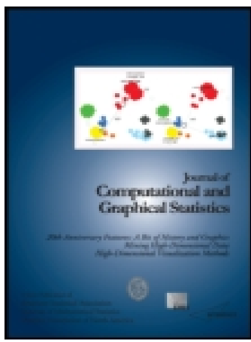


Journal of Computational and Graphical Statistics

ISSN: 1061-8600 (Print) 1537-2715 (Online) Journal homepage: http://www.tandfonline.com/loi/ucgs20

Scalable Detection of Anomalous Patterns With Connectivity Constraints

Skyler Speakman, Edward McFowland III & Daniel B. Neill

To cite this article: Skyler Speakman, Edward McFowland III & Daniel B. Neill (2015) Scalable Detection of Anomalous Patterns With Connectivity Constraints, Journal of Computational and Graphical Statistics, 24:4, 1014-1033, DOI: 10.1080/10618600.2014.960926

To link to this article: http://dx.doi.org/10.1080/10618600.2014.960926

Accepted author version posted online: 07 Oct 2014.
Published online: 10 Dec 2015.

Submit your article to this journal

Article views: 54

View related articles

View Crossmark data

Full Terms & Conditions of access and use can be found at
http://www.tandfonline.com/action/journalInformation?journalCode=ucgs20

Download by: [Carnegie Mellon University] **Date:** 23 December 2015, At: 14:18

Downloaded by [Carnegie Mellon University] at 14:18 23 December 2015

Scalable Detection of Anomalous Patterns With Connectivity Constraints

Skyler SPEAKMAN, Edward MCFOWLAND III, and Daniel B. NEILL

We present GraphScan, a novel method for detecting arbitrarily shaped connected clusters in graph or network data. Given a graph structure, data observed at each node, and a score function defining the anomalousness of a set of nodes, GraphScan can efficiently and exactly identify the most anomalous (highest-scoring) connected subgraph. Kulldorff's spatial scan, which searches over circles consisting of a center location and its $k - 1$ nearest neighbors, has been extended to include connectivity constraints by FlexScan. However, FlexScan performs an exhaustive search over connected subsets and is computationally infeasible for $k > 30$. Alternatively, the upper level set (ULS) scan scales well to large graphs but is not guaranteed to find the highest-scoring subset. We demonstrate that GraphScan is able to scale to graphs an order of magnitude larger than FlexScan, while guaranteeing that the highest-scoring subgraph will be identified. We evaluate GraphScan, Kulldorff's spatial scan (searching over circles) and ULS in two different settings of public health surveillance. The first examines detection power using simulated disease outbreaks injected into real-world Emergency Department data. GraphScan improved detection power by identifying connected, irregularly shaped spatial clusters while requiring less than 4.3 sec of computation time per day of data. The second scenario uses contaminant plumes spreading through a water distribution system to evaluate the spatial accuracy of the methods. GraphScan improved spatial accuracy using data generated from noisy, binary sensors in the network while requiring less than 0.22 sec of computation time per hour of data.

Key Words: Biosurveillance; Event detection; Graph mining; Scan statistics; Spatial scan statistic.

1. INTRODUCTION

The ability to detect patterns in massive datasets has multiple applications in policy domains such as public health, law enforcement, and security. The "subset scan" approach to pattern detection treats the problem as a search over subsets of data, with the goal of finding the most anomalous subsets. One major challenge of the "subset scan" approach is the computational problem that arises from attempting to search over the exponentially many subsets of the data. Linear time subset scanning (LTSS; Neill 2012) is a novel

Skyler Speakman, Edward McFowland III, and Daniel B. Neill, Event and Pattern Detection Laboratory, Carnegie Mellon University, Pittsburgh, PA 15213 (E-mail: *neill@cs.cmu.edu*).

© 2015 *American Statistical Association, Institute of Mathematical Statistics, and Interface Foundation of North America*
Journal of Computational and Graphical Statistics, Volume 24, Number 4, Pages 1014–1033
DOI: 10.1080/10618600.2014.960926
Color versions of one or more of the figures in the article can be found online at *www.tandfonline.com/r/jcgs*.

approach to anomalous pattern detection that addresses this issue by identifying the most anomalous subset of the data without requiring an exhaustive search, reducing computation time from years to milliseconds. Although LTSS provides a valuable speed increase, there are applications where LTSS by itself will provide less than ideal results as it is focused on detecting the most anomalous subset without additional constraints.

This work proposes GraphScan, a new method for event and pattern detection in massive datasets that have an underlying graph structure. Given a graph structure with vertices and edges $G = (V, E)$, and a time series of counts c_i^t for each vertex V_i in G, GraphScan detects emerging patterns by finding connected subgraphs $S \subseteq G$ such that the recent counts of the vertices V_i in S are significantly higher than expected. This process will be described in more detail below.

As one concrete example of the application of GraphScan, we consider the problem of disease outbreak detection. In this setting, LTSS with proximity constraints (Neill 2012) can be used to quickly detect spatially compact clusters of anomalous locations. However, consider an outbreak from a waterborne illness that leads to an increased number of hospital visits from patients that live in zip codes along a river or coastline. This noncompact spatial pattern would be hard to detect using proximity constraints. Taking advantage of an underlying graph structure based on zip code adjacency allows GraphScan to consider *connected* subsets of zip codes and therefore have increased power to detect these irregularly shaped clusters.

A second motivating example focuses on identifying contaminant plumes in a water distribution system equipped with noisy, binary sensors. We demonstrate that GraphScan's ability to exactly identify the most anomalous connected subset of sensors (nodes) increases spatial accuracy compared to heuristic methods such as the upper level set (ULS) scan statistic.

To clarify, our approach differs in both form and function from other recent work in graph mining. We are not attempting "community" or cluster detection (Flake, Lawrence, and Giles 2000). Also, unlike Wang et al. (2008), the anomalousness of the connected subsets we wish to identify is not based on the density of edges within the subgraph, but rather on properties of the nodes. We simply require that the detected subset of nodes be connected rather than looking for an anomalous collection of edges. Recent work by Leskovec et al. (2007) is also concerned with detecting events in networked data. Their goal is to determine the optimal placement of sensors within the network, while we address the complementary problem of fusing noisy data from multiple sensors for a given placement. Once these sensors are placed, scalable methods are still needed to detect events in the resulting large datasets with an underlying network structure.

1.1 SPATIAL EVENT DETECTION

This work applies GraphScan to the spatial event detection domain, using the additional connectivity constraints defined by the graph structure to detect irregularly shaped but connected subsets of locations. Our goal is to find the most interesting spatial (or spatiotemporal) subset of locations S, subject to the connectivity constraints, by maximizing the score function $F(S)$.

Downloaded by [Carnegie Mellon University] at 14:18 23 December 2015

In particular, let the domain of interest, $D = \{R_1 \ldots R_N\}$, be a set of N locations (or data records in a more general setting) and let $F(S)$ be a function mapping a subset of locations $S \subseteq D$ to a real number. These scoring functions are typically likelihood ratio statistics, assuming a parametric model such as Poisson- or Gaussian-distributed counts. The null hypothesis H_0 assumes that all counts are generated from the expected distribution (which can be spatially and temporally varying), while the alternative hypothesis $H_1(S)$ assumes that the recent counts for locations in subset S are increased by a multiplicative factor. Therefore, the ratio of the likelihoods of these two hypotheses, $F(S) = \frac{P(D \mid H_1(S))}{P(D \mid H_0)}$, provides the "score" of a region S, and we are interested in detecting the most anomalous (highest-scoring) connected region.

Spatial event detection methods in disease surveillance monitor a data stream (such as Emergency Department visits with respiratory complaints, or over-the-counter medication sales) across a collection of spatial locations and over time. These streams are represented as a series of counts c_i^t, from location s_i, and time step t. These counts are also used to determine the historical baseline (expected count) b_i^t for each location s_i at each time step t. Our goal is to determine the spatial or spatiotemporal region (subset of locations within a time window consisting of the past W days, for some $W = 1 \ldots W_{\max}$) that has an elevated level of activity indicating the early stages of a potential disease outbreak. The counts and baselines for each location in a region S are aggregated to form the count $C(S) = \sum_{s_i \in S} \sum_{t=1 \ldots W} c_i^t$ and baseline $B(S) = \sum_{s_i \in S} \sum_{t=1 \ldots W} b_i^t$. The amount of activity is quantified by the scoring function, $F(S) = F(C(S), B(S))$. For the expectation-based Poisson (EBP) statistic used here, the log-likelihood ratio is $F_{\text{EBP}}(S) = C \log(\frac{C}{B}) + B - C$, if $C > B$, and $F_{\text{EBP}}(S) = 0$ otherwise (Neill et al. 2005).

