NetSpot: Spotting Significant Anomalous Regions on Dynamic Networks

Misael Mongiovì* Petko Bogdanov* Razvan Ranca* Evangelos E. Papalexakis†
Christos Faloutsos† Ambuj K. Singh*

Abstract

How to spot and summarize anomalies in dynamic networks such as road networks, communication networks and social networks? An anomalous event, such as a traffic accident, a denial of service attack or a chemical spill, can affect several near-by edges and make them behave abnormally, over several consecutive time-ticks. We focus on spotting and summarizing such *significant* anomalous regions, spanning space (i.e. nearby edges), as well as time.

Our first contribution is the problem formulation, namely finding all such *Significant Anomalous Regions* (SAR). The next contribution is the design of novel algorithms: an expensive, exhaustive algorithm, as well as an efficient approximation, called NetSpot. Compared to the exhaustive algorithm, NetSpot is up to *one order of magnitude* faster in real data, while achieving less than 4% average relative error rate. In synthetic datasets, it is more than *30 times* faster and solves large problem instances that are otherwise infeasible. The final contribution is the validation on real data: we demonstrate the utility of NetSpot for inferring accidents on road networks and detecting patterns of anomalous access to subnetworks of Wikipedia. We also study NetSpot's scalability in large social, transportation and synthetic evolving networks, spanning in total *up to 50 million edges.*

1 Introduction

Given a road network with segments associated with their traffic at every time-tick, how can we identify the main unexpected congestions to report, say, to the highway patrol authorities? Given the Wikipedia pages/links network, annotated with the rate of page accesses, how can we spot subnetworks and time intervals with an unexpectedly high number of requests? We want to report the areas (i.e. set of adjacent, connected segments or links) as well as the time-intervals, that best summarize the extent of "anomaly" in our input.

Consider road networks (Fig. 1). A dynamic road network has a fixed graph structure with edges corresponding to road segments and nodes corresponding to road intersections. Edges are associated with values that model their state (average speed) over time. The network can be viewed as a sequence of isomorphic graphs (slices) for discrete time stamps (Fig. 1(a)). Within the above setting, anomalies due to unexpected events (e.g. traffic accidents, music festivals, road work) manifest as a localized abnormal behavior in both time and network structure (Fig. 1(b)). For example, an accident on a highway may induce lower average speeds along the same highway as well as intersecting roads, and this effect can persist through several time slices until the cause of abnormal behavior is removed. Within this scenario we set out to answer the following question: *how can we compute a comprehensive summary of all unusual traffic congestions and their time of occurrence, and report it to the police or to urban planners, by mining the full history of a large road network?*

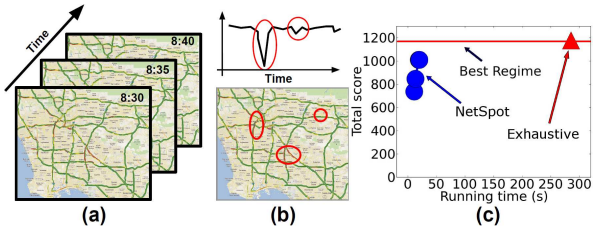
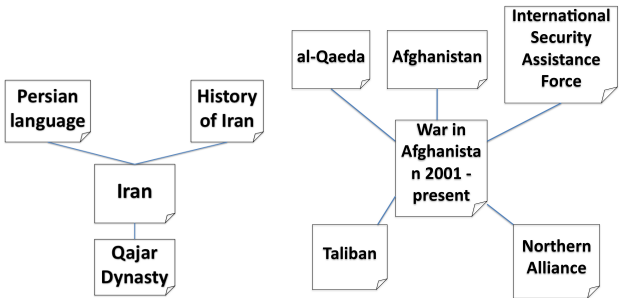


Figure 1: (a) A time-evolving road network. (b) Temporal anomalous regions with time and network extent. The combinatorial nature of subgraph anomalies renders employing exhaustive techniques inefficient even in small networks, while our approach NetSpot achieves high quality fast (c).



(a) Iran Elections (b) Afghanistan

Figure 2: Anomalous patterns of increased Wikipedia page views at the time of 2009 presidential elections in Iran 2(a), and clashes of Taliban militants and police in Afghanistan in late 2009 2(b).

In Wikipedia, an instance of a dynamic information network, we set out to detect abnormal levels of access to a subnetwork, hinting at external events which trigger significantly elevated information need. Figure 2 shows two of the anomalous network regions (patterns) which we identify in the daily views of interlinked Wikipedia

*Dept. of Computer Science. UC Santa Barbara
†Dept. of Computer Science. Carnegie Mellon University

pages. The first pattern (Fig. 2(a)) corresponds to the page of Iran and related pages about its history and language. The temporal span of this pattern is shortly after the controversial presidential elections of 2009, when the government cracked down on protesters using force. A possible explanation for this anomaly is that once the news from Iran are reflected in major media, more than expected Internet users access Wikipedia's subnetwork related to Iran in search of more information on the issue and its background. We identify another pattern in late 2009 (Fig. 2(b)) shortly after the start of the second term of president Karzai in Afghanistan, and coinciding with reports of "23 Taliban militants" being killed by foreign and Afghan forces in "southern and eastern Afghanistan" [1].

Besides traffic and information networks, anomalies that extend in time and network are abundant in other application domains. Significant increase in the rate of communication among a group of people in a company may correspond to a project delivery deadline. In computer networks, an anomalous region may correspond to nodes' coordination within a virus-infected botnet [31]. Similarly, abnormally low concentration of chlorine within a region of a water distribution network may indicate a contamination [19]. All these example domains fall within the same setting of (relatively) stable topology and dynamic attributes associated with network elements.

