Carnegie Mellon University

H. JOHN HEINZ III COLLEGE
School of Information Systems and Management

DISSERTATION
By
Skyler David Speakman

Submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy
in
Information Systems and Management

Fast Constrained Subset Scanning for Pattern Detection

September 24, 2014

Accepted by the
Dissertation Committee:

[illegible]	10/10/14
Professor Daniel B. Neill, Chair	Date
[illegible]	9/24/2014
Professor Christos Faloutsos	Date
[illegible]	Sept. 24, 2014
Professor Rema Padman	Date

Approved by the
Dean

[illegible]	10/20/2014
Ramayya Krishnan	Date

Fast Constrained Subset Scanning for Pattern Detection

Dissertation submitted by

Skyler Speakman

H. John Heinz III College, Carnegie Mellon University

Dissertation Committee:

Daniel B. Neill (Chair)

Christos Faloutsos

Rema Padman

September 2014

To my loving wife, Megan,

for her amazing support these past years.

Contents

List of Figures

List of Tables

List of Algorithms

Acknowledgements

I would like to give a special thanks to my advisor, Daniel Neill, for his motivation, guidance, and edits and re-edits of this work in its many forms.

Edward McFowland and Sriram Somanchi are more than just colleagues. I want to thank them for their code edits and hours spent at a white-board, but more importantly for their friendship.

Thanks to R. Ravi for providing additional insight to the development of the Additive GraphScan algorithm.

This work was partially supported by NSF grants: IIS-0916345, IIS-0911032, and IIS-0953330.

Chapter 1

Introduction and Background

*The contributions of this thesis lie at the intersection of novel pattern detection techniques, large-scale noisy data, and public health concerns.*The ability to detect patterns in massive data sets has multiple applications in diverse domains such as public health, law enforcement, human mobility, sustainability, smart power grids, sanitation, agriculture & food supply chains, and security. For example, spatial scan statistics are commonly used to alert public health officials to an unexpected increase in the number of Emergency Department complaints from patients in some spatial region (i.e., set of nearby zip codes) which may indicate the early stages of an emerging disease outbreak or bio-terrorist attack. Another example is the spread of contaminant plumes in a water distribution system equipped with noisy, binary sensors. Tracking, source-tracing, and predicting these dynamic events is a non-trivial pattern detection problem. The scope and complexity of these "societal scale" problems will soon be matched by vast amounts of data being generated by ubiquitous sensors and devices. The complementary factors needed to address these situations are data mining methods designed to detect relevant patterns that are emerging in the data and invoke a timely, targeted response. Although the focus of this thesis is on public health surveillance, the methods developed in this work are easily generalizable to any space-time data, including data both with and without an underlying graph structure.

The methodology of this work expands on the "subset scanning" approach to pattern detection. These techniques treat the detection problem as a search over subsets of data elements, with the goal of identifying the subset that best matches a pattern of interest, such as increased number of Emergency Room visits in nearby ZIP codes. Unlike "bottom-up" approaches such as density-based clustering [12] that find and aggregate individual anomalies, and "top-down" approaches that detect globally anomalous trends and then localize them to specific subsets of the data [14], subset scanning approaches maintain high detection power for both highly localized and global trends [28, 29].

However, subset scanning approaches pose two main challenges. First is appropriately evaluating the anomalousness of a given subset, and second is the computational issue of searching through the 2^N possible subsets of a data set containing N elements. Previous approaches such as spatial scan statistics [19, 33, 28, 29] have addressed the first concern by "scoring" each subset using a log-likelihood ratio statistic and identifying the highest-scoring subsets. For spatial data, the computational challenge of subset scanning has been addressed in several ways: limiting the search space to only consider regions of a given shape, such as circles [19] or rectangles [31, 44], or performing a heuristic search over subsets which is not guaranteed to find the most anomalous subsets [9, 1]. Such approaches enable efficient computation at the expense of reduced detection power and spatial accuracy [29].

The interplay between 1)appropriately evaluating a subset's anomalousness through probabilistically founded scoring functions and 2)developing algorithms to efficiently *scan over subsets of the data, finding those subsets which maximize the scoring functions subject to various constraints, are persistent themes throughout this thesis.* The rest of this chapter outlines previous work in spatial and spatial-temporal event detection, provides the necessary terminology and background information, and summarizes the contributions of the individual chapters.

