The Thirty-Second AAAI Conference
on Artificial Intelligence (AAAI-18)

Mining Heavy Temporal Subgraphs: Fast Algorithms and Applications

Jose Cadena, Anil Vullikanti
Department of Computer Science and Biocomplexity Institute, Virginia Tech, Blacksburg, VA 24061
{jcadena,vsakumar}@vt.edu

Abstract

Anomaly detection is a fundamental problem in dynamic networks. In this paper, we study an approach for identifying anomalous subgraphs based on the Heaviest Dynamic Subgraph (HDS) problem. The HDS in a time-evolving edge-weighted graph consists of a pair containing a subgraph and sub-interval whose sum of edge weights is maximized. The HDS problem in a static graph is equivalent to the Prize Collecting Steiner Tree (PCST) problem with the Net-Worth objective—this is a very challenging problem, in general, and numerous heuristics have been proposed. Prior methods for the HDS problem use the PCST solution as a heuristic, and run in time quadratic in the size of the graph. As a result, they do not scale well to large instances. In this paper, we develop a new approach for the HDS problem, which combines rigorous algorithmic and practical techniques and has much better scalability. Our algorithm is able to extend to other variations of the HDS problem, such as the problem of finding multiple anomalous regions. We evaluate our algorithms in a diverse set of real and synthetic networks, and we find solutions with higher score and better detection power for anomalous events compared to earlier heuristics.

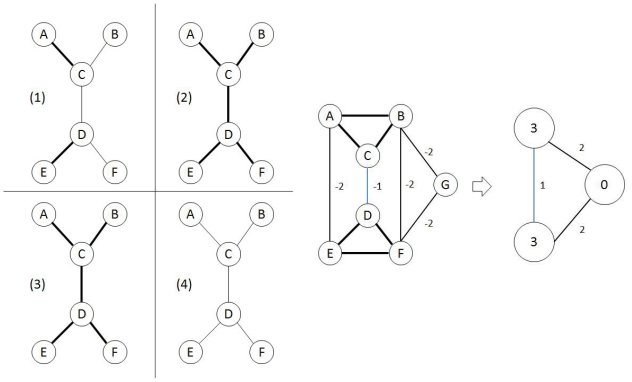
1 Introduction

Networks in most applications are dynamic and evolve over time. An important problem in temporal data is to detect "anomalous" events, e.g., an unusual rate of incidence of a disease in a region. As discussed in the survey of (Akoglu, Tong, and Koutra 2014), the different directions on dynamic network anomaly detection are formalized in terms of changes in properties of nodes and edges (Kang et al. 2011), spectral properties (Sun et al. 2008) or communities (Peel and Clauset 2014), or window-based (Bogdanov, Mongiovì, and Singh 2011; Mongiovì et al. 2013).

We focus on the window-based approach in a weighted temporal graph. In this setting, we are given a fixed graph $G = (V, E)$ with a weight $f^t(e)$ for an edge $e = (u, v) \in E$ for timestamps $t = 1, \ldots, T$, which may reflect the significance of the interaction at some time. The objective of the *Heaviest Dynamic Subgraph* (HDS) problem (Bogdanov, Mongiovì, and Singh 2011) is then to find a connected subgraph $G' = (V', E')$ and a time interval $[i, j]$ whose sum of edge scores is maximized (see Section 2 for a formal definition). When the time interval is fixed (i.e., the graph is static), we refer to this as the *Heaviest Subgraph* (HS) problem. The HDS has been shown to be able to identify interesting events in social media and traffic data networks. The heaviest subgraph problem for a static graph is the complement of the well-known *Prize Collecting Steiner Tree* (PCST) problem—this is equivalent to the PCST problem with the *Net-Worth* objective, which is known to be very challenging in general (Bateni, Hajiaghayi, and Liaghat 2013). Despite a lot of work, no rigorous algorithms are known for the NetWorth objective, but usually the Goemans-Williamson (Goemans and Williamson 1997) algorithm for PCST is used as a heuristic.

In this paper, we develop a new approach for the HDS problem and its extensions that combines rigorous algorithmic and practical techniques. Our contributions are:
(1) We design a new temporal filtering method that selects a sub-quadratic set of time intervals and provably preserves the maximum HDS solution, within a factor of 2. Together, with a faster algorithm for the PCST problem (by adapting the technique of (Cole et al. 2001)), we obtain a near-linear time algorithm, both in terms of the number of time intervals and the size of the network.
(2) Our algorithms can be applied to various extensions of the HDS problem, such as finding multiple anomalous regions, finding anomalous regions *rooted* at a subset of interest, and finding regions up to a size constraint.
(3) We evaluate our algorithms in real and synthetic networks. Our proposed algorithm for the HDS problem is several orders of magnitude (up to four) faster than existing methods. For the generalization of HDS to multiple anomalous regions, we discover more regions while either matching or improving on the objective value of the existing state-of-the-art method. Further, we show that the subgraphs identified by our algorithms are relevant in their respective domains.

Since the Steiner tree and PCST are commonly used heuristics in a number of problems (Rozenshtein et al. 2014; Sadeghi and Frohlich 2013; Mongiovì et al. 2013; Wu et al. 2016), we expect our techniques for speeding up PCST will be useful more generally. Many details are omitted because of the space limitation, and are available in the full version at (Cadena and Vullikanti 2017).



(a) (b)

Figure 1: (a) Example of an instance of HDS with 4 time intervals. Thick edges have weight $+1$ and thin edges have weight -1. The HDS includes all the edges and spans sub-interval $[2, 3]$. For comparison, the HS in sub-interval [1,1] has a score of 1 by using either (A, C) or (D, E). (b) Reduction of an instance of HS to an instance of NW. Nodes A, B, and C are merged into a node of prize 3 (weights of (A, B), (B, C), and (C, A)). Nodes D, E, and F are merged similarly. Negative edges become positive.

