2019 IEEE 35th International Conference on Data Engineering (ICDE)

Finding Densest Lasting Subgraphs in Dynamic Graphs: a Stochastic Approach

Xuanming Liu
Computer Science
University of Massachusetts, Lowell
xliu@cs.uml.edu

Tingjian Ge
Computer Science
University of Massachusetts, Lowell
ge@cs.uml.edu

Yinghui Wu
EECS
Washington State University
yinghui@eecs.wsu.edu

***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 then devise EMU algorithms for the densest lasting subgraph problem. Using real-world graph data, we experimentally verify the effectiveness and efficiency of our techniques, and compare with two prior approaches on dense subgraph detection.**

I. 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 in dynamic graphs is especially evident in telecommunication, traffic networks, and social network analysis. Consider the following examples.

Communication hotspots. In a dynamic mobile phone network, each user is a vertex, and a phone call session between users 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 groups of 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 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]

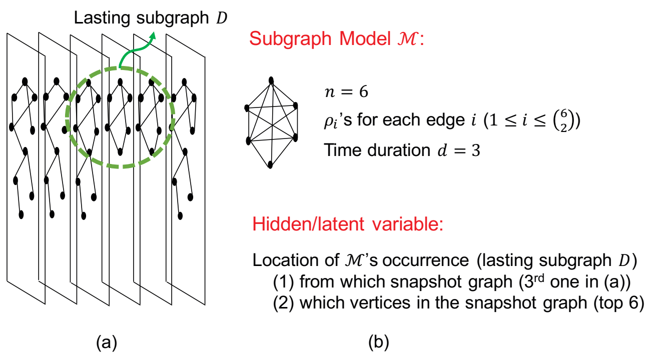

Lasting subgraph D
Subgraph Model $\mathcal{M}$:
$n = 6$
ρ_i's for each edge i ($1 \leq i \leq \binom{6}{2}$)
Time duration $d = 3$
Hidden/latent variable:
Location of $\mathcal{M}$'s occurrence (lasting subgraph D)
(1) from which snapshot graph (3rd one in (a))
(2) which vertices in the snapshot graph (top 6)
(a)
(b)

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

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 II).
- We propose a novel EMU framework, and create utility functions for our problem (Section III).
- We devise an algorithm under EMU, and prove its correctness as it is in the MM framework (Section IV).
- Using four real-world dynamic datasets, we perform a comprehensive empirical study (Section VI).

Related Work. We categorize the related work as follows. *Dense subgraphs in static graphs*. Discovering dense subgraphs has been studied for static graphs [8]–[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 I and VI). 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.

II. Preliminaries

A. Subgraph Models: A Probabilistic Characterization

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.

Based on the model, we introduce lasting subgraphs.

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 IV.

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 III). These techniques on the other hand readily extend to other classes of utility functions, as verified in Section V.

B. EM and Metropolis-Hastings Revisited

Before we introduce our framework, we briefly overview Expectation Maximization and related techniques. Readers who are familiar with these can skip to Section III.

Expectation Maximization (EM). EM is an *iterative* method to find *maximum likelihood* or *maximum a posteriori* (MAP) estimates of parameters in statistical models, which depend on unobserved *latent variables*. The EM iteration alternates between (1) an *expectation* (E) step, which, given the current estimate of model parameters, computes a distribution of the latent variables and the expectation, and (2) a *maximization* (M) step, which, given the expectation/estimate of latent variables in E step, computes a new estimate of model parameters. These parameter-estimates are then used to determine the distribution of the latent variables in the next E step. We refer the reader to [6] for details.

Metropolis-Hastings (MH). MH is a Markov chain Monte Carlo (MCMC) sampling method. Suppose we want to draw samples from a probability distribution $P(x)$ which is complex or even unknown. We can nevertheless compute the value of a function $f(x)$ that is proportional to the density of P. MH defines an MCMC sampling process, in a way that the stationary distribution of the Markov chain is $P(x)$. Suppose the current sample value is x_n. We define a *proposal function* (distribution) $Q(x^*|x_n)$, where x^* is called a *candidate*. Then we accept x^* as the next sample x_{n+1} with probability min $(1, \frac{P(\mathcal{X}^*)}{P(\mathcal{X}_n)} \cdot \frac{Q(\mathcal{X}_n|\mathcal{X}^*)}{Q(\mathcal{X}^*|\mathcal{X}_n)})$, which is just $\min(1, \frac{f(\mathcal{X}^*)}{f(\mathcal{X}_n)} \cdot \frac{Q(\mathcal{X}_n|\mathcal{X}^*)}{Q(\mathcal{X}^*|\mathcal{X}_n)})$, due to the discussion above as $f(x)$ can be obtained, but not $P(x)$. Furthermore, if $Q(x_n|x^*) = Q(x^*|x_n)$, where we call Q a *symmetric* proposal function, this probability is further simplified as $\min(1, \frac{f(\mathcal{X}^*)}{f(\mathcal{X}_n)})$. If x^* is not accepted, the current sample is still x_n. We refer the reader to [15] for details.