1.1 Expectation-based Scan Statistics for Spatial Event Detection

Spatial event detection methods 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. A stream is represented as a series of counts x_i^t, from location s_i, and time step t. This stream of counts is also used to determine the historical baseline (expected count) μ_i^t for each location s_i at each time step t.

In the subset scanning framework, the goal is to identify a subset of the data $S \subseteq D$ that maximizes a score function $F(S)$. The counts and baselines of a subset S are aggregated such that $x_S = \sum_{s_i \in S} \sum_{t=1 \ldots W} x_i^t$ and $\mu_S = \sum_{s_i \in S} \sum_{t=1 \ldots W} \mu_i^t$ for some temporal window $W = 1 \ldots W_{max}$. Likelihood ratio statistics have been commonly used as score functions [19, 33, 28, 29]. The log-likelihood ratio statistic is defined as

$$F(S) = \log \frac{P(D \mid H_1(S))}{P(D \mid H_0)} \tag{1.1}$$

where the alternative hypothesis $H_1(S)$ assumes an event occurring in region $S \subseteq \{s_1, s_2, \ldots, s_N\}$ and the null hypothesis H_0 assumes that all counts are generated from the expected distribution (which can be spatially and temporally varying). In other words, it assumes that no events are occurring. For the expectation-based scan statistics [28], the alternative hypothesis $H_1(S)$ assumes that counts x_i are drawn with mean $q\mu_i$ inside region S and mean μ_i outside region S, for some constant multiplicative factor $q > 1$ known as the *relative risk* or severity. Therefore the ratio of the likelihoods of these two hypotheses provides the "score" of a region S, and the goal is to identify the most anomalous (highest-scoring) subset. Statistical significance of the highest-scoring subset, as well as other high scoring subsets, can be obtained by randomization testing. See [28] for details.

The log-likelihood ratio for the expectation-based scan statistic can be written as

$$F(S) = \max_{q>1} \sum_{s_i \in S} \left(\log P(x_i \mid q\mu_i) - \log P(x_i \mid \mu_i) \right). \tag{1.2}$$

The distributions from the exponential family are written as $\log P(x \mid \mu) = T(x)\theta(\mu) - \psi(\theta(\mu)) = T(x)\theta(\mu) - \mu\theta(\mu) + \phi(\mu)$, where $T(x)$ is the sufficient statistic, $\theta(\mu)$ is a function mapping the mean μ to the natural parameter θ, ψ is the log-partition function, and ϕ is the convex conjugate of ψ. Plugging this form of the exponential family into (1.2) gives

$$F(S) = \max_{q>1} \sum_{s_i \in S} \left(T(x_i) \left(\theta(q\mu_i) - \theta(\mu_i) \right) + \mu_i\theta(\mu_i) - q\mu_i\theta(q\mu_i) + \phi(q\mu_i) - \phi(\mu_i) \right). \tag{1.3}$$

This form of scoring functions plays an important role in past and current work in expectation-based scan statistics and constrained subset scanning.

1.2 Linear Time Subset Scanning

Linear Time Subset Scanning (LTSS) [29] is a recently developed approach to anomalous pattern detection that addresses the computational complexity of subset scanning by identifying the most anomalous subset of the data without requiring an exhaustive search, reducing computation time from years to milliseconds.

The subset scanning 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 data sets. For score functions satisfying the LTSS property [29], 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 j^{th}-highest priority record. For clarification, $R_{(1)}$ is considered 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, only these N subsets need to be considered 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. Any function $F(S)$ which is quasi-convex, increases with x_S, and is restricted to positive values of μ_S will satisfy the LTSS property [29].

Here is an alternative interpretation of the LTSS property that will be referenced the GraphScan algorithm described in Chapter 2. For any subset of locations S, if there exist locations $s_{in} \in S$ and $s_{out} \notin S$ such that $G(s_{in}) \leq G(s_{out})$, then $F(S) \leq \max(F(S \setminus \{s_{in}\}), F(S \cup \{s_{out}\}))$, and thus subset S is suboptimal. In other words, the score of the subset S could be increased by either including s_{out} or excluding s_{in}.