2 Preliminaries

An *edge-evolving network* over a set $\mathcal{T} = \{t_1, \ldots, t_2\}$ of timestamps is a tuple $(G = (V, E), F, \mathcal{T})$, where: (1) $G = (V, E)$ is an undirected graph with set $V = V(G)$ of nodes and set $E = E(G)$ of edges; and (2) $F = \{f^{t_1}, \ldots, f^{t_2}\}$ is a family of weighing functions that assigns weights to edges. In most cases, $\mathcal{T}$ will just be a time interval $[t_1, t_2]$. Each function f^i is associated with a timestamp $i \in [t_1, t_2]$. For two timestamps $i \leq j$, such that $t_1 \leq i \leq j \leq t_2$, we say that $[i, j]$ is a *sub-interval* of $[t_1, t_2]$. Informally, the score $f^k(e)$ of an edge e represents the *importance* or *anomalousness* of the edge. For example, a positive edge may indicate increased interaction between two users in an online social network. Mongiovì et al. (Mongiovì et al. 2013) define $f^t(e) = -\log p^t(e)/\mu$, where $p^t(e)$ is the p-value associated with the edge at time t, and μ denotes a significance level threshold. We use $f^{[i,j]}(e) = \sum_{t \in [i,j]} f^t(e)$. A *temporal subgraph* of G is a pair $(G' = (V', E'), [i, j])$, where G' is a connected subgraph of G and $[i, j]$ is a sub-interval of $[t_1, t_2]$. The *score* of a subgraph G' in the interval $[i, j]$ is the sum of the weights of the edges in E' during the interval $[i, j]$, or, formally, $score(G', F, [i, j]) = \sum_{e \in E'(G')} \sum_{k=i}^{j} f^k(e)$. When the sub-interval of the temporal subgraph is clear from context, we omit it and refer only to G'.

Problem 2.1 (HDS and HS problems)**.** *Given an instance $(G, F, [t_1, t_2])$ of an edge-evolving network, the objective of the* HEAVIEST DYNAMIC SUBGRAPH *problem,* $\text{HDS}(G, F, [t_1, t_2])$*, is to find a temporal subgraph $(G', [i, j])$, over all possible subgraphs G' of G and sub-intervals $[i, j]$ of $[t_1, t_2]$, such that $score(G', F, [i, j])$ is maximized. The objective of the* HEAVIEST SUBGRAPH *problem,* $\text{HS}(G, f^{[t_1,t_2]})$*, is to find a temporal subgraph G' of maximum score in a fixed interval $[t_1, t_2]$ over all possible subgraphs G'.*

Notation	Description
$(G, f^{[t_1,t_2]})$	HS instance
$(G, F, \mathcal{T})$	Edge-evolving network
$(G, F, [t_1, t_2])$	HDS instance
$(G, F, [t_1, t_2], \tau)$	SAR instance
$\mathcal{R} = \{R_1 \ldots R_k\}$	SAR solution (or region set)
$\phi_{\text{HDS}}(G, F, [i, j])$	value of optimal solution for HDS instance
$(H, \mathbf{w}, \pi) = \mathcal{P}(G, [t_1, t_2])$	PCST instance equivalent to $(G, [t_1, t_2])$
$\phi_{\text{PCST}}(H, \mathbf{w}, \pi)$	Minimization PCST objective
$\phi_{\text{NW}}(H, \mathbf{w}, \pi)$	Net-worth PCST objective
w_e, π_i	Weight of edge e, prize of node i

Table 1: Summary of the notation used in the paper

The HS problem is closely related to the well-studied Prize-Collecting Steiner Tree (PCST) problem (Johnson, Minkoff, and Phillips 2000). In PCST, we are given an undirected graph $H = (V', E')$; each node i of the graph has a non-negative prize π_i, and each edge e has a non-negative weight w_e. The objective is to find a tree T that minimizes $\phi_{\text{PCST}}(H, \mathbf{w}, \boldsymbol{\pi}) = \sum_{e \in T} w_e + \sum_{i \notin T} \pi_i$. Goemans and Williamson (Goemans and Williamson 1997) develop an algorithm (referred to as Algorithm GW) that gives a 2-approximation to this problem but has a running time of $O(n^2 \log n)$, where $n = |V|$. Another objective that has been considered is the Net-Worth maximization (NW) problem, where the goal is to find a tree T that maximizes $\phi_{\text{NW}}(H, \mathbf{w}, \boldsymbol{\pi}) = \sum_{i \in T} \pi_i - \sum_{e \in T} w_e$. To avoid notational clutter, we will sometimes refer to these two objectives as ϕ_{PCST} and ϕ_{NW}, respectively, without mentioning the instance $(H, \mathbf{w}, \boldsymbol{\pi})$ whenever it is clear from the context. Observe that $\phi_{\text{NW}} = \sum_{i \in V'} \pi_i - \phi_{\text{PCST}}$, but an approximation to the PCST objective does not imply a similar approximation to the NW objective.