Most of the existing anomaly detection approaches focus on link and node behavior anomalies. Our goal is related, but complementary. We observe that unusual behavior often propagates along the network and persists in time. Such anomalies can be caused by diffusion-like phenomena, such as accident-induced congestions in transportation networks, contaminations in water networks or increased information exchanged among computer nodes within a botnet infected with a virus. An anomaly detection algorithm will assign 'anomaly-scores' to each edge on a graph (road network, in our example). This is exactly the input to *our* algorithm, that will summarize and report anomalous regions.

In the space and spatio-temporal domains, methods based on spatio-temporal scan statistics (STSS) have been proposed to detect anomalies [21, 22, 23, 30]. These methods aim to spot regions of the space that are anomalous, or may correspond to outbreaks. A simple adaptation of these methods to dynamic networks would require searching in the combined exponential space of possible subgraphs and quadratic space of time intervals, which would be prohibitive on most real-world network instances. Naive adaptations of existing methods for dynamic networks are also inefficient on large problem instances (see Exhaustive in Fig. 1(c)). In our experimental analysis, one such approach required almost four hours of evaluation on a traffic network with $6,000$ road segments evolving over one month.

Our contributions include:

- *Novelty*: We propose a novel problem formulation for detecting all Significant Anomalous network Regions (SAR) in time.
- *Scalability*: Our proposed algorithm NetSpot for SAR *scales linearly* to large network instances and outperforms the exhaustive counterpart by more than *10 times*.
- *Quality*: NetSpot produces *high quality* results, often matching the results of an exhaustive (but very slow) solution.
- *Real world relevance*: We run NetSpot on a large traffic network and demonstrate its ability to spot unreported traffic accidents. We also show the ability of NetSpot to discover interesting events by analyzing the access rate to Wikipedia pages.

2 Preliminaries

In this section, we provide a formal definition of anomalous regions in weighted time-evolving networks. We also introduce some problems and properties that are relevant to our method.

Problem 1. *An edge-weighted dynamic network $G = (V, E, W)$ (hereinafter simply dynamic network) is an undirected connected graph where V is the set of vertices, E is the set of edges, and $W = \{w^1, w^2, \ldots, w^T\}$ is a family of weight functions of the kind $w^t : E \to \mathcal{R}$ that associate each edge $e \in E$ with an anomaly score. Each function w^t corresponds to a discrete timestamp t.*[1]

The weights of edges quantify their time-dependent level of anomaly. Our approach can be applied on the output of any existing anomaly detection method for general time series data. In this work, we use a statistical measure based on *p-value* (details in Sec. 4). A high positive weight means high anomaly level, while a negative weight corresponds to normal behavior. Our goal is to find contiguous regions, and hence we allow adjacent anomalous edges (in either time or network neighborhood) to combine and form larger regions of higher score. We aggregate the participating edges' weights to quantify the level of anomaly for a region.

Definition 2.1. *A* temporal network region *(hereinafter simply* region*) in a dynamic network $G = (V, E, W)$ is defined as a pair $R = (G', [i, j])$, where $G' = (V', E')$ is a connected subgraph of G and $[i, j]$ is a sub-interval of $[1, T]$ (i.e. $1 \le i \le j \le T$). The score (strength of anomaly) of a region is given by the*

[1] A similar definition is possible for dynamic networks with node weights. It can be shown that the two settings are equivalent (see also [10]).

aggregated anomaly score of its edges: $score_G(R) = \sum_{e \in E'} \sum_{t=i}^{j} w^t(e)$

Our goal is to construct a comprehensive set of anomalous regions occurring in possibly different network locations and time periods. A special case of this problem is the problem of finding the single highest score region. This is an NP-hard problem known as *Heaviest Dynamic Subgraph (HDS)* [7]. Some special cases of HDS are discussed in the Supplemental material [33]. Here we bring attention to two special cases: (i) the *Maximum Score Subsequence* (MSS), which calls for finding the contiguous subsequence that maximizes its score, and (ii) the *Heaviest Subgraph* (HS) [7], which calls for finding the connected subgraph such that the sum of its edge weights is the highest.

3 Problem definition

Our goal is to construct a comprehensive summary of all significant anomalous regions within a dynamic network for consumption by domain experts such as the police, spam protection analysts or water distribution network planners. To discover anomalous regions, we need to (i) characterize the average behavior of network edges, (ii) score edges in time according to how unusual their behavior is with respect to the average and (iii) define an algorithm that can compute extended regions of anomalous edges.

3.1 Anomalous score of a single edge The original edge weight reflects a quantity of interest in the specific domain, for example, the average speed in road networks, the number of transmitted packets in computer networks or the number of exchanged emails in social networks. Given an edge and its weight at a given time, we measure the significance of observing this weight as its statistical p-value, according to the empirical distribution of weights on the edge. The p-value is computed as the fraction of timestamps in which an equal or higher weight is observed on the same edge. The set of considered observations can be extended to the whole time horizon or limited to time periods that are expected to have similar characteristics (e.g. the same day of the week, the same month of the year). The lower the p-value of an observed score, the more anomalous the observation.

We denote the p-value of an edge e at time-stamp t with $p^t(e)$. Edges are weighted by comparing their p-value to a significance level threshold μ (typically 0.01). We compute the negative logarithm of the fraction of the p-value and the significance threshold. Extracting the logarithm allows us to sum up the weights when computing the significance of a region. Specifically $w^t(e) = -\log(p^t(e)/\mu)$. In this log-odds scoring scheme, a positive score corresponds to a p-value lower than μ and hence highly unexpected behavior, and vice versa.

