers and data centers, where long-lasting and dense computer network request regions (e.g., large file transfers) should be provided with more resources.

Spam network filtering. Dense subgraph detection has been used for community detection [2]. Dense *long-lasting* subgraph patterns in communication/phone-call networks often indicate true communities, while conventional community detection will also include spam call subgraphs that are dense but typically quite short.

Traffic control. In road traffic networks, each road intersection (or critical points such as highway entries/exits) is a vertex, and a real-time report of traffic condition between two vertices suggests an edge with a time duration and a label (e.g., high-congestion, slowness, or smoothness). Dense lasting subgraphs indicate traffic congestion that lasts long and hence is the most significant [3]. Detecting such congestion in time benefits interventions and overall traffic effectiveness optimization.

While detecting dense subgraphs has been studied over static graphs, not much has been done to detect dense lasting subgraphs over dynamic networks. (1) Aggarwal et al. [4] propose a two-phase solution for finding frequently occurring dense subgraphs in dynamic graphs. In the first phase, they identify vertices that tend to appear together. In the second phase, they further find which vertices also form a dense subgraph in the snapshots where they appear together. Nevertheless, the method is based on set similarity—it may return vertices which are correlated in co-occurrence, but which still appear rarely over time. Detecting dense subgraphs that can last for a long period is not addressed. (2) Ma et al. [5] study fast computation of dense temporal subgraphs that pertain to the same set of nodes and edges with time-varying edge weights. The density is aggregated as the total edge weights of a subgraph. The approach first detects "promising" time intervals; instances of subgraphs in each time interval is then computed. In a nutshell, none of these previous approaches performs a direct, principled optimization of an objective function for the densest lasting subgraph problem as we do.

Problem and framework overview. We develop a general, stochastic approach to detecting densest lasting subgraphs. We consider a dynamic graph as a sequence of graph snapshots, each of which pertains to the same set of vertices but may contain a different set of edges (Figure 1a). Part of our goal is to compute a *probabilistic subgraph model* (Figure 1b top). The model has three critical parameters: number of vertices, probabilities (ρ_i) of each edge, and time duration (number of consecutive snapshots it appears in). In addition to this model, we need to find the value of a latent variable, which indicates the "location" index of the occurrence of this

The work is supported by NSF grant IIS-1633271. Wu is supported by NSF under CNS-1932574, OIA-1937143, ECCS-1933279, CNS-2028748, DoE under DE-IA0000025, and PNNL Data-Model Convergence initiative.

- *X. Liu and T. Ge are with the Department of Computer Science, University of Massachusetts, Lowell, MA 01854. E-mail: {xliu, ge}@cs.uml.edu*
- *Y. Wu is with the Department of Computer & Data Sciences, Case Western Reserve University, Cleveland, OH 44106. E-mail: yxw1650@case.edu*

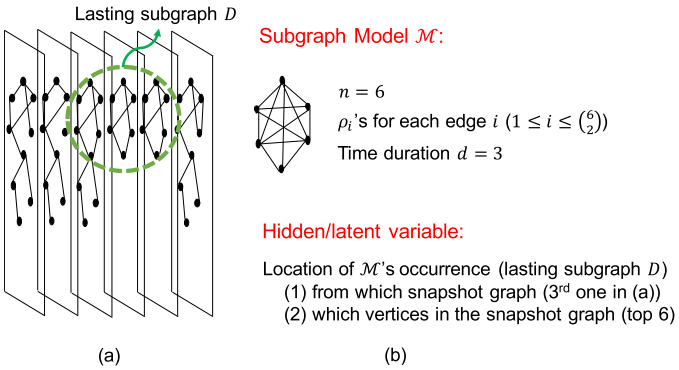


Fig. 1. Overview: Dynamic graph (a) and target subgraph model (b).

subgraph model within the graph snapshots: from which snapshot it starts, and which vertices it maps to in that snapshot (Figure 1b bottom).

At the core of our work is a novel framework, namely, Expectation Maximization with a utility function (EMU), which nontrivially extends the Expectation Maximization (EM) [6] method by incorporating a "utility" component that characterizes the need of detecting dense lasting subgraphs. We theoretically justify the extension by proving that the new two-step iterative algorithm converges to the optimum since EMU falls into the Minorization-Maximization (MM) framework [7] in statistics. We propose novel utility functions for EMU, show their connections with previous work, and devise an algorithm under the utility functions. The algorithm iterative refines both the model and the latent variable value of the occurrence location (Figure 1b). In summary:

- We formalize the problem of finding densest lasting-subgraph in a dynamic graph (Section 2).
- We propose a novel EMU framework, and create utility functions for our problem (Section 3).
- We devise an algorithm under EMU, and prove its correctness as it is in the MM framework (Section 4).
- We extend EMU to study other timing characteristics, including the densest subgraph spikes (Section 5).
- Using four real-world dynamic datasets, we perform a comprehensive empirical study (Section 6).

Related Work. We categorize the related work as follows. *Dense subgraphs in static graphs*. Discovering dense subgraphs has been studied for static graphs [8], [9], [10], [11], [12]. Dense subgraphs in static graphs are usually characterized by induced subgraphs with high edge-node count ratio, such as edge density [9], k-cores [10], α-quasi-cliques [11], among other variants. Decomposition algorithms are developed to find approximately dense subgraphs with optimaility guarantees [8], [9]. As observed in [12], dense subgraphs defined by edge-node counts tend to produce large subgraphs—for example, a graph can itself be a k-core, while quasi-cliques are often too small. The semantics in [12] incorporates an objective function on a notion of edge surplus over the expected number of edges under the random-graph model. This characterization subsumes several conventional semantics, such as edge/vertex count ratio and α-quasi-cliques, and leads to densest subgraphs with a more balanced size.

Dense subgraph detection in dynamic graphs. Previous work in this direction is significantly less than its static graph counterpart. A streaming algorithm is proposed to improve the algorithm in [8] for large graphs. The methods are nevertheless still developed for static graphs rather than temporal graphs. As remarked earlier, the approaches developed in [4], [5] either do not address time intervals, or do not focus on finding dense subgraphs that also last long (the details of comparisons are in Sections 1 and 6). The work by Bogdanov et al. [13] shares the same problem model as [5], but [5] improves the performance of [13] (thus we only compare with [5]). Angel et al. [14] consider weighted graphs with a constant number of vertices, and there are a number of weight updates at each time interval. Even though one can use weight increase and decrease to simulate the presence and absence of edges, the major difference of [14] from our work is the semantics: [14] finds dense subgraphs in each graph snapshot without considering the density across snapshots or time duration.

A preliminary version of this paper was published in IEEE ICDE'19 [15]. Here, we have significantly extended the ideas in [15]. Specifically, we add a new Section 5 where we propose a universal and composable utility function framework that addresses different timing characteristics such as densest subgraph spikes. We also add the corresponding experiments in Section 6.

2 Preliminaries

We define a *dynamic graph* $\mathcal{G}_T$ over a period of time as a sequence of graph snapshots $\{G_1, \ldots, G_T\}$. Each snapshot $G_t = (V, E_t)$ at timestamp t ($t \in [1, T]$) is an undirected graph with a vertex set V and an edge set E_t. We next introduce our subgraph model (shown in Figure 1).

Definition 1. ***(Subgraph Model)*** *Given a dynamic graph $\mathcal{G}_T$, a subgraph model $\mathcal{M}(n, \rho, d)$ consists of three (sets) of parameters: (1) n vertices , (2) the existence probabilities ρ_i of each edge e_i from a total of $\binom{n}{2}$ possible edges, and (3) a time depth d that the model spans in $\mathcal{G}_T$* (i.e., *d adjacent snapshots).*

The probabilistic subgraph model allows approximate characterization of edge appearance in a temporal graph, in terms of its probability. Moreover, it provides the flexibility for us to specify a class of utility functions (to be discussed) that characterize the properties of desired subgraphs. Subgraphs with predefined properties (*e.g.*, k-cores) lack such flexibility. Note that the edges in the data graph $\mathcal{G}_T$ are deterministic in each snapshot; the probabilities ρ_i are only used in the *subgraph model* $\mathcal{M}$ to characterize the frequency in which edge i appears within the d snapshots of the match instance in the data, as in the definition below.

Definition 2. ***(Lasting Subgraph)*** *A lasting subgraph $\mathcal{G}_d$ in $\mathcal{G}_T$ specified by a subgraph model $\mathcal{M}(n, \rho, d)$ is a dynamic graph that consists of n vertices, and spans d contiguous snapshots in $\mathcal{G}_T$; moreover, there is a one-to-one mapping f from the vertices of $\mathcal{M}$ to the vertices of $\mathcal{G}_d$. We say that $\mathcal{G}_d$ is an occurrence of $\mathcal{M}$, with a probability $Pr[\mathcal{G}_d|\mathcal{M}]$.*

Intuitively, the existence of a lasting subgraph in $\mathcal{G}_T$ is induced by an occurrence $\mathcal{G}_d$ of a corresponding subgraph model $\mathcal{M}$ (n, ρ, d) in $\mathcal{G}_T$, specified by the node mapping

from $\mathcal{M}$ to $\mathcal{G}_d$ and the lasting duration d; the likelihood of its existence is quantified by the model probability $\Pr[\mathcal{G}_d|\mathcal{M}]$. Note that the edges of each snapshot of $\mathcal{G}_d$ is induced by the node mapping. We discuss model probability in Section 4.

We are now ready to introduce the densest lasting subgraph problem. To this end, we introduce a *utility function*, denoted as $u(\mathcal{M})$, to measure the "quality" of subgraph models. The utility function allows us to integrate various density measures to lasting subgraph models. As such, intuitively, finding densest lasting subgraphs is to discover and compare subgraph models with higher $u(\mathcal{M})$ values, and moreover, more likely to have the corresponding occurrences in $\mathcal{G}_T$.