As observed by (Bogdanov, Mongiovì, and Singh 2011), the HS problem for an instance $(G, f^{[t_1,t_2]})$ and NW problems are equivalent in an approximation-preserving sense. Let $V_1, \ldots, V_r$ be the connected components of the graph $G[E^+]$ induced by the set E^+ of edges, where $E^+ = \{e \in E : f^{[t_1,t_2]}(e) \geq 0\}$ is the set of edges with non-negative score in this interval. Let $H = (V', E')$ be a weighted graph with $V' = \{V_1, \ldots, V_r\}$ and $(i, j) \in E'$ if there exists an edge $e = (u, v) \in E$ with $u \in V_i, v \in V_j$. For $e = (V_i, V_j) \in E'$, we define $w_e = \min\{f^{[t_1,t_2]}(e') : e' = (u', v')$ and $u' \in V_i, v' \in V_j\}$ to be the minimum score of an edge between V_i and V_j. For $V_i \in V'$, we define $\pi_i = \sum_{e=(u,v):u,v \in V_i} \max\{f^{[t_1,t_2]}(e), 0\}$ to be the sum of the positive edge scores for edges with both end points in V_i. An example of this reduction is shown in Figure 1. We will refer to $(H, \mathbf{w}, \pi) = \mathcal{P}(G, f^{[t_1,t_2]})$ as the PCST instance equivalent to $(G, f^{[t_1,t_2]})$. It can be verified that the score of $\text{HS}(G, f^{[t_1,t_2]})$ is equal to $\phi_{\text{NW}}(H, \mathbf{w}, \pi)$. Similarly, a subgraph H' of H can be mapped to a subgraph G' of G with equal score.

Next, we consider an extension of HDS to multiple subgraphs, as defined by (Mongiovì et al. 2013).

Problem 2.2 (Significant Anomalous Regions)**.** *Given an edge-evolving network $(G, F, \mathcal{T})$ and a threshold τ, the objective is to find a set of regions (temporal subgraphs) $\mathcal{R} = \{R_1, R_2, \ldots, R_k\}$ in decreasing order of scores, such that the score of a region R_i, without considering the score of positive edges overlapping with higher-scoring regions, is not below τ.*

Note that an edge can appear in multiple subgraphs, but it contributes to the score of only one of them. This allows for overlaps.

3 Proposed Methods

3.1 Small certificate for HDS: selection of a small number of time intervals

Algorithm 1 SELECTINTERVALS(t): Produce set of $O(t \log t)$ intervals.

Input: Time interval $[1, t]$
Output: Set $\mathcal{A}$ of sub-intervals from $[0, t]$
For $i \in [1, \log t]$, let $A_1(i)$ denote the set all the intervals of the form $[1, 2^i], [2^i, 2(2^i)] \ldots [(t - 2^i), t]$.
Let $A_1 = \cup_i A_1(i)$
For each interval $[a, b] \in A_1$, let $A_2(a, b)$ denote the sub-intervals that have either a or b as an end-point, i.e., intervals $[a, c]$, for all $c \in [a + 1, b]$ and $[d, b]$, for all $d \in [a, b - 1]$.
Let $A_2 = \cup_{[a,b] \in A_1} A_2(a, b)$.
Let A_3 be the set of intervals $[i, i + 1], [i, i + 2], \ldots, [i, \min\{i + \log t, i + t\}]$ for each $i \in [1, t]$.
return $\mathcal{A} = A_1 \cup A_2 \cup A_3$

Let $(G, F, [1, t])$ be an instance of HDS, where $[1, t]$ is chosen only to simplify notation. Instead of solving HDS for the instance by considering the HS solution on the graphs for each of the $O(t^2)$ time intervals, Algorithm SELECTINTERVALS picks $O(t \log t)$ intervals, such that a solution within a factor of two of the optimal HDS solution is preserved. Note that the set A_3 of intervals selected by Algorithm SELECTINTERVALS is not needed in the proof of Lemma 2; however, it turns out to give significant improvements in practice.

Lemma 1. *Let $1 \leq a < b < d \leq t$. Then, $\max\{\phi_{HDS}(G, F, [a, b]), \phi_{HDS}(G, F, [b, d])\} \geq \phi_{HDS}(G, F, [a, d])/2$.*

The proofs of this lemma and the next are presented in (Cadena and Vullikanti 2017).

Lemma 2. *Let $\mathcal{A}$ be the set of intervals returned by Algorithm* SELECTINTERVALS(t)*. Then, $|\mathcal{A}| = O(t \log t)$ and $\max_{(i,j) \in \mathcal{A}} \phi_{HDS}(G, [i, j]) \geq \phi_{HDS}(G, [1, t])/2$.*

3.2 Algorithm FASTGW for the HS problem

Next, we present algorithm FASTGW for the HS problem, which combines ideas from three prior results in a non-trivial manner. We improve upon the algorithm of (Cole et al. 2001), who adapt the primal-dual algorithm of (Goemans and Williamson 1997) (denoted by GW), using improved pruning and data structures, leading to an improved running time. We briefly describe the intuition and then present algorithm FASTGW. Some of the details are described in (Cadena and Vullikanti 2017).

Let $(H = (V', E'), \mathbf{w}, \pi)$ be an instance of PCST. The rooted version of the PCST problem with root r can be formulated as the following integer program (PCST-IP):

$$\min \sum_e w_e x_e + \sum_{T \subseteq V' - \{r\}} z_T \pi(T) \text{ such that}$$

$$\sum_{e \in \delta(S)} x(e) + \sum_{T \supseteq S} z_T \geq 1 \quad \forall S \subseteq V' - \{r\}$$

$$\sum_{T \subseteq V' - \{r\}} z_T \leq 1$$

$$x_e, z_T \in \{0, 1\}\ \forall e \in E, T \subset V' - \{r\}$$

The linear relaxation (PCST-LP) of (PCST-IP) is obtained by replacing the constraints $x_e \in \{0, 1\}$ and $z_T \in \{0, 1\}$ by $x_e \in [0, 1]$ and $z_T \in [0, 1]$. The dual (PCST-D) of (PCST-LP) is:

$$\max \sum_{S \subset V' - \{r\}} y_S \text{ such that}$$

$$\sum_{S: e \in \delta(S)} y_S \leq w_e,\ e \in E$$

$$\sum_{S \subseteq T} y_S \leq \pi(T),\ T \subseteq V' - \{r\}$$

$$y_S \geq 0,\ S \subseteq V' - \{r\}$$

The GW algorithm maintains variables y_S in (PCST-D) for each cluster S; these are interpreted as the cluster's *potential*. The algorithm has two phases: Growth and Pruning. In the **Growth phase**, we maintain a set of clusters. If a cluster has non-zero potential, we say that it is *active*; otherwise, the cluster is *inactive*. Initially, all nodes are in active singleton clusters, except for the root, which is inactive. Let S_u denote the cluster containing node u. The algorithm maintains a quantity d_u for each node u, which captures the sum of all dual variables (i.e., potential) y_S, such that $u \in S$ over the past rounds. For any edge $e = (u, v)$, the algorithm will ensure that $d_u + d_v \leq w_e$. In a given round, the dual variables associated with all the active clusters grow at the same rate until one of the following events happen: (1) For some edge $e = (u, v)$ with S_u being active and S_v being the root component, we have $d_u + d_v = w_e$. In this case, we say that the edge (u, v) becomes *tight*, S_u is merged with the root cluster, and it becomes inactive. (2) For some edge $e = (u, v)$ with S_u and S_v being distinct active clusters, we have $d_u + d_v = w_e$. In this case, we say that the edge (u, v) becomes tight, and the clusters S_u and S_v are merged to form a new cluster; (3) For some cluster S, we have $\sum_{T \subseteq S} y_T = \pi(S)$. In this case the cluster S becomes *inactive*. The Growth phase ends when all the clusters are inactive, and the solution returned is the set of tight edges, F. In the **Pruning phase**, we find a tree $F' \subset F$ by discarding edges whose removal does not degrade the quality of the initial solution. (Goemans and Williamson 1997) show that this algorithm runs in $O(n^2 \log n)$ time.

Algorithm 2 FASTGW$(H, \mathbf{w}, \pi)$

Input: $(H, \mathbf{w}, \pi)$
Output: Tree T, a solution to PCST
Growing Phase:
Set $F = \emptyset$, $\mathcal{H} = \{v_c | v \in V\}$
while $\mathcal{H}$ is not empty **do**
  Set $S_c = \mathcal{H}.pop()$
  Set $i = S.pop()$
  if i is an edge **then**
    if $i = (S, S')$ is a terminal edge **then**
      MERGE(S, S')
      Add PARENT(i) to F
    else
      EDGESPLIT(i)
      Add S_c back to $\mathcal{H}$
    end if
  else
    DEACTIVATECOMPONENT(S)
  end if
end while
Pruning Phase
Set $F' =$ ALLROOTEDNW(F)
return F'

FASTGW algorithm We adapt the approach of (Cole et al. 2001) to develop a much faster algorithm for PCST (Algorithm 2). We describe the main ideas below.

1. **Growth Phase and Edge Splitting**: The main idea is to avoid having an edge whose two endpoints are both active clusters—this is referred to as *edge splitting* by (Cole et al. 2001). Initially, every edge (u, v) in the graph is "split" in half by adding an artificial node t between u and v; this effectively creates two edges, (u, t) and (t, v). In every Growth phase round, if two clusters are merged and, as a result, two active clusters become neighbors, the edge is split again ensuring that at most one of the endpoints is active. Of course, the edge cannot be split indefinitely; an edge that cannot be split is called a *terminal*. The user determines how many times an edge will be split via a parameter.
2. **Pruning phase**: This step takes an edge-weighted tree and finds the optimal Heaviest Subgraph by solving the net-worth problem on it optimally. Note that this step is different from (Cole et al. 2001).
3. During the Growth phase, we maintain a binomial heap $\mathcal{H}$ of active clusters. For every active component S, $\mathcal{H}$ contains a heap S_c of edges that have S as an endpoint. S_c also contains a reference to itself. In every Growth phase round, we pop a heap S_c from $\mathcal{H}$ and, subsequently, an element i from S_c. The element i corresponds to either an edge that has just become tight, or a cluster that now has zero potential. In the former case, if the edge is a terminal, its endpoints are merged (procedure MERGE); otherwise, the edge is split (EDGESPLIT). In the latter case, the component becomes deactivated, so we discard it (DEACTIVATECOMPONENTS).
4. In order to obtain the the desired running time, we implemented a system of *clocks*. The Growth phase of GW can be interpreted as a process in which dual variables grow as time passes. We maintain 1) a global clock W, 2) an internal clock for each cluster, S_W, and 3) an internal clock for each element in a heap i_W. A cluster clock S_W indicates the the time from the beginning of the algorithm at which S will either merge with some other component or be deactivated.

3.3 Putting it all together: improved algorithm for the HDS problem

We describe the main idea for FASTGW-HDS (Algorithm 3) below. It iterates over the intervals returned by the SELECTINTERVALS procedure. The second step involves converting G into an instance of PCST, H. We define a procedure called HS-TO-PCST that executes the instance transformation described in Section 2. For further improving the running time, we use the two upper bounds, UB_{SOP} and UB_{STR}, from (Bogdanov, Mongiovì, and Singh 2011), which allow us to discard low-scoring intervals without having to compute the PCST solution.

Algorithm 3 FASTGW-HDS: Find a solution to HDS

Input: Dynamic Network $G = (V, E, W)$
Output: Dynamic Subgraph $R = (G', [i, j])$
Set $z = 0$,$R = \emptyset$
Let $\mathcal{A} = A_1 \cup A_2 \cup A_3 =$ SELECTINTERVALS(T)
for subinterval $[a, b] \in A_1$ **do**
  if the graph for subinterval $[a, b]$, passes the upper-bound checks, $UB_{SOP}(G_{[a,b]}) \geq z$ and $UB_{STR}(G_{[a,b]}) \geq z$ **then**
    Set $(H, \mathbf{w}, \pi) =$ HS-TO-PCST(G)
    Set $score =$ FASTGW$(H, \mathbf{w}, \pi)$
    if this score is better than the current best **then**
      Let H' be the subgraph of H returned by FASTGW$(H, \mathbf{w}, \pi)$
      Set $z = score$
      Set $R = (G', [a, b])$, which is the temporal subgraph of $(G, [a, b])$ equivalent to H'
    end if
  end if
end for
Repeat above loop for each subinterval $[a, b] \in A_2 \cup A_3$
return R

Lemma 3. *Algorithm* FASTGW-HDS *runs in time* $O(T \log T(m + n) \log^2 n)$ *and requires space* $O(m)$.