1.3 Enforcing Constraints on Subset Scanning

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. Enforcing hard constraints in subset scanning changes the search space (i.e. some subsets are excluded from consideration). For example, with hard connectivity constraints enforced, the search space only includes subsets that are connected in an underlying graph structure. In contrast, enforcing soft constraints changes the scoring function, penalizing some subsets and rewarding others (i.e., changing the definition of what it means for a subset to be anomalous). In particular, soft constraints in the subset scanning framework can be interpreted as the prior log-odds for a particular data record to be included in the highest scoring penalized subset [38].

1.3.1 Hard Connectivity Constraints

Chapter 2 provides a theoretical basis and practical implementation ("GraphScan") for scalable pattern detection in graph or network data. Constraints in the form of zip code adjacency allow scanning methods to scan over *connected* subsets of locations, increasing power to detect irregularly shaped clusters of activity along rivers or interstate highways. Although similar to the previously proposed FlexScan algorithm [40], GraphScan is able to scale to much larger graphs, with a 450,000-fold increase in speed compared to FlexScan for neighborhoods with 30 locations.

GraphScan was tested against the circular space-time scan statistic [19] and the upper level set scan statistic [35] on synthetic disease outbreaks injected into real-world Emergency Department data from 97 zip codes in Allegheny County, PA. 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.

1.3.2 Soft Constraints and Penalized Scoring Functions

Chapter 3 introduces and formalizes the Additive Linear Time Subset Scanning (ALTSS) property, which allows exact and efficient optimization of *penalized* likelihood ratio scan statistics over all subsets of data elements. ALTSS is incorporated into a Penalized Fast Subset Scan (PFSS) framework which enables the scan statistics to be efficiently optimized with or without including additional, element-specific penalty terms. The critical insight is that the scoring function $F(S)$ may be written as an additive function, summing over all data elements $s_i \in S$, when conditioning on the relative risk q. Moreover, only a small (linear rather than exponential) number of values of the relative risk q must be considered, making the computation of the highest scoring penalized subset $S^* = \arg\max_S \max_{q>1} F_{pen}(S \mid q)$ computationally tractable.

These additional element-specific penalty terms may be interpreted as the prior log-odds of a given record to be included in the highest scoring penalized subset. If the alternative hypothesis $H_1(S)$ is true for some subset S, the highest scoring penalized subset can be interpreted as a maximum a posteriori estimate of the true affected subset S. Critically, this extension opens up a wide range of well-founded founded probabilistic models to be incorporated into the subset scanning framework [38].

1.3.3 Soft Proximity Constraints

Chapter 4 provides a straight-forward example of the Penalized Fast Subset Scanning method in practice. Soft constraints on spatial proximity reflect the probabilistic model that for a given local neighborhood under consideration, locations closer to the center are assumed to be more likely to have been affected. This incorporation of prior knowledge as soft constraints is not possible in the original LTSS framework.

PFSS with soft proximity constraints is applied to the task of detecting anthrax bio-attacks, comparing its detection power and spatial accuracy to the current state of the art. PFSS demonstrates strong results, beating the traditional, circular spatial scan statistic [19] and the unpenalized Fast Subset Scan (FSS) [29] in both detection power and spatial accuracy.

1.3.4 Soft Temporal Consistency Constraints

Chapter 5 introduces the Dynamic Subset Scan for detecting, tracking, and source-tracing *dynamic* patterns that change the affected subset over time. Temporal consistency constraints are enforced on temporally adjacent, spatial subsets. These constraints are a fruitful compromise between traditional spatial-temporal scan statistics that do not allow the detected region to change over time (i.e., enforcing hard temporal consistency constraints) and the other extreme where temporal information is ignored. Critically, these temporal consistency constraints were derived to allow temporal information to be shared both forward and backward in time. Dynamic Subset Scan with temporal consistency and connectivity constraints provides a scalable solution for future work in dynamic pattern detection.

1.3.5 Soft Constraints with Hard Connectivity Constraints

Chapter 6 develops the Additive GraphScan algorithm, which allows the Dynamic Subset Scan to enforce both soft temporal consistency constraints *and* hard connectivity constraints while scaling to large, real world networks. Additive GraphScan is a fast, heuristic alternative to GraphScan introduced in Chapter 2. However, the results demonstrate an approximation ratio of over 99%, suggesting a very small sacrifice in detection accuracy for dramatic gains in speed and scalability.