Densest Lasting Subgraph Problem. Given a dynamic graph $\mathcal{G}_T$ and a specified utility function $u(\mathcal{M})$, the *densest lasting subgraph problem* is to discover a subgraph model $\mathcal{M}^*(n, \rho, d)$ and the associated lasting subgraph ${\mathcal{G}_d}^*$, such that

$$(\mathcal{M}^*, {\mathcal{G}_d}^*) = \arg\max_{\mathcal{M}, \mathcal{G}_d} (u(\mathcal{M}) \cdot \Pr[\mathcal{G}_d|\mathcal{M}, \mathcal{G}_T]).$$

We shall introduce and focus on a specific utility function to present our algorithms (Section 3). These techniques on the other hand readily extend to other classes of utility functions, as verified in Section 5. For example, in Figure 1(a), the green dashed oval encloses a lasting subgraph that has 6 vertices and spans 3 contiguous snapshots in the data, which is an occurrence of the subgraph model $\mathcal{M}$ in Figure 1(b). Intuitively, the densest lasting subgraph problem is to find the optimal subgraph model $\mathcal{M}^*$ and associated lasting subgraph instance in data with the highest utility. Readers unfamiliar with EM or the Metropolis-Hastings method may refer to [15] for some background.

3 EMU: EM WITH A UTILITY FUNCTION

In this section, we introduce our general algorithm framework called EMU (EM with utility function). The idea is to integrate a utility function into the EM process, such that the process is guided by the utility function towards maximizing the likelihood of subgraph models with the desired density property.

3.1 Utility Function for Densest Lasting Subgraph

Intuitively, the subgraph models and their occurrences with more edges and larger time depth should be favored, given a specified number of vertices. We justify this intuition by providing a utility function to characterize "good" models.

A probabilistic perspective. Given a subgraph model $\mathcal{M}$, consider $\mathcal{M}$ as an "agent" that generates the observed data. If $\mathcal{M}$ generates $\mathcal{G}_d$, then a "reward" $u(\mathcal{M})$ is granted. Otherwise, $\mathcal{M}$ gets no reward. Define a random variable U that refers to the utility the model is rewarded in this process. The goal is to find a model agent that achieves the highest expected value of U. In other words, we want to maximize

$$\mathbf{E}[U] = u(\mathcal{M}) \cdot \Pr[\mathcal{G}_d|\mathcal{M}] \tag{1}$$

which justifies our objective function in Section 2.

Utility function. In particular, our utility function for a densest subgraph model $\mathcal{M}(n, \rho, d)$ is defined as

$$u(\mathcal{M}(n, \rho, d)) = \prod\nolimits_{j \in E_c(\mathcal{M})} e^{d(\rho_j - \alpha)} \tag{2}$$

where ρ_j is the existence probability of edge j in the *complete edge set* $E_c(\mathcal{M})$ of the model (edges induced by every vertex pair), d is the time duration of $\mathcal{M}$, and α is a constant that (implicitly) balances contrasting terms of edge abundance and node size of occurrences generated by the probabilistic model agent. Intuitively, the utility function favors subgraph models with higher aggregated edge probability (thus denser occurrences) and larger time duration d.

We next provide a justification by bridging the utility function to a widely adopted semantics for *static* dense graphs [12]. The density of a subgraph with edges E_S induced by a set V_S of n vertices in [12] is quantified by edge surplus of V_S, which is defined as $|E_S| - \alpha\binom{n}{2}$, where α is a counterbalancing factor that penalizes subgraphs with too many vertices. Thus the semantics strikes a balance between contrasting measures of edge size and node size, by favoring subgraphs that are neither "too small" nor "too large". We show the following. For proofs of theorems/lemmas not shown in this paper, please refer to [15].

Theorem 1. *The problem of computing densest subgraph that maximizes edge surplus [12] is equivalent to finding a densest lasting subgraph that maximizes $u(\mathcal{M}(n, \rho, d))$ where $d = 1$ and ρ is either 0 or 1 for each edge $e_j \in E_c(\mathcal{M})$.*

Theorem 1 suggests that our subgraph model subsumes edge surplus of vertex set V_S over expected edge size under the random-graph model. Given Theorem 1, one can also verify that our problem is in general NP-hard. Indeed, computing optimal static densest subgraph with edge surplus, as a special case of our problem, is already intractable [12].

3.2 The General EMU Framework

While a standard EM method with the Maximum Likelihood estimation [6] can be used to compute subgraph models that are likely to occur in $\mathcal{G}_T$, it may yield occurrences that are neither dense nor lasting. We now introduce our general EMU framework incorporating a utility function.

Overview. Similar to EM, EMU methods also interleave the E step and the M step. The difference is that in the M step, instead of using the *maximum likelihood estimate* to get the model parameters for the next iteration, EMU estimates the model parameters by maximizing Equation (1). It is easy to see that, if the utility function $u(\mathcal{M})$ is a positive constant value, then EMU is equivalent to EM. Thus, EMU can be deemed a *generalization* of EM, expressing preference over some property of the model to be searched for.

Specifically, a model $\widehat{\mathcal{M}}_i$ at iteration i of EMU consists of three (sets of) parameters: (1) the number of vertices $\hat{n}_i$, (2) the probability $\hat{\rho}_{ji}$ of the j-th edge in a complete graph of $\hat{n}_i$ vertices ($1 \leq j \leq \binom{\hat{n}_i}{2}$), and (3) the time depth $\hat{d}_i$.

4 THE EMU ALGORITHMS

We next show that the general EMU framework gives birth to efficient algorithms to compute the densest lasting subgraphs in large $\mathcal{G}_T$. In the E step of EMU, given $\widehat{\mathcal{M}}_i$, we estimate a probability distribution $\hat{L}_i$ of the location of $\widehat{\mathcal{M}}_i$'s

occurrence in $\mathcal{G}_T$. In the M step, based on the (expected) information collected from $\hat{L}_i$ in $\mathcal{G}_T$, we estimate a new model $\widehat{\mathcal{M}}_{i+1}$ that maximizes $\mathbf{E}[U]$, and continue with the next iteration of E step. For the M step, we use the utility function in Section 3.1 by default, with generalized edge probability $\rho_j \in [0,1]$ (beyond the binary case in [12]), and for the case $d > 1$, to characterize the densest lasting subgraphs desired in many real-world applications.

Recall that a subgraph model $\mathcal{M}$ consists of a set of n vertices $V_{\mathcal{M}}$ (out of the N vertices V of the dynamic graph), edge probability ρ_j for each edge j, and the time depth d. During EMU, we need to match $\mathcal{M}$ with subgraphs in $\mathcal{G}_T$, starting from some snapshot. To perform this subgraph match, one would need to enumerate all permutations of the n vertices for isomorphism and examine ρ_j. We first introduce a technique to reduce the cost of subgraph matching between a subgraph model and $\mathcal{G}_T$, used by our EMU algorithms.

Linearized Vertex Order. To simplify the model evaluation, we assign an arbitrary, but fixed order to the N vertices (V) of the dynamic graph (let the list be $v_1, \ldots, v_N$), as well as to the n vertices $V_{\mathcal{M}}$ of $\mathcal{M}$ (let the list be $u_1, \ldots, u_n$). When we evaluate any subset of n vertices $v_{i_1}, \ldots, v_{i_n}$ from the dynamic graph against $\mathcal{M}$ (to get the matching probability in our EMU algorithms that follow), $v_{i_1}, \ldots, v_{i_n}$ are sorted in their linear ID order in V, and are mapped one-to-one with $u_1, \ldots, u_n$ in $\mathcal{M}$. The vertex linearization ensures that a subset of n vertices is matched against $\widehat{\mathcal{M}}$ as one subgraph rather than $n!$ subgraphs (all vertex permutations). As such, we avoid enumerating all permutations of $v_{i_1}, \ldots, v_{i_n}$ in the dynamic graph $\mathcal{G}_T$.

Lemma 1. *Assigning a fixed order to V and a fixed order to $V_{\mathcal{M}}$, and matching any subset of n vertices from V with $V_{\mathcal{M}}$ following this order do not miss any occurrence of the densest lasting subgraph models using the EMU framework.*

The intuition of Lemma 1 is that, even though we give an order to the vertices in V and those in $V_{\mathcal{M}}$, the EMU algorithm has the full freedom to set the probabilities of all the edges in the model $\mathcal{M}$, so that its vertices one-to-one match those in the optimal instance ($\mathcal{G}_d$). We next introduce our EMU algorithm. As remarked earlier, EMU follows EM by interleaving E steps and M steps. We present the E step first and show the M step in Section 4.2. The algorithm, in the end, returns the subgraph model $\mathcal{M}$, and the latent variable value—the model's best location L in $\mathcal{G}_T$.

4.1 Generalized E Step

Consider iteration i of EMU. In the generalized E step, we assume that the model $\widehat{\mathcal{M}}_i$ is given ($\widehat{\mathcal{M}}_0$ is initialized arbitrarily in the first iteration). The goal of E step is to estimate the location distribution $\hat{L}_i$ of this model in $\mathcal{G}_T$. However, there are $\binom{N}{\hat{n}_i}(T - \hat{d}_i)$ "locations" to examine, where $\hat{n}_i$ and $\hat{d}_i$ are the number of vertices and time depth of $\widehat{\mathcal{M}}_i$, respectively, for any subset of $\hat{n}_i$ vertices starting from the first $T - \hat{d}_i$ snapshots. While the vertex linearization avoids vertex enumeration cost, it is still quite expensive to examine every locations and compute the probabilities of matching.