Proof. (Sketch) The algorithm runs FASTGW for all the intervals in $\mathcal{A}$. For each such interval, the bounds UB_{SOP} and UB_{STR} are computed, and the HS instance is converted to the equivalent PCST instance, and back. All of these can be done in linear time. Since FASTGW takes $O((n+m) \log^2 n)$ time, the total time is $O(T \log T(m+n) \log^2 n)$. Finally, the algorithm only needs space for running the instance corresponding to one interval in $\mathcal{A}$, so the lemma follows. □

4 C-SAR: An algorithm for the SAR Problem

We describe Algorithm C-SAR (Component SAR) for the SAR problem. C-SAR has two main parts that we describe below (see (Cadena and Vullikanti 2017) for more details).
(1) *Candidate-generation step*. We generate the candidate subgraph set for an interval $[i, j]$ by running a modified version of FASTGW, which we call FORESTGW. This algorithm returns a collection of trees of the PCST instance $(H, \mathbf{w}, \pi)$ and their respective scores. The candidate subgraph set $\mathcal{C}_{i,j}$ is precisely this collection. Notice that we do not map the forest to the corresponding edges in G at this point, as there is no need to retrieve the original edges for a subgraph unless it gets selected as the next region to be output.
(2) *Update step*. The update step takes place in an iteration k after region $R_k = (G', [a, b])$ is selected to be output. We need to update the candidate subgraphs belonging to intervals that overlap with $[a, b]$, so that the positive edges of G' do not contribute to the scores of the subgraphs in these overlapping intervals in future iterations.

Algorithm 4 C-SAR Find all regions with anomaly score above a given threshold

Input: $(G, F, [t_1, t_2], \tau)$
Output: Set of regions $\mathcal{R} = \{R_1, R_2, \ldots, R_k\}$
Set $\mathcal{R} = \emptyset$
Set $\mathcal{I} =$ FORESTGW-HDS$(G, F, [t_1, t_2], \tau)$
Set $k = 1$
while there is a candidate interval $\mathcal{I}$ **do**
  Find the interval $[a, b]$ with subgraph of maximum score:
  Set $[a, b] = argmax_{[i,j]\in\mathcal{I}}\{\max_{G'\in\mathcal{C}_{i,j}}\{score(G')\}\}$
  Find the subgraph G^k in $[a, b]$ with maximum score:
  $G^k = argmax_{G'\in\mathcal{C}_{a,b}}\{score(G')\}$
  Implicitly remove G^k from $\mathcal{C}_{i,j}$
  Set $\mathcal{R} = \mathcal{R} \cup \{(G^k, [a, b])\}$
  for each sub-interval $[i, j] \in \mathcal{I}$ that overlaps with $[a, b]$ **do**
    Update edge weight function:
    for subgraph $H' \in \mathcal{C}_{i,j}$ **do**
      for node $v \in H'$ with prize π_v **do**
        compute new prize π'_v:
        if $\pi'_v \neq \pi_v$ **then**
          mark H' for update
        end if
      end for
      if H' is marked for update **then**
        Remove $\{H'\}$ from $\mathcal{C}_{i,j}$
        Set $new\_sub =$ FORESTGW$(H', \mathbf{w}', \mathbf{pi}')$
        Add the updated components in new_sub back $\mathcal{C}_{i,j}$:
        Set $\mathcal{C}_{i,j} = \mathcal{C}_{i,j} \cup \{g \in new\_sub | score(g) \geq \tau\}$
      end if
    end for
    Comment: Implicitly discard $[i, j]$ if $\mathcal{C}_{i,j} = \emptyset$
  end for
  $k \leftarrow k + 1$
end while

Table 2: Sizes of experimental networks

Dataset	Nodes	Edges	Timestamps	Resolution
Twitter Venezuela	2,645	17,018	182	1 day
Hospital	75	1,139	5,820	1 minute
Traffic	1,870	1,993	2,160	20 minutes
Wikipedia	992	1,408	723	1 day
AS-CAIDA	31,379	101,945	122	weekly to monthly

5 Experiments

5.1 Datasets and Setup

We evaluate our algorithms on five real networks (see Table 2): (i) a Twitter follower graph with communication evolving over 6 months, (ii) the highway network of Los Angeles County, California and its activity on May, 2014[1], (iii) a contact network of health-care workers and patients in a hospital ward during 4 days (Vanhems et al. 2013), (iv) a sample of Wikipedia page view statistics, and (v) an autonomous systems network. Additionally, we generated semi-synthetic data by randomly assigning edge weights with probability p to the Hospital, Twitter, and Wikipedia networks. For the results below, we use values of p of 0.01, 0.05, 0.10, 0.30. A detailed description of each dataset can be found in (Cadena and Vullikanti 2017).
Experimental setup We compare the quality of FASTGW to the Basic and *MEDEN* algorithms of (Bogdanov, Mongiovì, and Singh 2011) for HDS. Also, we compare C-SAR to the NetSpot algorithm of (Mongiovì et al. 2013), which is the state-of-the-art for the SAR problem.