We will nontrivially extend the above techniques for our solution to the densest lasting subgraph problem, and show its feasibility over large scale dynamic graphs.

III. 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.

A. 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 II.

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.

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 $e_j \in E_c(\mathcal{M})$.*

Proof. We show Theorem 1 by constructing the equivalence between the problem of computing static dense subgraph with edge surplus and a special case of our problem. Since the log function is monotonic, maximizing Equation (2) is equivalent to maximizing

$$\ln u(\mathcal{M}) = \sum\nolimits_{j \in E_c(\mathcal{M})} d(\rho_j - \alpha) \tag{3}$$

Consider a "flat" deterministic subgraph $G = (V_\mathcal{M}, E)$ where the time depth is $d = 1$ and an edge probability ρ_j is either 0 or 1, corresponding to a static graph over $V_{\mathcal{M}}$. Let $n = |V_{\mathcal{M}}|$. Equation (3) becomes

$$\begin{aligned}\ln u(\mathcal{M}) &= \sum\nolimits_{j \in E}(1-\alpha) + \sum\nolimits_{j \in E_c - E}(-\alpha) \\ &= (1-\alpha)|E| + (-\alpha)\left[\binom{n}{2} - |E|\right] \\ &= |E| - \alpha\binom{n}{2}\end{aligned} \tag{4}$$

where we abbreviate $E_c(\mathcal{M})$ as E_c, denoting the complete set of $\binom{n}{2}$ edges in the subgraph model. Observe that Equation (4) is exactly of the same form as edge surplus [12]. Theorem 1 thus follows. □

Equation (4) 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].

B. 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 $\hat{d}_i$. 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 EMU.

IV. The EMU Algorithms

We next show that the general EMU framework gives birth to efficient algorithms to compute the densest lasting subgraphs in large $\mathcal{G}_T$. We use the utility function in Section III-A 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.*

Proof. Consider an arbitrary fixed order $v_1, \ldots, v_N$ of V. Suppose the ground-truth densest lasting subgraph $\mathcal{G}_d$ consists of vertices $v_{i_1}, \ldots, v_{i_n}$ also in this order ($i_1 < \cdots < i_n$ may not be consecutive). There will always exist a subgraph model $\mathcal{M}$ with vertices $u_1, \ldots, u_n$ that one-to-one map to $v_{i_1}, \ldots, v_{i_n}$, respectively, and the edge probabilities ρ_j of $\mathcal{M}$ match the edge connectivity in $\mathcal{G}_d$. Thus, a correct EMU algorithm should identify the lasting subgraph $\mathcal{G}_d$ induced by $v_{i_1}, \ldots, v_{i_n}$ and the corresponding $\mathcal{M}$ simultaneously. □

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 IV-B. The algorithm, in the end, returns the subgraph model $\mathcal{M}$, and the latent variable value—the model's best location L in $\mathcal{G}_T$.

A. 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 (reviewed in Section II-B) 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 $G_{t_0 \dots t_0+\hat{d}_i}$ 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 $G_{1\dots T-\hat{d}_i}$
 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 $G_{1\dots T-\hat{d}_i}$, 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 VI. $\hat{L}_i$ 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 VI.

"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 IV-B. 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*

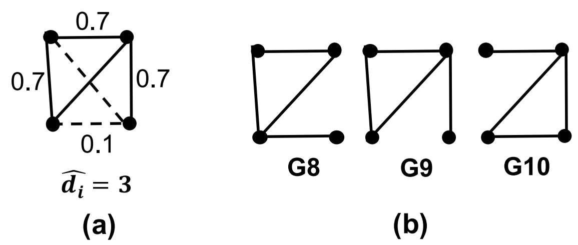


Fig. 2. Illustrating details of GENERALIZEDE. (a) The model $\widehat{\mathcal{M}}_i$ where $\hat{n}_i = 4, \hat{d}_i = 3$, and each solid edge has probability 0.7 and each dashed one 0.1 (simplified for clarity). (b) A component across three snapshots G_8 to G_{10} with the same four vertices.