We tackle this challenge by adapting a statistical technique called the Metropolis-Hastings (MH) method to the lasting subgraph model discovery. The idea is to *selectively* get samples from the whole space of $\binom{N}{\hat{n}_i}(T - \hat{d}_i)$ locations, in such a way that they form a Markov chain that has a stationary distribution, in which the probability of "hitting" a location sample is *proportional* to the probability that $\widehat{\mathcal{M}}_i$ occurs in that location. Thus, this guided search tends to find the true occurrence locations of $\widehat{\mathcal{M}}_i$ quickly.

We present the E step as Algorithm GENERALIZEDE.

Algorithm 1: GENERALIZEDE $(\widehat{\mathcal{M}}_i, \mathcal{G}_T)$

Input: model $\widehat{\mathcal{M}}_i(\hat{n}_i, \hat{\rho}_{ji}, \hat{d}_i)$, dynamic graph $\mathcal{G}_T$
Output: a location distribution $\hat{L}_i$

1 $C \leftarrow$ getEdgeWalkComponent $(\hat{n}_i, \hat{d}_i, \mathcal{G}_T)$
2 $p_c \leftarrow \prod_{j \in E(C)} \hat{\rho}_{ji} \cdot \prod_{j \notin E(C)} (1 - \hat{\rho}_{ji})$
3 $\hat{L}_i \leftarrow \{(C, p_c)\}$
4 **while** $|\hat{L}_i| < n_c$ **do**
5 $C_{prev} \leftarrow C$
6 $r \leftarrow$ random(0, 1)
7 **if** $r \geq p_{teleport}$ **then**
8 $t_0 \leftarrow C.t$ or $C.t + 1$ or $C.t - 1$ with equal probability
9 **if** $C.V$ *is a connected component in* $G_{t_0 \ldots t_0 + \hat{d}_i}$ **then**
10 $C.t \leftarrow t_0$
11 **with** *probability* $1/2$ **do**
12 $e \leftarrow$ pick an edge randomly from $N_e(C)$
13 $C.V \leftarrow C.V \cup \{e$'s endpoint not in $C.V\}$
14 remove a random $v \in C.V$ s.t. C is still a component
15 **else**
16 $C \leftarrow$ getEdgeWalkComponent $(\hat{n}_i, \hat{d}_i, \mathcal{G}_T)$
17 $p_c \leftarrow \prod_{j \in E(c)} \hat{\rho}_{ji} \cdot \prod_{j \notin E(C)} (1 - \hat{\rho}_{ji})$
18 $\alpha \leftarrow \min(1, \frac{p_c}{p_{c_{prev}}})$
19 with probability $1 - \alpha$, set $C \leftarrow C_{prev}$
20 $\hat{L}_i \leftarrow \hat{L}_i \cup \{(C, p_c)\}$
21 **return** $\hat{L}_i$

1 **Function** *getEdgeWalkComponent* $(\hat{n}_i, \hat{d}_i, \mathcal{G}_T)$
2 $e \leftarrow$ pick an edge uniformly at random from $G_{1 \ldots T - \hat{d}_i}$
3 $C.t \leftarrow e.t$; $C.d \leftarrow \hat{d}_i$
4 $C.V \leftarrow \{$two end points of $e\}$
5 **while** $|C.V| < \hat{n}_i$ **do**
6 $e \leftarrow$ pick an edge uniformly at random from $N_e(C)$
7 $C.V \leftarrow C.V \cup \{e$'s endpoint not in $C.V\}$
8 **return** C

EMU Algorithm: E step. We introduce the details of E step.

Function getEdgeWalkComponent. We start with a procedure invoked by GENERALIZEDE, denoted as getEdgeWalkComponent, to randomly "grow" an $\hat{n}_i$ vertex component starting from a selected edge (thus denser areas in $\mathcal{G}_T$ have a higher chance to be reached). As shown in line 2 of getEdgeWalkComponent, it chooses an edge uniformly from $G_{1 \ldots T - \hat{d}_i}$, which denotes snapshots G_1 to $G_{T - \hat{d}_i}$ in $\mathcal{G}_T$. In line 3 of the function, we initialize a *component* object C (which in the end will grow to the same size as $\widehat{\mathcal{M}}_i$ and be returned). We set its *starting time* field $C.t$ to be the random

edge's time, and its time depth field $C.d$ to the current model's time depth. Line 4 of getEdgeWalkComponent initializes component C's vertex set $C.V$ as the endpoints of the first edge. The loop (lines 5-7) grows C by randomly selecting edges from $N_e(C)$, where $N_e(C)$ refers to the set of neighboring edges of C (i.e., those edges with exactly one endpoint included in C).

Main algorithm. The algorithm GENERALIZEDE invokes function getEdgeWalkComponent to obtain a component C (line 1). It then calculates the probability of generating the specific component C, based on the edge probabilities $\hat{\rho}_{ji}$'s for each edge j in $\widehat{\mathcal{M}}_i$, and the set of edges $E(C)$ that are in component C (between $C.V$ in the $C.d$ snapshots from $C.t$). In line 3, the component and probability pair is added to the location distribution set $\hat{L}_i$. The loop in lines 4-20 will select more components to add to $\hat{L}_i$, until the number is n_c, a performance/accuracy tradeoff parameter we shall study in Section 6. $\hat{L}_i$ is finally returned in line 21.

"Teleport" or "Stay"?. In each iteration, the algorithm GENERALIZEDE decides whether to "teleport" to reinitialize a component C or to continue to perform local incremental update to the current component C. This is decided by a probability threshold $p_{teleport}$ (line 6). (1) With probability $1 - p_{teleport}$, we do not "teleport", i.e., to arbitrarily jump to anywhere in the dynamic graph by calling getEdgeWalkComponent (line 16). (2) Otherwise, we randomly change the component's starting time $C.t$ in its ± 1 interval (lines 8-10). Then in lines 11-14 we do minor adjustment to the vertex set $C.V$. The intuition of introducing *teleport* is to strike a balance between *exploitation* (sticking with local good candidates) and *exploration* (exploring remote good locations). This is especially necessary when the graph is not connected. We will further examine the parameter $p_{teleport}$ in Section 6.

"Accept" or "Reject"? Line 17 computes the probability of the current component C given the model $\widehat{\mathcal{M}}_i$. The parameter α at line 18 denotes the "acceptance" probability of the current component C, which is the ratio between p_C (the probability of C given the model) and the probability of the previous (accepted) component, but it should not exceed 1. At line 19, with probability $1 - \alpha$, C is rejected and set to the previous one. Note that GENERALIZEDE returns a distribution of the locations (latent variable); the "expectation" (as in "E" of EM) will be readily performed in the M step in Section 4.2. Moreover, upon the return of the last run of E step, the location (i.e., match instance C) in $\hat{L}_i$ with the maximum probability p_C is considered as the best match instance of the final model.

We illustrate GENERALIZEDE in Example 1.

Example 1. *Figure 2(a) shows the current model $\widehat{\mathcal{M}}_i$ where the time depth is $\hat{d}_i = 3, \hat{n}_i = 4$, and the edge probabilities are as shown—for clarity, all solid edges in Figure 2(a) have probability 0.7 and all dashed edges have probability 0.1. Figure 2(b) shows a component C involving the same four vertices across three consecutive snapshots G_8, G_9, and G_{10}, i.e., $C.t = 8$ and $C.d = 3$. Then the probability of this component as calculated in line 2 or 17 is $p_c = (0.7^3 \cdot 0.3 \cdot 0.1 \cdot 0.9) \cdot (0.7^4 \cdot 0.9^2) \cdot (0.7^3 \cdot 0.3 \cdot 0.1 \cdot 0.9)$, where the three sets of parentheses correspond to the edges from G_8, G_9, and G_{10}, respectively. For example, for G_8, the contribution to p_C is $0.7^3 \cdot 0.3 \cdot 0.1 \cdot 0.9$, because among the four solid edges in $\widehat{\mathcal{M}}_i$ of Figure 2(a), G_8 has three of them (each with probability 0.7) and the missing one is with probability 0.3, while for the two dashed edges, one is there (with probability 0.1), and one is absent (with probability 0.9).*

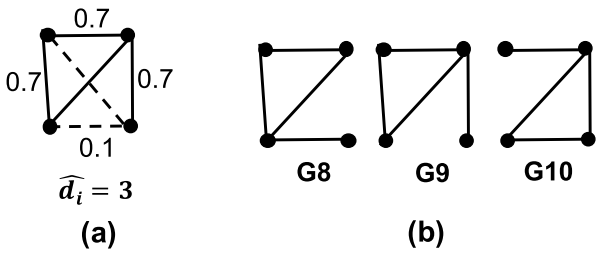


Fig. 2. Illustrating details of GENERALIZEDE. (a) The model $\widehat{\mathcal{M}}_i$ where $\hat{n}_i = 4, \hat{d}_i = 3$, and each solid edge has probability 0.7 and each dashed one 0.1 (simplified for clarity). (b) A component across three snapshots G_8 to G_{10} with the same four vertices.

We now prove some property of the GENERALIZEDE algorithm, as will be part of the correctness proof/validation of the whole EMU algorithms in Section 4.3.