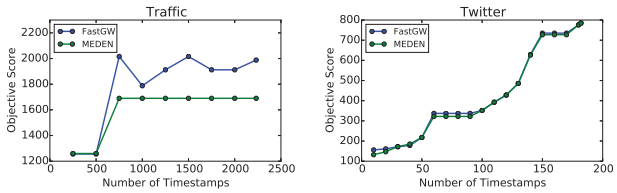
We focus on answering the following questions:
(1) **Quality:** What is the accuracy of FASTGW-HDS compared to *MEDEN*? How does TopDown compare to our algorithms for approximating the score of a heaviest subgraph on standard benchmarks?
(2) **Scalability:** How fast is FASTGW-HDS compared to (Bogdanov, Mongiovì, and Singh 2011), and how does it scale with the network size and number of intervals?
(3) **Detection Power:** Are the subgraphs identified by our algorithms relevant or anomalous in their respective domains?

5.2 Quality of FASTGW-HDS

We compare the objective value of the solution produced by our algorithms to the solution produced by *MEDEN* on real data. Figure 2 shows the score of the HDS obtained by both algorithms for the highway network and for Twitter. We compute heaviest subgraph for different lengths of time intervals up to the maximum number of intervals for each network. For Twitter, the two algorithms exhibit similar performance. However, as discussed in Section 5.5, our algorithms find qualitatively better results for this dataset. Results for the other datasets can be found in (Cadena and Vullikanti 2017).

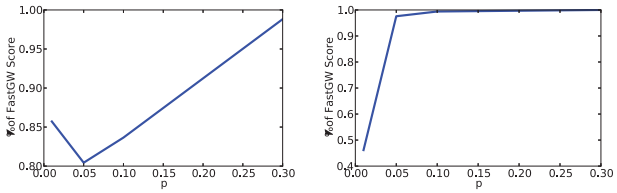
Figure 3 shows the score obtained by *MEDEN* relative to the FASTGW-HDS score as the number of positive edges in the network increases. We plot relative performance in this case because (i) FASTGW-HDS always has a higher score and (ii) the scores for different values of p are very different

[1] http://pems.dot.ca.gov/



(a) (b)

Figure 2: Objective score of FASTGW-HDS and *MEDEN* for (a) the Traffic dataset and (b) Twitter for a varying number of timestamps. Most of the time, the solution produced by FASTGW-HDS is at least as good as the *MEDEN* solution and significantly better for the Traffic dataset



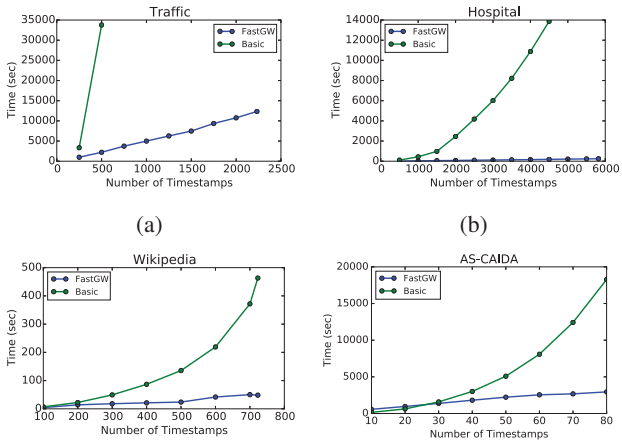
(a) (b)

Figure 3: *MEDEN* score relative to FASTGW-HDS as the positive edges of a network increase. Notice that the y-axes on both plots have different ranges. FASTGW-HDS outperforms *MEDEN* for instances with a low percentage of positive weights. As more edges become positive, the instance becomes easier to solve and both algorithms obtain the same scores.

making it difficult to appreciate the gap between the two algorithms on an absolute scale. We note that both algorithms achieve the same score across all values of p for the Hospital network, suggesting that this dataset is an easy instance to solve.

5.3 Scalability of FASTGW-HDS

Figure 4 shows the running times of FASTGW-HDS and Basic on the Traffic (4a), Hospital (4b), Wikipedia (4c) and AS (4d) datasets. We ran both algorithms varying the number of timestamps of the input up to the maximum number of timestamps for each network. We can see FASTGW-HDS is much faster than Basic for these datasets, even for a small number of intervals. The most notorious difference is the Hospital network, where our algorithm is up to four orders of magnitude faster, but we point out that this dataset is also the largest with respect to number of intervals. For Wikipedia, we see an improvement of one order of magnitude, and for the Traffic dataset, we could only run Basic with up to 500 time intervals. Given that *MEDEN* is approximately one order of magnitude faster than Basic (Bogdanov, Mongiovì, and Singh 2011), our results show evidence that FASTGW-HDS has running time competitive to or superior to that of *MEDEN*.



(a) (b)

(c) (d)

Figure 4: Running times (in seconds) of FASTGW-HDS and Basic for the (a) Traffic, (b) Hospital, (c) Wikipedia, and (d) AS datasets. FASTGW-HDS is much faster than Basic in all instances for up to four orders of magnitude.

Algorithms		Twitter	Hospital	Traffic	Wikipedia
C-SAR	**Avg.**	315.53	189.14	1,577.87	27.33
	Med.	269	198	1,539	18
	Count	72	14	31	9
	Range	[153, 786]	[114, 241]	[1,012, 2,505]	[15, 76]
NetSpot	**Avg.**	323.77	137.90	1,303.57	29.714
	Med.	293	134	1,262	18
	Count	56	10	7	7
	Range	[153, 722]	[112, 180]	[1,026, 1,657]	[15, 76]

Table 3: Quality statistics for C-SAR and NetSpot

5.4 Quality of C-SAR

Table 3 shows statistics of running C-SAR and NetSpot in our datasets. For every network, we set the threshold $\mathcal{T}$ low enough to capture subgraphs with a score at least half of the heaviest subgraph— sometimes much lower as in the case of Twitter, where $\mathcal{T}$ is 150 and the heaviest subgraph found has score 786. We note that the largest regions found by C-SAR for each dataset are larger than those found by NetSpot (equal for Wikipedia). The biggest improvement is on the traffic dataset, where our algorithm finds a region 50% larger than the NetSpot solution. Also, C-SAR finds a larger number of regions for the chosen NetSpot parameter. Average and median scores also show overall better quality for C-SAR, except for Twitter. An explanation for this is that the regions found by NetSpot span only one time interval for this dataset, whereas our algorithm covers almost half of the time intervals for the biggest region. This makes it possible for NetSpot to "pack" more intervals of higher scores in subsequent runs.