Previous methods have approached spatial event detection by reducing the search space of possible subsets, only considering regions that correspond to a particular shape such as circles (Kulldorff and Nagarwalla 1995; Kulldorff 1997), rectangles (Neill and Moore 2004), or ellipses (Kulldorff et al. 2006). Kulldorff's original spatial scan approach uses a circular (spatial) or cylindrical (space-time) window to detect regions of increased activity. While these approaches reduce the computational complexity from exponential to polynomial time, they have reduced power to detect clusters that do not correspond to the given shape.

Our work is not the first to address detecting events in graph or network data. The flexible scan statistic (FlexScan) has shown the utility of using adjacency constraints when detecting irregularly shaped spatial clusters (Tango and Takahashi 2005). FlexScan considers all subsets formed by a center node and a connected subset of its $k - 1$ nearest neighbors. Unfortunately, the run time of FlexScan scales exponentially with the neighborhood size k, and thus FlexScan becomes computationally infeasible for neighborhoods larger than 30 nodes. A more efficient method is required to scale to even moderately sized datasets. This increase in efficiency does not have to come at the price of a using a heuristic; our GraphScan method makes larger problems tractable while guaranteeing that the highest-scoring connected subset will be identified.

Other approaches rely on heuristics to accelerate the subset selection process. These are not guaranteed to find the most anomalous subset, and in some cases may perform arbitrarily badly as compared to the true optimum. For example, Duczmal and Assuncao (2004) detected clusters of homicides in a large urban dataset using simulated annealing

Downloaded by [Carnegie Mellon University] at 14:18 23 December 2015

Downloaded by [Carnegie Mellon University] at 14:18 23 December 2015

to search over the space of connected subgraphs. The upper level set scan statistic (ULS) by Patil and Taillie (2004) has impressive speed and scalability, but can fail to detect the highest-scoring connected subset even in a simple four-node graph, as shown by Neill (2012).

Neill (2012) proposed a method that exploits a property of scoring functions called linear-time subset scanning (LTSS). This property allows us to find the highest-scoring subset of N locations without exhaustively searching over the exponentially many subsets. However, it is highly nontrivial to extend LTSS to detect connected subsets of locations, and thus LTSS will often return disconnected clusters. This is the limitation addressed by our current work. We demonstrate that our GraphScan algorithm can efficiently and exactly detect the highest-scoring connected subset. This is different than both FlexScan (which is computationally intractable for large neighborhoods) and ULS (which does not guarantee an exact solution).

2. FAST SUBSET SCANNING WITH CONNECTIVITY CONSTRAINTS

Our approach to event detection is based on both *efficiently* and *exactly* identifying the highest-scoring connected subset of the data, thus providing high detection power while being able to scale to large datasets. For score functions satisfying the LTSS property (Neill 2012), the highest-scoring subset of records can be found by ordering the records according to some priority function $G(R_i)$ and searching over groups consisting of the top-j highest priority records for some (unknown) value of j. Formally, for a given dataset D, the scoring function $F(S)$ and priority function $G(R_i)$ satisfy the LTSS property if and only if $\max_{S \subseteq D} F(S) = \max_{j=1\ldots N} F(\{R_{(1)} \ldots R_{(j)}\})$, where $R_{(j)}$ represents the jth-highest priority record. For clarification, we consider $R_{(1)}$ to be the highest priority record, $G(R_{(1)}) \geq G(R_{(i)})$ for all $i > 1$, and $R_{(N)}$ to the be lowest priority record. In other words, the highest-scoring subset is guaranteed to be one of the linearly many subsets composed of the top-j highest priority records, for some $j \in \{1 \ldots N\}$. Therefore, in the search for the highest-scoring subset, we only need to consider these N subsets instead of the exponentially many possible subsets. The sorting of the records by priority requires $O(N \log N)$ time. However, if the priority sorting has already been completed, searching over subsets requires only $O(N)$ computation time.

For any subset of locations S, Neill (2012) showed that, if there exist locations $R_{\text{in}} \in S$ and $R_{\text{out}} \notin S$ such that $G(R_{\text{in}}) \leq G(R_{\text{out}})$, then $F(S) \leq \max(F(S \setminus \{R_{\text{in}}\}), F(S \cup \{R_{\text{out}}\}))$, and thus subset S is suboptimal. This property extends intuitively from single records to subsets of records. As above, let $C(S) = \sum_{s_i \in S} c_i^t$ and $B(S) = \sum_{s_i \in S} b_i^t$, and we define the priority of subset S to be $G(S) = \frac{C(S)}{B(S)}$, the ratio of the total count within S to the total baseline within S. Then if there exist subsets of locations $S_{\text{in}} \subseteq S$ and S_{out}, $S \cap S_{\text{out}} = \emptyset$, such that $G(S_{\text{in}}) \leq G(S_{\text{out}})$, then $F(S) \leq \max(F(S \setminus S_{\text{in}}), F(S \cup S_{\text{out}}))$, and thus subset S is suboptimal.

When connectivity constraints are introduced, the above inequality between subsets S, $S \setminus S_{\text{in}}$, and $S \cup S_{\text{out}}$ still holds. However, for a connected subset S, the subsets $S \setminus S_{\text{in}}$ and $S \cup S_{\text{out}}$ may not be connected. Thus S is only guaranteed to be suboptimal if two conditions

Downloaded by [Carnegie Mellon University] at 14:18 23 December 2015

hold: (i) simultaneously removing all records $R_i \in S_{\text{in}}$ would not disconnect S; and (ii) at least one of the records in S_{out} is adjacent to S, and therefore simultaneously adding all records $R_i \in S_{\text{out}}$ would allow the subset to remain connected. Thus we can state the LTSS GraphScan logic as follows: "If subset S_{in} is included in the highest-scoring connected subset S, and removing S_{in} would not disconnect S, then no connected subset S_{out} adjacent to S can have higher priority than S_{in}."

We now consider the various types of scoring functions that satisfy LTSS and hence can be optimized by the GraphScan algorithm. Neill (2012) proved that any function $F(C, B)$ which is quasi-convex, increases with C, and is restricted to positive values of B will satisfy the LTSS property. Kulldorff's original spatial scan statistic (Kulldorff 1997), also used as the score function for the FlexScan algorithm (Tango and Takahashi 2005), satisfies LTSS. Therefore, GraphScan could be used in place of the circular scan, to scan over connected clusters instead of circles, in any of the large number of application domains to which Kulldorff's approach and FlexScan have been applied. The corresponding priority function for Kulldorff's spatial scan statistic is $G(R_i) = \frac{c_i}{b_i}$.

Additionally, LTSS holds for expectation-based scan statistics (Neill 2009b) in the separable exponential family, including but not limited to the Poisson, Gaussian, and exponential distributions. In these cases, the additive sufficient statistics C and B may be different: for example, $c_i = \frac{x_i \mu_i}{\sigma_i^2}$ and $b_i = \frac{\mu_i^2}{\sigma_i^2}$ for the expectation-based Gaussian scan statistic with means μ_i, standard deviations σ_i, and observed values x_i. The priority function $G(R_i) = \frac{c_i}{b_i}$ also applies to expectation-based scan statistics. Typically, scan statistics are used to detect *increased* activity where counts are higher than expected. However, the expectation-based scan statistics can also be used to detect decreased counts while still satisfying LTSS. Intuitively, the corresponding priority function in this setting is $G(R_i) = \frac{b_i}{c_i}$, reversing the original ordering. Finally, LTSS can also be applied to a variety of nonparametric scan statistics, as described in McFowland, Speakman, and Neill (2013), and GraphScan can be used to detect connected clusters in these settings as well.

3. GRAPHSCAN ALGORITHM

Operating naively, identifying the highest-scoring connected subset for a graph of N nodes requires an exhaustive search over all $O(2^N)$ possible connected subsets. GraphScan performs this search over connected subsets using a depth-first search with backtracking, but gains speed improvements by ruling out subsets that are provably suboptimal. First, we rule out subsets violating the LTSS GraphScan property. If there exist two subsets S_{in} and S_{out} as defined above, with the priority of S_{out} exceeding the priority of S_{in}, then S is suboptimal. Second, we apply a "branch-and-bounding" technique to rule out groups of subsets that are guaranteed to be lower scoring than the best connected subset found thus far.