Theorem 2. GENERALIZEDE *performs a Markov chain Monte Carlo sampling, specifically, the Metropolis-Hastings method with a symmetric proposal function, returning a location sample drawn from the probabilistic match instances of $\widehat{\mathcal{M}}_i$ in $\mathcal{G}_T$ for iteration i.*

4.2 Generalized M Step

We now proceed to describing the generalized M step of EMU. Given a location distribution $\hat{L}_i$ returned by GENERALIZEDE, the goal of M step is to estimate (the parameters of) an updated model $\widehat{\mathcal{M}}_{i+1}$, which in turn will be used by the next iteration $(i+1)$'s E step. Our general idea is to estimate the parameters of $\widehat{\mathcal{M}}_{i+1}$ that maximizes $\mathbf{E}[U]$, which is to get three parts: number of vertices $\hat{n}_{i+1}$, edge probabilities $\hat{\rho}_{j,i+1}$, and time depth $\hat{d}_{i+1}$. The key idea is to apply the result of solving the optimization problem associated with the utility function in Equation (2) to set the edge probabilities (Theorem 3), and to use Coordinate Ascent [16] and Gradient Ascent [17] to optimize $\hat{n}_{i+1}$ and $\hat{d}_{i+1}$. The algorithm is presented as GENERALIZEDM.

EMU Algorithm: M step. Lines 2-17 perform iterative Coordinate Ascent optimization [16] over three sets of parameters of $\widehat{\mathcal{M}}_{i+1}$, namely $\hat{\rho}_{j,i+1}, \hat{n}_{i+1}$, and $\hat{d}_{i+1}$. Like the E-step, the algorithm does Metropolis-Hastings sampling to figure out the best model. In line 4, with equal probability, the algorithm tries the adjacent values of the previous iteration's number of vertices. Lines 5-10 revise each component accordingly. Line 11 invokes the function *gradientAscentTimeDepth* (below the main function) to optimize the time depth $\hat{d}_{i+1}$. Line 3 of *gradientAscentTimeDepth* uses the current setting of $\hat{L}_i$ and $\hat{n}$, and first try time depth $\hat{d}_{i+1} + h$, a value slightly greater than the current $\hat{d}_{i+1}$ (e.g., $h = 2$). Then it tries the time depth $\hat{d}_{i+1} - h$. In either case, it calls the function *getModel* to set the edge probabilities.

Line 2 of the function *getModel* iterates over each edge of the subgraph components identified by $\hat{L}_i$ and $\hat{d}$, and retrieves the "expectation" result as in EM. Recall that $\hat{L}_i$

Algorithm 2: GENERALIZEDM $(\widehat{\mathcal{M}}_i, \mathcal{G}_T)$

Input: distribution $\hat{L}_i$, model $\widehat{\mathcal{M}}_i(\hat{n}_i, \hat{\rho}_{ji}, \hat{d}_i)$, dynamic graph $\mathcal{G}_T$
Output: model $\widehat{\mathcal{M}}_{i+1}$

1 $\widehat{\mathcal{M}}_{i+1} \leftarrow null$; $\hat{L}_{prev} \leftarrow \hat{L}_i$; $\hat{n}_{prev} \leftarrow \hat{n}_i$; $\hat{d}_{prev} \leftarrow \hat{d}_i$
2 **while** $\widehat{\mathcal{M}}_{i+1}$ *not converged* **do**
3 $\quad \hat{L} \leftarrow \hat{L}_{prev}$; $\hat{d} \leftarrow \hat{d}_{prev}$
4 $\quad \hat{n} \leftarrow \hat{n}_{prev}$ or $\hat{n}_{prev} - 1$ or $\hat{n}_{prev} + 1$ with equal probability
5 $\quad$ **for** *each component* C *in* $\hat{L}$ **do**
6 $\quad\quad$ **if** $\hat{n} > \hat{n}_{prev}$ **then**
7 $\quad\quad\quad e \leftarrow$ an edge uniformly at random from $N_e(C)$
8 $\quad\quad\quad C.V \leftarrow C.V \cup \{e$'s endpoint not in $C.V\}$
9 $\quad\quad$ **else if** $\hat{n} < \hat{n}_{prev}$ **then**
10 $\quad\quad\quad$ remove random $v \in C.V$ s.t. C is still connected
11 $\quad \hat{d} \leftarrow$ gradientAscentTimeDepth $(\hat{L}, \hat{n}, \hat{d})$
12 $\quad \mathcal{M} \leftarrow$ getModel$(\hat{L}, \hat{n}, \hat{d})$
13 $\quad$ **if** $\widehat{\mathcal{M}}_{i+1} = null$ *or* $u(\mathcal{M}) > u(\widehat{\mathcal{M}}_{i+1})$ **then**
14 $\quad\quad \widehat{\mathcal{M}}_{i+1} \leftarrow \mathcal{M}$
15 $\quad \alpha \leftarrow \min\,(1, \frac{u(\mathcal{M})}{u(\widehat{\mathcal{M}}_{i+1})})$
16 $\quad$ **with** *probability* α **do**
17 $\quad\quad \hat{L}_{prev} \leftarrow \hat{L}$; $\hat{n}_{prev} \leftarrow \hat{n}$; $\hat{d}_{prev} \leftarrow \hat{d}$
18 **return** $\widehat{\mathcal{M}}_{i+1}$

1 **Function** *gradientAscentTimeDepth* $(\hat{L}, \hat{n}, \hat{d}_{i+1})$
2 $\quad$ **while** *true* **do**
3 $\quad\quad \mathcal{M}_{+h} \leftarrow$ getModel $(\hat{L}_i, \hat{n}, \hat{d}_{i+1} + h)$
4 $\quad\quad \mathcal{M}_{-h} \leftarrow$ getModel $(\hat{L}_i, \hat{n}, \hat{d}_{i+1} - h)$
5 $\quad\quad \Delta u \leftarrow u(\mathcal{M}_{+h}) - u(\mathcal{M}_{-h})$
6 $\quad\quad$ **if** $\Delta u < \epsilon$ **then**
7 $\quad\quad\quad$ **break**
8 $\quad\quad \hat{d}_{i+1} \leftarrow \hat{d}_{i+1} + \gamma \frac{\Delta u}{2h}$
9 $\quad$ **return** $\hat{d}_{i+1}$

1 **Function** *getModel* $(\hat{L}_i, \hat{n}, \hat{d})$
2 $\quad$ **foreach** *edge* j *in the subgraphs identified by* $\hat{L}_i$ *and* $\hat{d}$ **do**
3 $\quad\quad n_j^+ \leftarrow$ **E**[number of snapshots that has edge j] from $\hat{L}_i$
4 $\quad\quad \hat{\rho}_j \leftarrow \sqrt{\frac{n_j^+}{\hat{d}}}$
5 $\quad$ **return** $\mathcal{M}(\hat{n}, \hat{\rho}_j, \hat{d})$

consists of pairs (C, p_C). Suppose edge j in line 2 is between vertices u and v in the subgraph model, which are mapped to vertices u_i and v_i in each component i, respectively. The idea is to perform a weighted "average" (expectation) over the $|C|$ components, to draw a conclusion whether edge j, i.e., (u, v), exists in each of the $\hat{d}$ snapshots. The expectation will be a value in $[0, 1]$. Summing this expectation over the $\hat{d}$ snapshots (and based on the linearity of expectation), the result is the expected number of snapshots that contain a match for edge j, which is n_j^+ in line 3 of *getModel*. Then in line 4, the edge probability $\hat{\rho}_j$ is set based on the optimization result using the utility function (Theorem 3).

Example 2. *Revisiting the example in Figure 2, suppose Figure 2(a) is the current model and Figure 2(b) is only one of the $|C|$ components in $\hat{L}_i$. For clarity, suppose there are only two components $|C| = 2$, and the component shown in Figure 2(b) has probability 0.8, while the other component (not shown) has probability 0.2. Line 2 of getModel iterates through each edge of the model in Figure 2(a); let us take one edge as an example, the left vertical solid edge. In the component shown in Figure 2(b), this edge appears in 2 snapshots (G_8 and G_9) out of $\hat{d} = 3$ snapshots. In the other component not shown, suppose this edge appears in all 3 snapshots. Then the expected value n_j^+ calculated in line 3 of getModel is $2 \times 0.8 + 3 \times 0.2 = 2.2$. The edge probability in line 4 is $\hat{\rho}_j = \sqrt{\frac{2.2}{3}} = 0.856$. This is repeated for all other edges of the model in Figure 2(a).*

Back to the *gradientAscentTimeDepth* function, in lines 5 and 8, it estimates the gradient of the model utility function and adjusts $\hat{d}_{i+1}$ with a value proportional to it (where γ is a small constant), based on Gradient Ascent [17]. Lines 6-7 are to exit the loop at convergence. This function estimates the optimal $\hat{d}_{i+1}$ under the current $\hat{n}$ and ρ_j's.

After *gradientAscentTimeDepth* is invoked in line 11 and the best $\hat{d}$ is obtained, the algorithm retrieves the currently chosen model in $\mathcal{M}$ at line 12, and at lines 13-14 sets it to $\widehat{\mathcal{M}}_{i+1}$ if it is the best so far. At lines 15-17 it completes the MH sampling by setting the acceptance probability α. Once the current candidate is accepted, its $\hat{L}, \hat{n}$, and $\hat{d}$ are bookkept as the next iteration's starting point (line 17).

EMU iteratively interleaves GENERALIZEDE and GENERALIZEDM, until the model converges, and the maximum probability component is returned as the densest lasting subgraph. We analyze the correctness of GENERALIZEDM.

Theorem 3. *With the utility function in Equation (2), given the parameters $\hat{n}$ and $\hat{d}$ of the model and a location distribution $\hat{L}$ of the model in the dynamic graph, the edge probability parameters $\hat{\rho}_j$ that maximizes $\boldsymbol{E}[U]$ in Equation (1) is $\hat{\rho}_j = \sqrt{\frac{n_j^+}{\hat{d}}}$, where n_j^+ is the expected number of occurrences of edge j in $\hat{L}$ (as in line 3 of the getModel function).*

Theorem 3 justifies the choice of the *getModel* function (line 4). We now justify the correctness of GENERALIZEDM.