5.5 Detection power

For the **traffic** dataset, the HDS obtained by FASTGW-HDS corresponds to the stretch of highway I10-E between high-

ways I405 and I5, a road prone to congestion during peak traffic hours. The HDS occurs on Friday, May 9, from 15:00 to 18:20. For **Wikipedia**, at least three of the highest-scoring subgraphs found using C-SAR exactly match those found by NetSpot. Therefore, we can draw conclusions similar to the NetSpot authors regarding the relevance of the discovered patterns. For the rest of this section, we focus on analyzing our **Twitter** datasets. We explore different ways of studying Twitter follower graphs in terms of edge-evolving networks.

In order to model information diffusion, we make an assumption similar to the one in (Bogdanov, Mongiovì, and Singh 2011): if two neighboring users send similar tweets in the same timestamp, it is possible that one is influencing the other. Furthermore, we model anomalous activity in an edge as deviations from a Poisson distribution specific to that edge. Formally, let n_e^t be the number of tweets that pass through edge e at time t; we model n_e^t as a draw from a Poisson distribution with parameter λ_e. We take a Bayesian approach and consider λ_e to be drawn from a Gamma distribution with parameters α_e and β_e. These parameters are updated as we see new data every day. Finally, the weight of an edge is given by $-\log(P(n_e^t|n_e^{[1,t-1])})/\mu$, where $P(n_e^t|n_e^{[1,t-1]})$ is the posterior probability of n_e^t given the counts in the previous days, and $\mu = 0.05$ is a significance threshold. This weighing function was proposed by (Mongiovì et al. 2013); it has has the desired properties of being positive-increasing if the posterior probability is less than μ and negative-decreasing otherwise.

We ran FASTGW-HDS on the Twitter dataset of 182 days. The temporal subgraph obtained spans the time period from January 4, 2014 to March 31, 2014, which aligns with a period of large-scale national protests in the country. We note that our result is more interpretable than the subgraph obtained by *MEDEN*, which covers the entire 182 days. Figure 5 shows the number of civil-unrest events in Venezuela according to the GSR dataset (Ramakrishnan et al. 2014); this dataset is a compilation of events from major newspapers in Latin America processed manually by analysts and approved by political scientists who are experts in the region.

Using HDS in this setting we find the heaviest subgraph, but it may be interesting to also find other events of high score. In this case, the SAR problem is a natural fit. We ran NETSPOT (Mongiovì et al. 2013) (with $\tau = 0$) to find the 100 heaviest subgraphs. We find that the timeline of heavy subgraphs found for Venezuela aligns with the timeline of GSR events. In particular, most of the activity for both timelines occur in the period of January to March 2014. Figure 5 shows the weekly number of events for GSR superimposed with the weekly number of anomalous subgraphs. The y-axis is normalized by the sum of instances in the respective series.

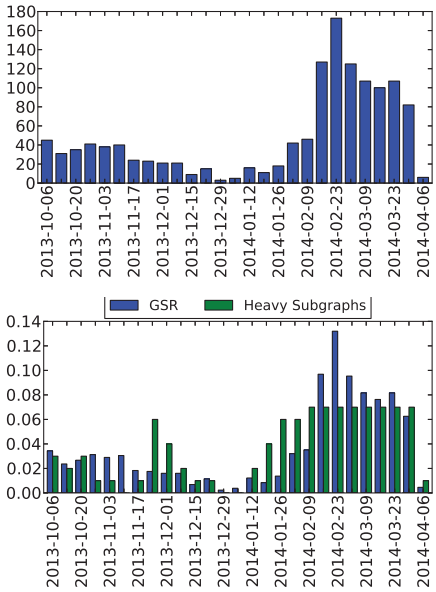


Figure 5: Top: Number of GSR protest events and heavy subgraphs per week in Venezuela. Bottom: Proportion of GSR events compared to the proportion of anomalous regions found by C-SAR each week

6 Related Work

Anomaly detection in dynamic networks has received a lot of attention in recent years, and has been used in a number of applications, including computer and social networks and water distribution systems e.g., (Ma et al. 2011; Zeidanloo and Manaf 2010; Akoglu and Faloutsos 2010). The survey by Akoglu et. al. (Akoglu, Tong, and Koutra 2014) gives a good introduction to the research on this topic. It classifies the different approaches on anomaly detection in dynamic graphs into the following broad classes, depending on the graph characteristic that is used and the kind of events that are detected. The first approach uses graph features, such as degree distribution, diameter and eigenvalues, and identifies anomalies based on changes in these features, e.g., (Kang et al. 2011; Shoubridge et al. 2002; Leskovec et al. 2008). A related approach examines changes in community structure, e.g., (Peel and Clauset 2014; Aggarwal, Zhao, and Yu 2011). The third approach uses matrix decomposition of time-evolving graphs, and formalizes anomalies in terms of eigenvectors and eigenvalues, e.g., (Sun et al. 2008; Shoubridge et al. 2002). In this paper, we focus on what is referred to as window-based approach. This attempts to formalize anomalous patterns in the graph within a time window, which is used for identifying anomalies in the entire stream (Bogdanov, Mongiovì, and Singh 2011; Mongiovì et al. 2013). Bogdanov et al. (Bogdanov, Mongiovì, and Singh 2011) show that the HS problem is NP-Hard even when the edge weights in $\{-1, 1\}$. Section 2 presents some discussion of their results and those of Mongiovì et al. (Mongiovì et al. 2013) for the SAR problem.