3.1 Subgraph Creation and Definitions of Common Terms

We define *seed records* as records that have higher priority than all of their neighbors. Let seeds $\subseteq D$ be the set of all seed records in G. For each seed record $R_{(j)} \in$ seeds, GraphScan forms a subgraph G_j such that all records with higher priority than $R_{(j)}$, and the neighbors

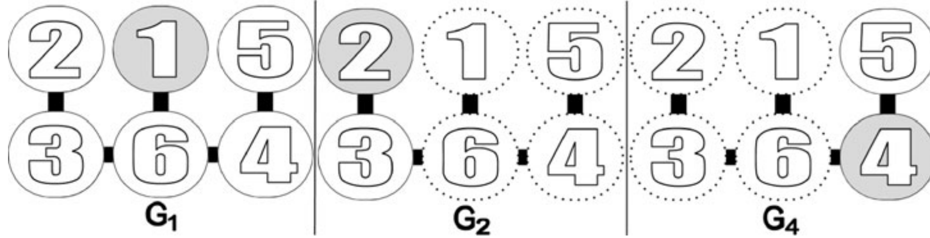


Figure 1. A graph broken into three subgraphs, one for each seed record (darkened). Nodes with dashed bevels are not included in a given subgraph. In G_2, $R_{(6)}$ has been removed because it is a neighbor of $R_{(1)}$. We remove $R_{(4)}$ and $R_{(5)}$ because they can no longer be reached from $R_{(2)}$. Subgraphs G_1, G_2, and G_4, respectively, represent 32, 2, and 2 of the 64 subsets under consideration. The remaining 28 subsets have been ruled out by the subgraph creation process.

Downloaded by [Carnegie Mellon University] at 14:18 23 December 2015

of these higher-priority records, are excluded from G_j. Additionally, records that are no longer reachable from $R_{(j)}$ are excluded. An example is provided in Figure 1.

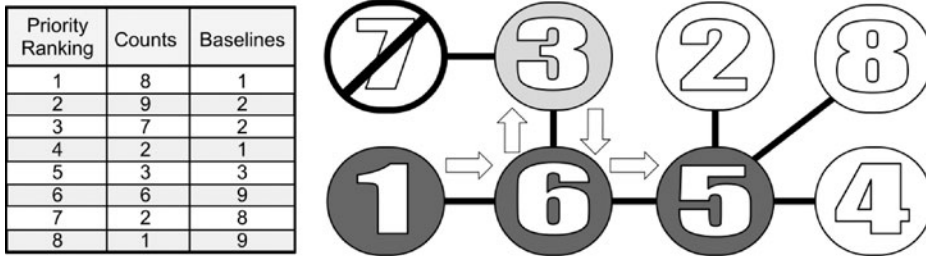
To conduct a depth-first search within each subgraph, we define a route to be a data structure with five components. First is the *subset* of records included and excluded from the route. These are stored in a priority-ordered N_j-bit string, where N_j is the number of nodes remaining in that subgraph. The kth bit, X_k, represents the inclusion or exclusion of the kth highest priority record $R_{(k)}$. All records included in the route are represented as $X_k = 1$ and excluded records are represented as $X_k = 0$. Any records that have yet to be considered are marked with $X_k = ?$. Second is the route's *current path*, which ends at its *current location*. This is a sparse representation of records ordered by inclusion in the route, and allows for backtracking. Third are the route's current *sidetracks*. Sidetracks are connected subsets of records which have been backtracked through by the depth-first search procedure; they are included in the route's subset but are not on the current path and no longer have potential for further exploration. Note that removal of any sidetrack will not disconnect the current subset, and thus a route's S_{in} is defined as the lowest priority sidetrack contained in that route. Finally, a route's S_{out} is the highest priority excluded neighbor of the route; alternatively, we can consider a broader definition of S_{out} as detailed below.

GraphScan keeps track of all candidate routes for a given subgraph using a priority queue. New routes under consideration will either be ruled out by the LTSS GraphScan property, ruled out by "branch and bounding," or added back to the queue for further processing. Any connected subset S which is not pruned will have its score $F(S)$ computed, and GraphScan keeps track of the highest-scoring connected subset found during its search.

3.2 PROCESSING A SUBGRAPH

After identifying seed records and forming a subgraph for each seed record, the task is to efficiently process each subgraph to identify its highest-scoring connected subset. The highest score over all subgraphs is returned as the final solution. At each step of the GraphScan algorithm, a route is removed from the queue and multiple child routes are propagated as either an extension or backtrack of the current path. Cycles are avoided by not considering child nodes that are also neighbors of the current path. Assuming that

Downloaded by [Carnegie Mellon University] at 14:18 23 December 2015



Priority Ranking	Counts	Baselines
1	8	1
2	9	2
3	7	2
4	2	1
5	3	3
6	6	9
7	2	8
8	1	9

Figure 2. A possible route for an 8-node subgraph. The number in each node represents the node's priority ranking. The current subset is [1, ?, 1, ?, 1, 1, 0, ?] , and the current path is [1, 6, 5]. $S_{\text{in}} = \{R_{(3)}\}$ with priority 3.5, because $R_{(3)}$ is included in the subset and removing it would not disconnect the subset. $S_{\text{out}} = \{R_{(7)}\}$ with priority 0.25. Four child routes must be considered: extending the path to $R_{(2)}$; excluding $R_{(2)}$ and extending to $R_{(4)}$; excluding $R_{(2)}$ and $R_{(4)}$ and extending to $R_{(8)}$; excluding $R_{(2)}$, $R_{(4)}$, and $R_{(8)}$ and backtracking to $R_{(6)}$. All but the first route are provably suboptimal and would not be reinserted into the queue. Specifically, excluding $R_{(2)}$ from the route would increase the route's S_{out} priority to $\frac{9}{2} = 4.5$, higher than the priority of S_{in}.

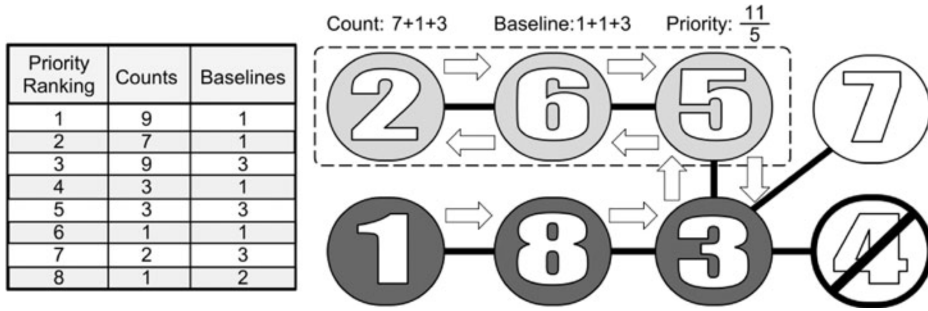
the current location is $R_{(i)}$ with C child nodes $R_{(j_1)} \ldots R_{(j_C)}$ in priority order, we consider $C + 1$ child routes for reinsertion into the queue: one route extending the path to each child node $R_{(j_c)}$, and one backtracked route.

When extending the current path from record $R_{(i)}$ to record $R_{(j_c)}$, $1 < c \leq C$, we exclude the $c - 1$ neighbors of $R_{(i)}$ that have a higher priority than $R_{(j_c)}$. The route's S_{out} is updated if one of the newly excluded neighboring records has a higher priority than the route's current S_{out}. If the priority of the route's S_{out} exceeds that of S_{in} then this new route is *not* reinserted into the queue because it represents a provably suboptimal subset of records. See Figure 2 for an example.

When backtracking, we exclude all of the C neighbors of $R_{(i)}$ and change the current location to the previous node on the current path. In addition to potentially updating a route's S_{out}, backtracking may also change the route's S_{in} and requires some additional attention. When backtracking, GraphScan must recalculate the priority of the entire current sidetrack. To that end, the new current location aggregates the counts and baselines of the backtracked record with its own. This is done for every backtrack, and therefore the new current location inherits the counts and baselines (and therefore, the priority as well) of the entire current sidetrack. It is this priority that we must consider when updating a route's S_{in}. See Figure 3 for an example.

If this ratio of aggregated counts and baselines is lower than the priority of the route's current S_{in}, then we update the route's S_{in} before attempting to reinsert it into the queue. If S_{in} has lower priority than the route's S_{out} then it is not reinserted to the queue because it represents a provably suboptimal subset. This updating and comparing of S_{in} and S_{out} as each route propagates allows GraphScan to prune a large number of subsets from its search space.