The Dynamic Subset Scan with connectivity constraints is evaluated on data provided through the "Battle of the Water Sensor Networks" [3]. Dynamic scan succeeded in detecting contamination events sooner and tracking these events more accurately compared to other competing methods. These gains are due to Dynamic Scan's constrained flexibility: competing methods either failed to capture the dynamics of the spreading plume or were susceptible to over-fitting from lack of constraints. In scenarios with a weaker signal to be detected, incorporating information from a node's neighbors in the Dynamic Scan proved worthwhile, leading to substantial gains in performance on the detection, tracking, and source-tracing tasks.

1.4 Summary of Contributions

This thesis makes significant contributions to pattern detection through subset scanning in three broad categories: development of scalable methods, introduction of novel, probabilistic-based theory, and domains of application related to public health. Table 1.4 summarizes these contributions broken down by chapter.

	Method	**Theory**	**Application**
Chapter 2	GraphScan enforces connectivity constraints on the subset scan. It efficiently identifies anomalous connected subset of locations 450,000x faster than state-of-the-art.	GraphScan is guaranteed to identify most anomalous subset despite the large speed increase. GraphScan is not a heuristic.	GraphScan increases detection power of simulated respiratory outbreaks along highways and rivers compared to heuristics and unconstrained searches.
Chapters 3 and 4	Penalized Fast Subset Scanning with soft proximity constraints penalizes spatially dispersed clusters while rewarding dense clusters.	The ALTSS property allows penalty terms to be included in the subset scan while remaining exact and efficient. Only *O(n)* subsets need to be evaluated.	Soft proximity constraints increased detection power and spatial accuracy of simulated aerosolized anthrax bio-attacks released over a populated area.
Chapters 5 and 6	The Dynamic Scan detects patterns that change the affect subset over time. Additive GraphScan is a heuristic for maximizing additive functions over connected subsets.	Temporal consistency constraints are derived from a generative model to allow temporal information to be passed forward and backward in time.	The Dynamic Subset Scan increases source-tracing, tracking, and prediction ability for contaminant plumes spreading in a water distribution system equipped with noisy binary sensors.

Table 1.1: Summary of contributions.

Chapter 2

Fast Subset Scanning with Connectivity Constraints

This chapter proposes GraphScan, a new method for event and pattern detection in large data sets that have an underlying graph structure. Given a graph structure with vertices and edges $G = (V, E)$, and a time series of counts x_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 a concrete example of the application of GraphScan, we consider the problem of disease outbreak detection. In this setting, LTSS with proximity constraints [29] 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 non-compact 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.

To clarify, this approach to event detection on graph data differs in both form and function from other recent work in graph mining. For example, it is not attempting "community" or cluster detection [13]. Also, unlike [43], the anomalousness of the connected subsets 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, [23] is also concerned with detecting events in networked data. Their goal is to determine the optimal placement of a limited number 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 data sets with an underlying network structure.

GraphScan is applied 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. The goal is to find the most interesting spatial (or spatio-temporal) subset of locations S, subject to the connectivity constraints, by maximizing the score function $F(S)$.

This 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 [40]. 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 in order 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, simulated annealing has been used to detect clusters of homicides in a large urban data set to search over the space of connected subgraphs [9]. The Upper Level Set scan statistic (ULS) [35] 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 [29].

Neill [29] 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 non-trivial 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 the 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).

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 data sets. For score functions satisfying the LTSS property [29], 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 j^{th}-highest priority record. 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 [29] shows that, if there exist locations $R_{in} \in S$ and $R_{out} \notin S$ such that $G(R_{in}) \leq G(R_{out})$, then $F(S) \leq \max(F(S \setminus \{R_{in}\}), F(S \cup \{R_{out}\}))$, 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 $S_{in} \subseteq S$ and S_{out}, $S \cap S_{out} = \emptyset$, such that $G(S_{in}) \leq G(S_{out})$, then $F(S) \leq \max(F(S \setminus S_{in}), F(S \cup S_{out}))$, and thus subset S is suboptimal.

When connectivity constraints are introduced, the above inequality between subsets S, $S \setminus S_{in}$, and $S \cup S_{out}$ still holds. However, for a connected subset S, the subsets $S \setminus S_{in}$ and $S \cup S_{out}$ may not be connected. Thus S is only guaranteed to be suboptimal if two conditions hold: i) simultaneously removing all records $R_i \in S_{in}$ 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 $R_i \in S_{out}$ 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 [29] proves 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 [19], also used as the score function for the FlexScan algorithm [40], 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 [28] 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 non-parametric scan statistics, as described in [24], and GraphScan can be used to detect connected clusters in these settings as well.

