2011 11th IEEE International Conference on Data Mining

Mining Heavy Subgraphs in Time-Evolving Networks

Petko Bogdanov†, Misael Mongiovì†, Ambuj K. Singh
Department of Computer Science, University of California
Santa Barbara, CA 93106-5110
Email: {petko,misael,ambuj}@cs.ucsb.edu

***Abstract*—Networks from different genres are not static entities, but exhibit dynamic behavior. The congestion level of links in transportation networks varies in time depending on the traffic. Similarly, social and communication links are employed at varying rates as information cascades unfold. In recent years there has been an increase of interest in modeling and mining dynamic networks. However, limited attention has been placed in high-scoring subgraph discovery in time-evolving networks.**

We define the problem of finding the highest-scoring temporal subgraph in a dynamic network, termed *Heaviest Dynamic Subgraph (HDS)*. We show that *HDS* is NP-hard even with edge weights in $\{-1, 1\}$, and devise an efficient approach for large graph instances that evolve over long time periods. While a naïve approach would enumerate all $O(t^2)$ sub-intervals, our solution performs an effective pruning of the sub-interval space by considering $O(t \cdot \log(t))$ groups of sub-intervals and computing an aggregate of each group in logarithmic time. We also define a fast heuristic and a tight upper bound for approximating the static version of HDS, and use them for further pruning the sub-interval space and quickly verifying candidate sub-intervals.

We perform an extensive experimental evaluation of our algorithm on transportation, communication and social media networks for discovering subgraphs that correspond to traffic congestions, communication overflow and localized social discussions. Our method is two orders of magnitude faster than a naïve approach and scales well with network size and time length.

I. Introduction

Time-evolving networks arise in multiple application domains. They can characterize traffic variation in transportation networks, information flow in communication networks, variation of trade rates in a network of trading agents or phases of pathway switching in gene interaction networks. Despite their rich representative power, time-evolving networks have received limited attention from the data mining and database communities. Most existing dynamic network research focuses on link formation [1] and community discovery [2], [3]. Limited attention has been placed to the problem of finding high scoring subgraphs in time-evolving networks.

Approaches for finding high-scoring subgraphs have been widely studied for static networks [4], [5]. They have many applications in Biology [4], network design [6] and other fields. Extending the notion of high-scoring subgraphs to time-evolving networks can enable several interesting applications. For example, the road network is characterized by a fixed network topology whose edge utilization varies with time according to traffic [7]. High scoring sub-networks correspond to congested locations over an extent in time. Similarly, in the biological domain, the activity within a cell can be represented as a network of interactions between cell components (DNA, proteins, RNA, small molecules). The network structure is fixed (an edge between two components represents a potential interaction), while the state of each interaction (active/inactive) changes during the execution of biological processes [8]. Finding a connected set of active interactions that persist in time is useful to identify functional modules and analyze their evolution in time [9].

In this paper we introduce the *Heaviest Dynamic Subgraph (HDS)* problem, i.e. the problem of finding the highest-scoring connected temporal subgraph in an edge-weighted network whose weights evolve in time (edge-evolving network for short). A temporal subgraph is characterized by its participating edges and a time sub-interval. The score of a temporal subgraph is defined as the sum of the edge weights. Our problem can be naturally extended to finding the top-k high-scoring non-overlapping subgraphs.

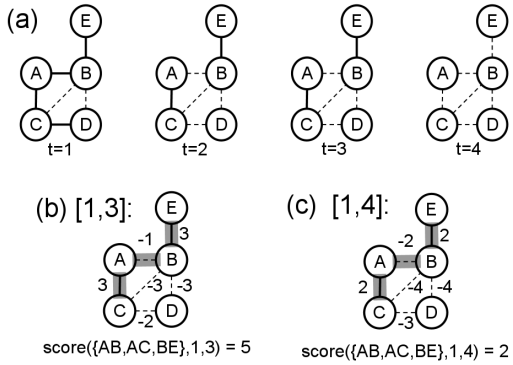


Figure 1. An example of time-evolving network (a). Edges in each time step are scored either 1 or -1 (solid or dashed, respectively). The heaviest subgraph is defined over the sub-interval $[1, 3]$ and consists of edges AC, AB, BE (b). (c) shows the scoring of the same subgraph in the interval $[1, 4]$

An example for HDS is depicted in Fig. 1(a). Solid lines

†These authors contributed equally to this work

1550-4786/11 $26.00 © 2011 IEEE
DOI 10.1109/ICDM.2011.101

represent edges with score 1, while dashed lines represent edges of score -1. A possible social network interpretation is the unfolding of a discussion among friends over time. At the beginning the pairs $\{(C,D)(A,C),(A,B),(E,B)\}$ initiate a discussion. Although B "knows" C, they do not interact directly in this discussion. The discussion then persists for pairs (A,C) and (E,B) (steps 2 and 3) and terminates at step 4. The heaviest dynamic subgraph in this example is the set of edges AB, AC, BE over the sub-interval $[1,3]$. HDS captures the backbone of communication channels employed in the discussion process, while excluding the edges that are not used (such as BC). Similar interpretations can be derived for transportation and biological networks.

HDS balances the cost of considering negative edges with the benefit of connecting high-scoring components. For example in Fig. 1, the edges AC and BE are connected through the edge AB even though its weight is prevalently negative. Joining the two edges is advantageous since it produces a larger temporal subgraph whose overall score (value 5) is higher than the score of the single temporal subgraphs (value 3 each). Admitting solutions that contain negative edges has several advantages. First, since weights change over time, not admitting negative edges would tend to discard longer sub-intervals. Second, in many applications, it may be worthwhile to pay a small cost for producing a larger component. Finally, it makes the method robust to noise, since sporadic negative values resulting from noise do not significantly affect the result.

A naïve method for solving HDS would reduce the problem to the static case by enumerating all the possible $\frac{t\cdot(t+1)}{2}$ time sub-intervals and finding the highest-scoring subgraph in each one of them. However, this approach would not scale for two main reasons. First, the number of sub-intervals is quadratic in the length of the time interval. Second, finding the highest-scoring subgraph in a static network is NP-hard, thus solving it optimally is infeasible on large graphs. Although efficient heuristics exist [10], executing them on each sub-interval would be infeasible for large HDS problem instances. Reducing the quadratic dependence on the time length is a challenge, since neither the optimal score is convex in the sub-interval space nor the score of an interval can be related to the score of sub-intervals.

We propose a novel filter-and-verify approach for pruning sub-intervals named *MEDEN (Mining EDge-Evolving Networks)*. Besides defining tight bounds of the optimal solution for pruning sub-intervals that are not promising, we introduce a novel sub-interval aggregation scheme that allows us to prune groups of sub-intervals without inspecting their members. Our filtering method has complexity $O(t \cdot log^2(t) \cdot |E|)$, where t is the total time length of the graph evolution. Our method achieves high pruning power by guaranteeing that only highly overlapping sub-intervals are grouped together. We propose a fast heuristic for the verification phase. We experiment extensively with our method on several real and synthetic datasets and show that it scales well with time length and network size.