These problems are closely related to the generalized Steiner tree and PCST problems, which have been studied very extensively, e.g., (Goemans and Williamson 1997; Cole et al. 2001; Johnson, Minkoff, and Phillips 2000; Chudak, Roughgarden, and Williamson 2001; Bateni, Hajiaghayi, and Liaghat 2013), including node weights and other objectives. While efficient approximation algorithms are known for the PCST problem, the NetWorth objective

remains open. Bateni et al., (Bateni, Hajiaghayi, and Liaghat 2013) showed strong inapproximability results for the rooted version.

7 Conclusions

Mining temporal networks is a challenging problem with broad applicability. We present new algorithms for the HDS and HS problems. FastGW-HDS performs very well in practice, improving on prior approaches in terms of quality, with comparable running time. Our temporal pruning technique is also likely to be useful in other temporal network analysis problems. We develop a much faster algorithm for the PCST problem, which extends easily to other variants, as discussed in (Cadena and Vullikanti 2017).

Acknowledgements

This work was partially supported by the following grants: DTRA CNIMS Contracts HDTRA1-11-D-0016-0010, HDTRA1-17-D-0023, and NSF grants IIS-1633028, ACI-1443054.

References

Aggarwal, C.; Zhao, Y.; and Yu, P. 2011. Outlier detection in graph streams. In *ICDE*.

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

Akoglu, L.; Tong, H.; and Koutra, D. 2014. Graph based anomaly detection and description: a survey. *Data Mining and Knowledge Discovery*.

Bateni, M.; Hajiaghayi, M.; and Liaghat, V. 2013. Improved approximation algorithms for (budgeted) node-weighted steiner problems. In *ICALP*.

Bogdanov, P.; Mongiovì, M.; and Singh, A. 2011. Mining heavy subgraphs in time-evolving networks. In *ICDM*.

Cadena, J., and Vullikanti, A. 2017. Mining heavy temporal subgraphs: Fast algorithms and applications. https://tinyurl.com/ycgwqgbm.

Chudak, F.; Roughgarden, T.; and Williamson, D. 2001. Approximate k-msts and k-steiner trees via the primal-dual method and lagrangean relaxation. In *IPCO*.

Cole, R.; Hariharan, R.; Lewenstein, M.; and Porat, E. 2001. A faster implementation of the goemans-williamson clustering algorithm. In *ACM SODA*.

Goemans, M. X., and Williamson, D. P. 1997. The primal-dual method for approximation algorithms and its application to network design problems. *SIAM Journal of Computing*.

Johnson, D.; Minkoff, M.; and Phillips, S. 2000. The prize collecting steiner tree problem: Theory and practice. In *ACM SODA*.

Kang, U.; Papadimitriou, S.; Sun, J.; and Tong, H. 2011. Centralities in large networks: Algorithms and observations. In *SDM*.

Leskovec, J.; Backstrom, L.; Kumar, R.; and Tomkins, A. 2008. Microscopic evolution of social networks. In *Proc. of KDD*.

Ma, X.; Xiao, H.; Xie, S.; Li, Q.; Luo, Q.; and Tian, C. 2011. Continuous, online monitoring and analysis in large water distribution networks. In *ICDE*.

Mongiovì, M.; Bogdanov, P.; Ranca, R.; Singh, A.; Papalexakis, E.; and Faloutsos, C. 2013. Netspot: Spotting significant anomalous regions on dynamic networks. In *SDM*.

Peel, L., and Clauset, A. 2014. Detecting change points in the large-scale structure of evolving networks. CoRR, abs/1403.0989.

Ramakrishnan, N.; Butler, P.; Muthiah, S.; Self, N.; Khandpur, R.; Saraf, P.; Wang, W.; Cadena, J.; Vullikanti, A.; Korkmaz, G.; Kuhlman, C.; Marathe, A.; Zhao, L.; Hua, T.; Chen, F.; Lu, C. T.; Huang, B.; Srinivasan, A.; Trinh, K.; Getoor, L.; Katz, G.; Doyle, A.; Ackermann, C.; Zavorin, I.; Ford, J.; Summers, K.; Fayed, Y.; Arredondo, J.; Gupta, D.; and Mares, D. 2014. Beating the news with embers: Forecasting civil unrest using open source indicators. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '14, 1799–1808. New York, NY, USA: ACM.

Rozenshtein, P.; Anagnostopoulos, A.; Gionis, A.; and Tatti, N. 2014. Event detection in activity networks. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '14, 1176–1185. New York, NY, USA: ACM.

Sadeghi, A., and Frohlich, H. 2013. Steiner tree methods for optimal sub-network identification: an empirical study. *BMC Bioinformatics* 14.

Shoubridge, P.; Kraetzl, M.; Wallis, W.; and Bunke, H. 2002. Detection of abnormal change in a time series of graphs. *Journal of Interconnection Networks*.

Sun, J.; Xie, Y.; Zhang, H.; and Faloutsos, C. 2008. Less is more: Sparse graph mining with compact matrix decomposition. *Statistical Analysis and Data Mining*.

Vanhems, P.; Barrat, A.; Cattuto, C.; Pinton, J.-F.; Khanafer, N.; Régis, C.; Kim, B.-a.; Comte, B.; and Voirin, N. 2013. Estimating potential infection transmission routes in hospital wards using wearable proximity sensors. *PloS one* 8(9):e73970.

Wu, N.; Chen, F.; Li, J.; Zhou, B.; and Ramakrishnan, N. 2016. Efficient nonparametric subgraph detection using tree shaped priors. In *AAAI*.

Zeidanloo, H. R., and Manaf, A. B. A. 2010. Botnet detection by monitoring similar communication patterns. CoRR, abs/1004.1232.