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).

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 IV-C.

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.*

Proof. First, the way in which GENERALIZEDE generates component (i.e., location) samples follows a Markov chain. This is because the next sample, either through lines 8-14 or through line 16 (getEdgeWalkComponent) only depends on the previous step sample (at most). Second, this Markov chain is *finite* (a finite number of states), *irreducible* (i.e., from one component, the algorithm can get to any other component with non-zero probability), and *aperiodic* (lines 7-16 ensure that with non-zero probability, the next sample/state stays the same—hence aperiodic). Thus, this Markov chain must have a *unique* stationary distribution [16].

Finally, GENERALIZEDE follows the Metropolis-Hastings method. In particular, the proposal function is symmetric, as the probability of going from a component C_{prev} to the next component C is the same as the probability of going from C to C_{prev}. The C computed from lines 6-16 is a "candidate" in Metropolis-Hastings, and the α in line 18 is its acceptance probability. Since the distribution of match instances of $\widehat{\mathcal{M}}_i$ in $\mathcal{G}_T$ fits the MH sampling process, it must be the *unique* stationary distribution. □

B. 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 [17] and Gradient Ascent [18] to optimize $\hat{n}_{i+1}$ and $\hat{d}_{i+1}$. The algorithm is presented as GENERALIZEDM.

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

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

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

EMU Algorithm: M step. Lines 2-17 perform iterative Coordinate Ascent optimization [17] 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 $\hat{L}_i$ 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 [18]. 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 $\mathbf{E}[U]$ 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).*

Proof. Given a model $\mathcal{M}(\hat{n}, \rho, \hat{d})$, the probability of the weighted average component of $\hat{L}$ (as in line 3 of *getModel* and Example 2) is:

$$\Pr[\mathcal{G}_d|\mathcal{M}] = \prod_{j \in E} \rho_j \cdot \prod_{j \notin E} (1 - \rho_j) \tag{5}$$

where E is the set of edges in this component. From Equations (2) and (5), we get the expected utility

$$\begin{aligned}\mathbf{E}[U] &= u(\mathcal{M}) \cdot \Pr[\mathcal{G}_d|\mathcal{M}] \\ &= \prod_{j \in E_c(\mathcal{M})} e^{\hat{d}(\rho_j - \alpha)} \cdot \prod_{j \in E} \rho_j \cdot \prod_{j \notin E} (1 - \rho_j)\end{aligned} \tag{6}$$

Taking the *log* on Equation (6), we get

$$\begin{aligned}&\ln \mathbf{E}[U] = \\ &\sum_{j \in E_c(\mathcal{M})} \hat{d}(\rho_j - \alpha) + \sum_{j \in E} \ln \rho_j + \sum_{j \notin E} \ln(1 - \rho_j)\end{aligned} \tag{7}$$

Thus, to get maximum **E**[U], setting

$$\frac{\partial \ln \mathbf{E}[U]}{\partial \rho_j} = \hat{d} + \frac{n_j^+}{\rho_j} - \frac{\hat{d} - n_j^+}{1 - \rho_j} = 0 \tag{8}$$

gives us $\hat{\rho}_j = \sqrt{\frac{n_j^+}{\hat{d}}}$ as given in the theorem. ☐

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 $\hat{d}$.*

Proof. The loop in lines 2-17 (particularly lines 4, 15-17) of GENERALIZEDM does Metropolis-Hastings sampling of $\hat{n}$, with acceptance probability α, and the sample probability is proportional to the utility of the corresponding model. Line 11 calls the function *gradientAscentTimeDepth* to set $\hat{d}$ using gradient ascent. Finally, the algorithm sets $\hat{\rho}_j$ in line 4 of the *getModel* function as proved in Theorem 3. Overall, this simulates a coordinate ascent optimization where we optimize three groups of parameters alternately. ☐

Both GENERALIZEDE and GENERALIZEDM employ Metropolis-Hastings sampling—although there are no theoretical guarantees when it will converge to the stationary distribution, in practice [15], 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 VI, 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 [18] where ϵ is the allowed error in line 6 of $gradientAscentTimeDepth$.

C. Validation of EMU Algorithms

In Sections IV-A and IV-B, we have individually shown the correctness of the generalized E step and M step, respectively, 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).