Theorem 4. *Given the location distribution $\hat{L}_i$ from* GENERALIZEDE, *the* GENERALIZEDM *algorithm does Coordinate Ascent to optimize three groups of parameters of model $\widehat{\mathcal{M}}$: $\hat{n}$, $\hat{\rho}_j$'s, and $\hat{d}$.*

Both GENERALIZEDE and GENERALIZEDM employ Metropolis-Hastings sampling—although there are no theoretical guarantees when it will converge to the stationary distribution, in practice [18], several thousand iterations are typically used as the "burn-in" period. After that, the cost of GENERALIZEDE is linear to the model size, given that the distribution size n_c is a constant (we study n_c in Section 6, of which we use 50 as the default). Similarly, the cost of GENERALIZEDM has the cost of gradient ascent as a linear factor, which takes $O(1/\epsilon)$ iterations [17] where ϵ is the allowed error in line 6 of $gradientAscentTimeDepth$.

4.3 Validation of EMU Algorithms

In Sections 4.1 and 4.2, we have individually shown the correctness of the generalized E step and M step, respec-

tively, in terms of their own roles. It remains to show that our extension from EM to EMU is valid, i.e., the EMU algorithms still *converge* to the optimum as EM does. We do so by proving that EMU falls into a more general statistical optimization framework called MM [7], which has been proven to converge to the optimal objective values.

Preliminary on MM. The MM algorithm framework is an iterative optimization method which exploits the convexity of a function in order to find their maxima or minima. The MM stands for "Majorization-Minimization" or "Minorization-Maximization", depending on whether it is a minimization or a maximization problem, respectively. EM can be treated as a special case of MM, although EM has been more widely known for its applications.

Let $f(\theta)$ be the objective function for which we want to find the location of the *maximum* value, as illustrated in Figure 3 (the minimization problem is similar). The MM algorithm works by finding a *surrogate* function $g(\theta|\theta_i)$ that *minorizes* $f(\theta)$, meaning that $f(\theta) \geq g(\theta|\theta_i)$ for all θ, and $f(\theta_i) = g(\theta_i|\theta_i)$, as shown in Figure 3. Note that θ_i denotes the parameter value at the i'th iteration of MM. Thus, θ_i (with a subscript) is a constant, while θ is a variable. $g(\theta|\theta_i)$ is a function over θ. The above minorization condition says that the function curve/surface $g(\theta|\theta_i)$ lies below that of $f(\theta)$, and is tangent to it at the current iteration $\theta = \theta_i$ (Figure 3).

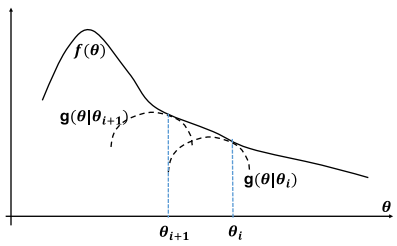


Fig. 3. Illustrating MM. $f(\theta)$ is the objective function, and $g(\theta|\theta_i)$ is the surrogate function we maximize at each iteration i instead. This continues for iteration $i+1$, and so on.

MM is also iterative, where each iteration has two "M" steps. The construction of the minorizing function $g(\theta|\theta_i)$ constitutes the first M step, and the second M step maximizes the surrogate $g(\theta|\theta_i)$ rather than $f(\theta)$ directly. The marching of θ_i and the surrogate functions relative to the objective function is shown in Figure 3. We refer the reader to [7] for more details of MM.

Theorem 5. *The EMU algorithm (given in Sections 4.1 and 4.2) is also an MM (Minorization-Maximization) algorithm.*

We also provide some insight on our random search approach in [15].

5 Discovering Subgraphs with Different Timing Characteristics

In this section, we generalize our method to the discovery of subgraphs with different timing characteristics:

- Sudden short spikes and bursts with dense subgraphs.
- Sudden densification of an area of the network.
- Sudden dissolution of a dense area of the network.
- Discovery from snapshots weighted by time of occurrence—e.g., more recent ones have higher weights.

It turns out that, interestingly, we can use our EMU framework to achieve any of these goals too, by defining different utility functions, or by customizing the weights of match instances.

5.1 Discovering Densest Subgraph Spikes, Densification, and Dissolution

5.1.1 Motivations

Let us first look at a few motivating applications.

Example 3. *In brain science and neuroscience, complex network topologies represent the necessary substrate to support complex brain functions [19]. In this network, nodes are neurons, and edges model the connections (morphological or functional) among the neurons. Neurons communicate with other neurons in the form of all-or-none action potentials, i.e., short-time spikes. These spikes are the brain's language for encoding information, both extracted from external stimuli and sent by internal sources. For instance, in acute slices, network bursts may be regarded as a sign of* epilepsy *[20]. Therefore, it is very important to effectively find the densest subgraphs that last for very short time periods, surrounded by periods of relative quiescence.*

Example 4. *Some security attacks in computer networks, such as port scan or denial of service, are marked by a large dense subgraph that lasts for a short time [21]. They often evade the conventional network security tools due to its extremely short time duration. In addition, the fact that there is virtually no edge connection activities before and after the attack is also among the salient characteristics of such attacks. Thus, from the network logs, it will be very helpful to discover such short but dense subgraphs with significantly reduced or no activities before and after dense period.*

In both examples above, our previously selected utility function and algorithm cannot be directly applied, for the following two reasons:

1) It will select dense subgraphs that also last for a long time, and will almost certainly miss the ones that are dense but lasts for a very short time.
2) It will not take into consideration the characteristic quiescence periods before and after the dense period.

Thus, we will need a different method. To the best of our knowledge, finding densest subgraph spikes in general graphs has not been well-studied thus far. For instance, Eswaran et al.'s work [21] only addresses bipartite graphs and does not optimize the appearance and disappearance of a dense subgraph simultaneously.

5.1.2 A Universal Framework to Design Utility Functions

We present a simple and elegant universal framework to design utility functions for a broad range of optimization problems that fall into a general model.

We illustrate our general model of the utility function in Figure 4. The key idea is to generalize the utility function

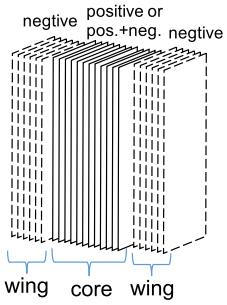


Fig. 4. Illustrating our general model of the utility function. There are *core* layers (snapshots) in the middle, with optional left and/or right *wings* before and after in time. The core may consist of either all positive layers/edges (e.g., in the case of densest lasting subgraphs) or a positive layer followed by negative ones (e.g., in finding spike dense subgraphs). The wings, if they exist, typically are only negative layers.

model proposed in Equation (2) as follows. It is comprised of *core* layers (i.e., graph snapshots) at the center (in time), along with optional left and/or right *wings* before and after them. The utility function in Equation (2) is a special case in which we only have the core part. Intuitively, the wings are to enforce the requirements of "suddenness"—the time instants before and/or after the core densest subgraphs devoid of the edges in the model.

In our stochastic model $\mathcal{M}$ for finding densest subgraphs, and in the utility function of Equation (2), we call the term $e^{\rho_j-\alpha}$ for edge j in $\mathcal{M}$ a *positive edge-factor*, as it rewards the presence of an edge in $\mathcal{M}$. Model $\mathcal{M}$ has depth d, which we call d *layers*, and edge j has d positive edge-factors over them, i.e., $(e^{\rho_j-\alpha})^d = e^{d(\rho_j-\alpha)}$ in Equation (2). This is true for every edge in $\mathcal{M}$; we say that the model $\mathcal{M}$ has d *positive layers* in its core.

More generally, we may let the core of $\mathcal{M}$ have any number of positive and negative layers, where a *negative layer*, analogously, rewards the absence of an edge in it. In particular, in the problem of finding spike densest subgraphs, we have one positive layer and $d-1$ negative layers, i.e., a thinner core has a higher utility.

Accordingly, we propose that a *negative edge-factor* for edge j is of the form $e^{1-\rho_j-\alpha}$. Thus, a *negative edge* in a *negative layer* is a logical complement of a positive edge—a positive edge has probability ρ_j in $\mathcal{M}$, while a negative edge has probability $1-\rho_j$. This applies to all edges in a negative layer.

The same simple and uniform treatment applies to the (optional) wings too. Edge j in a layer of a wing has a negative edge factor $e^{1-q_j-\alpha}$, where q_j is the existence probability of edge j in a wing. As can be seen by now, this is essentially a *composable* and universal utility function framework. Several interesting instantiations of this general framework are as follows:

- Only positive layers in the core and no wings. This is the densest lasting subgraph problem studied above.
- One positive layer and $d-1$ negative layers in the core, along with two wings of negative layers. This is the *densest subgraph spike* problem.
- All positive core plus two negative wings. This identifies the sudden appearance and disappearance of a dense region.
- All positive core plus a negative left wing. This identifies the sudden densification of an area of the network.
- All positive core plus a negative right wing. This corresponds to a sudden dissolution of a densely active region.

In particular, for the densest subgraph spike problem, the utility function is:

$$u(\mathcal{M}) = \prod\nolimits_{j\in E(\mathcal{M})} e^{\rho_j-\alpha+(d-1)(1-\rho_j-\alpha)} \prod\nolimits_{j\in E(\mathcal{M})} e^{2w(1-q_j-\alpha)} \tag{3}$$

where the first product is for the core and the second for the two wings, each of which has w layers, and q_j is the probability of edge j in the wings. In the first product, $e^{\rho_j-\alpha}$ is for the positive layer, while $e^{(d-1)(1-\rho_j-\alpha)}$ is for the $d-1$ negative layers. Thus far, it appears that all our utility functions are of the form of a product of natural exponential functions. This is partly for the convenience of algebraic manipulation in maximizing Equation (1), and partly for magnifying the effect of ρ_j for fast convergence to the desired density. However, in principle, as long as the M step can maximize Equation (1), other functions could work too, which we leave for future study.