Our contributions include:

1) we formulate the HDS problem and prove that it is NP-hard even with edge weights in $\{-1,1\}$;
2) we design a filter-and-verify approach for HDS based on a tight upper bound for the optimal solution in a specific sub-interval;
3) we propose a novel sub-interval aggregation scheme and give an effective method for pruning the quadratic sub-interval space in time $O(t \cdot log^2(t) \cdot |E|)$;
4) we introduce an efficient heuristic for verifying promising sub-intervals that scales with the graph size;
5) we experiment with our method on dynamic networks of different genres and sizes and demonstrate its scalability.

The rest of this paper is structured as follows: We formally define the HDS problem in section II. We develop a filter-and-verify solution for HDS that relies on a scalable heuristic and a tight upper bound in Section III. Our space partitioning scheme and scalable mining algorithm *MEDEN* are developed in Section IV. Section V presents experimental results and section VI discusses related work. We conclude in Section VII.

II. Preliminaries and Problem Formulation

In this section we first define the problem *Heaviest Subgraph* (HS) for static graphs and discuss its relation to the known *Prize-Collecting Steiner Tree* (PCST) problem [4]. We then define HDS for time-evolving networks and show that it is NP-hard.

Definition 2.1: Given an edge-weighted graph $\bar{G} = (V,E,w)$, the Heaviest Subgraph (HS) problem calls for finding a subgraph $\bar{G}' = (V',E')$ of $\bar{G}$ such as its score, defined as $\sum_{e\in E'} w(e)$ is maximal.

The PCST problem [10] takes as input a network $\bar{G} = (V,E,w)$, with positive vertex weights and negative edge weights[1]. There are two different formulations of this problem. The *Goemans-Williamson Minimization* (GW-PCST) finds the connected subgraph that minimizes the sum of the node weights that are not included in the solution minus the sum of the edge weights within the solution. More precisely, the score of a connected subgraph $\bar{G}' = (V',E')$ of $\bar{G} = (V,E,w)$ is $\sum_{v\in V\setminus V'} w(v) - \sum_{e\in E'} w(e)$. The *Net Worth Maximization* (NW-PCST) maximizes the score $\sum_{v\in V'} w(v) + \sum_{e\in E'} w(e)$. Although the optimal solutions of the two formulations are equivalent, GW-PCST can be approximated to a factor $2-\frac{1}{n-1}$ using *GW-algorithm* [10], while NW-PCST cannot be approximated polynomially within any constant factor [11].

[1]In [10] edge weights are defined to be positive and their score is inverted in the scoring function.

HS and PCST can be reduced to each other in linear time. HS can be reduced to PCST by collapsing all the connected components with positive edges in one node with weight equal to the sum of the weights of the collapsed edges. Similarly, PCST can be reduced to HS by adding a new edge per node and assigning the node weight to it. Formal proofs of these reductions are omitted for space reasons.

For convenience, we define a summation $(+)$ and $\max$ operators for edge-weighted graphs as follows.

Definition 2.2: Let $\bar{G}_1 = (V, E, w_1)$ and $\bar{G}_2 = (V, E, w_2)$ be two edge-weighted isomorphic graphs that may have different edge weights. We define (i) the aggregate graph $\bar{G} = \bar{G}_1 + \bar{G}_2$ as $\bar{G} = (V, E, w)$ such that $w(e) = w_1(e) + w_2(e), \forall e \in E$ and (ii) the domination graph $\bar{G}_M = \max(\bar{G}_1, \bar{G}_2)$ as $\bar{G}_M = (V, E, w_M)$ such that $w_M(e) = \max(w_1(e), w_2(e)), \forall e \in E$. Both operators can also be applied to sets of graphs.

Next, we introduce the *Heaviest Dynamic Subgraph (HDS)* problem and state that it is NP-hard.

Definition 2.3: An edge-evolving network $G = (V, E, F)$ is an undirected connected graph where V is the set of vertices, E is the set of edges, and $F = \{f^1, f^2, \ldots, f^{|F|}\}$ is a family of functions of the kind $f^i : E \to \{-1, 1\}$ that associate each edge $e \in E$ with a weight. Each function f^i is associated with a discrete timestamp i.

Note that we restrict the edge weights to assume values in $\{+1, -1\}$. Below we prove that, even in this case, the problem is NP-hard. Due to its simplicity, combined with expressiveness, we adopt this scheme in the rest of the paper. Our method and results are naturally applicable to time-varying networks with arbitrary edge weights.

Definition 2.4: A temporal subgraph of an edge-evolving network $G = (V, E, F)$ is a pair $(G' = (V', E'), [i, j])$, where G' is a connected subgraph of G and $[i, j]$ is a sub-interval of $[1, |F|]$, i.e. $1 \leq i \leq j \leq |F|$. The score of G' is defined as:

$$score(G', i, j) = \sum_{e \in E'} \sum_{k=i}^{j} f^k(e)$$

Definition 2.5: The HDS problem is defined as finding the temporal subgraph $(G', [i, j])$ of an edge-evolving network G such that $score(G', i, j)$ is maximal.

HDS is an optimization problem in which the scoring function has to be maximized over all possible subgraphs $G' \in G$ and all possible sub-intervals of the whole interval. Note, that the HDS formulation allows for parameter-free discovery of high-scoring solutions that are either small subgraphs with a large time extent or large subgraphs with a smaller time extent. The score of a temporal subgraph can be informally considered as the importance of the process that it represents. For example, the importance of a congestion may be considered high due to either its long time duration or the size of the road network that it affects.

Theorem 2.1: HDS is NP-hard.

The proof of NP-hardness is based on a reduction from the Minimum *Thumbnail Rectilinear Steiner Tree* (TRST) problem, a variant of Minimum Steiner Tree in which nodes are embedded in a 2D grid and segments are restricted to be horizontal or vertical. The reduction considers a grid graph that represents the TRST grid and a high-scoring path for each terminal point of the TRST. The formal proof is omitted due to space limitation.

Definition 2.6: Given an edge-evolving network $G = (V, E, F)$ and a sub-interval $[i, j]$ with $i, j \in \{1, 2, \ldots |F|\}$, we define the *aggregated* graph *induced* by $[i, j]$ as the graph $\bar{G}(i, j) = (V, E, w)$, where $w(e) = \sum_{k=i}^{j} f^k(e)$ is a function that associates edges with the sum of their respective weights in $[i, j]$.

HDS can be solved by enumerating all the possible sub-intervals $[i, j]$ of $[1, |F|]$ and solving HS on the aggregated graph induced by each sub-interval. This solution, however, would require solving the NP-hard HS problem $\frac{t \cdot (t+1)}{2} = O(t^2)$ times, and hence would not scale with time length.