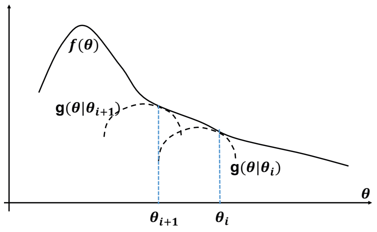


Fig. 3. Illustrating MM. $f(\theta)$ is the objective function, and $g(\theta|\theta_i)$ is the surrogate function we maximize at each iteration i instead. This continues for iteration $i + 1$, and so on.

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 IV-A and IV-B) is also an MM (Minorization-Maximization) algorithm.*

Proof. From the EMU algorithm, we construct the corresponding $f(\theta)$ and the surrogate function $g(\theta|\theta_i)$ in MM. The very first question is what θ corresponds to in our EMU algorithm. Let θ be a tuple $(\mathcal{M}, L)$, where $\mathcal{M}$ is the subgraph model parameters (edge probabilities, number of vertices, and time depth) and L is the location (and the component there) of this model in $\mathcal{G}_T$. Then we define:

$$f(\theta) = f(\mathcal{M}, L) \equiv \max_{\mathcal{M}'}[u(\mathcal{M}') \cdot \Pr[(L|\mathcal{M}')]] \tag{9}$$

$$g(\theta|\theta_i) = g(\mathcal{M}, L|\mathcal{M}_i, L_i) \equiv u(\mathcal{M}) \cdot \Pr[(L_i|\mathcal{M})] \tag{10}$$

where $\mathcal{M}_i$ and L_i are the model parameters and location/component obtained in iteration i, respectively. In words, $f(\theta)$, which is just $f(\mathcal{M}, L)$, ignores its input argument $\mathcal{M}$, but only uses L; it is defined as the *maximum* product of the utility of some model $\mathcal{M}'$ and the probability of the input data location/component given this model $\mathcal{M}'$. $f(\theta)$ can be deemed the objective function, as essentially we look for L such that $f(\theta)$ is maximized, i.e., the optimal data location such that, with the suitable model there, $\mathbf{E}[U]$ is maximized.

The surrogate function $g(\theta|\theta_i)$, which is also a function of $\mathcal{M}$ and L (given $\mathcal{M}_i$ and L_i computed in iteration i of EMU), is the product of the utility of $\mathcal{M}$ and the probability of the computed constant L_i at iteration i (data location/component) assuming model $\mathcal{M}$. Note that, opposite to $f(\theta)$, $g(\theta|\theta_i)$ ignores its input argument L, but only uses $\mathcal{M}$.

We construct the two functions (9) and (10) following this, so that they satisfy the definition of "surrogate function" of MM as remarked earlier (Figure 3). That is, it holds that $g(\theta|\theta_i) \leq f(\theta)$ for all θ, since $f(\theta)$ uses the "best" model $\mathcal{M}'$ w.r.t. the input argument L. Note that we assume the input argument L always matches $\mathcal{M}$, i.e., given $\mathcal{M}$, L is the location of $\mathcal{M}$ with the highest probability. Then we can see that $g(\theta|\theta_i) \leq f(\theta)$, as $f(\theta) \geq u(\mathcal{M}) \cdot \Pr[(L|\mathcal{M})] \geq u(\mathcal{M}) \cdot \Pr[(L_i|\mathcal{M})] = g(\theta|\theta_i)$. The other condition (tangent point) also holds, i.e., $f(\theta_i) = g(\theta_i|\theta_i)$, because $f(\mathcal{M}_i, L_i) = g(\mathcal{M}_i, L_i|\mathcal{M}_i, L_i)$, as the M step of EMU exactly sets the optimal $\mathcal{M}$ based on L_i at this iteration.

The first M step of MM corresponds to the E step of EMU, which constructs the surrogate function $g(\theta|\theta_i)$ at θ_i of iteration i. Basically, it is to compute L_i given the $\mathcal{M}_i$ resulted from the previous iteration. Thus, $g(\theta|\theta_i)$ is ready for operation. The second M step of MM corresponds to the M step of EMU, which, given L_i, maximizes the surrogate function $g(\theta|\theta_i)$ by computing a new $\mathcal{M}_i$, ready for the next iteration ($\mathcal{M}_i$ will be the initial $\mathcal{M}_{i+1}$ for iteration $i + 1$).