3.2 Significant Anomalous Regions (SAR) Next, we define the problem of detecting all *Significant Anomalous Regions (*SAR*)* in a dynamic network. Our problem formulation considers a single-region score threshold $\mathcal{T}$. This parameter can be determined by dataset-specific score significance analysis, as we discuss in the Supplemental material [33]. An equivalent alternative is to fix k and report the top-k patterns.

DEFINITION 3.1. Significant Anomalous Regions (SAR)*: given a dynamic network* $G = (V, E, W)$ *and a threshold* $\mathcal{T}$*, find an ordered set of regions* $R_1, R_2, \ldots R_k$*, in decreasing order by score, such that the score of region* R_i *(defined as in Def. 2.1), computed without considering the contribute of positive edges overlapping with higher score regions, is not below* $\mathcal{T}$*.*

Our definition establishes an order of regions, specified by the index i. Higher index regions have lower scores than their predecessors and their score does not include contribution from edges in regions with lower index. This reduces the overlap in the resulting set. Further details are given in the Supplemental material [33].

SAR is NP-hard since it generalizes HDS (finding R_1 is equivalent to HDS). SAR can be solved naively by iteratively computing HDS and erasing the scores of the newly found region. More precisely, after a region R_1 is discovered from a graph $G = (V, E, W)$, a new graph G_1 is generated from G as follows: $G_1 = (V, E, W_1)$ has the same structure as G and its edge weights are obtained from those in G by erasing all positive edge weights contained in region R_1. More precisely, $w_1^t(e) = 0$ if $w^t(e) > 0$ and $e \in R_1$, and $w_1^t(e) = w^t(e)$ otherwise. The procedure is repeated iteratively. We call this approach Exhaustive. The attractiveness of such a naive extension is diminished by its inherent inefficiency. The main reason is that it requires scanning the network multiple times. Instead, we resort to a scalable and accurate solution for SAR based on a very large-scale neighborhood search approach.

4 Proposed Method

Our solution for SAR is based on an efficient *very large-scale neighborhood search* approach for approximating HDS in a dynamic network. Like other local search approaches, the quality of the final solution is highly dependent on its initial solution, because of the possibility to get stuck in a local maximum. However, our approach considers a large set of neighboring solutions at each step, and hence it is more likely to overcome local maxima compared to standard local search approaches [3]. Moreover, instead of starting the search from a random point, we propose an effective heuristic for generating initial solutions (seeds) which tend to converge to global optima.

The core of our algorithm is the procedure NETAMOEBA (Alg. 4.1) which approximates HDS. It alter-

nates between optimizing in the graph space and optimizing in the time domain via the following two steps:

1. *compute_max_score_subsequence*, which considers a fixed subgraph and optimizes the time interval that produces the highest score.

2. *compute_heaviest_subgraph*, which considers a fixed time interval and finds the best subgraph in this interval by using the TopDown heuristic (see Sect. 2) for HS.

ALGORITHM 4.1. NETAMOEBA: starting from a seed, find a near optimal *single* region

Require: Dynamic network $G = (V, E, W)$
Require: Seed $(G_{seed}, [t, t])$
Output: Temporal network region $R = (G', [i, j])$
  $R_{prev} \leftarrow (G_{seed}, [t, t])$
  $[i, j] \leftarrow compute\_max\_score\_subsequence(G_{seed})$
  $G' \leftarrow compute\_heaviest\_subgraph([i, j])$
  while $s(R_{prev}) \leq s((G', [i, j]))$ **do**
    $R_{prev} \leftarrow (G', [i, j])$
    $[i, j] \leftarrow compute\_max\_score\_subsequence(G')$
    $G' \leftarrow compute\_heaviest\_subgraph([i, j])$
  end while
  return R_{prev}

Our overall algorithm (Alg. 4.2) takes as input a score threshold $\mathcal{T}$ and a parameter h (number of failures before stopping, typically 10), and returns a set of anomalous regions whose score exceeds $\mathcal{T}$. The algorithm executes NETAMOEBA (Alg. 4.1) iteratively and uses a seed generation procedure to initialize the search. Next, it erases from the network the positive weights of edges that are within the newly found region. The algorithm stops when the last h discovered regions have score lower than $\mathcal{T}$. The idea is that if a region with score higher than $\mathcal{T}$ is not found after h consecutive times, then it is unlikely that such a region can be found later on. Higher values of h produce better quality, while lower values exhibit higher efficiency.

ALGORITHM 4.2. NETSPOT: iteratively find all regions with anomaly score above a given threshold $\mathcal{T}$.

Require: Dynamic network $G_0 = (V, E, W)$
Require: Score threshold $\mathcal{T}$
Require: Stopping condition h (# failures, normally 10)
Output: Set of regions $\mathcal{R} = \{R_1, R_2, \ldots R_k\}$
  $\mathcal{R} = \phi$
  $i \leftarrow 0$
  repeat
    $S \leftarrow generate\_seed(G_i)$
    $R_i \leftarrow$ NETAMOEBA(G_i, S)
    $\mathcal{R} = \mathcal{R} \cup \{R_i\}$ if $score_{G_i}(R_i) \geq \mathcal{T}$
    $G_{i+1} \leftarrow erase(G_i, R_i)$
    $i \leftarrow i + 1$
  until $score_{G_i}(R_i) \leq \mathcal{T}$ for h consecutive times
  return $\mathcal{R}$

The parameter h in Alg. 4.2 is needed as the score of consecutively found regions is not guaranteed to decrease monotonically (as for Exhaustive). If, however, the scores of found patterns is close to monotonic (i.e. high score patterns are found first) the accuracy of NETSPOT will be significantly higher. In order to maintain the order close to monotonic, we develop an effective seed generation strategy.