Further speed improvements can be made by including an additional check before a route is inserted into the queue. Recall that the route contains information about which records have yet to be included or excluded, that is, the records with $X_k = ?$. If the highest priority of all such records is lower than the priority of S_{out}, then we may also prune this route after scoring the current subset.



Priority Ranking	Counts	Baselines
1	9	1
2	7	1
3	9	3
4	3	1
5	3	3
6	1	1
7	2	3
8	1	2

Figure 3. A possible route for an 8-node subgraph. This example demonstrates aggregating counts and baselines during the backtracking step of the GraphScan algorithm. Currently, $S_{\text{out}} = \{R_{(4)}\}$, with a priority of 3, and $S_{\text{in}} = \{R_{(2)}, R_{(6)}, R_{(5)}\}$, with a priority of $\frac{11}{5} = 2.2$. Note that $R_{(5)}$ has a priority of $\frac{3}{3} = 1$ when considered by itself. However, we cannot assign $S_{\text{in}} = \{R_{(5)}\}$ because removing only $R_{(5)}$ would disconnect the subset. If we remove $R_{(5)}$ we must also remove the rest of the sidetrack. Thus S_{in} is the minimum priority of $R_{(2)}$ alone (priority = 7), $R_{(2)}$ and $R_{(6)}$ (priority = 4), and $R_{(2)}$, $R_{(5)}$, and $R_{(6)}$ (priority = 2.2). This particular route would not be reinserted into the queue because the priority of S_{in} is less than that of S_{out} (2.2 < 3).

Algorithm 1 presents GraphScan without "branch and bounding" or proximity constraints. These additional extensions to the GraphScan algorithm are discussed below. Note that steps 8 and 13 prune any subsets that are provably suboptimal by not reinserting them into the queue.

Algorithm 1 GraphScan

1: Identify *seed records* as records with higher priority than their neighbors.
2: **for** each seed record **do**
3: Form subgraph and initialize priority queue with route originating at seed record.
4: **while** priority queue not empty **do**
5: Remove highest priority route from queue and note its current location, S_{in}, and S_{out}.
6: **for** each neighbor of current location not on or adjacent to the path **do**
7: Extend the path by setting the current location to that neighbor, and exclude higher priority neighbors.Update S_{out} if necessary.
8: **if** priority of S_{out} < priority of S_{in} **then**
9: Score the subset and insert route into priority queue for further processing.
10: **end if**
11: **end for**
12: Backtrack the path by setting the current location to the previous location on the path, and exclude all neighbors. Update S_{out} and S_{in} if necessary.
13: **if** priority of S_{out} < priority of S_{in} **then**
14: Score the subset and insert route into priority queue for further processing.
15: **end if**
16: **end while**
17: **end for**
18: Return highest scoring subset across all subgraphs.

Downloaded by [Carnegie Mellon University] at 14:18 23 December 2015

3.3 Proof of GraphScan's Exactness

We now prove that the GraphScan algorithm is guaranteed to identify the highest-scoring connected subset despite the large reduction in the search space. Since GraphScan performs a depth-first search over the space of all connected subsets, it is clear that the highest-scoring connected subset would be found if no pruning was performed. Thus we must show that, for all connected subsets S pruned at each step of the algorithm, there exists some connected subset S' which is not pruned and has $F(S') \geq F(S)$. Our first proof will focus on partitioning the problem into subgraphs based on *seed records*, and our second proof will focus on the exclusion of routes within each subgraph. Let $\text{IN}(S)$ denote the set of all nonempty subsets $S_{\text{in}} \subseteq S$ such that $S \setminus S_{\text{in}}$ is connected (or empty), and $\text{OUT}(S)$ denote the set of all nonempty subsets S_{out} such that $S \cap S_{\text{out}} = \emptyset$ and $S \cup S_{\text{out}}$ is connected. We can then prove the following theorems:

Lemma 1. For any connected subset S, if there exist $S_{\text{in}} \in \text{IN}(S)$ and $S_{\text{out}} \in \text{OUT}(S)$ such that $G(S_{\text{in}}) \leq G(S_{\text{out}})$, then subset S is suboptimal.

Proof. This follows directly from the facts that $F(S) \leq \max(F(S \setminus S_{\text{in}}), F(S \cup S_{\text{out}}))$ and that the subsets $S \setminus S_{\text{in}}$ and $S \cup S_{\text{out}}$ are connected.

□

Theorem 1. (*Exactness of Subgraph Creation*). For any connected subset S that is pruned by the subgraph creation process described in Section 3.1, there exists some connected subset S' which is not pruned and has $F(S') \geq F(S)$.

Proof. Let $\mathbf{S}$ be the set of all possible connected subsets and let $\mathbf{S}_j$ represent all connected subsets in which record $R_{(j)}$ is the highest priority included record. Note that $\mathbf{S} = \bigcup_{j=1}^{N} \mathbf{S}_j$, and thus we can reduce the problem to finding the highest-scoring subset for each $\mathbf{S}_j$. However, GraphScan only forms subgraphs for each seed record, pruning all subsets for which the highest-priority record is not a seed record. Also, for a given subgraph G_j, GraphScan prunes all subsets in $\mathbf{S}_j$ which contain a neighbor of any record with higher priority than $R_{(j)}$. In either case, for all pruned subsets S, there exists a record $R_{\text{out}} \notin S$ which is adjacent to S and has higher priority than all records in S. The suboptimality of region S follows from applying Lemma 1 with $S_{\text{in}} = S$ and $S_{\text{out}} = \{R_{\text{out}}\}$. More precisely, we know that $F(S) \leq F(S \cup \{R_{\text{out}}\})$ and that $S \cup \{R_{\text{out}}\}$ is connected. Finally, the exclusion of nodes which are no longer reachable from $R_{(j)}$ during subgraph formation does not prune any subsets in $\mathbf{S}_j$, since all such subsets would be disconnected.

□

Theorem 2. (*Exactness of Route Propagation*). For any connected subset S that is pruned by the route propagation process described in Section 3.2, there exists some connected subset S' which is not pruned and has $F(S') \geq F(S)$.

Proof. For a given route Z, let S_{incl} denote the set of all "included" records $R_{(k)}$ (i.e., records with $X_k = 1$), and let S_{excl} denote the set of all "excluded" records $R_{(k)}$ (i.e., records with

Downloaded by [Carnegie Mellon University] at 14:18 23 December 2015

$X_k = 0$). Let $\mathbf{S}$ denote the set of all subsets still under consideration for the current route, that is, all subsets S such that $S_{\text{incl}} \subseteq S$ and $S \cap S_{\text{excl}} = \emptyset$. When route Z is propagated, C child routes $Z_1 \ldots Z_C$ are formed by conditioning on the highest-priority included child node $R_{(j_c)}$, and an additional child route Z_0 is formed assuming that all child nodes are excluded. Let $\mathbf{S}_c$ denote the set of all subsets still under consideration for child route Z_c. We first note that $\bigcup_{c=0}^{C} \mathbf{S}_c = \mathbf{S}$, and thus if no pruning was performed, GraphScan would search exhaustively over all connected subsets.

However, GraphScan will prune any route Z which has $G(S_{\text{in}}) \leq G(S_{\text{out}})$, where $S_{\text{in}} \subset S_{\text{incl}}$ is a sidetrack and $S_{\text{out}} \subseteq S_{\text{excl}}$ is a subset that is excluded from, but adjacent to, S_{incl}. For any subset $S \in \mathbf{S}$ which is still under consideration for the route, we know that $S_{\text{in}} \in \text{IN}(S)$, since $S_{\text{in}} \subset S_{\text{incl}} \subseteq S$ and removal of the sidetrack S_{in} will not disconnect S. Also, we know that $S_{\text{out}} \in \text{OUT}(S)$, since $S_{\text{out}} \in \text{OUT}(S_{\text{incl}})$ and $S \cap S_{\text{out}} = \emptyset$. These facts imply that S is suboptimal by Lemma 1, as its score would be improved by either excluding S_{in} or including S_{out}.

GraphScan also compares each route's S_{out} to the highest-priority record $R_{(k)}$ yet to be included in the subset (i.e., the smallest k such that $X_k = ?$). If the priority $G(\{R_{(k)}\}) \leq G(S_{\text{out}})$, then the route's currently included subset S_{incl} is scored but the route is not reinserted into the queue. In this case, for any other subset $S \in \mathbf{S}$ which is still under consideration for the route, we know that $G(S \setminus S_{\text{incl}}) \leq G(S_{\text{out}})$. Since S_{incl} is connected, we know that $S \setminus S_{\text{incl}} \in \text{IN}(S)$. Also, we know that $S_{\text{out}} \in \text{OUT}(S)$, since $S_{\text{out}} \in \text{OUT}(S_{\text{incl}})$ and $S \cap S_{\text{out}} = \emptyset$. Thus S is suboptimal by Lemma 1, as its score would be improved by either excluding $S \setminus S_{\text{incl}}$ or including S_{out}.