This completes the proof of Theorem 5. □

Remarks. Our EMU algorithm belongs to the category of *random search* algorithms. Random search algorithms have been shown to have a potential to solve large-scale problems efficiently in a way that is not possible for deterministic algorithms, especially for NP-hard problems [19]. Theorem 5 shows that our algorithm belongs to MM, and hence converges to the optimal solution. Here we give some intuitions on the efficiency of finding good-quality (dense) subgraphs, even if they were very rare. Suppose there are only k high-quality subgraph components matching a good model after some iterations (for a very small k, such as 1 or 2); assume each of them matches the model with a high probability p^*, while each of the remaining $n-k$ low-quality subgraph components matches with a low probability $\frac{p^*}{r}$ (for a large ratio r). From Theorem 2, GENERALIZEDE samples the n subgraphs with a probability proportional to their respective matching probabilities to the model. Thus, in one iteration, the probability one of the k high-quality subgraphs is chosen is $\epsilon = \frac{kp^*}{kp^*+(n-k)\frac{p^*}{r}}$. After m iterations, the probability that at least one of the high-quality subgraphs is chosen is $Pr(hit) = 1-(1-\epsilon)^m = 1-(\frac{n-k}{n+k(r-1)})^m$, for a small integer k and a large $r \gg 1$. Since $\frac{n-k}{n+k(r-1)} < 1$, the probability $Pr(miss)$ of *not* finding a high-quality model decreases exponentially fast with the sample size m. Moreover, it is intuitive that as the number of good subgraph components k increases, the decrease of $Pr(miss)$ is even faster; perhaps surprisingly, this is also true when r increases. This is because as the difference between the good and other match instances is more pronounced, the weight/ratio of getting a good model is higher since each instance is chosen with a probability proportional to their matching probabilities.

V. EXTENSIONS TO MULTIPLE EDGE/VERTEX LABELS

In this section, we discuss several extensions to our basic EMU utility function for the densest lasting subgraph problem. Our discussion in Section IV only considers the density from the perspective of graph structures, without taking into account edge or vertex labels. In practice, edge and vertex labels are also an important part of the information in determining and reporting the densest structure. For example, in Twitter graphs where vertices are people and edges are tweet messages, edge labels may indicate various hashtags or message content topics. Then one may be interested in dense subgraphs over time considering, as well as reporting "dense" message topics. More formally, we add to our formulation in Section II a label function $L : V \cup E \to \Sigma$ that assigns a vertex/edge a label from a finite alphabet Σ, where $E = \bigcup_{t=1}^{T} E_t$.

Adding node/edge labels. We show, in response, that edge (or vertex) label distributions can be readily incorporated to the model. Given the subgraph model $\mathcal{M}$, in addition to $\hat{n}$ and $\hat{d}$, we extend the edge distributions $\hat{\rho}_j$ into edge-label distributions $\hat{\rho}_{lj}$($l \in \Sigma$ is a label) for edge j. Vertex labels are similar, and we omit the discussion here. Moreover, we use $\hat{\rho}_{\perp j} = 1 - \sum_{l\in\Sigma} \hat{\rho}_{lj}$ to denote the probability that edge j does not exist. Accordingly, we extend the utility function to

$$f(\mathcal{M}) = \prod_{j\in E_c(\mathcal{M})} e^{d(\sum_{l\in\Sigma}\hat{\rho}_{lj}-\alpha)} \tag{11}$$

Theorem 6. *Given the utility function in Equation (11), a location distribution $\hat{L}$, and the current parameters $\hat{n}$ and $\hat{d}$ of the model, to maximize $\mathbf{E}[U]$ (Section III-A), the edge label probability $\hat{\rho}_{lj}(l \in \Sigma)$ of the model 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 $\hat{L}$.*

Proof. Equation (6) now becomes

$$\begin{aligned}\mathbf{E}[U] &= f(\widehat{\mathcal{M}}) \cdot \Pr[\mathcal{G}_d|\widehat{\mathcal{M}}] \\ &= \prod_{j\in E_c(\mathcal{M})} e^{\hat{d}(\sum_{l\in\Sigma}\hat{\rho}_{lj}-\alpha)} \cdot \prod_{j\in E, l_j\in\Sigma\cup\{\perp\}} \hat{\rho}_{l_j j}\end{aligned} \tag{12}$$