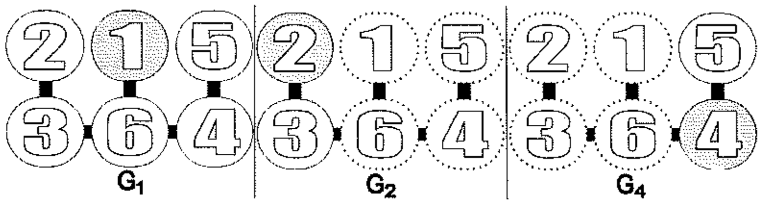

Figure 2.1: A graph broken into 3 subgraphs, one for each seed record (darkened). Nodes with dashed bevels are not included in a given subgraph. In G_2, $R_{(6)}$ has been removed because it is a neighbor of $R_{(1)}$. We remove $R_{(4)}$ and $R_{(5)}$ because they can no longer be reached from $R_{(2)}$. 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.

2.1 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.

2.1.1 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 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 2.1.

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 k^{th} bit, X_k, represents the inclusion or exclusion of the k^{th} 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.

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

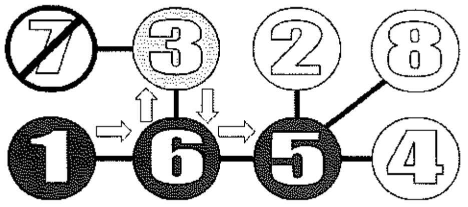

Figure 2.2: A possible route for an 8 node subgraph. The number in each node represents the node's priority ranking. The current subset is $[1, ?, 1, ?, 1, 1, 0, ?]$, and the current path is $[1, 6, 5]$. $S_{in} = \{R_{(3)}\}$ with priority 3.5, because $R_{(3)}$ is included in the subset and removing it would not disconnect the subset. $S_{out} = \{R_{(7)}\}$ 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 re-inserted 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}.

2.1.2 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 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

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

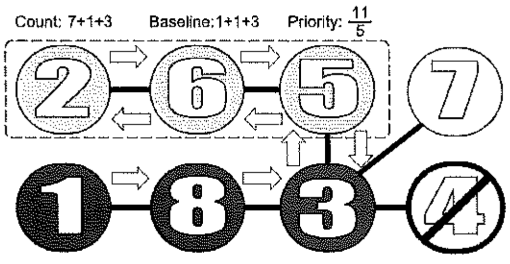

Figure 2.3: A possible route for an 8 node subgraph. This example demonstrates aggregating counts and baselines during the backtracking step of the GraphScan algorithm. Currently, $S_{out} = \{R_{(4)}\}$, with a priority of 3, and $S_{in} = \{R_{(2)}, R_{(6)}, R_{(5)}\}$, with a priority of $\frac{11}{5} = 2.2$. Note that $R_{(5)}$ has a priority of $\frac{3}{3} = 1$ when considered by itself. However, we cannot assign $S_{in} = \{R_{(5)}\}$ 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} ($2.2 < 3$).

of S_{in} then this new route is *not* reinserted into the queue because it represents a provably suboptimal subset of records. See Figure 2.2 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 2.3 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, i.e. 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.

Algorithm 2.1 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.

2.1.3 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 $IN(S)$ denote the set of all non-empty subsets