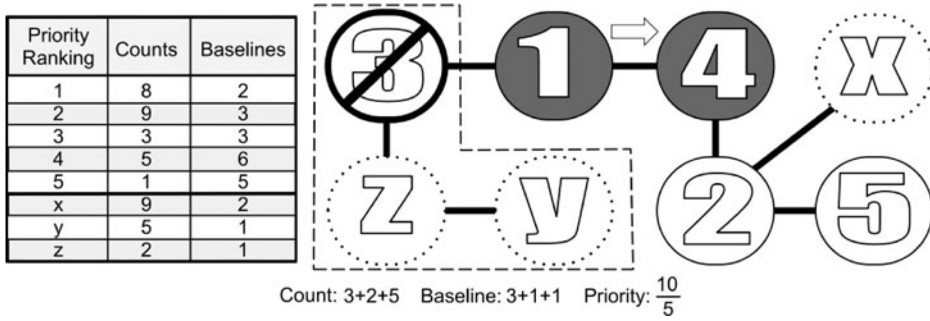
□

3.4 SPEEDING UP SUBGRAPH PROCESSING WITH BETTER ESTIMATION OF S_{OUT}

We have introduced the GraphScan algorithm with an effective but simplistic understanding of a route's S_{out} by restricting it to be a single record (the highest priority neighboring record excluded from a route). We now allow for S_{out} to be a connected subset of records that have all been excluded from a given route. To do so, recall that S_{out} is a connected subset of records not contained in S such that at least one of the records in S_{out} is adjacent to S, and therefore simultaneously adding all records $R_i \in S_{\text{out}}$ would allow the subset to remain connected.

Consider a subgraph G_j, for $j > 1$. This subgraph excludes all records with priority higher than $R_{(j)}$ as well as the neighbors of these higher priority records. GraphScan uses records that have been excluded from G_j to expand a route's S_{out}. Let $R_{(i)}$ be a record contained in G_j which has a neighbor $R_{(k)}$, $k < i$, that has been excluded from G_j. If $R_{(i)}$ is excluded from a route in G_j, then it benefits us to consider the priority of the subset $S_{\text{out}} = \{R_{(i)}, R_{(k)}\}$, which will be higher than the priority of $R_{(i)}$. Even if $k > i$, $R_{(k)}$ may have high-priority neighbors that have also been excluded from G_j. This insight leads to a goal of establishing a high-priority subset S_{out} of connected records that have all been excluded from G_j but include at least one record adjacent to potential routes contained in G_j. It is this subset's priority that is used when determining the route's highest-priority excluded subset, rather than the priority of a single excluded record.

Downloaded by [Carnegie Mellon University] at 14:18 23 December 2015



Priority Ranking	Counts	Baselines
1	8	2
2	9	3
3	3	3
4	5	6
5	1	5
x	9	2
y	5	1
z	2	1

Figure 4. A possible route for a 5-node subgraph with additional information from records excluded from the subgraph. Naively, we would use $S_{out} = R_{(3)}$ with a priority of $\frac{3}{3} = 1$. During the creation of the subgraph, it is noted that nodes R_y and R_z (their priority ranking does not matter because they are excluded from the subgraph) are connected to $R_{(3)}$ in the original graph. Therefore, when excluding $R_{(3)}$ from the route we may actually set the highest excluded priority of the route to $\frac{3+2+5}{3+1+1} = 2$ and $S_{out} = \{R_{(3)}, R_y, R_z\}$. This operation is not limited to excluding records from the subgraph. Consider extending the current path to $R_{(2)}$. By including $R_{(2)}$, we are able to further increase the highest excluded priority to $\frac{9}{2} = 4.5$ and set $S_{out} = R_{(x)}$.

Although finding a high priority S_{out} is preferred, the exactness of the GraphScan algorithm does not require us to find the *highest* priority S_{out}. Therefore, a simple greedy heuristic is used to aggregate the counts and baselines of connected records that have been excluded from G_j. Searching over only records that have been excluded from G_j, the heuristic iteratively adds the highest-priority neighbor until either there are no more records to add or the priority of the subset begins to decrease. This extension can substantially increase the priority of S_{out} for a given route, resulting in much more pruning of the search space. Finally, we note that these priorities are precalculated during the creation of the subgraph. During route propagation, when extending the current path by including a neighboring record and excluding higher priority neighbors, the priority of S_{out} is established by referencing these precalculated priorities rather than relying solely on the single highest priority excluded record. See Figure 4 for more details.

3.5 BRANCH AND BOUNDING WITH UNCONSTRAINED LTSS

The unconstrained LTSS property of scoring functions is applied in the GraphScan algorithm through *branch and bounding* (Land and Doig 1960). Branch and bounding is intelligently enumerating candidate solutions by systematically ruling out large subsets of fruitless ones. In practice, branch and bounding allows the algorithm to interrupt the route propagation when all subsets represented in a route are guaranteed to be lower scoring than a currently known connected subset. This is possible because we can quickly determine the "upper bound" (unconstrained score) of a route through the property of LTSS. Since the set of records is already sorted by priority, the unconstrained score can be calculated in linear time. This process involves consecutively adding the next highest priority record with $X_k = ?$ (ignoring connectivity constraints) and then scoring all records contained in the (now, possibly disconnected) subset. The highest-scoring subset from this process is guaranteed by the LTSS property to be the highest-scoring unconstrained subset in that route. If this bound is less than or equal to the current high score, then the maximum score

Downloaded by [Carnegie Mellon University] at 14:18 23 December 2015

of all connectivity-constrained subsets within the route cannot be greater than the current high-scoring connected subset, and thus we do not need to continue processing the route.

We define two scoring functions which map a route to real numbers LBound(route) and UBound(route). LBound(route) is the score of the connected subset formed by only including the records in the current subset. UBound(route) is the score of the highest-scoring unconstrained subset of the route, efficiently determined by the LTSS property as described above.

Before being inserted into the queue, the upper and lower bounds of the route are found and compared to the current best score of a connected subset with the following outcomes:

- Current best score $<$ LBound(route) $<$ UBound(route): This signifies that route's current subset is the *new* current best scoring connected subset. The subset is noted and the new best score updated before inserting the route back into the queue.
- Current best score $<$ LBound(route) = UBound(route): This signifies that the current subset is the *new* current best scoring connected subset as well as the highest-scoring subset in the entire route. The subset is noted and the new best score is updated but the route is not reinserted into the queue.
- UBound(route) $<$ Current best score: This signifies that all of the route's subsets (even without enforcing connectivity constraints) are lower scoring than the highest-scoring connected subset found so far. The route is not reinserted into the queue.
- LBound(route) $<$ Current best score $<$ UBound(route): This signifies that no new information is gained through branch and bounding. The route is reinserted into the queue.

The order in which the routes are processed within a branch and bounding framework can affect the runtime of the algorithm. We sort the queue based on the LBound(route) value. This ordering had minor but noticeable improvement in runtime ($\sim$23% faster than random ordering).

3.6 INCORPORATING PROXIMITY CONSTRAINTS

The major contribution of GraphScan is combining *connectivity constraints* with the LTSS property to efficiently determine the highest-scoring connected subset of records. However, if the dataset has spatial information as well, then we may use both *proximity* and *connectivity* constraints simultaneously. Given a metric which specifies the distance $d(R_i, R_j)$ between any two records R_i and R_j, we may identify a "local neighborhood" of records around a central record R_c. For example, in the disease surveillance domain, we use the latitude and longitude coordinates of the centroid of each zip code. GraphScan forms "local neighborhoods" by considering a central record R_c and its $k - 1$ nearest neighbors for a fixed constant k. There are N of these neighborhoods formed with each one centered around a different record R_c. GraphScan finds the highest-scoring connected cluster *within* each neighborhood by forming and processing a connectivity graph consisting of only the records in that neighborhood, and then reports the single highest-scoring connected subset found from these N searches.