In addition, we have the constraints that $\sum_{l\in L_e\cup\{\perp\}} \hat{\rho}_{lj} = 1$. Taking the log over Equation (12), and using Lagrange Multipliers [18] over the constraints, we have:

$$\begin{aligned}\Lambda(\hat{\rho}_{lj}, \lambda) =& \hat{d}\sum_{j\in E_c(\widehat{\mathcal{M}})} (\sum_{l\in\Sigma} \hat{\rho}_{lj} - \alpha) + \\ & \sum_{j\in E, l_j\in\Sigma\cup\{\perp\}} \ln \hat{\rho}_{l_j j} + \lambda(\sum_{l\in\Sigma\cup\{\perp\}} \hat{\rho}_{lj} - 1)\end{aligned} \tag{13}$$

Thus, for $l \in \Sigma$, we set

$$\frac{\partial\Lambda}{\partial\hat{\rho}_{lj}} = \hat{d} + \frac{n^+_{lj}}{\hat{\rho}_{lj}} + \lambda = 0 \tag{14}$$

Moreover,

$$\frac{\partial\Lambda}{\partial\hat{\rho}_{\perp j}} = \frac{n^+_{\perp j}}{\hat{\rho}_{\perp j}} + \lambda = 0 \tag{15}$$

Given Equations (14) and (15), together with $\sum_{l\in\Sigma\cup\{\perp\}} \hat{\rho}_{lj} = 1$ and $\sum_{l\in\Sigma\cup\{\perp\}} n^+_{lj} = \hat{d}$, we get $\lambda = -\hat{d} - \sqrt{\hat{d}^2 - \hat{d}n^+_{\perp j}}$, which, combined with Equation (14), gives $\hat{\rho}_{lj} = \frac{n^+_{lj}}{\sqrt{\hat{d}^2 - \hat{d}n^+_{\perp j}}}$. □

Given Theorem 6, we can modify the *getModel()* function in the GENERALIZEDM to set the model parameters accordingly. Likewise, in lines 2 and 17 of the GENERALIZEDE, we modify the calculation of probability p_C following Equation (12).

Adding node/edge 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). We omit these discussions and will leave them as future work.

VI. EXPERIMENTS

A. Datasets and Setup

Datasets. We use four real-world datasets: **(1) Twitter data.** We use the Twitter Stream API [20] to retrieve real-time Twitter graphs from January 22, 2017 to May 21, 2017. For communication datasets (Twitter and Stack Overflow below), we treat users as vertices and edges as communications, and the duration of an edge 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 data.** The trip data is about 30GB, containing the information of all taxi trips in NYC in 2013 [21]. It has 14 attributes. Each trip from the pick-up to the drop-off locations is considered as an edge, and the duration of the edge is the trip time. We treat the number of passengers as an edge label. **(3) Stack Overflow data [22].** This is a network of interactions on the web site Stack Overflow [23]. 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 data [24].** This dataset contains historical traffic and transit events in NY. We partition the NY state area into 0.0011-degree latitude by 0.0012-degree longitude grid areas (i.e., 122 meters by 100 meters). 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 (vertices), and the duration of the event is the duration of the edges. We report the statistics of the datasets in Table I.

TABLE I
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

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 (above a threshold). There are a few issues in this approach for the problem we study. First, the "similarity" above is based on the set similarity, for the set of snapshots each vertex appears in. This is too restrictive. Second, the dense pattern approach returns "small" subgraphs, i.e., a small number of vertices. 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. The "dense temporal subgraph" approach in [5] simplifies the search into 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 this approach is efficient, there are a few major issues for the problem that we target at. First, it uses a model where all edges are present in all graph snapshots, while only edge weights vary from one snapshot to another. 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.

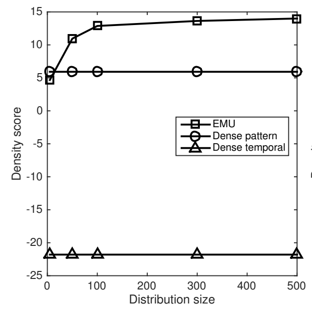
B. Experimental Results

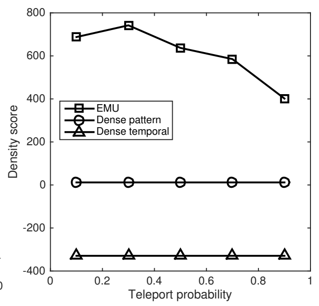
We first study the effectiveness of our EMU approach, i.e., the quality/density of the subgraphs discovered. We compare our method with the two related methods, denoted as "Dense pattern" [4] and "Dense temporal" [5], respectively. For a fair comparison, we define a metric called *density score* $s = [m - \alpha\binom{n}{2}]d^{\beta}$, where n is the number of vertices in the found subgraph pattern, d is the time duration (number of consecutive snapshots), 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.