4.1 Seed generation Although there are a number of candidate seed generation strategies (random, maximum edge, matrix factorization), none of them lead to high quality results (see Supplemental material [33] and Sect. 5). Instead we resort to a novel approach, namely **Heaviest Subgraph, Maximum Subsequence (HSMS)**, which captures locality both in time and in the graph. At each step, HSMS selects the edge/timestamp e/t that maximizes the product of the heaviest subgraph score that contains e in slice t and the maximum subsequence score that contains timestamp t in the sequence of weights of edge e. The seed is then generated by considering the approximated heaviest subgraph that contains edge e in slice t. Compared to the previous strategies, HSMS is more likely to discover a seed contained in a large anomalous region as it analyzes both time and network dimensions. As we will see in the experimental section, the HSMS strategy is robust in selecting a good seed and hence improves the overall performance of NETSPOT by reducing the number of steps. However, it introduces a computational challenge as it requires computing (i) the heaviest subgraph and (ii) the maximum subsequence for every edge in time. If approached naively, this method introduces significant performance overhead and possibly worsen the overall running time. In what follows, we present a novel linear time algorithm for computing HSMS.

HSMS requires solving the following two subproblems:

- *All rooted HS*: for every edge e of a graph, find the Heaviest Subgraph that contains e. We refer to e as the root edge.

- *All rooted MSS*: given a sequence of real values, for every element t, find the Maximum Score Subsequence that contains t.

All rooted MSS is a special case of *All rooted HS*, where the graph is a simple path. Therefore we will discuss only *All rooted HS*. The results can be update incrementally, as discussed in the Supplemental material [33], thus avoiding re-running the whole process at every iteraction.

4.2 All rooted HS Given an edge e, the rooted Heaviest Subgraph calls for finding the heaviest subgraph that contains e. This variant of HS can be approximated in linear time by the same algorithm for HS

discussed in [7]. Unfortunately, this approach is inefficient since Rooted HS needs to be computed for every edge in the graph, and hence the overall complexity would be quadratic.

We propose a novel algorithm for computing All rooted HS for every edge in a tree in linear time. We extend it on graphs by computing the maximum spanning tree and computing All rooted HS on the resulting tree. To reduce the error, the weight of positive edges that do not belong to the spanning tree is added to adjacent positive edges that belong to the spanning tree (we can show that this is always a feasible operation). Given a tree G and an edge $(u, v) \in G$, we introduce the following quantities:

- bidirectional score $s_{\leftrightarrow}(u, v)$: the score of the HS rooted on edge (u, v);
- directional right score $s_{\rightarrow}(u, v)$: the score of the HS rooted on node v after removing all edges incident to v except (u, v);
- directional left score $s_{\leftarrow}(u, v)$: the score of the HS rooted on node u after removing all edges incident to u except (u, v).

Informally, $s_{\rightarrow}(u, v)$ is the part of score that can be propagated from edge (u, v) to node v. If $s_{\rightarrow}(u, v)$ is negative, edge (u, v) does not participate in the solution rooted in v. Note that $s_{\rightarrow}(u, v) = s_{\leftarrow}(v, u)$. We denote the weight of (u, v) as $w(u, v)$.

The relationship among the above scores is stated by the following lemma (proof skipped for brevity):

LEMMA 4.1. *Given a tree $G = (V, E, W)$, the following relation holds:*

$$s_{\leftrightarrow}(u, v) = s_{\rightarrow}(u, v) + s_{\rightarrow}(v, u) - w(u, v) \quad (4.1)$$

The following lemma (proof skipped for brevity) gives a recurrence for $s_{\rightarrow}(u, v)$ that allows us to compute this quantity for every edges in a tree. Combined with Lemma 4.1, it suggests a linear time algorithm for All rooted HS.

LEMMA 4.2. *Let G be a tree and (v, u) be an edge in G. The following relation holds:*

$$s_{\rightarrow}(u, v) = \sum_{x \in N(u) \setminus \{v\}} \max(0, s_{\rightarrow}(x, u)) + w(u, v) \quad (4.2)$$

The complete algorithm proceeds as follows: first the maximum spanning tree is computed and a root is picked arbitrarily. In order to preserve the score, the weight of positive edges that do not belong to the spanning tree is assigned to one of the adjacent positive edges. Next, the algorithm computes the quantities above by performing aggregations in bottom-up and then top-down direction on the tree. During the bottom-up aggregation, scores $s_{\leftarrow}$ are propagated from the leaves to the root by using Eq. 4.2. Next, the opposite directional scores $s_{\rightarrow}$ are propagated from the root to the leaves and the final score $s_{\leftrightarrow}$ is computed by using Eq. 4.1. An example is given in Fig. 3. One can show that this procedure computes the scores correctly on trees, and explores each edge exactly twice. Its running time is linear in the number of edges.

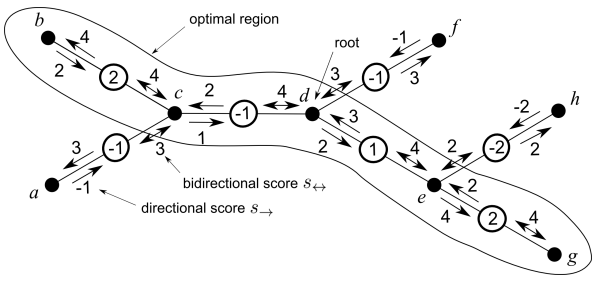


Figure 3: An example of computing All rooted HS on a tree. The score of the HS rooted at each edge is reported as the bidirectional score. The quantity $s_{\leftrightarrow}(u, v)$ is computed as a function of the directional scores by Eq. 4.1. The directional scores are propagated from the leaves to the root and then vice-versa. For example, if d is chosen as a root, $s_{\rightarrow}(c, d) = \max(0, s_{\rightarrow}(a, c)) + \max(0, s_{\rightarrow}(b, c)) + w(c, d) = 0 + 2 - 1 = 1$. Scores are computed in the following order: $s_{\leftarrow}(c, a)$, $s_{\leftarrow}(c, b)$, $s_{\leftarrow}(d, c)$, $s_{\leftarrow}(e, g)$, $s_{\leftarrow}(e, h)$, $s_{\leftarrow}(d, e)$, $s_{\leftarrow}(d, f)$, $s_{\rightarrow}(d, c)$, $s_{\rightarrow}(c, a)$, $s_{\rightarrow}(c, b)$, $s_{\rightarrow}(d, e)$, $s_{\rightarrow}(e, g)$, $s_{\rightarrow}(e, h)$, $s_{\rightarrow}(d, f)$