Although HS is equivalent to PCST (in static networks), this result is not generalizable to HDS with arbitrary edge weights. Consider the natural extension of PCST to dynamic networks (let's call it *dynamic PCST*), i.e. the problem of finding the highest-scoring temporal subgraph in a time-evolving network with positive node weights and negative edge weights. An instance of dynamic PCST can be reduced to HDS with arbitrary edge weights by repeating the process slice by slice. A similar reduction from an instance of HDS to an instance of dynamic PCST is not possible since it would require considering a different network structure for each slice. For this reason, we focus on HDS as a more general problem than dynamic PCST. Clearly all the results in this paper extend to dynamic PCST.

III. A FILTER-AND-VERIFY FRAMEWORK FOR HDS

An HDS solution is characterized by two main dimensions (i) subgraph: the set of edges forming a connected subgraph and (ii) time extent: a sub-interval. Since the problem in the time dimension can be solved by multiple applications of the Heaviest Subgraph (HS) problem for static graphs and this last problem is NP-hard (Theorem 2.1), we first present our heuristic for HS. We then develop an efficient filtering solution to prune the quadratic sub-interval space. The combination of the two components gives a *Basic* filter-and-verify procedure for HDS.

A. A heuristic for Heaviest Subgraph (HS)

In this section we propose *TopDown*, a scalable heuristic for solving HS. TopDown's performance is comparable to GW-algorithm (applied after reducing HS to PCST, Section II). While neither TopDown nor GW-algorithm can guarantee a constant factor approximation for our scoring

Algorithm 1 TopDown

Input: An edge-weighted graph $\bar{G} = (V, E, w)$
Output: an HS solution $\bar{G}' \subseteq \bar{G}$
1: $T \leftarrow$ Max. Spanning Tree of $\bar{G}$
2: Move positive weights to adjacent nodes in T
3: $T' \leftarrow$ NW-PCST-Tree(T)
4: $G' \leftarrow T' \cup \{$Positive edges, adjacent to $T'\}$
5: **return** G'

function, TopDown has the advantage of lower time complexity. Our approach is flexible, allowing to use either heuristic, according to the desired trade-off between quality and efficiency.

TopDown obtains a global solution by initially including all nodes in $\bar{G}$ and refining it by excluding non-optimal subgraphs (Algorithm 1). In *Step 1* we compute a Maximum Spanning Tree T of the edge-weighted graph $\bar{G}$. Next, we construct a tree instance of *NW-PCST* by moving the weight of each positive edge to one of its adjacent nodes in T (*Step 2*). Although NW-PCST is hard to approximate for general graphs, it can be computed exactly on trees in time linear in the number of nodes. NW-PCST-Tree (*Step 3*) is equivalent to the strong pruning phase of the PCST approximation by Johnson et al. [10]. It computes the spanning tree PCST solution T' which is then expanded to G' by restoring the original edge weights and adding all positive edges adjacent to T' (*Step 4*).

The complexity of TopDown is dominated by the Maximum Spanning Tree construction (Step 1), which can be performed by Prim's algorithm in time $O(|E| \cdot \log(|V|))$ using a binary heap, or in time close to linear in the number of edges due to later improvements. In contrast to TopDown, GW-algorithm [10] has a super quadratic running time $O(|V|^2 \cdot \log|V|)$. In our experimental analysis, TopDown achieves on average more than 90% of the best score computed by GW-algorithm. Any of the two algorithms can be used in our overall solution for HDS.

B. Upper bounds for HS

A naïve approach for HDS would solve HS in all $\frac{t(t-1)}{2}$ sub-intervals. The complexity can be reduced by quickly pruning non-promising sub-intervals. The pruning procedure bounds the solution in each sub-interval and discards the sub-intervals whose upper bound is smaller than a previously found solution. A simple upper bound, named UB_{sop} (sum-of-positive), can be obtained by summing all the positive edges. Although UB_{sop} is very fast and can be used for a first screening, it is not tight due to many positive edges distributed on a graph. We define a tighter upper bound, UB_{str} (structural), by relaxing the graph structure and connecting the positive components "optimistically".

Lemma 3.1: Let $\bar{G} = (V, E, w)$ be an edge-weighed graph and $\bar{G}'$ be an optimal solution for HS with score s.

Algorithm 2 Basic

Input: An edge-evolving graph $G(V, E, F)$
Output: $(G', [i, j])$, a solution for HDS
1: Compute $UB_{SOP}, \forall [l, r] \in [1, |F|]$
2: Estimate a Lower Bound LB for HDS
3: **for all** $[l, r] \in [1, |F|]$ **do**
4: Prune $[l, r]$ if $UB_{SOP} \leq LB$ or $UB_{STR} \leq LB$
5: **end for**
6: **for all** Not pruned sub-intervals $[l, r]$ **do**
7: $TopDown(\bar{G}(l, r))$
8: **end for**
9: **return** The highest-score temporal subgraph $(G', [i, j])$

Let also $C = \{(P, N)\}$ be a set of pairs where P represents the score of a connected component of positive edges in $\bar{G}$ and N represents the lowest score among negative edges with one endpoint in the corresponding positive component. We have:

$$UB_{STR} = \sum_{(P,N)\in C} \max(0, P + N) - \min_{(P,N)\in C}(N) \geq s$$

We do not report the proof due to space limitations. Since UB_{STR} preserves more structural information, it is significantly tighter than UB_{SOP}. When the positive components in G are "close" to each other, UB_{STR} is close to the score of the optimal solution.

The time necessary to compute both UB_{SOP} and UB_{STR}, is linear in the number of graph edges. In practice UB_{SOP} takes much less time and both bounds are significantly faster than TopDown. As we show in the experimental evaluation, both upper bounds have good pruning power in real and synthetic problem instances.

C. A filter-and-verify algorithm for HDS

A filter-and-verify approach for HDS, called *Basic*, which uses the developed bounds is presented in Alg. 2. *Basic* takes as input an edge-evolving network. In *Step 1* we compute UB_{SOP} for all $O(t^2)$ sub-intervals. Next, we estimate a lower bound LB for HDS by applying TopDown on the k sub-intervals that have the highest UB_{SOP} (*Step 2*). The intuition is that if a sub-interval induces many positive edges it is likely to contain a good solution.

In *Steps 3-5* we check each of the $\frac{t(t-1)}{2}$ sub-intervals progressively and prune irrelevant sub-intervals by first applying UB_{SOP} and then UB_{STR}. All intervals that are not pruned are next verified by using *TopDown* (*Steps 6-8*). We maintain the best temporal subgraph $(G', [i, j])$ from all applications of TopDown and return it as the solution (*Step 7*). Since UB_{SOP} and UB_{STR} are upper bounds for HS, the filtering procedure does not have any loss in quality. *Basic* reduces the running time by orders of magnitude with respect to a naïve solution by discarding most of the search space. However, *Basic* does not scale well with the time length

t, as it requires the computation of UB_{SOP} for all $O(t^2)$ intervals. We reduce the complexity of the initial filtering phase to $O(t{\cdot}\log^2(t){\cdot}|E|)$ by grouping intervals, as described in the next section.