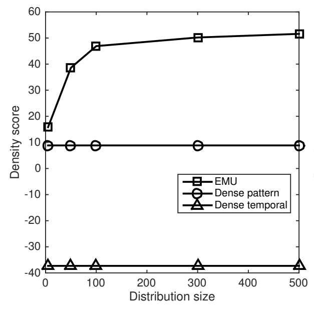
Effectiveness on Real Datasets. We report our findings below.

Twitter. We first evaluate the methods using the Twitter dataset. Recall that two parameters in our algorithms are *the location distribution size* and the *teleport* probability in GENERALIZEDE. Figure 4 shows the density scores of the densest lasting-subgraphs returned by EMU over Twitter data, as we vary the location distribution size. The same figure also shows the density scores of the results returned by previous work dense pattern [4] and dense temporal [5] respectively (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 distribution size. As explained earlier, the dense pattern approach [4] tends to return small subgraphs, which get smaller (but usually positive) density scores. The dense temporal approach [5], on the other hand, tends to return subgraphs that are very large, resulting in negative density scores (i.e., the number of edges are few compared to the complete graph over those vertices).







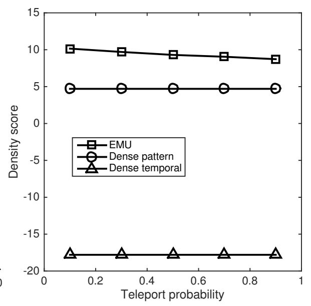
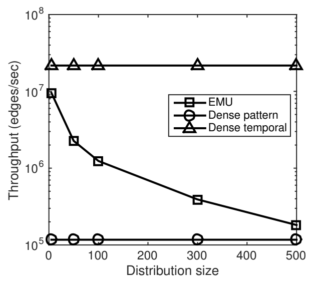
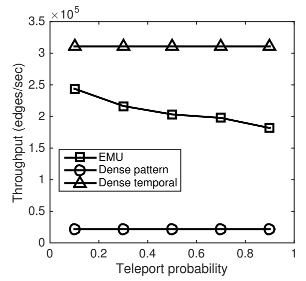
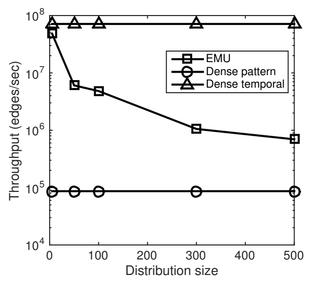


Fig. 4. Density vs distrib. size(Twitter) Fig. 5. Density vs teleport prob.(Taxi) Fig. 6. Density vs distrib. size(Stack) Fig. 7. Density vs teleport prob.(Events)







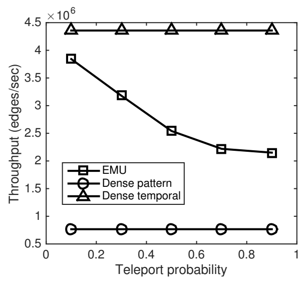
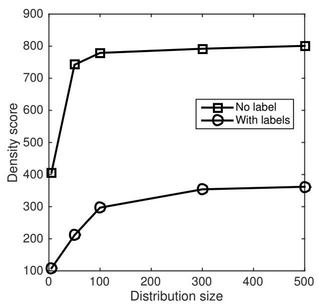
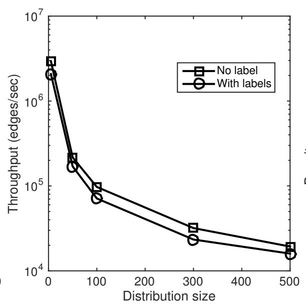
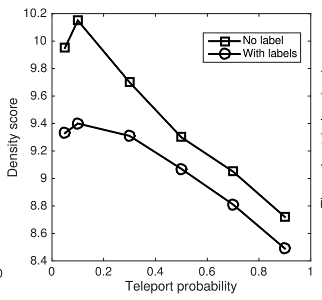


Fig. 8. Speed vs distrib. size(Twitter) Fig. 9. Speed vs teleport prob.(Taxi) Fig. 10. Speed vs distrib. size(Stack) Fig. 11. Speed vs teleport prob.(Events)







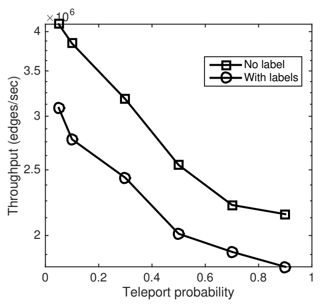


Fig. 12. Density w/ edge labels(Taxi) Fig. 13. Speed w/ edge labels(Taxi) Fig. 14. Density w/ vertex labels(Events) Fig. 15. Speed w/ vertex labels(Events)

Taxi. In Figure 5, we vary the teleport probability between 0.1 and 0.9, and measure the density scores over Taxi.