Downloaded by [Carnegie Mellon University] at 14:18 23 December 2015

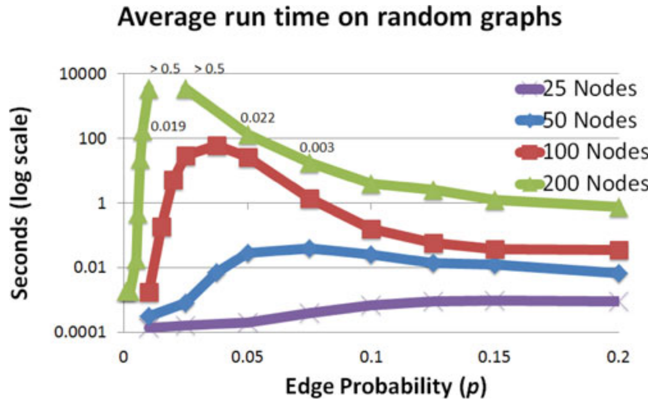


Figure 5. Performance of GraphScan on Erdös-Renyi random graphs of varying size and edge probability. Labeled data points are the proportion of graphs where run time exceeded 1 hr.

The implementation of proximity constraints within GraphScan is similar to the constraints used in FlexScan (Tango and Takahashi 2005), with a slight difference. FlexScan uses an identical approach to form the neighborhoods of each data record, but it only considers the connected subsets that *include* the central record R_c. In other words, it determines the highest-scoring connected cluster consisting of R_c and a subset of its $k-1$ nearest neighbors. GraphScan does not require the central record to be in the subset and considers *all* possible connected subsets for each group of k records. In practice, this minor difference has negligible impact on detection power, and thus the only substantial difference between FlexScan and GraphScan is in runtime.

4. EVALUATION OF RUN TIME ON RANDOM GRAPHS

We first evaluate the average amount of time taken for GraphScan to identify the highest-scoring connected subgraph for Erdös-Renyi random graphs of varying size n and edge probability p. Erdös-Renyi graphs are formed by placing each of the $\binom{n}{2}$ possible edges in the graph with probability p. Figure 5 provides the average run times for graphs of size 25, 50, 100, and 200 nodes with varying edge probability. For each combination of n and p, at least 1000 different Erdös-Renyi graphs were created, processed with GraphScan, and the average run time was reported. Some of the 200-node graphs resulted in runtimes exceeding 1 hr. In these instances, the excessive run times were not used in the calculation of the mean, but the proportion of runs that exceeded this 1-hr threshold are provided as a reference on the point. For example, for 200-node graphs with an edge probability of $p = 0.05$, 97.8% of the runs finished with an average of 135.2 sec each. However, 2.2% of the graphs exceeded 1 hr of processing time and had their run times removed from the overall calculation.

Not surprisingly, increased graph size resulted in longer run times; however, the role of edge probability is interesting and worthy of further discussion. In Erdös-Renyi graphs, the edge probability p has theoretical thresholds that change the nature of the graph (Erdös and Renyi 1959). For example, when $p < \frac{1}{n}$, the entire graph is composed of smaller subgraphs

Downloaded by [Carnegie Mellon University] at 14:18 23 December 2015

that are disconnected from each other. As p increases beyond $\frac{1}{n}$, a single *giant component* begins to emerge which contains the majority of the nodes. This giant component increases in size with increasing p, until $p = \frac{\ln n}{n}$. At this point the giant component will (almost surely) contain all of the n nodes in the graph, resulting in a single component graph. Increasing p beyond this threshold increases the overall connectedness of the graph and decreases its diameter. These stages are evident in the performance of GraphScan. The peak in run time occurs near $p = \frac{\ln n}{n}$ for each of the various graph sizes. As edge probability drops below this threshold value, we see improved performance because the majority of calculation time is spent on the giant cluster that is decreasing in size. As edge probability increases above the threshold, the giant component is no longer increasing in size but is now decreasing in diameter, also resulting in improved performance.

Downloaded by [Carnegie Mellon University] at 14:18 23 December 2015

5. EVALUATION ON SPATIAL DISEASE SURVEILLANCE

We present empirical results of GraphScan's run time performance, time to detect (average number of days needed to detect an outbreak) and detection power using a set of simulated respiratory disease outbreaks injected into real-world Emergency Department data from Allegheny County, Pennsylvania. We compare results for multiple methods: "Circles" (traditional approach introduced by Kulldorff; returns the highest-scoring circular cluster of locations), "All subsets" (LTSS implemented without proximity or connectivity constraints; returns the highest-scoring unconstrained subset of locations), "ULS" (returns a high-scoring connected subset based on the ULS scan statistic within a neighborhood size of k) and "GraphScan" (returns the highest-scoring connected subset within a neighborhood size of k).

The Emergency Department data come from 10 Allegheny County hospitals from January 1, 2004 to December 31, 2005. By processing each case's ICD-9 code and free text "chief complaint" string, a count dataset was created by recording the number of patient records with respiratory symptoms (such as cough or shortness of breath) for each day and each zip code. The resulting dataset had a daily mean of 44.0 cases, and standard deviation of 12.1 cases. There were slight day-of-week and seasonal trends, with counts peaking on Mondays and in February.

In Figure 6, we present the average run times per day of Emergency Department data for three different algorithms. The FlexScan algorithm naively enumerates all 2^{k-1} subsets containing the center record for each group of k records. GraphScan's speed improvements come from two different sources: reduction of the search space by applying the LTSS property with connectivity constraints, and by branch and bounding (direct application of LTSS without connectivity constraints). We provide run times for GraphScan with and without branch and bounding for values of $k = 10, 15, \ldots, 70$. For $k = 30$, GraphScan achieves over 450,000x faster computation time than FlexScan, and FlexScan was computationally infeasible for $k > 30$. The addition of branch and bounding to GraphScan results in a further 50x speed increase for $k = 50$. ULS, like GraphScan, required only seconds to process each day of data. However, while GraphScan is guaranteed to find the highest-scoring subset, ULS was only able to find the highest-scoring subset 1.1% of the time, while 14.2% of the time ULS returned a subset with score less than half of the maximum.

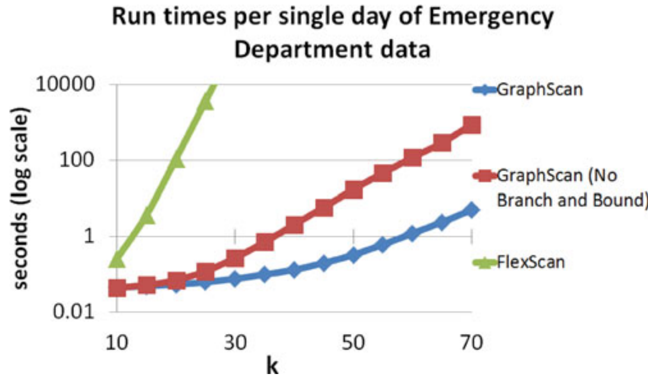


Figure 6. Run time analysis for FlexScan and GraphScan with and without branch and bounding. The x-axis denotes the "neighborhood size" as various values of k.

We note that the worst case complexity of GraphScan is exponential in the neighborhood size. If no pruning was performed, GraphScan would evaluate all connected subsets, requiring $O(2^k)$ run time; however, GraphScan is able to rule out many connected subsets as provably suboptimal, reducing complexity to $O(q^k)$ for some constant $1 < q < 2$, where q is dependent on the proportion of subsets that are pruned. For the Emergency Department data, we empirically estimate $q \approx 1.2$. For graphs that are sufficiently dense, runtime of GraphScan becomes linear in k as in the unconstrained LTSS case, while for sufficiently sparse graphs, few subsets are connected.

5.1 Simulating and Detecting Outbreaks

Our semisynthetic testing framework for evaluating the performance of disease outbreak detection algorithms artificially increases the number of disease cases in the affected region by injecting simulated counts into real-world background data. This allows us to simulate disease outbreaks of varying duration and severity while taking into account the noisy nature of real world data. The simulation of realistic disease outbreak scenarios is a large and active research area. Simulators such as those used in Buckeridge et al. (2004) and Wallstrom, Wagner, and Hogan (2005) combine current background data with that of past outbreaks to create a realistic new outbreak injected into current data. In this work, we implement a much simpler outbreak model that linearly increases the number of cases over the duration of the outbreak. We acknowledge that this is not a realistic model of the temporal progression of an outbreak. However, it allows for a precise comparison of the different detection methods under consideration, by gradually increasing the severity of the outbreak over its duration. On each day t of the outbreak $t = 1 \ldots 14$, the simulator injects Poisson(t) cases over the affected zip codes.

We created six spatial injects that correspond to natural or man-made geographical features of Allegheny County, Pennsylvania, shown in Figure 7. Three of the regions are formed with zip codes along the Allegheny and Monongahela rivers, simulating a waterborne disease outbreak. The other three regions follow the path of two major U.S. interstates that traverse the county.