IV. Scalable filtering by sub-interval grouping

In this section we define a sub-interval aggregation scheme that partitions the quadratic space of candidate sub-intervals into $O(t \cdot log(t))$ disjoint groups. We introduce a fast filtering phase that prunes groups without considering their individual members and hence has a sub-quadratic complexity of $O(t \cdot log^2(t) \cdot |E|)$. The grouping of sub-intervals follows an intuitive principle: sub-intervals with significant overlap produce similar aggregated graphs and hence solutions of similar score. Our final algorithm for HDS, called *MEDEN*, achieves an order of magnitude improvement over *Basic* (Alg. 2) in our experiments.

A. Aggregation scheme

The sub-interval aggregation scheme maintains a constant minimum fraction of overlap for any pair of sub-intervals within the same group. Groups are composed of *left-aligned* sub-intervals, i.e. sub-intervals that share the same starting point. To ensure a minimum overlap of α (with $0 \leq \alpha < 1$) within groups, we combine short sub-intervals in smaller size groups and larger sub-intervals in larger size groups. The relationship between the number of groups and interval length has an exponential form, ensuring that the total number of disjoint groups is $O(t \cdot log(t))$.

We define the fraction of *overlap* between two sub-intervals as the ratio of the lengths of their intersection and union. The overlap varies between 0 (the sub-intervals do not intersect) to 1 (the sub-intervals are exactly the same).

A *left-aligned* group $S(i, k_1, k_2)$ is a group of sub-intervals that start from the same position i and end at positions k_1 through $k_2, k_2 \geq k_1$. Formally:

$$S(i, k_1, k_2) = \{[i, j] | k_1 \leq j \leq k_2\}.$$

The *minimum overlap* in a group of left-aligned sub-intervals $S(i, k_1, k_2)$ is the ratio between the minimum and maximum length of intervals within the group, i.e. $(k_1 - i + 1)/(k_2 - i + 1)$.

Let $G = (V, E, F)$ be an edge-evolving network with total time period $t = |F|$. Let α be the minimum admitted overlap within a group. For every position $i, 1 \leq i \leq t$, we divide the sub-intervals starting at i in left-aligned groups as follows:

$$\mathcal{S}^i_{G,\alpha} = \Big\{ S\Big(i, i + \Big\lfloor \frac{1}{\alpha^{j-1}} \Big\rfloor, i + \Big\lfloor \frac{1}{\alpha^{j}} \Big\rfloor - 1, t\Big)$$

$$\Big| 1 \leq j \leq n-1 \text{ and } \Big\lfloor \frac{1}{\alpha^{j-1}} \Big\rfloor \leq \Big\lfloor \frac{1}{\alpha^{j}} \Big\rfloor - 1 \Big\} \bigcup$$

$$\Big\{ S\Big(i, i, i\Big) \Big\} \bigcup \Big\{ S\Big(i, i + \Big\lfloor \frac{1}{\alpha^{n-1}} \Big\rfloor, t\Big) \Big\},$$

where n is the smallest integer satisfying $i + \left\lfloor \frac{1}{\alpha^n} \right\rfloor - 1 \geq t$. The key concept in the above aggregation strategy is that the set of all left-aligned intervals starting at i is divided in a logarithmic number of groups, since the difference between the earliest and latest interval ends within a group grows exponentially with the group index j. We also consider the extreme cases $S(i, i, i)$ and $S(i, i + \lfloor 1/\alpha^{n-1} \rfloor, t)$ for the sake of completeness.

Building on the partitioning for one starting point, we define the complete partition of intervals for G as $\mathcal{S}_{G,\alpha} = \bigcup_{1 \leq i \leq t} \mathcal{S}^i_{G,\alpha}$. Clearly all sub-intervals of $[1, t]$ are included in at least one set of $\mathcal{S}_{G,\alpha}$ and the sets are mutually disjoint.

The following lemma states that the aggregation scheme produces a partition of size $O(t \cdot \log(t))$ and the minimum overlap of a pair within each group exceeds α.

Lemma 4.1: Let $G = (V, E, F)$ be a time-varying network ($t = |F|$), and α be a real value in $[0, 1]$. Then, the complete partition of intervals $\mathcal{S}_{G,\alpha}$ has the following properties:

1) $|\mathcal{S}_{G,\alpha}| = O(t \cdot \log(t))$
2) $overlap([i, j], [k, l]) \geq \alpha, \forall [i, j], [k, l] \in S, \forall S \in \mathcal{S}_{G,\alpha}$

Proof: 1) Given a starting time i such that $1 \leq i \leq t$, by definition $|\mathcal{S}^i_{G,\alpha}| \leq n + 1$ where n is the minimum integer satisfying $i + \left\lfloor \frac{1}{\alpha^n} \right\rfloor - 1 \geq t$. Consequently, we have $i + \left\lfloor \frac{1}{\alpha^{n-1}} \right\rfloor - 1 < t$ and hence $i + \frac{1}{\alpha^{n-1}} - 2 < t$. We obtain the following bound for a single starting point: $|\mathcal{S}^i_{G,\alpha}| \leq n + 1 < \frac{\log(t-i+2)}{-\log(\alpha)} + 2$. Therefore:

$$|\mathcal{S}_{G,\alpha}| = \sum_{1 \leq i \leq t} |\mathcal{S}^i_{G,\alpha}| < t \cdot \Big(\frac{\log(t - i + 2)}{-\log(\alpha)} + 2 \Big)$$

2) Given a set $S \in \mathcal{S}_{G,\alpha}$, with S not extreme, by definition the minimum overlap is:

$$overlap_{min}(S) = \frac{\lfloor \frac{1}{\alpha^{j-1}} \rfloor + 1}{\lfloor \frac{1}{\alpha^{j}} \rfloor},$$

where j is between 1 and $n - 1$. It is easy to check that for any j we have $overlap_{min}(S) \geq \alpha$. The condition holds for the extremes as well. ■

B. Bounding sub-interval groups

Next we show that a group of intervals can be pruned at once by computing an upper bound for the optimal solution on a *dominating graph*, representing the whole group. The idea is to assign a weight to each edge of the dominating graph that is higher than the corresponding edge weight of any aggregated graph induced by an interval of the group. This guarantees that an upper bound for the group is an

upper bound for the optimal solution in any interval of the group.

Given an edge-evolving network $G = (V, E, F)$, we define the *dominating graph* $\hat{G}(i, k_1, k_2)$ of a group of left-aligned intervals $S(i, k_1, k_2)$ as:

$$\hat{G}(i, k_1, k_2) = \max_{\{j | k_1 \leq j \leq k_2\}} \bar{G}(i, j) \quad (1)$$

Lemma 4.2: For an edge-evolving network $G = (V, E, F)$, let $S(i, k_1, k_2)$ be a group of left-aligned intervals with corresponding dominating graph $\hat{G}(i, k_1, k_2)$. Then, an optimal solution of HS in $\hat{G}(i, k_1, k_2)$ is an upper bound for the optimal solution of HS in any aggregated graph induced by an interval in $S(i, k_1, k_2)$.