THEOREM 4.1. *Given a tree G, the described algorithm computes the exact scores of the HSs rooted in every edge (bidirectional scores) with time complexity $O(|E|)$.*

5 Experiments

5.1 Implementation We implement all discussed algorithms and perform the evaluation on a Linux server with processor Intel Xeon 2.0 GHz 4MB cache (only one processor used) and 98 GB RAM. To assess accuracy and scalability, we compare our method NETSPOT described in Sect. 4 with the Exhaustive approach described in Sect. 3.2.

Our two variations of Exhaustive use the MEDEN filter-and-verify framework [7] for reducing HDS to multiple application of HS. The first version uses the Top-Down heuristic (defined in [7], see Supplemental material [33]) for solving HS, while the second version uses ILP, and hence achieves the optimal. We also implement our very large-scale neighborhood search approach (Sect. 4) with two alternative seed generation strategies, namely *VLNS-Rand* (pick an edge at random) and *VLNS-Max* (pick the edge with maximum weight). Further details on these alternative strategies are given in the Supplemental material [33]. Our NETSPOT implementation uses the very large-scale neighborhood search

Table 1: Sizes of the experimental networks

Dataset	Nodes	Edges	Slices	Slice length
Traffic small	100	128	8640	5 min
Traffic	1923	6208	8640	5 min
Wikipedia	5000	1944	731	1 day
Enron	1598	6244	925	1 day
Synthetic	500	1000	8000	

approach (Sect. 4) and the Heaviest Subgraph Maximum Subsequence (HSMS) seed generation strategy (Sect. 4.1). Unless differently specified, the parameter h (number of failures) used in the following experiments is 10.

5.2 Datasets We evaluate NetSpot on three real-world dynamic networks: (i) a small and large highway transportation networks from Los Angeles, California evolving during the month of April 2011[2], (ii) the Enron email dataset[3] and (iii) a sample of Wikipedia. We also use synthetic networks to evaluate both scalability and accuracy of our method. Table 1 lists the sizes of all datasets used for evaluation. Note that considering all slices, the largest network (Traffic) contains in total 53 million of edges. Further details on the employed datasets are given in the Supplemental material [33].

5.3 Results Our experimental analysis aims to answer the following questions:

- Scalability: How fast is NetSpot compared to Exhaustive? how does it perform when the data size increases?
- Quality: What is the accuracy of NetSpot with respect to the slow Exhaustive approach?
- Real world relevance: Is NetSpot able to spot interesting regions? Is it able to infer unreported accidents in road networks better than naive approaches? Can it discover interesting events by analyzing accesses to Wikipedia?

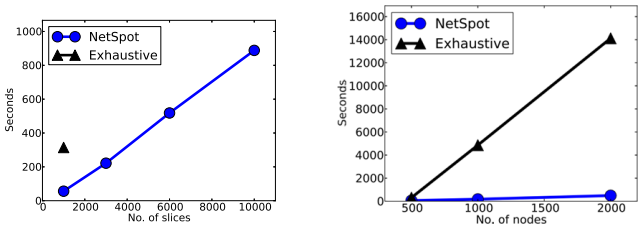
Scalability. Table 2 reports the running time of NetSpot in comparison with Exhaustive, VLNS-Max and VLNS-Rand on various datasets. NetSpot outperforms Exhaustive by more than one order of magnitude in all datasets except Wikipedia. Since the number of slices of Wikipedia is small, this dataset is "easy" to analyze for Exhaustive, therefore the gain of our method is less pronounced. VLNS-Max and VLNS-Rand are faster than NetSpot since they spend little effort in seed generation. However they perform poorly, as we discuss below (Fig. 5).

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

[3] http://www.cs.cmu.edu/~enron/

Table 2: NetSpot is more than one order of magnitude faster than Exhaustive on long datasets. Running times in seconds.

Dataset	NetSpot	Exhaustive	VLNS-Max	VLNS-Rand
Traffic small	14.6	196.1	7.9	0.2
Traffic	706.5	11271.4	443.8	11.8
Enron	122.8	1778.1	179.0	5.9
Wikipedia	386.0	931.0	134.0	10.7



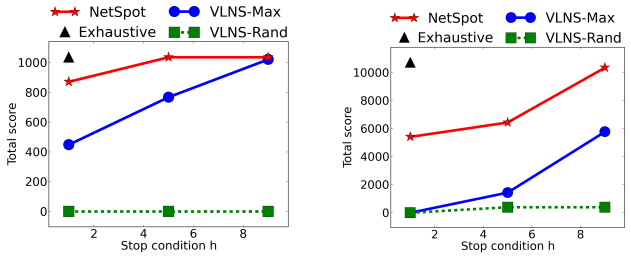
(a) **Scalab. in #slices** (b) **Scalab. in #nodes**

Figure 4: NetSpot scales linearly in the number of (a) time slices and (b) edges. In contrast Exhaustive was not able to complete in 10 hours on size 3,000 slices. The parameter h of NetSpot is set to 10.