Downloaded by [Carnegie Mellon University] at 14:18 23 December 2015

Figure 7. Outbreak regions used in the semisynthetic tests. Regions #1 and #2 follow rivers and #4 and #5 follow interstates. #3 is the union of #1 and #2; #6 is the union of #4 and #5.

Once the simulated cases have been created and injected into the real-world background data, our focus turns to detecting the outbreak. First, we obtain a score $F^* = \max_S F(S)$ (using the same search space and scoring function as the method under consideration) for each day in the original dataset *without* any injected cases. This provides a background distribution of scores which is used to provide a realistic false positive rate that is more accurate than those obtained through Monte Carlo simulation (Neill 2009a). Then for every day t of the simulated outbreak, we compute the day's maximum region score and determine the proportion of background days for which F^* exceeds it. Therefore, for a *fixed* false positive rate r, the number of days required to detect a gradually increasing outbreak is a good measure of detection power. We allow a false positive rate of 1 per month, a level considered to be acceptable by many public health departments (Neill 2006).

We provide results for detection power for the four different methods under consideration: circles, all subsets, ULS, and GraphScan, with the last two considering various neighborhood sizes, k. For each of the six different S_{inject} regions, 200 simulated injects were created and randomly inserted in the two-year time frame of our data. At the fixed false positive rate of 1 per month, the total number of outbreaks detected and the average number of days to detection (counting missed outbreaks as 14 days to detect) were recorded.

Figure 8 provides the time to detect and overall detection rate for the outbreaks along rivers. GraphScan with a neighborhood size of $k = 15$ detects 2.00 days earlier than circular scan and detects 29.1% more of the outbreaks. ULS has similar performance to GraphScan for $k = 5$ and $k = 10$, but GraphScan delivers the overall best performance at $k = 15$, and outperforms ULS for almost all values of k. Similarly, Figure 9 provides the time to detect and overall detection rate for the outbreaks along the interstate corridors. GraphScan with a neighborhood size of $k = 15$ detects 1.97 days earlier than circles with fewer than half as many missed outbreaks.

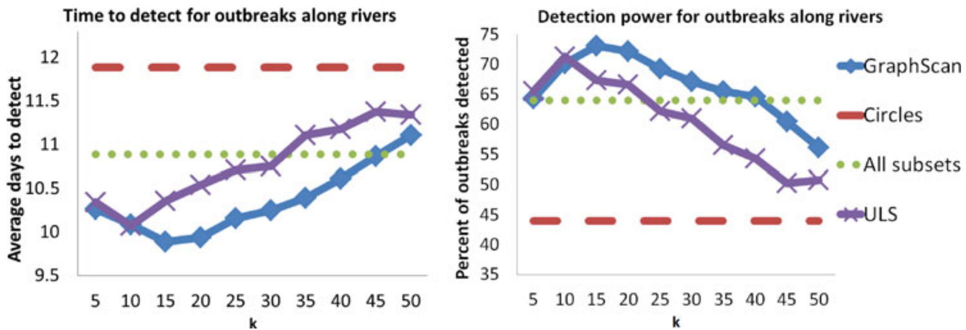


Figure 8. Detection time (average number of days to detect) and power at a fixed false positive rate of 1 per month for outbreaks along the rivers.

Downloaded by [Carnegie Mellon University] at 14:18 23 December 2015

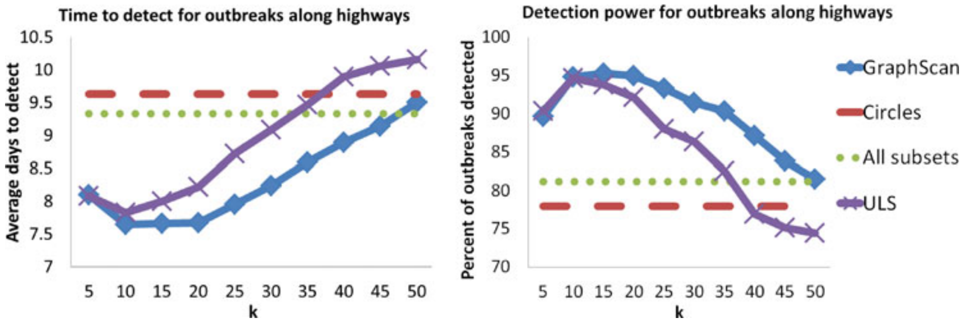


Figure 9. Detection time (average number of days to detect) and power at a fixed false positive rate of 1 per month for outbreaks along the highways.

6. LOCATING CONTAMINANTS IN A WATER DISTRIBUTION SYSTEM

Our second application of GraphScan focuses on locating contaminant plumes in a water distribution system equipped with noisy, binary sensors. The "Battle of the Water Sensor Networks" (BWSN) (Ostfeld et al. 2008) provided real-world data to teams tasked with placing *perfect* sensors to locate contaminants in the network of water pipes. Our work focuses on the complementary problem of fusing data collected from *noisy* sensors assuming a given placement and network structure to identify which locations have been contaminated.

We proceed by modeling simple, binary sensors at each of the 129 pipe junctions (graph nodes) in the system. We assume that a fixed false positive rate (e.g., FPR = 0.1) and true positive rate (e.g., TPR = 0.9) are known and that each sensor operates independently of the others in the network. This makes the expectation-based binomial (EBB) scan statistic (Kulldorff 1997) a logical scoring function to optimize. For fixed false and true positive rates, the EBB scan statistic becomes an *additive* function over the subset S. More specifically, $F_{\text{EBB}}(S) = \sum_{R_i \in S}(c_i \log(\frac{\text{TPR}}{\text{FPR}}) + (1 - c_i)\log(\frac{1-\text{TPR}}{1-\text{FPR}}))$ where sensor R_i produces a "trigger" $c_i \sim$ Bernoulli (FPR) under H_0 or $c_i \sim$ Bernoulli (TPR) under H_1. It can be trivially shown that additive functions satisfy LTSS with priority function $G(R_i) = F(R_i)$, and hence GraphScan can efficiently and exactly identify the highest scoring or *most positive* connected subgraph.

We use a graph radius r to define "local neighborhoods" of sensors (nodes). For example, a neighborhood with $r = 3$ would include the center node and all nodes within three edges of the center node. For a neighborhood radius of $r = 12$, GraphScan's average processing time on the water distribution network was 0.21 sec. With no neighborhood constraints, GraphScan was able to process the entire 129-node network in 0.04 sec.

We used 400 contaminant plumes provided in the BWSN data to generate sensor readings over the course of 12 one-hour intervals. As above, we present results for four competing methods: "Circles," "All Subsets," "ULS," and "GraphScan." In this setting, we note that All Subsets returns the subset consisting of all "triggered sensors" with $c_i = 1$, while ULS returns the largest connected subset of triggered sensors contained within a local neighborhood. For GraphScan and ULS, we report results as a function of the neighborhood

Downloaded by [Carnegie Mellon University] at 14:18 23 December 2015

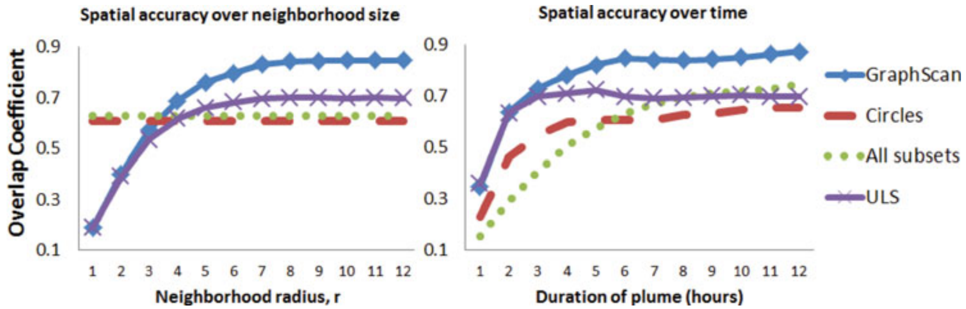


Figure 10. Spatial accuracy for contaminant plumes in a water distribution system. The left panel is accuracy as a function of neighborhood radius. The right panel is accuracy as a function of time since the beginning of the plume in hours.