Proof: Let $\bar{G}'_{i,j} = (V', E', w_{i,j})$ be an optimal solution for HS in an arbitrary interval $[i, j] \in S(i, k_1, k_2)$ with score $score(\bar{G}'_{i,j}) = s'$. Then, by definitions of graph *max* operator and dominating graph, the score of the corresponding subgraph $\hat{G}'$ of $\hat{G}(i, k_1, k_2)$ is:

$$score(\hat{G}') = \sum_{e \in E'} \max_{[i,r] \in S(i,k_1,k_2)} w_{ir}(e) \geq \sum_{e \in E'} \sum_{k=i}^{j} f^k(e) = s'$$

Therefore an optimal solution score for HS in $\hat{G}(i, k_1, k_2)$ exceeds the score of HS in any subgraph $\bar{G}'_{i,j}$. ■

Clearly any upper bound for the optimal solution in the dominating graph of a group is also an upper bound for the optimal solution for any group member. Therefore, we can apply the upper bounds UB_{SOP} and UB_{STR} (see Sect. III) on the dominating graphs of each group with total time complexity $O(t \cdot log(t) \cdot |E|)$, assuming that dominating graphs in the groups are precomputed. If we compute the dominating graphs for each group naïvely by considering all group members, we will again incur a quadratic cost. In order to reduce the quadratic cost of group filtering we need to be able to compute dominating graphs in time that is sub-linear in t.

C. Computing a dominating graph in $O(\log(t))$

We compute the dominating graph for a group of left-aligned intervals using a binary tree index that stores partially-constructed graphs at different coarseness levels. Both building the index and its storage have complexity $O(t \cdot |E|)$.

Lemma 4.3: Let $G = (V, E, F)$ be an edge-weighted evolving network and i, k, j be three time points such that $1 \leq i \leq k \leq j \leq |F|$. The following recursive composition rules can be used to generate the dominating graph $\hat{G}$ of any left-aligned group of intervals:

1) $\bar{G}(i, j) = \bar{G}(i, k) + \bar{G}(k + 1, j)$
2) $\hat{G}(i, k, j) = \bar{G}(i, k) + \hat{G}(k + 1, k + 1, j)$
3) $\hat{G}(i, i, j) = \max(\hat{G}(i, i, k), \hat{G}(i, k + 1, j))$

Rule 1 states that graphs induced by consecutive intervals can be aggregated to form a graph for the whole interval. Rule 2 generates the dominating graph for a left-aligned group with common prefix $[i, k]$, by summing the aggregated graph of the prefix and the dominating graph of all suffixes. Rule 3 shows how to generate the dominating graph for all left-aligned intervals in $[i, j]$ by taking the maximum of the dominating graphs of (i) intervals up to a midpoint k and (ii) the intervals with common prefix $[i, k + 1]$.

We partition the total time interval $[1, t]$ in non-overlapping sub-intervals that span it completely at decreasing length resolutions that are powers of 2. We place all sub-intervals in a binary tree. The leaves of the tree represent t single slice intervals. At the second level we have $\lceil \frac{t}{2} \rceil$ non-overlapping intervals of size 2, each having two sub-interval children from the lower level. We proceed with this grouping until we reach the root representing the whole time length $[1, t]$. Each interval tree node $[l, r]$ maintains both the dominating graph of all its left-aligned sub-intervals $\hat{G}(l, l, r)$ and its aggregated graph $\bar{G}(l, r)$. An example index for the time interval $[1, 8]$ and its node structure are presented in Fig. 2.

We build the tree in a bottom-up fashion, starting from the leaves. Every internal node is computed in constant time in t, by composing its $\bar{G}$ and $\hat{G}$ graphs from its children, according to the composition rules in Lemma 4.3. Since the number of nodes of the binary tree is linear in the number of its leaves, the building time and storage complexity of our index are both $O(t \cdot |E|)$.

Constructing the dominant graph of any group in our space partitioning $\mathcal{S}_{G,\alpha}$ (defined in Section IV-A) can be performed efficiently using our tree index. The dominating graph corresponding to any group in our partitioning can be composed by querying the index. The evaluation of an example query is demonstrated in Fig. 2. We show how we obtain the dominating graph $\hat{G}(4, 4, 6)$ corresponding to the left-aligned group of intervals with common prefix $[4, 4]$ and with extent up to 6. We perform a depth-first search on the tree, starting from the root and following left children links first. Sub-interval nodes that intersect the query (but are not contained in the query) are visited and the process continues in one of their children that is either intersected or contained in the query. In our example, nodes that are visited are shaded in light gray, while those that are used to compose the dominating graph $[4, 4]$ and $[5, 6]$ are shaded in dark gray. The nodes, used in the composition are processed in increasing time order and the partial dominating and aggregated graphs are maintained in the recursive invocation.

Lemma 4.4: The time complexity of constructing the dominating graph of a single group is $O(\log(t) \cdot |E|)$.

The proof for Lemma 4.4 is based on the observation that only a logarithmic number of nodes are used for the construction of a dominating graph (the darkest shaded nodes in the example in Fig. 2) and that the number of visited nodes is within a constant factor.

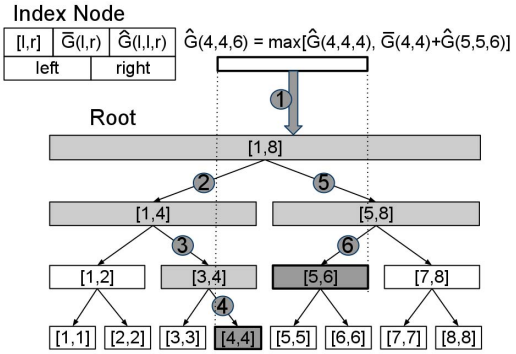


Figure 2. The structure of the interval index, its nodes and an example of interval query processing.

Algorithm 3 MEDEN

Input: An edge-evolving graph $G(V, E, F)$, α
Output: $(G', [i, j])$ a solution for HDS

```
1: Build an index on G
2: for all Left-aligned groups S ∈ S_{G,α} do
3:    Construct the group dominating subgraph Ĝ
4:    Compute Group UB_SOP
5: end for
6: Estimate a Lower Bound LB for HDS
7: for all Left-aligned groups S ∈ S_{G,α} do
8:    Prune S if UB_SOP ≤ LB or UB_STR ≤ LB
9: end for
10: for all Not pruned groups S and ∀[l, r] ∈ S do
11:    Costruct the aggregated graph Ḡ(l, r)
12:    Prune [l, r] if UB_SOP ≤ LB or UB_STR ≤ LB
13: end for
14: for all Not pruned intervals [l, r] do
15:    TopDown(Ḡ(l, r))
16: end for
17: return The highest-score evaluated subgraph (G', [i, j])
```

D. MEDEN

Our final algorithm for HDS called *MEDEN* is described in Alg. 3. It augments *Basic* with the scalable group-based filtering phase. After building an interval index on G (Step 1), we group the intervals, compute the dominating graphs and compute UB_{SOP} for each group in the partitioning $\mathcal{S}_{G,\alpha}$ (Steps 2-5). We estimate a lower bound (Step 6) by interpolating UB_{SOP} for intervals for which it is not computed due to the grouping. Next, we prune whole groups by progressively applying our upper bounds on the group dominating graphs (Steps 7-9). The intervals from all groups that cannot be pruned are then processed as in *Basic*, while using the index to obtain their aggregated graphs. The best temporal subgraph is returned as a result.