5.1.3 Setting the Parameters

We first discuss how to set the parameter α in Equation (3) for the densest subgraph spike problem. Since we are looking for spikes, it does not make sense for the core duration d to be too large, as otherwise it would not be a "spike"—thus, let the upper bound allowed by the application be $d \le D$. We first state the following desideratum of a reasonable utility function.

Desideratum 1 (Monotonicity). *For the same depth d in a densest subgraph spike model $\mathcal{M}$, extending $\mathcal{M}$ with more edges (thus making it denser) should increase its utility.*

We are now ready to state the following result for selecting the parameter α.

Theorem 6. *To achieve Desideratum 1, we must set the parameter $\alpha \le \frac{1}{D}$, where $D \ge 2$ is the upper bound of the model depth d.*

Proof. Consider two densest subgraph spike models $\mathcal{M}_1$ and $\mathcal{M}_2$ where $\mathcal{M}_1$ has m edges and $\mathcal{M}_2$ has km edges ($k \ge 1$), with all other parameters being the same. Then Desideratum 1 requires that $u(\mathcal{M}_1) \le u(\mathcal{M}_2)$. From Equation (3), for the simple case where all edges have the same probability $\rho > \alpha$, we have $e^{(\rho-\alpha)m(2-d)+m(1-2\alpha)(d-1)} \le e^{(\rho-\alpha)km(2-d)+km(1-2\alpha)(d-1)}$.

Simplifying the above inequality, we have $(\rho-\alpha)(2-d)+(1-2\alpha)(d-1) \ge 0$. For $d=1$, this inequality holds as long as $\rho \ge \alpha$ (which is required for it to be a dense subgraph). For $d=2$, it holds as long as $\alpha \le \frac{1}{2} \le \frac{1}{D}$, as $D \ge 2$. Finally, for $d>2$, we must have $\rho \le \alpha + \frac{(1-2\alpha)(d-1)}{d-2}$, which will hold if $\alpha + \frac{(1-2\alpha)(d-1)}{d-2} \ge 1$. With some algebraic manipulation, this leads to $\alpha \le \frac{1}{d} \le \frac{1}{D}$. □

Theorem 6 gives us insight on how to set the model parameter α. It also shows the validity of our universal utility

function framework. For example, it is easy to verify that an alternative design of setting the negative edge factor to $e^{-(q_j-\alpha)}$ (instead of $e^{1-q_j-\alpha}$) does not satisfy Desideratum 1 for any α. We next discuss how to iteratively set the model parameters ρ_j and q_j for each edge j in $\mathcal{M}$ in the revised GENERALIZEDM algorithm.

Theorem 7. *For the densest subgraph spike problem using Equation (3) as the utility function, to maximize $\mathbf{E}[U]$ in Equation (1), ρ_j and q_j are set as follows. Let n_j^+ and n_j^- be the expected numbers of occurrences of edge j in the core and wings in $\hat{L}$ (getModel function), respectively. Then, if $d = 1$, $\hat{\rho}_j = \sqrt{n_j^+}$; if $d = 2$, $\hat{\rho}_j = \frac{n_j^+}{2}$; if $d > 2$, $\hat{\rho}_j = \frac{d-1}{d-2} - \sqrt{(\frac{d-1}{d-2})^2 - \frac{n_j^+}{d-2}}$. Moreover, $\hat{q}_j = 1 - \sqrt{1 - \frac{n_j^-}{2w}}$.*

Proof. Given a model $\mathcal{M}$, the probability of the weighted average component of $\hat{L}$ is

$$\Pr[\mathcal{G}_d|\mathcal{M}] = \prod_{j\in E} \rho_j \cdot \prod_{j\notin E}(1-\rho_j) \prod_{j\in E_w} q_j \cdot \prod_{j\notin E_w}(1-q_j) \tag{4}$$

where E and E_w are the set of edges in the core and wings in this component in data, respectively.

Based on Equations (3) and (4), taking the derivative of the log of Equation (1) to get maximum $\mathbf{E}[U]$ gives us

$$\frac{\partial \ln \mathbf{E}[U]}{\partial \rho_j} = 2 - d + \frac{n_j^+}{\rho_j} - \frac{d - n_j^+}{1-\rho_j} = 0 \tag{5}$$

From Equation (5), we further get $(d-2)\rho_j^2 + 2(1-d)\rho_j + n_j^+ = 0$, which gives us the results on $\hat{\rho}_j$ for the cases of $d = 1$, $d = 2$, and $d > 2$ as stated in the theorem. Likewise, we have

$$\frac{\partial \ln \mathbf{E}[U]}{\partial q_j} = -2w + \frac{n_j^-}{q_j} - \frac{2w - n_j^-}{1-q_j} = 0 \tag{6}$$

which finally gives us $\hat{q}_j = 1 - \sqrt{1 - \frac{n_j^-}{2w}}$. □

To see a concrete numeric example, suppose the model core depth is $d = 6$, and a particular edge j in expectation appears 4 times (i.e., $n_j^+ = 4$) in the match component distribution ($\hat{L}$), and in expectation appears 0.36 fraction in the two wings (i.e., $\frac{n_j^-}{2w} = 0.36$). Then from Theorem 7, we set $\hat{\rho}_j = 0.5$ and $\hat{q}_j = 0.2$ in the GENERALIZEDM algorithm.

A final remark is that we can set the parameters in an analogous manner as Theorem 7 for other instantiations of our universal utility function and other versions of the problem as listed in Section 5.1.2.

5.2 Other Generalizations and Extensions

Adding node/edge labels. Our powerful utility function framework also allows us to handle graphs of different variants. For example, we may include the distinctions of *edge and vertex labels* in the selection of densest temporal subgraphs by extending the utility function, for which we have the following result:

Theorem 8. *In our algorithms for the densest lasting subgraph problem, given the location distribution $\hat{L}$, and the current parameters $\hat{n}$ and $\hat{d}$ of the model, to maximize $\mathbf{E}[U]$ (Section 3.1), the edge label probability of the model $\hat{\rho}_{lj}$ for edge j and label $l \in \Sigma$ should be set to $\hat{\rho}_{lj} = \frac{n_{lj}^+}{\sqrt{\hat{d}^2 - \hat{d}n_{\perp j}^+}}$, where n_{lj}^+ (resp., $n_{\perp j}^+$) is the expected number of snapshots in which edge j has label l (resp., does not exist) among the $\hat{d}$ snapshots in $\hat{L}$.*

The details, as well as the proof of Theorem 8, are in [15].

Adding node/edge weights and adding time weights. In the same vein, we may take advantage of the flexibility of the EMU framework by setting the utility function for other scenarios, such as giving weights to edge/vertex labels for appearing in the densest lasting subgraph, as well as handling parallel edges (i.e., multiple edges at the same time between two vertices).

Finally, we may also give priorities and weights to data at different times in the past. For instance, in practice it often makes sense to prefer finding more recent dense subgraphs than older ones. Interestingly, for this extension, it is different from generalizing the utility function. Instead, one can multiply a time factor t^γ (where $\gamma > 0$ is a constant) to the probabilities of each component in the component location distribution $\hat{L}$. In this way, a more recent component (i.e., match instance) in data will have higher weights (i.e., greater t^γ).

6 EXPERIMENTS

6.1 Datasets and Setup

Datasets. We use four real-world datasets: **(1) Twitter.** Twitter Stream API [22] is used to retrieve data from Jan. 22 to May 21, 2017. For Twitter and Stack Overflow below, we treat users as vertices and edges as communications, and the edge duration is the time period in which the user who initiates the communication keeps active and communicating with one or more users (without pausing for more than 30 seconds). **(2) Taxi.** The trip data is about 30GB, containing the information of all taxi trips in NYC in 2013 [23]. It has 14 attributes. Each trip from the pick-up to the drop-off locations is an edge, with the trip time as edge duration and the number of passengers as edge label. **(3) Stack Overflow [24].** This is a network of interactions on the web site Stack Overflow [25]. There are three types of interactions represented by a directed edge (u, v, t), where user u answered or commented on user v's question or comment at time t. **(4) NY events [26].** It contains traffic and transit events in NY. We partition the NY state into 0.0011-degree latitude by 0.0012-degree longitude grid areas. Each grid area is a vertex and the event type is a vertex label. When a transportation event happens at a vertex (e.g., an accident), we create eight edges indicating its impact to its eight surrounding grid areas, and the event duration is the edge duration. The statistics of the datasets are in Table 1.

Methods. We compare our methods with two related approaches in [4] and [5] briefly reviewed below.