Next, we assess the scalability of our approach in both size of the underlying graph and length of the time interval on synthetic datasets. For scalability in time length, we increase the number of slices from $1,000$ to $10,000$, while keeping the size of the graph fixed to 500 nodes, and report the running time. Fig. 4(a) shows a comparison of the running time of NetSpot and Exhaustive. Our algorithm's running time increases linearly with the size of the problem instance in time, while Exhaustive increases super linearly. Indeed Exhaustive was not able to complete in 10 hours on a dataset of 3,000 slices. The reason is that Exhaustive performs an expensive bounding at every iteration to get to the best next pattern, while we rely on our effective HSMS seed generation, combined with our large-scale neighborhood search approach.

In the scalability experiments for graph size we vary the number of nodes from 500 to $2,000$ in a synthetically generated dataset with $1,000$ slices. Results of this comparison are presented in Fig. 4(b). NetSpot scales much better than Exhaustive, and performs 30 times faster in the largest dataset.

Quality We evaluate the accuracy of NetSpot at varying the stop condition parameter h (number of failures) in comparison to Exhaustive, VLNS-Max and VLNS-Rand. Results on Traffic are presented in Fig. 5 (on Enron and Wikipedia we obtain similar results, see Suppl. material [33]). NetSpot consistently produces high quality regions, achieving more than 96% relative



(a) Traffic small, $\mathcal{T}$=10, μ=0.002

(b) Traffic, $\mathcal{T}$=30, μ=0.002

Figure 5: Quality of our algorithm, compared to Exhaustive on Traffic. The HSMS seed generation (NetSpot), combined with our NetAmoeba procedure, produces good quality regions

quality with respect to Exhaustive on real networks. At the same time, NetSpot is one order of magnitude faster than Exhaustive, as we discussed above.

The other seed generation strategies are more efficient (see Table 2), but they perform poorly in obtaining a high score solution. For example, in the traffic dataset, the random seed generation (VLNS–Rand) is able to find only a few regions before it terminates. This can be explained by the relatively small number of positive edges in this dataset, and hence the smaller chance that a randomly chosen edge is contained in a good region. The maximal-edge seed generation (VLNS–Max) is more consistent in the quality of obtained regions. However, NetSpot significantly outperforms it, reaching a good quality even for small values of h. For example, although VLNS–Max converges to a quality close to NetSpot on Traffic small, it reaches its peak quality at $h = 9$, while NetSpot is close to its peak score at $h = 1$. This difference is an evidence of higher stability of the NetSpot's seed generation procedure in generating good seeds at the beginning of the evaluation.

We also performed an evaluation using an optimal algorithm that uses an exact ILP solution for the HS problem on Traffic small, as opposed to approximating it using a heuristic (not reported). The optimal solution has a similar score, but it takes twice as much time as Exhaustive. The solution found by NetSpot on this dataset is within 0.1% error from Exhaustive. A similar evaluation of the bigger datasets did not terminate in reasonable time.

Real world relevance Apart from score-based quality of the patterns, we are also interested in the ability of the proposed framework to infer the existence of unexpected events. To this end, we report anomalous patterns in Wikipedia and also measure the ability of NetSpot to infer accidents in transportation traffic using accident reports as ground truth.

We apply NetSpot on the Wikipedia daily number of views network and discover patterns of varying size and shapes that all reflect real world events. Such patterns give insight into the factual information seeking process as a result of major news on a given subject. Table 3 lists some of the top patterns in the Wikipedia network. The pattern of highest score involves 37 articles on airplane models, airlines and accidents involving commercial airlines. This pattern coincides with the tragic event of an Air France flight crash. Pattern $2, 3, 5, 7, 9$ are all related to football with the highest score one coinciding with the day of the draw of groups in the 2009 UEFA Champions league. The fourth pattern involves articles on various counties in the US and occurs a day after the US presidential elections. Pattern 6 is on the Iran elections (Fig. 2(a)), while pattern 8 is the Afghanistan pattern from Fig. 2(b). Finally pattern 10 is related to a region in the Philippines that attracted media attention in 2009 for pre-election violence.

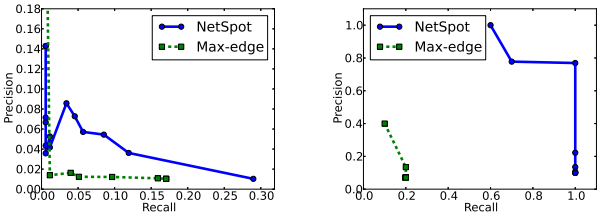
Note that some of the patterns found in Wikipedia are short-lived and have big-network-span $(1, 2, 3, 4, 6, 7, 9)$, while others affect smaller portion of the network, but extend in a longer time period $(5, 8, 10)$. If we use anomaly detection methods based on single edge/node analysis, we will not discover the full range of patterns and many of the events reported in Table 3 will be missed. Instead, NetSpot successfully discovers patterns of different shapes and elucidates the information foraging process of Internet users as an outcome of major news events. In the specific example of Wikipedia, one can use the output of NetSpot to identify articles that are vulnerable to false-information attacks (these are the articles that participate in the reported anomalous regions). In a more general setting, applying NetSpot on other information and social networks will allow for real-time detection of anomalous subnetworks that may correspond to bursty information spread, abnormal information demand on specific topics and unexpected communication patterns among users of an online social network.

Next, we compare precision and recall of NetSpot for inferring car accidents in the Traffic Dataset. Every reported accident falling within 4 hops and 30 minutes before a discovered pattern is considered as detected, while accidents falling outside of any region are considered as "false negatives". We execute our method for different values of the score threshold $\mathcal{T}$ and report a precision-recall curve. For this experiment, up to $10,000$ regions are considered. Results are shown in Fig. 6(a) in comparison with a naive approach that chooses the top-k edges (with k up to $10,000$) with lowest p-value (namely Max-edge). As forNetSpot, an accident is considered detected if it is within 4 hops and 30 minutes backwards from one of the top-k edges.