Theorem 4.1: The complexity of the first filtering phase of MEDEN, including indexing, grouping, estimation and group filtering (Steps 1-9 in Alg. 3), is $O(t \cdot \log^2 t \cdot |E|)$. The proof of Theorem 4.1 follows from the fact that there are $O(t \cdot \log(t))$ groups in $\mathcal{S}_{G,\alpha}$ (Lemma 4.1) and the ability to construct the dominating graph of every group using the index in $O(\log(t) \cdot |E|)$ (Lemma 4.4). In addition, MEDEN's group filtering does not have any loss in quality since it does not discard any groups that contain the optimal solution (Lemma 4.2).

MEDEN can be naively augmented to finding a set of top-k heavy non-overlapping subgraphs. The top-k procedure computes the heaviest dynamic subgraph, marks all solution edges as $-\infty$ in the solution time slices and repeats the process until k solutions are found. Improving such an extension to reuse results from previous invocations, as well as admitting overlap are not as trivial and will be the subject of further investigation.

Table I
SIZES OF EXPERIMENTAL NETWORKS

Dataset	#Nodes	#Edges	#Slices	Slice length
Twitter	2605	14871	204	1 day
ENRON	1598	6244	925	1 day
LA Traffic	1923	6208	2160	20 min

V. EXPERIMENTAL ANALYSIS

We evaluate MEDEN on three real-world networks: (i) the highway transportation network of Los Angeles, California during April 2011, (ii) the Enron email dataset[2] and (iii) a subset of the Twitter follower graph and the corresponding tweeting activity during the second half of 2009 [12], [13]. MEDEN exhibits good scaling properties for increasing interval lengths and graph sizes and it is not sensitive to its main parameter α.

We use a fixed sampling rate of edge evolution in consecutive time slices. All edges are in one of the active $(+1)$ or inactive (-1) states at each time slice. The input size of an HDS instance includes the graph size and time length. Table I lists the sizes of our datasets.

Our *Traffic dataset* is the highway network of Los Angeles. The edges are highway segments. We use the average speeds (at a resolution of one slice every 20 min.) in highway segments over the duration of a month (April 2011) hosted by the *PeMS*[3] project. An edge is considered active if its average speed falls below the threshold of 30 mph. The HDS solution in this dataset corresponds to a sizable and long congestion.

The *ENRON* dataset is a collection of corporate email messages exchanged among employees of the Enron corporation [14]. An edge between two email accounts is present

[2]http://www.cs.cmu.edu/~enron/
[3]http://pems.dot.ca.gov/

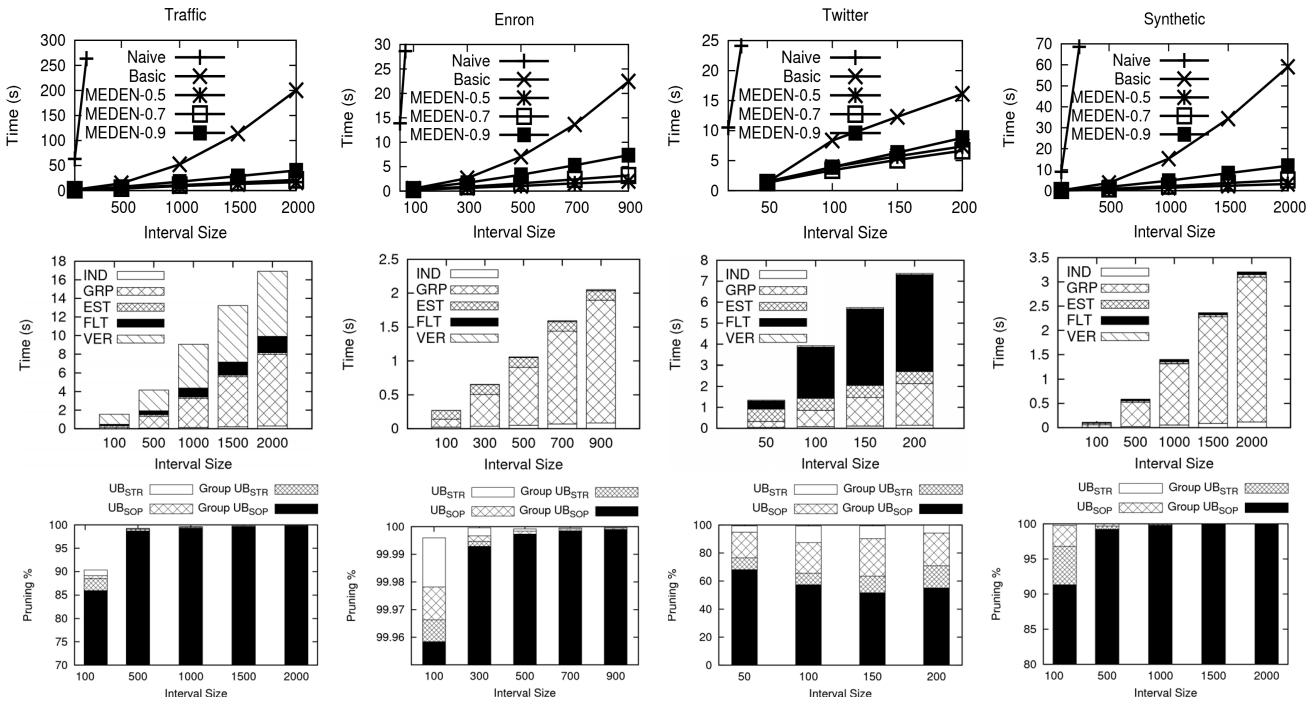


Figure 3. Performance of MEDEN on real and synthetic networks for increasing interval lengths. The sizes of the real networks are given in Table I. The synthetic network has 500 nodes and 1000 edges and is generated with 0.1 activation density and 0.3 propagation factor. The first row compares the total time of Basic, Naïve and MEDEN for different values of α (0.5, 0.7 and 0.9). The second row shows the running time of each of MEDEN's steps for $\alpha = 0.5$. The reported time components map to the phases of MEDEN (Alg. 3) as follows: *IND* is the time to build the index (line 1); *GRP* is the time to construct the dominating graphs (lines 2-5); *EST* is the time to estimate a LB (line 6); *FLT* is the filtering time (lines 7-13); and *VER* is the verification time (lines 14-16). The third row reports the percentage of intervals pruned by each filtering phase of MEDEN for $\alpha = 0.5$. *Group* UB_{sop} and *Group* UB_{str} are the fractions of intervals pruned by applying the upper bounds on whole groups, while UB_{sop} and UB_{str} refer to filtering the remaining individual intervals. In all cases most intervals are pruned by applying UB_{sop} on groups of intervals (note that the scale of the Y axes does not start from zero).