Downloaded by [Carnegie Mellon University] at 14:18 23 December 2015

radius r. The fast-spreading contaminant plumes in this setting provide an easy detection task: all four methods detected the plumes very early with no significant differences in time to detect. Thus, we instead compare the spatial accuracy of the methods as measured by the "overlap coefficient," Overlap $= \frac{|\text{Affected} \cap \text{Detected}|}{|\text{Affected} \cup \text{Detected}|}$. Overlap $= 1$ corresponds to perfect agreement between the affected and detected subsets, while Overlap $= 0$ means that the affected and detected subsets are disjoint. Figure 10 presents the average spatial accuracy for each of the methods. The left panel shows accuracy as a function of neighborhood radius r at a fixed point in time (6 hr after the plume began). The right panel shows accuracy as a function of time, assuming a fixed neighborhood radius of $r = 10$.

We see that both GraphScan and ULS have higher spatial accuracy for larger neighborhood sizes, since the smaller neighborhoods fail to capture the entire plume. The connectivity constraints in GraphScan and ULS allow for relatively high precision (i.e., few noncontaminated sensors are included in the detected subset) even for larger neighborhood sizes. As compared to ULS, GraphScan's higher accuracy stems from its ability to correctly include contaminated sensors that did not trigger (false negatives) to connect clusters of true positives. ULS is unable to "bridge" these false negatives without also including all other sensors in the given neighborhood.

We note that the choice of neighborhood size k (or neighborhood radius r) substantially affects detection power and spatial accuracy. In practice, choice of k can be either based on prior knowledge of the expected size of the event of interest or based on labeled training data. In the former case, we recommend choosing the lowest k such that the event of interest is typically contained within a neighborhood of size k. In the latter case, the value of k can be chosen to maximize the metric of interest (detection power or accuracy) on the set of labeled training examples.

7. CONCLUSIONS

This work has provided a theoretical basis and practical implementation for scalable pattern detection in graph or network data. Linear-time subset scanning is a versatile tool able to speed up algorithms in many applications. However, in the spatial event detection domain, unconstrained LTSS performs poorly because it may return dispersed sets of

locations which we do not believe to be significant events. Therefore, we have implemented connectivity constraints allowing LTSS to scan over *connected* subsets of locations and increasing its power to detect irregularly shaped clusters of activity. Although similar to the previously proposed FlexScan algorithm, GraphScan is able to scale to much larger graphs, with a 450,000-fold increase in speed compared to FlexScan for neighborhoods of size $k = 30$.

These speed improvements come from two sources. First, we reduce the search space by excluding any subset that is provably suboptimal through the LTSS GraphScan property: "If subset S_{in} is included in the highest-scoring connected subset S, and removing S_{in} would not disconnect S, then no connected subset S_{out} adjacent to S can have higher priority than S_{in}." Second, we apply the unconstrained LTSS property to quickly compute an upper bound for the score of a route. If this bound is less than the score of an already known connected subset, then the entire route may be ignored. Branch and bounding improved the run time of GraphScan by an additional factor of 50x for moderately sized neighborhoods (e.g., $k = 50$).

We tested the GraphScan algorithm against the circular scan statistic proposed by Kulldorff (1997) and the upper level set scan statistic proposed by Patil and Taillie (2004) in two different scenarios. The first setting used synthetic disease outbreaks injected into real-world Emergency Department data from 97 zip codes in Allegheny County, Pennsylvania. Compared to the competing methods, GraphScan had higher detection power with shorter time required to detect the events, as well as fewer missed events overall. The second setting compared spatial accuracy of the methods for locating contaminant plumes spreading through a water distribution system equipped with 129 noisy, binary sensors. GraphScan demonstrated improved spatial accuracy and increased robustness to the occurrence of false negatives, when sensors failed to trigger.

Downloaded by [Carnegie Mellon University] at 14:18 23 December 2015

ACKNOWLEDGMENTS

This work was partially supported by the National Science Foundation grants IIS-0916345, IIS-0911032, and IIS-0953330. Edward McFowland III was also supported by NSF Graduate Research Fellowship GRFP-0946825 and an AT&T Labs Fellowship.

[Received March 2013. Revised July 2014.]

REFERENCES

Buckeridge, D. L., Burkom, H. S., Moore, A. W., Pavlin, J. A., Cutchis, P. N., and Hogan, W. R. (2004), "Evaluation of Syndromic Surveillance Systems: Development of an Epidemic Simulation Model," *Morbidity and Mortality Weekly Report*, 53, 137–143. [1028]

Duczmal, L., and Assuncao, R. (2004), "A Simulated Annealing Strategy for the Detection of Arbitrary Shaped Spatial Clusters," *Computational Statistics and Data Analysis*, 45, 269–286. [1016]

Erdös, P., and Renyi, A. (1959), "On Random Graphs I," *Publicationes Mathematicae*, 6, 290–297. [1026]

Flake, G. W., Lawrence, S., and Giles, C. L. (2000), "Efficient Identification of Web Communities," in *Proceedings of the 6th International Conference on Knowledge Discovery and Data Mining*, pp. 150–160. [1015]

Kulldorff, M. (1997), “A Spatial Scan Statistic,” *Communications in Statistics: Theory and Methods*, 26, 1481–1496. [1016,1018,1030,1032]

Kulldorff, M., Huang, L., Pickle, L., and Duczmal, L. (2006), “An Elliptic Spatial Scan Statistic,” *Statistics in Medicine*, 25, 3929–3943. [1016]

Kulldorff, M., and Nagarwalla, N. (1995), “Spatial Disease Clusters: Detection and Inference,” *Statistics in Medicine*, 14, 799–810. [1016]

Land, A. H., and Doig, A. G. (1960), “An Automatic Method of Solving Discrete Programming Problems,” *Econometrica*, 28, 497–520. [1024]

Leskovec, J., Krause, A., Guestrin, C., Faloutsos, C., VanBriesen, J., and Glance, N. (2007), “Cost-Effective Outbreak Detection in Networks,” in *Proceedings of the 13th International Conference on Knowledge Discovery and Data Mining*, pp. 420–429. [1015]

McFowland, E., Speakman, S., and Neill, D. B. (2013), “Fast Generalized Subset Scan for Anomalous Pattern Detection,” *Journal of Machine Learning Research*, 14, 1533–1561. [1018]

Neill, D. B. (2006), “Detection of Spatial and Spatio-Temporal Clusters,” Technical Report CMU-CS-06-142, Ph.D. thesis, Carnegie Mellon University, School of Computer Science. [1029]

——— (2009a), “An Empirical Comparison of Spatial Scan Statistics for Outbreak Detection,” *International Journal of Health Geographics*, 8, 20. [1029]

——— (2009b), “Expectation-Based Scan Statistics for Monitoring Spatial Time Series Data,” *International Journal of Forecasting*, 25, 498–517. [1018]

——— (2012), “Fast Subset Scan for Spatial Pattern Detection,” *Journal of the Royal Statistical Society*, Series B, 74, 337–360. [1014,1015,1017,1018]

Neill, D. B., and Moore, A. W. (2004), “Rapid Detection of Significant Spatial Clusters,” in *Proceedings of the 10th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 256–265. [1016]

Neill, D. B., Moore, A. W., Sabhnani, M. R., and Daniel, K. (2005), “Detection of Emerging Space-Time Clusters,” in *Proceedings of the 11th International Conference on Knowledge Discovery and Data Mining*, pp. 218–227. [1016]

Ostfeld, A., Uber, J., and Salomons, E., et al. (2008), “The Battle of Water Sensor Networks: A Design Challenge for Engineers and Algorithms,” *Journal of Water Resources Planning and Management*, 134, 556–568. [1030]

Patil, G. P., and Taillie, C. (2004), “Upper Level Set Scan Statistic for Detecting Arbitrarily Shaped Hotspots,” *Environmental and Ecological Statistics*, 11, 183–197. [1017,1032]

Tango, T., and Takahashi, K. (2005), “A Flexibly Shaped Spatial Scan Statistic for Detecting Clusters,” *International Journal of Health Geographics*, 4, 11. [1016,1018,1026]

Wallstrom, G. L., Wagner, M. M., and Hogan, W. R. (2005), “High-Fidelity Injection Detectability Experiments: A Tool for Evaluation of Syndromic Surveillance Systems,” *Morbidity and Mortality Weekly Report*, 54, 85–91. [1028]

Wang, B., Phillips, J. M., Schrieber, R., Wilkinson, D., Mishra, N., and Tarjan, R. (2008), “Spatial Scan Statistics for Graph Clustering,” in *Proceedings of the 8th SIAM International Conference on Data Mining*, pp. 727–738. [1015]

Downloaded by [Carnegie Mellon University] at 14:18 23 December 2015