On real traffic data, NetSpot achieves almost always better precision than Max-edge in correspondence to the same level of recall, with up to 4 times increase in precision. At very low recall, the precision of Max-edge is higher than NetSpot (0.34 vs. 0.14). This indi-

Table 3: Top patterns discovered in the Wikipedia network, based on unexpected number of daily views

#	Duration	Size	Articles	Coinciding events
1	2-3 Jun 2009	37	Boeing 777, Lufthansa, AirFrance, Aeroflot, AirIndia, ...	Air France Flight 447 crashes.[4]
2	28 Aug 2009	42	FC Bayern Munchen, Juventus FC, 2009-10 UEFA Champions League, PFC Levski Sofia, FC Basel, ...	One day after the draw for group stage of UEFA Champ. League (08/27/2009)
4	5 Nov 2008	19	Race and Ethnicity in the United States Census, Blount County Alabama, Hardin County Kentucky, ...	One day after the US Presidential Elections on 11/4/2008
10	14-16 Nov 2009	4	Autonomous Region in Muslim Mindanao, Lakas–Christian Muslim Democrats, Lanao del Sur, Maguindanao	A Philippines region in which elections tension leads to the Maguindanao massacre.[5]



(a) Traffic (b) Synthetic

Figure 6: NetSpot outperforms significantly a single-edge-based method in spotting accidents. A precision-recall curve is shown for both Traffic and Synthetic. On real data (a), since congestions can be caused by many factors beside accidents, and only a few percent of accidents cause congestions, the absolute precision and recall values are limited. However NetSpot clearly outperforms a single-edge-based approach.

cates that edges with very low p-value are good markers in detecting major events. However, as soon as the p-value threshold increases, the precision of Max-edge drops drastically, while NetSpot maintains a significantly higher precision. We do not report the results for Exhaustive, since they are very similar to the ones reported by NetSpot.

The absolute values of precision and recall are relatively low since not all accidents reported by the highway patrol cause significant average speed reduction, and unexpected congestions can be caused by other events such as big concerts, sport events and road constructions. However, the significant increase in performance of NetSpot with respect to a single-edge-based approach, demonstrates the effectiveness of NetSpot in spotting interesting anomalous regions.

In addition, we evaluate our algorithm for its ability in spotting anomalous regions on the synthetic dataset. Fig. 6(b) shows the precision-recall curve for NetSpot and Max-edge on this dataset. NetSpot performs optimal precision at 60% recall. At higher recall, the precisimilsion reduces slightly due to the noise. In contrast, Max-edge is very sensitive to noise and performs less than 40% of precision at 10% recall and less then 10% precision at 20% recall.

6 Related Work

Most anomaly detection algorithms are complementary to our work, in the sense that their output (list of abnormal edges/nodes) can be used as input for NetSpot. This includes algorithms to detect unusual behavior in social, email and phone call networks [8, 12, 13], computer network traffic [14, 17, 32], smart grid sensor data [6] and water distribution networks [19]. Most of these approaches focus on link and node behavior anomalies [2, 5, 18, 27, 28, 9].

In the realm of static networks, Noble and Cook [24] introduce the concept of structural anomaly detection. Within this framework, a subgraph is considered anomalous if it is infrequent or parts of it are rarely repeated in the analyzed network. Jia et al. [16] introduce a framework for mining interesting or anomalous patterns and subgraphs out of noisy and distorted graphs. Eberle et al. [11] consider a substructure as anomalous if it deviates from a "normative" substructure, discovered by compression, based on the MDL principle. Later, Wang et al. [29] focus on the problem of finding the top-k most dissimilar subgraphs of fixed-size within a network. Besides being based on static networks, the above methods consider the degree of "coherence" of a subgraph structure with the rest of the network. In contrast, our definition renders a region anomalous if its dynamic behavior deviates from a norm.

In dynamic networks, the focus is to spot anomalous nodes or edges [2, 5], or to monitor global network parameters [15, 4]. While detecting anomalous nodes and edges is complementary to our method, global approaches are often non sensitive enough in detecting anomalies that involve small parts of the network. Recently, Chen et al. [8] proposed a method for anomalous community evolution discovery, which considers six possible types of community-dynamics anomalies: grown, shrunk, merged, split, born and vanished. In contrast, we aim to find regions in which the anomalous behavior persists in space (connected subnetworks) and time. In [26], Rossi et al. introduce a fully automated, parameter-free tool for identifying, representing and tracking the dynamics of roles within a network, as they evolve over time. Instead, we focus our search at the level of significant connected subnetworks.

Spatial scan statistics (SSS) and spatio-temporal scan statistics (STSS) methods [21, 22, 23, 30] are also

conceptually related to our formulation. They aim to spot and summarize anomalies in spatio-temporal domains. Extensions to dynamic networks [25, 20] are limited to detecting regions of predefined shapes such as disks and paths. Priebe et al. [25] compute anomalous regions by aggregating edge values in Enron, while restricting the region shapes to "disks" (neighborhood of order k). Although time is explicitly considered, slices of the network are evaluated independently. Neil et al. [20] restrict their patterns to paths and stars. In contrast to the methods above, our focus is to find *arbitrary-shape* anomalies that can possibly span *multiple time slices*.