Algorithm 2.1 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.
```

$S_{in} \subseteq S$ such that $S \setminus S_{in}$ is connected (or empty), and $OUT(S)$ denote the set of all non-empty subsets S_{out} such that $S \cap S_{out} = \emptyset$ and $S \cup S_{out}$ is connected. We can then prove the following theorems:

Lemma 2.1. *For any connected subset S, if there exist $S_{in} \in IN(S)$ and $S_{out} \in OUT(S)$ such that $G(S_{in}) \leq G(S_{out})$, then subset S is suboptimal.*

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

Theorem 2.1. ***Exactness of Subgraph Creation.*** *For any connected subset S that is pruned by the subgraph creation process described in §2.1.1, 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 $R_{out} \notin S$ which is adjacent to S and has higher priority than all records in S. The suboptimality of region S follows from applying Lemma 2.1 with $S_{in} = S$ and $S_{out} = \{R_{out}\}$. More precisely, we know that $F(S) \leq F(S \cup \{R_{out}\})$ and that $S \cup \{R_{out}\}$ 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 2.2. ***Exactness of Route Propagation.*** *For any connected subset S that is pruned by the route propagation process described in §2.1.2, 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 $X_k = 0$). Let $\mathbf{S}$ denote the set of all subsets still under consideration for the current route, i.e., all subsets S such that $S_{incl} \subseteq S$ and $S \cap S_{excl} = \emptyset$. 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 $G(S_{in}) \leq G(S_{out})$, where $S_{in} \subset S_{incl}$ is a sidetrack and $S_{out} \subseteq S_{excl}$ 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 $S_{in} \in IN(S)$, since $S_{in} \subset S_{incl} \subseteq S$ and removal of the sidetrack S_{in} will not disconnect S. Also, we know that $S_{out} \in OUT(S)$, since $S_{out} \in OUT(S_{incl})$ and $S \cap S_{out} = \emptyset$. These facts imply that S is suboptimal by Lemma 2.1, 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 $G(\{R_{(k)}\}) \leq G(S_{out})$, 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 $G(S \setminus S_{incl}) \leq G(S_{out})$.

Since S_{incl} is connected, we know that $S \setminus S_{incl} \in IN(S)$. Also, we know that $S_{out} \in OUT(S)$, since $S_{out} \in OUT(S_{incl})$ and $S \cap S_{out} = \emptyset$. Thus S is suboptimal by Lemma 2.1, as its score would be improved by either excluding $S \setminus S_{incl}$ or including S_{out}. □

2.1.4 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 $R_i \in S_{out}$ 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 $S_{out} = \{R_{(i)}, R_{(k)}\}$, 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,

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

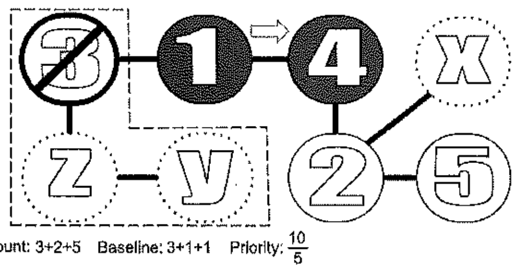

Figure 2.4: A possible route for a 5 node subgraph with additional information from records excluded from the subgraph. Naively, we would use $S_{out} = R_{(3)}$ with a priority of $\frac{3}{3} = 1$. During the creation of the subgraph, it is noted that nodes R_y and R_z (their priority ranking does 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 $S_{out} = R_{(x)}$.

rather than the priority of a single excluded record.

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 pre-calculated 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 pre-calculated priorities rather than relying solely on the single highest priority excluded record. See Figure 2.4 for more details.

2.1.5 Branch and Bounding with unconstrained LTSS

The unconstrained LTSS property of scoring functions is applied in the GraphScan algorithm through *Branch and Bounding* [21]. 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 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 (~23% faster than random ordering). Runtimes for GraphScan with and without Branch and Bounding are provided below. While Branch and Bounding substantially improves runtime, GraphScan is much more computationally efficient than FlexScan even without Branch and Bounding.

2.1.6 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 data set 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.

The implementation of proximity constraints within GraphScan is similar to the constraints used in FlexScan [40], 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. If desired, inclusion of the center record can be added as an additional constraint into the

GraphScan algorithm, immediately pruning any route for which that bit X_k is set to 0. This results in slightly faster runtime and does not impact detection power.