Dense pattern. The "Dense pattern" approach in [4] returns a set of vertices that are similar with respect to the snapshots they appear in, and that have dense edges between them. There are a few issues. First, the "similarity" above is based on set similarity, for the set of snapshots each vertex appears

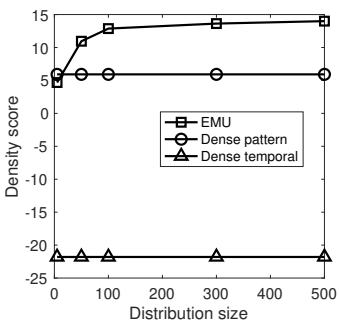


Fig. 5. Density vs distrib. size(Twitter)

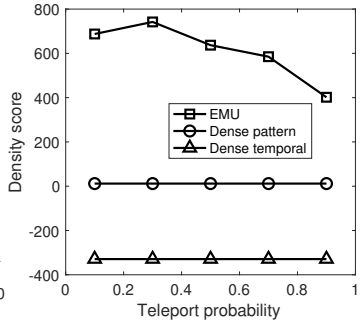


Fig. 6. Density vs teleport prob.(Taxi)

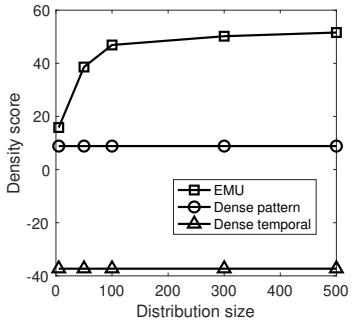


Fig. 7. Density vs distrib. size(Stack)

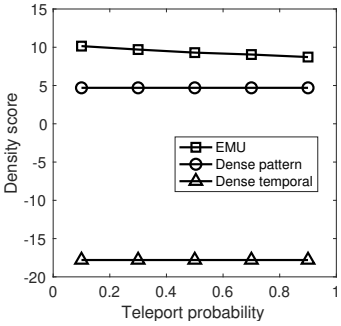


Fig. 8. Density vs teleport prob.(Events)

TABLE 1
Statistics of the datasets

Dataset	#vertices	#edges	Average inter-arrival time	Data Size
Twitter	17,274,424	119,604,457	0.1 sec	4.9GB
Taxi	5,654	169,100,000	0.186 sec	5.62GB
Stack Overflow	2,601,977	63,497,050	3.775 sec	1.64GB
NY events	429,456	10,148,112	161.62 sec	138.7MB

in. This is too restrictive. Second, it returns "small" subgraphs. In step 1, [4] uses a MinHash method for finding a set of vertices. If the probability that two vertices are similar is p, the probability that three vertices are similar is p^2, and so on. Thus, the probability that such similar pair of vertices joining a returned vertex set decreases exponentially. Finally, it does not consider "continuity" strength over time.

Dense temporal subgraph. This approach in [5] uses two phases: (1) locating top-k promising snapshot intervals without considering any subgraphs, and (2) finding the heaviest-weight subgraph in each of those intervals. Although it is efficient, there are a few major issues for our problem. First, it uses a model where all edges are present in all snapshots, while only the weights vary. The "evolving convergence phenomenon" relied upon in [5] is to assume that edge weights of the whole graph increase or decrease in the same direction at any time. This does not hold for the applications we look at. For example, for communication graphs, over a long period of time, the number of times of *starting* a communication should match the number of times of *ending* a communication, and the weight increase and decrease may happen simultaneously in any graph snapshot. Second, it would tend to return huge subgraphs. This is because the connection between two positive weight components is a single edge (with a minimum weight of -1 in each snapshot of the selected time interval). Thus, there is a good chance that the algorithm will merge the two positive components to have a higher total weight, resulting in a component too large to be a meaningful result.

We implement all the algorithms in Java. The experiments are performed on a MacBook Pro machine with OS X version 10.11.4, a 2.5 GHz Intel Core i7 processor, a 16 GB 1600 MHz DDR3 memory, and a Macintosh hard disk.

6.2 Experimental Results

We first study the effectiveness of our EMU approach, i.e., the quality/density of the subgraphs discovered. We compare our method with the two related methods, denoted as "Dense pattern" [4] and "Dense temporal" [5], respectively. For a fair comparison, we define a metric called *density score* $s = [m - \alpha\binom{n}{2}]d^{\beta}$. Here, n is the number of vertices in the found subgraph pattern, d is the time duration—the number of consecutive snapshots where each snapshot has a duration of 3 hours, and m is the number of edges in this subgraph that appears in at least half of the snapshots ($d/2$). Following [12], we set $\alpha = 1/3$ and $\beta = 1/2$ by default. Intuitively, this metric indicates the "surplus" number of edges compared to a (discounted) complete graph over all vertices in the selected subgraph and across d snapshots.

Effectiveness on Real Datasets

Twitter. Recall that two parameters are *location distribution size* and *teleport* probability in GENERALIZEDE. Figure 5 shows the density scores of the densest lasting-subgraphs returned by EMU over Twitter data, as we vary the location distribution size. It also shows the density scores of the results returned by previous work dense pattern [4] and dense temporal [5] (which remain constants as they do not have such a parameter). As the distribution size increases, the density scores improve, eventually converging. This is because exploring a larger location distribution gives a higher chance to get to a denser lasting-subgraph, indicating a trade-off between result quality and performance. We use 50 as the default size. As explained earlier, the dense pattern approach [4] tends to return small subgraphs, which get smaller density scores. The dense temporal approach [5], on the other hand, tends to return very large subgraphs, resulting in negative scores (i.e., the number of edges are few compared to the complete graph over those vertices).

Taxi. In Figure 6, we vary the teleport probability between 0.1 and 0.9, and measure the density scores over Taxi. (1) The maximum score occurs when the teleport probability is around 0.3 (which we use as default). Indeed, teleports help when the search is stuck at a local maximum. Too frequent teleports, however, may prematurely abort or delay a good location. (2) The density scores achieved over Taxi are much higher than Twitter in Figure 5, due to the nature of the datasets—traffic tends to be much denser with longer-lasting edges than communication networks. The dense pattern [4], however, still ranks small subgraphs as they are much more likely to be discovered, resulting in small scores. The dense temporal subgraph [5], on the other hand, returns subgraphs that are too large with negative scores.

Stack Overflow and NY Events. Similarly, we report the den-

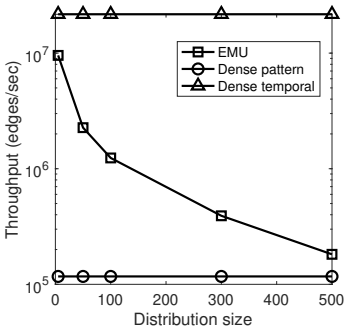


Fig. 9. Speed vs distrib. size(Twitter)

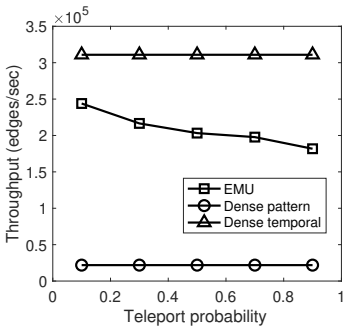


Fig. 10. Speed vs teleport prob.(Taxi)

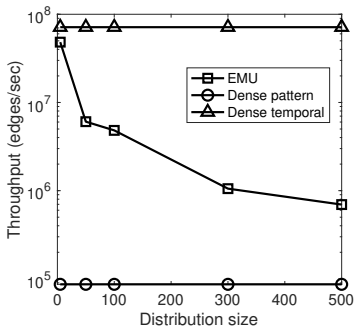


Fig. 11. Speed vs distrib. size(Stack)

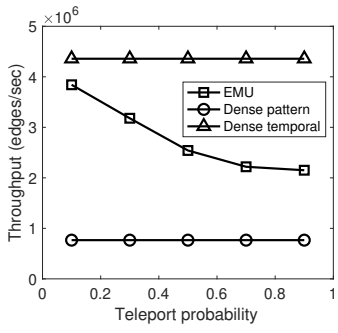


Fig. 12. Speed vs teleport prob.(Events)

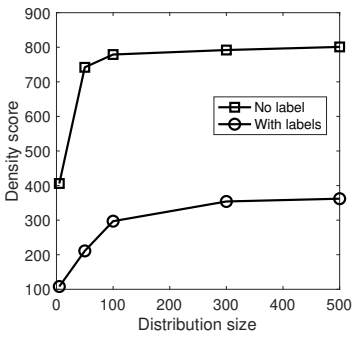


Fig. 13. Density w/ edge labels(Taxi)

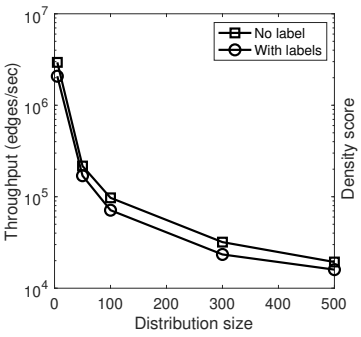


Fig. 14. Speed w/ edge labels(Taxi)

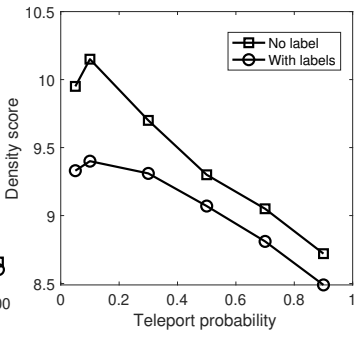


Fig. 15. Density w/ vertex labels(Events)

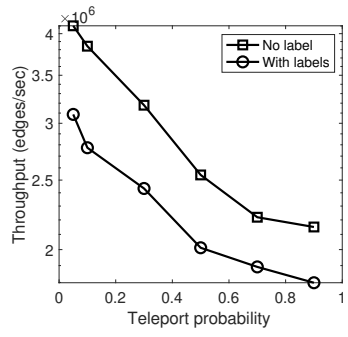


Fig. 16. Speed w/ vertex labels(Events)

sity scores vs. distribution sizes over the Stack Overflow data in Figure 7. The trend is consistent with that in Figure 5, except that the scores are higher. Varying the teleport probability, we report the result using NY events in Figure 8. The density scores are relatively low for this dataset, as the events such as accidents and delays are usually not very dense. Moreover, compared to Figure 6, the density scores are less sensitive to the teleport probability.