(1) The maximum density 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 the Taxi dataset in Figure 5 are much higher than the Twitter dataset in Figure 4. This is due to the nature of the datasets—traffic network 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, thus resulting in small positive density scores. The dense temporal subgraph [5], on the other hand, returns subgraphs that are too large with negative density score.

Stack Overflow and NY Events. Similarly, we report the density scores for various distribution sizes over the Stack Overflow dataset in Figure 6. The trend is consistent with that in Figure 4, except that the density scores are higher. Varying the teleport probability, we report the result using NY events data in Figure 7. 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 5, 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 8 for Twitter. For EMU algorithms, as we increase the location distribution size, the performance degrades. This is because the increase of distribution size 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 of the algorithms over the Taxi dataset for various teleport probabilities in Figure 9, over the Stack Overflow dataset for various distribution size in Figure 10, and over the NY events dataset for various teleport probabilities in Figure 11. The performance decreases as the transport probability increases, because performing a transport is more expensive as it needs to obtain a whole random subgraph component, while making local changes only slightly revises a component.

Adding vertex/edge labels. We also investigate the effectiveness and performance of EMU algorithms when we consider edge or vertex labels in the graph. The two approaches in [4] and [5] cannot cope with vertex/edge labels.

For the Taxi dataset, we consider the number of passengers in a trip as an edge label. Thus, the densest lasting-subgraph model found also contains matching edge labels. Our results indicate that the densest lasting-subgraph model contains edge labels with mostly 1 passenger and some 2 passengers. The density score comparison for with vs. without edge labels under various distribution sizes is shown in Figure 12. We can see that the density scores are in general lower when we consider edge labels. This is because the chance of matching an edge in the model with a specific label is smaller, compared to simply just matching an edge; thus it can even be proved that the density score of the densest lasting-subgraph will be smaller than that without edge labels.

In Figure 13, we show the performance comparison with and without edge labels for the Taxi dataset. It becomes slower when we consider edge labels. This is because essentially the *entropy* of the problem instance is higher when we consider edge labels, i.e., there are more variables in the dynamic graph. Thus, it takes longer time for the EMU algorithms to converge.

We next examine the effect of vertex labels using the NY events dataset, where each vertex indicates a grid area where an event happens, and the vertex label indicates the type of events. We show the results of density scores in Figure 14 and performance in Figure 15, as we vary the teleport probability. The densest lasting subgraph model discovered in our experimental result 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 discussed above for edge labels. We find that the optimal teleport probability is around 0.1.

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 outperforms 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, which cannot be supported by the baseline methods.

VII. Conclusions

We propose a novel probabilistic subgraph model to characterize densest lasting 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.

Acknowledgments. Tingjian Ge is supported by NSF grants IIS-1149417 and IIS-1633271. Yinghui Wu is supported by NSF IIS-1633629, USDA/NIFA 2018-67007-28797, Siemens and Huawei HIRP.

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] D. MacKay, "Introduction to Monte Carlo methods," *Learning in Graphical Models*, 1999.

[16] R. Serfozo, *Basics of Applied Stochastic Processes*, ser. Probability and Its Applications. Springer, 2009.

[17] S. J. Wright, "Coordinate descent algorithms," *Math. Program.*, 2015. [Online]. Available: http://dx.doi.org/10.1007/s10107-015-0892-3

[18] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge University Press, 2004.

[19] Z. B. Zabinsky, "Random search algorithms," *Wiley Encyclopedia of Operations Research and Management Science*, 2010.

[20] https://dev.twitter.com/streaming/overview, 2018.

[21] https://www.reddit.com/r/bigquery/comments/28ialf/173_million_2013_nyc_taxi_rides_shared_on_bigquery/, 2018.

[22] http://snap.stanford.edu/data/sx-stackoverflow.html, 2018.

[23] https://stackoverflow.com/, 2018.

[24] https://catalog.data.gov/dataset/511-ny-events-beginning-2010, 2018.