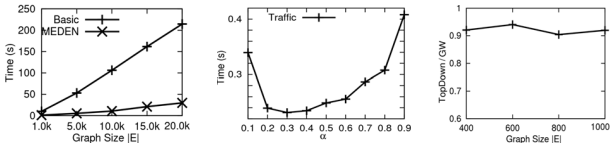
if at least one message is exchanged over the whole timeline. The dataset spans communication during the years $1999 - 2001$, and an edge is active during the day if there is at least one email message in any direction. The HDS solution on this dataset is the communication backbone of correspondence.

We extract a subset of the follower network of *Twitter* [12] and use 6 months of tweet messages of the included nodes [13]. We mark a follow edge as active if the tweets of the adjacent nodes are similar for the day. The intuition is that if a follower-followee pair tweets about similar things they may influence each other. We use cosine similarity to compare daily tweets and assign an edge as active if the similarity exceeds a value of 0.004.

The scalability of MEDEN is also evaluated for *synthetic data*. We define a synthetic generator for edge-evolving networks. The edge weights are generated by a flow propagation procedure, where a seed edge is activated at random, and then neighboring edges are activated based on a probability of influence that we call *propagation factor*. A high propagation factor corresponds to dynamic behavior in which the activation of an edge affects neighboring edges (graph locality) as well as the same edge in future time steps (time locality). The process is repeated multiple times until a fixed *activation density* (fraction of active time edges) is reached. We use random graphs for the underlying graph structure.

We first evaluate the scalability of *MEDEN* in comparison to a *Naïve* solution and the *Basic* algorithm described in Section III. Naïve enumerates all the intervals and executes TopDown for each of them. Basic differs from *MEDEN* in the lack of group filtering (Sect. IV).

The results for our three real-world and one synthetic networks are presented in Fig. 3. All comparison experiments (first row in Fig. 3) report the total processing time (filtering and verification) for increasing length of the time interval. The reported time is the average over a hundred executions on different intervals of the same length, chosen uniformly from the time interval of the whole dataset. Naïve is reported only for small intervals, since it does not scale with time. Even on small intervals, it takes two orders of magnitude longer than MEDEN to complete. Apart from the Twitter dataset, Basic scales super-linearly, due to its quadratic dependence on the total time length. In Twitter, the interval length is short, compared to the number of edges, therefore the super-linearity of Basic is not evident. MEDEN scales almost linearly on all datasets. It improves Basic's running



(a) Graph Size (b) Varying α (c) TopDown Quality

Figure 4. (a) Scalability comparison between MEDEN and Basic for increasing graph size. (b) Effect of α on the running time for the Traffic dataset ($t = 100$). (c) Quality of the TopDown solution score as a fraction of the score obtained by GW-algorithm on instances of size $|E| = [400, 1000]$.

time by an order of magnitude for evaluation intervals of size 1000 and higher, and outperforms Basic in every time length. The parameter α does not affect significantly the performance of MEDEN ($\alpha = 0.5$ results in optimal running time on all datasets except for Twitter).

The second row of experiments in Fig. 3 reports the running time for each of the main phases of MEDEN. In Enron and Synthetic, computing the dominating graphs (*GRP*) dominates the running time, and hence smaller values of α (i.e. fewer, but bigger groups) improve the performance. In Twitter, the time is dominated by the filtering phase. In this case smaller size groups ($\alpha = 0.7$) are pruned better leaving less intervals for the expensive filtering phase. The verification time is a small fraction of the overall time in all datasets except Traffic. In this network, many intervals have solutions of score similar to the optimal and hence are harder to prune.

The third row of Fig. 3 reports the percentage of pruned intervals for each variation of our upper bounds. For long evaluation intervals the group filtering (Group UB_{sop} and Group UB_{str} combined) prunes more than 99% of the candidate intervals. The Twitter dataset evolves over only 200 time steps, therefore it does not follow the same trend. A significant portion of remaining intervals is pruned by the tighter UB_{str} and the second step pruning on single intervals. For instance, in *Traffic* (for size 100), 86% of intervals are pruned by Group UB_{sop}. Over 4% of the remaining candidates are pruned by Group UB_{str} and the individual interval pruning. This percentage is higher for the other datasets. Regardless of the specifics of each dataset, MEDEN scales well with the interval size and completes in less than 20 seconds for intervals up to 2000 time slices on all datasets. In the Traffic dataset this input size corresponds to 2000 graphs (one for each slice) of 6000 edges each, i.e. 12 million total edges. The naïve approach would consider all sub-intervals and their aggregated graphs, amounting to processing about 24 billion edges.

Next, we study the scalability of *MEDEN* for increasing graph sizes. We use a synthetic graph of average node degree 2.5 and vary the number of edges from 1 to 20 thousand. The interval length is 500, the activation density 0.1 and the propagation factor 0.3. Fig. 4(a) reports the total processing time of Basic and MEDEN. Both algorithms scale linearly with the graph size as their filtering phases are linear in the number of edges. MEDEN is about 7 times faster than Basic on graphs with 20000 edges due to the efficient and effective group filtering.

Fig. 4(b) reports the total processing time of Basic and MEDEN on *Traffic* for varying α. The running time is not affected significantly by this parameter as long as middle-range values are chosen.

The relative performance of *TopDown* in comparison to GW-algorithm is presented in Fig. 4(c). Obtaining an exact solution for the HDS problem is infeasible, and hence our baseline is the best score obtained by GW-algorithm. The solution scores obtained by TopDown exceed 90% of GW's scores, regardless of the instance size. We also evaluated TopDown's quality on a set of hard benchmark instances from literature (groups of instances K and P from [6]) and obtained a similar performance of 90% of GW's scores and 85% of the absolute optimum for these instances. Due to its lower complexity ($O(|E|\cdot\log(|V|))$ vs. $O(|V|^2\log(|V|))$ for GW), TopDown is two orders of magnitude faster than GW for $|E| = [100, 1000]$.

The HDS reported by MEDEN in Enron for 2001 is a star structure with J. Dasovich (Govt. Affairs Exec.) in the center and including R. Shapiro (Sr. VP) and other personnel, with time extent 5 weeks starting from $05/01/01$. The increased communication between J.D. and R.S. in the very beginning of 2001 coincides with the energy commodity trading deregulation in California from which Enron benefited with increased revenue. The HDS in our Traffic network occurs in the afternoon of Friday, 04/01/11 at the intersection of HWY405 and HWY10 in Santa Monica, CA with extent of 1h20m and spanning a location of high congestion at peak traffic hours.