7 Conclusions

We propose a novel and intuitive formulation for the problem of detecting all significant anomalous regions in a time-evolving network. Our proposed algorithm is:

- *Scalable*: NetSpot *scales linearly* to large real and synthetic network instances. It outperforms the Exhaustive counterpart by more than *10 times* on real networks and *30 times* on synthetic networks;
- *Accurate*: NetSpot produces *high quality* results, often matching the results of an exhaustive (but very slow) solution.
- *Effective*: NetSpot was able to spot unreported traffic accidents from real highway speed data, with precision and recall significantly higher than a single-edge approach. It was also able to discover interesting events by monitoring the rate of Wikipedia page views.

References

[1] *Afgh. News Cntr.* http://www.afghanistannewscenter.com/news/2009/november/nov212009.html.

[2] J. Abello, T. Eliassi-rad, and N. Devanur, *Detecting Novel Discrepancies in Communication Networks*, ICDM, (2010).

[3] R. K. Ahuja, Ö. Ergun, J. B. Orlin, and A. P. Punnen, *A survey of very large-scale neighborhood search techniques*, Discr. Appl. Math., 123 (2002), pp. 75–102.

[4] L. Akoglu and C. Faloutsos, *Event detection in time series of mobile communication graphs*, in Army Sc. Conf., 2010.

[5] L. Akoglu, M. McGlohon, and C. Faloutsos, *OddBall: Spotting Anomalies in Weighted Graphs*, in PAKDD, 2010.

[6] Z. Baig, *On the use of pattern matching for rapid anomaly detection in smart grid infrastructures*, in SmartGridComm, oct. 2011, pp. 214 –219.

[7] P. Bogdanov, M. Mongiovi, and A. K. Singh, *Mining heavy subgraphs in time-evolving networks*, in ICDM, 2011.

[8] Y. Chen, S. Nyemba, W. Zhang, and B. Malin, *Leveraging social networks to detect anomalous insider actions in collaborative environments*, in ISI, 2011.

[9] M. Davis, W. Liu, P. Miller, and G. Redpath, *Detecting anomalies in graphs with numeric labels*, in CIKM, 2011.

[10] M. T. Dittrich, G. W. Klau, A. Rosenwald, T. Dandekar, and T. Mller, *Identifying functional modules in protein-protein interaction networks: an integrated exact approach*, J. of Bioinformatics, (2008).

[11] W. Eberle and L. Holder, *Discovering structural anomalies in graph-based data*, in ICDMW, 2007.

[12] W. Eberle and L. Holder, *Graph-based approaches to insider threat detection*, in CSIIRW, 2009.

[13] W. Eberle, L. Holder, and D. Cook, *Identifying threats using graph-based anomaly detection*, in Mach. Learn. in Cyber Trust, 2009.

[14] W. He, G. Hu, and Y. Zhou, *Large-scale IP network behavior anomaly detection and identification using substructure-based approach and multivariate time series mining*, Telecom. Syst., (2012).

[15] K. Henderson, T. Eliassi-Rad, S. Papadimitriou, and C. Faloutsos, *HCDF: A hybrid community discovery algorithm*, in SDM,2010.

[16] Y. Jia, J. Zhang, and J. Huan, *An efficient graph-mining method for complicated and noisy data with real-world applications*, Knowledge and Information Systems, 28 (2011), pp. 423–447.

[17] D. Q. Le, T. Jeong, H. E. Roman, and J. W.K. Hong, *Traffic dispersion graph based anomaly detection*, in SoICT, 2011.

[18] S. Lin and H. Chalupsky, *Unsupervised link discovery in multi-relational data via rarity analysis*, in ICDM, 2003.

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

[20] J. Neil, *Scan Statistics for the Online Detection of Locally Anomalous Subgraphs*, PhD thesis, U. of New Mexico, 2011.

[21] D. Neill and G. Cooper, *A multivariate bayesian scan statistic for early event detection and characterization*, Machine Learning, 79 (2010), pp. 261–282. 10.1007/s10994-009-5144-4.

[22] D. B. Neill and A. W. Moore, *Rapid detection of significant spatial clusters*, in KDD, 2004.

[23] D. B. Neill, A. W. Moore, M. Sabhnani, and K. Daniel, *Detection of emerging space-time clusters*, in KDD, 2005.

[24] C. C. Noble and D. J. Cook, *Graph-based anomaly detection*, in KDD, 2003.

[25] C. E. Priebe, J. M. Conroy, D. J. Marchette, and Y. Park, *Scan statistics on enron graphs*, Comput. Math. Organ. Theory, 11 (2005).

[26] R. Rossi, B. Gallagher, J. Neville, and K. Henderson, *Role-dynamics: fast mining of large dynamic networks*, in Proceedings of the 21st international conference companion on World Wide Web, ACM, 2012, pp. 997–1006.

[27] J. Sun, H. Qu, D. Chakrabarti, and C. Faloutsos, *Relevance search and anomaly detection in bipartite graphs*, KDD Explor. Newsl., (2005).

[28] X. Wan, E. Milios, N. Kalyaniwalla, and J. Janssen, *Link-based event detection in email communication networks*, in SAC, 2009.

[29] J. Wang, B.-H. Chou, and E. Suzuki, *Finding the k-Most Abnormal Subgraphs from a Single Graph*, in Discovery Science, LNCS, 2009.

[30] M. Wu, C. Jermaine, S. Ranka, X. Song, and J. Gums, *A model-agnostic framework for fast spatial anomaly detection*, ACM Trans. Knowl. Discov. Data, 4 (2010), pp. 20:1–20:30.

[31] H. R. Zeidanloo and A. B. A. Manaf, *Botnet detection by monitoring similar communication patterns*, CoRR, abs/1004.1232 (2010).

[32] Y. Zhou, G. Hu, and W. He, *Using graph to detect network traffic anomaly*, in ICCCAS, 2009.

[33] *Supplemental material.* http://www.cs.ucsb.edu/~dbl/papers/mongiovi_sdm_2013_supplement.pdf.