Efficiency on Real Datasets. We next evaluate the efficiency, reported as the number of processed edge changes per second, as shown in Figure 9 for Twitter. The performance of EMU algorithms degrades as we increase the location distribution size, because the increase significantly slows down both the E and M steps. We also see that the dense temporal approach [5] is significantly faster than EMU, while the dense pattern approach is slower. Dense temporal approach is faster because its phase (1) discards a large amount of data, as it only heuristically selects a small number of intervals, which also eases its subgraph search in phase (2). The dense pattern approach [4], however, needs to process a large "transaction table" for the dataset.

In the same vein, we also measure the performance over Taxi for various teleport probabilities in Figure 10, over Stack Overflow for various distribution sizes in Figure 11, and over NY events for various teleport probabilities in Figure 12. The performance decreases as the teleport probability increases, as performing a teleport is more expensive for obtaining a whole random subgraph component, while making local changes only slightly revises a component.

Adding Vertex/Edge Labels. We now look into adding edge or vertex labels, which the two approaches in [4] and [5] cannot handle. For Taxi, we consider the number of passengers in a trip as an edge label. Thus, the densest lasting-subgraph found contains matching edge labels. The density score comparison for with vs. without edge labels under various distribution sizes is shown in Figure 13. The scores are in general lower when we consider edge labels. This is because the chance of matching an edge with a specific label is smaller, compared to simply matching an edge. In Figure 14, we show the performance comparison. It becomes slower with edge labels because the *entropy* of the problem instance is higher with edge labels, i.e., there are more variables in the dynamic graph. Thus, it takes longer for EMU to converge.

We next examine the effect of vertex labels using NY events, where a vertex label indicates the type of events at the vertex (grid area). We show the density scores in Figure 15 and performance in Figure 16 vs. teleport probability. The densest lasting subgraph model discovered contains vertex labels "delays" and "accident", indicating that they are the most common event types. The comparison result with and without vertex labels is similar to its counterpart with and without edge labels—the density scores and performance are lower with vertex labels, for the same reason as for edge labels. We find that the optimal teleport probability is around 0.1.

Discovering Densest Subgraph Spikes. We now evaluate our approach on discovering densest subgraph spikes, as presented in Section 5.1. We use the Twitter and the Taxi datasets, where looking for sudden short subgraph spikes is more practically significant. For Twitter data it means sudden appearance and disappearance of a hot topic, while for Taxi data it means sudden densification followed by quickly dissolution of traffic possibly due to a short-term event.

We naturally adapt our previously used density score definition to this new use as $s = [m - \alpha\binom{n}{2}]d^\beta + [\binom{n}{2} - m - \alpha\binom{n}{2}](2w)^\beta$, where $\beta = -1/2$, the first term corresponds to

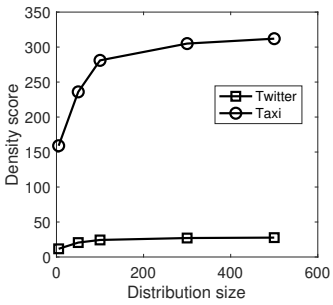


Fig. 17. Density scores

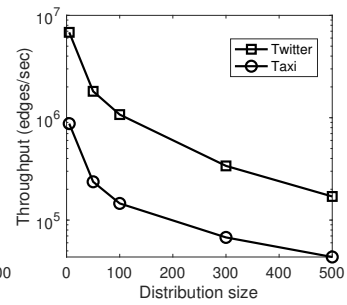


Fig. 18. Throughput

the core part of the utility, and the second term corresponds to the negative layers at the wings. We show the density scores using the two datasets in Figure 17 for various distribution sizes and the throughput in Figure 18.

We can see that for both datasets, the density scores increase as we increase the distribution sizes, while the throughput decreases. This is due to the tradeoff between processing overhead and accuracy. Density scores level off as we keep increasing the distribution size. Moreover, we obtain higher density scores with the Taxi data than the Twitter data, demonstrating that the Taxi data has more dramatic spikes than the Twitter data.

Summary. Our experimental study using four real-world data sets demonstrates that our EMU framework is effective in solving the problem of finding densest lasting-subgraph as observed in many dynamic graph applications. Our algorithms outperform two state-of-the-art densest subgraph discovery methods in [4] and [5] in effectiveness measured by density score, and strikes a balance between too small and too large dense subgraphs. The location distribution size of our EMU algorithms provides a tradeoff between result quality and performance, and the optimal teleport probability is often between 0.1 and 0.3 in our empirical study. Finally, EMU algorithms readily extend to coping with edge or vertex labels and other timing characteristics, which cannot be supported by the baseline methods.

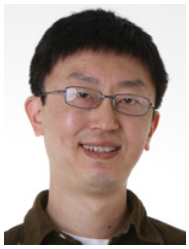
7 Conclusions

We propose a novel probabilistic subgraph model to characterize densest temporal subgraphs in a dynamic graph, and a stochastic approach, EMU, which nontrivially extends EM with a utility function for the desired objective. Based on the semantics that we propose for densest lasting subgraphs, we devise EMU algorithms using MH sampling and coordinate and gradient ascent, and prove the correctness of EMU by showing its membership in MM algorithms. Our experiments over four real-world datasets verify the effectiveness and efficiency of our algorithms.

References

[1] T. R. Weiss, "In emergencies, can cell phone network overload be prevented?" *Computer World*, 2007.
[2] J. Chen and Y. Saad, "Dense subgraph extraction with application to community detection," *TKDE*, 2012.
[3] A. Stathopoulos and M. G. Karlaftis, "Modeling duration of urban traffic congestion," *Journal of Transportation Engineering*, 2002.
[4] C. C. Aggarwal, Y. Li, P. S. Yu, and R. Jin, "On dense pattern mining in graph streams," *VLDB*, 2010. [Online]. Available: http://dx.doi.org/10.14778/1920841.1920964
[5] S. Ma, R. Hu, L. Wang, X. Lin, and J. Huai, "Fast computation of dense temporal subgraphs," in *ICDE*, 2017.
[6] M. R. Gupta and Y. Chen, "Theory and use of the EM algorithm," *Found. Trends Signal Process.*, 2011.
[7] D. R. Hunter and K. Lange, "A tutorial on MM algorithms," *Amer. Statist*, 2004.
[8] M. Charikar, "Greedy approximation algorithms for finding dense components in a graph," in *APPROX*, 2000.
[9] A. V. Goldberg, "Finding a maximum density subgraph," Berkeley, CA, USA, Tech. Rep., 1984.
[10] N. Tatti and A. Gionis, "Density-friendly graph decomposition," in *WWW*, 2015.
[11] J. Abello, M. G. C. Resende, and S. Sudarsky, "Massive quasi-clique detection," in *LATIN*, London, UK, UK, 2002.
[12] C. Tsourakakis, F. Bonchi, A. Gionis, F. Gullo, and M. Tsiarli, "Denser than the densest subgraph: Extracting optimal quasi-cliques with quality guarantees," in *KDD*, 2013.
[13] P. Bogdanov, M. Mongiovì, and A. K. Singh, "Mining heavy subgraphs in time-evolving networks," in *ICDM*, Dec 2011, pp. 81–90.
[14] A. Angel, N. Sarkas, N. Koudas, and D. Srivastava, "Dense subgraph maintenance under streaming edge weight updates for real-time story identification," *VLDB*, 2012.
[15] X. Liu, T. Ge, and Y. Wu, "Finding densest lasting subgraphs in dynamic graphs: A stochastic approach," in *ICDE*, 2019.
[16] S. J. Wright, "Coordinate descent algorithms," *Math. Program.*, 2015. [Online]. Available: http://dx.doi.org/10.1007/s10107-015-0892-3
[17] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge University Press, 2004.
[18] D. MacKay, "Introduction to Monte Carlo methods," *Learning in Graphical Models*, 1999.
[19] D. Poli, V. P. Pastore, and P. Massobrio, "Functional connectivity in in-vitro neuronal assemblies," *Frontiers in Neural Circuits*, 2015.
[20] F. Zeldenrust1, W. J. Wadman, and B. Englitz, "Neural coding with bursts–current state and future perspectives," *Frontiers in Computational Neuroscience*, 2018.
[21] D. Eswaran, C. Faloutsos, S. Guha, and N. Mishra, "SpotLight: Detecting anomalies in streaming graphs," in *KDD*, 2018.
[22] https://dev.twitter.com/streaming/overview, 2018.
[23] https://www.reddit.com/r/bigquery/comments/28ialf/173_million_2013_nyc_taxi_rides_shared_on_bigquery/, 2018.
[24] http://snap.stanford.edu/data/sx-stackoverflow.html, 2018.
[25] https://stackoverflow.com/, 2018.
[26] https://catalog.data.gov/dataset/511-ny-events-beginning-2010, 2018.

Xuanming Liu is currently a PhD candidate in the Computer Science department at the University of Massachusetts, Lowell. He received his BS degree in Software Engineering from Beijing Jiaotong University in 2011 and his MS degree in Computer Science from University of Massachusetts, Lowell in 2015. His research interests include data streams, graph mining, and distributed systems.



Tingjian Ge received his BS degree from Tsinghua University in 1994, his MS degree from University of California at Davis in 1998, and his PhD degree from Brown University in 2009, all in Computer Science. He is currently a Professor in the Computer Science department at the University of Massachusetts, Lowell. His research interests include databases, data mining, and data science in general, as well as data streams and graph analysis in particular.

Yinghui Wu is currently an Assistant Professor in the Department of Computer and Data Sciences at the Case Western Reserve University. Yinghui received his PhD from the University of Edinburgh, UK in 2011, and BS from Peking University, China in 2007, both in Computer Science. His research interests include databases, data management and analytics, cost-effective data systems for information and knowledge extraction, and graph analytics.