VI. Related Work

Dynamic networks have been considered for evolutionary clustering [2], [3], frequent temporal subgraph mining [15], [16], [17], evolution rules mining [18] and novel event detection [19], [20]. Lin et al. [3] and later Kim et al. [2] propose community discovery algorithms for time-varying graphs. The main difference between our problem and clustering is that the latter maximizes the intra-cluster connectivity, while the former maximizes the total weight of a connected component of arbitrary topology. Dynamic pattern mining methods [15], [16] extend the notion of frequent subgraphs to the dynamic case and hence they differ from MEDEN by maximizing the frequency as opposed to edge score.

Closer to our formulation are methods on novelty and event detection [19], [20] in which the target is to discover deviating edge/node behavior. A distinctive difference from our technique is the objective of finding outstanding edges and vertices [19] or significant global shift in the behavior of nodes with respect to their history [20]. In contrast,

our goal is to discover general connected subgraphs. Our HDS formulation considers jointly the temporal behavior of edges, as well as their connectivity. In addition to mining approaches, there has been recent work on modeling and theoretical analysis of edge evolving networks under an edge independence assumption [21], [22]. In contrast, we focus on the collective dynamics of edges in the form of heavy dynamic subgraphs.

Algorithms for high-scoring subgraphs in static networks have been widely studied [4], [5], [6]. Common approaches consider the *Prize-Collecting Steiner Tree* (PCST) problem [10] and the related Maximum Weight Connected Graph (MCG) problem [5]. The latter calls for finding the maximum-scoring connected subgraph of a fixed vertex size in a node-weighted network. Both PCST and MCG are defined on static networks and are either approached by approximation algorithms [10], or solved on relatively small instances to optimality [6]. Our TopDown heuristic is closely related to the PCST heuristics, but scales much better.

VII. Conclusion

We introduce the *Heaviest Dynamic Subgraph (HDS)* problem for edge-evolving networks and propose MEDEN, an algorithm for HDS that scales to large networks of long evolution extents. MEDEN uses tight upper bounds of the optimal solution to prune irrelevant time intervals. We avoid the quadratic enumeration of all possible intervals by MEDEN's efficient and effective group filtering phase of complexity $O(t \cdot \log^2(t) \cdot |E|)$. This significant reduction is possible due to our interval grouping scheme that combines intervals of high overlap that are likely to produce similar solutions and enables their joint pruning. Each interval grouping is performed in time $O(\log(t))$ using an efficient index. Our extensive experimental analysis demonstrates that MEDEN is scalable on real and synthetic networks and is not sensitive to its single parameter α. Our method achieves several orders of magnitude of improvement compared to a naïve approach.

Acknowledgements

Research was sponsored by the Army Research Laboratory and was accomplished under Cooperative Agreement Number W911NF-09-2-0053. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Laboratory or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation here on. Research was also partially sponsored by the National Science Foundation under Grant No. IIS-0917149.

References

[1] J. Leskovec, L. Backstrom, R. Kumar, and A. Tomkins, "Microscopic evolution of social networks," in *Proc. of KDD*, 2008.

[2] M. Kim and J. Han, "A Particle-and-Density Based Evolutionary Clustering Method for Dynamic Networks," in *Proc. of VLDB*, 2009.

[3] Y. Lin, Y. Chi, S. Zhu, H. Sundaram, and H. Sundaram, "Facetnet : A framework for analyzing communities and their evolutions in dynamic networks," in *Proc. of WWW*, 2008.

[4] M. T. Dittrich, G. W. Klau, A. Rosenwald, T. Dandekar, and T. Mller, "Identifying functional modules in proteinprotein interaction networks: an integrated exact approach," *J. of Bioinformatics*, 2008.

[5] H. F. Lee and D. R. Dooly, "Algorithms for the constrained maximum-weight connected graph problem," in *Proc. of Naval Research Logistics*, 1996.

[6] I. Ljubić, R. Weiskircher, U. Pferschy, G. W. Klau, P. Mutzel, and M. Fischetti, "An Algorithmic Framework for the Exact Solution of the Prize-Collecting Steiner Tree Problem," *J. of Math. Prog.*, 2005.

[7] P. Bickel, C. Chen, J. Kwon, and J. Rice, "Traffic Flow on a Freeway Network," *J. of Electrical Eng.*, 2001.

[8] R. Jin, S. Mccallen, C. chi Liu, E. Almaas, and X. J. Zhou, "Identifying dynamic network modules with temporal and spatial constraints," in *Pac. Symp. Biocomput.*, 2009.

[9] G. Krouk, P. Mirowski, Y. Lecun, D. E. Shasha, and G. M. Coruzzi, "Predictive network modeling of the high-resolution dynamic plant transcriptome in response to nitrate." *J. of Genome Biology*, 2010.

[10] D. S. Johnson, M. Minkoff, and S. Phillips, "The Prize Collecting Steiner Tree Problem : Theory and Practice," *Proc. of SODA*, 2000.

[11] J. Feigenbaum, N. Haven, C. H. Papadimitriou, U. C. Berkeley, and S. Shenker, "Sharing the Cost of Multicast Transmissions," *J. Computer and System Sciences*, 2001.

[12] H. Kwak, C. Lee, H. Park, and S. Moon, "What is twitter, a social network or a news media?" in *Proc. of WWW*, 2010.

[13] J. Yang and J. Leskovec, "Temporal Variation in Online Media." 2011.

[14] J. Diesner, T. L. Frantz, and K. M. Carley, "Communication Networks from the Enron Email Corpus It's Always About the People. Enron is no Different," *J. of Computational and Mathematical Organization Theory*, 2006.

[15] K. Borgwardt, H. Kriegel, and P. Wackersreuther, "Pattern Mining in Frequent Dynamic Subgraphs," *Proc. of ICDM*, 2006.

[16] B. Wackersreuther, P. Wackersreuther, A. Oswald, C. Böhm, and K. M. Borgwardt, "Frequent Subgraph Discovery in Dynamic Networks," *Proc. of MLG*, 2010.

[17] T. Oshino, Y. Asano, and M. Yoshikawa, "Time Graph Pattern Mining for Web Analysis and Information Retrieval," *WAIM LNCS*, 2010.

[18] M. Berlingerio and F. Bonchi, "Mining graph evolution rules," in *Proc. of ECML PKDD*, 2009.

[19] J. Abello, T. Eliassi-Rad, and N. Devanur, "Detecting Novel Discrepancies in Communication Networks," *Proc. of ICDM*, 2010.

[20] L. Akoglu and C. Faloutsos, "Event detection in time series of mobile communication graphs," in *Proc. of Army Science Conference*, 2010.

[21] A. Clementi, A. Monti, F. Pasquale, and R. Silvestri, "Information Spreading in Stationary Markovian Evolving Graphs," *Informatica*, 2009.

[22] C. Avin, M. Kouck, and Z. Lotker, "How to explore a fast-changing world," in *Automata, Languages and Programming*, 2008.