2.2 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 Erdos-Renyi random graphs of varying size n and edge probability p. Erdos-Renyi graphs are formed by placing each of the $\binom{n}{2}$ possible edges in the graph with probability p. Figure 2.5 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 Erdos-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 hour. 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-hour 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 seconds each. However, 2.2% of the graphs exceeded 1 hour 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 Erdos-Renyi graphs, the edge probability p has theoretical thresholds that change the nature of the graph [11]. For example, when $p < \frac{1}{n}$, the entire graph is composed of smaller subgraphs 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

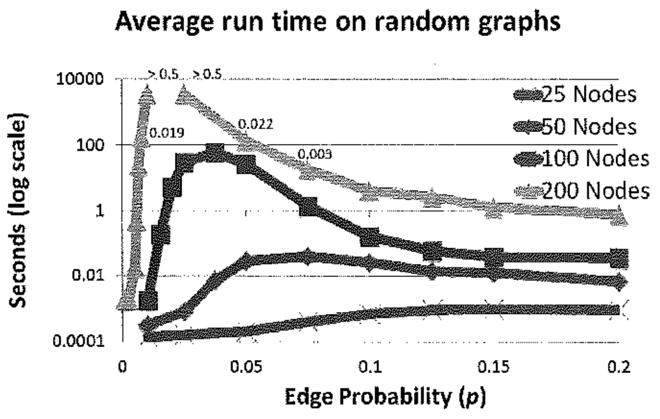

Figure 2.5: Performance of GraphScan on Erdos-Renyi random graphs of varying size and edge probability. Labeled data points are the proportion of graphs where run time exceeded 1 hour.

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.

2.3 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 original implementation of ULS did not incorporate proximity constraints, equivalent to using a neighborhood size $k = N$, but detection performance of both ULS and GraphScan can be improved with proximity constraints.

The Emergency Department data comes from ten Allegheny County hospitals from January 1, 2004 to December 31, 2005. The data was cleaned by removing records where the home zip code or admission date was missing or the home zip code was outside Allegheny County. By processing each case's ICD-9 code and free text "chief complaint" string, a count data set 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 data set 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.

Table 2.1 provides empirical results for ULS applied to the Emergency Department data from Allegheny County. While GraphScan is guaranteed to identify the highest-scoring connected subset S, ULS finds the highest-scoring connected subset only 1.1% of the time. This result marks a true distinction between these two methods: ULS reduces the search space for speed, requiring only seconds to process each day of data, while GraphScan only excludes subsets which are provably suboptimal,

Threshold as % of maximum score	Percentage of highest-scoring ULS meeting or exceeding threshold
100%	1.1%
90%	9.0%
80%	29.8%
70%	49.6%
60%	69.1%
50%	85.8%

Table 2.1: Empirical results of ULS on Emergency Department data.

guaranteeing that the highest-scoring connected region will be found.

2.3.1 Run time performances

In Figure 2.6, 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. GraphScan may also be used without incorporating proximity constraints, finding the highest-scoring

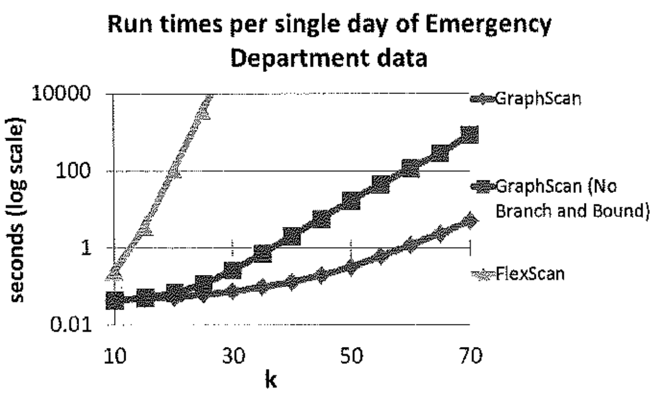

Figure 2.6: Run time analysis for FlexScan and GraphScan with and without Branch and Bounding. The x-axis denotes the "neighborhood size" as various values of k.

of all connected subsets. GraphScan averaged 2.50 seconds per day of ED data when processing the entire graph, while FlexScan would have required an estimated 5×10^{17} years. For outbreak detection, ignoring proximity constraints and attempting to detect patterns across the entire graph typically results in lower detection power. In this setting, proximity constraints serve to speed up run times as well as increase detection power.

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.

2.3.2 Simulating and Detecting Outbreaks

Our semi-synthetic 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 [7] and [42] combine current background data with that of past outbreaks to create a realistic new outbreak injected into current data. Settings such as outbreak duration and severity can be incorporated into the simulator and allow for a variety of outbreak scenarios. In this chapter, 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. The outbreak simulator requires three parameters: duration, severity, and the set of zip codes affected S_{inject}. For this work, I assume that every injected outbreak has a duration of $T = 14$ days and a severity of $\Delta = 1$. The start date of the outbreak is chosen uniformly at random. On each day t of the outbreak $t = 1 \ldots 14$, the simulator injects Poisson(t) cases over the affected zip codes S_{inject}.

We created six spatial injects that correspond to natural or man-made geographical features of Allegheny County, Pennsylvania, shown in Figure 2.7. Three of the regions are formed with zip codes along the Allegheny and Monongahela rivers, simulating a waterborne disease outbreak. The three other regions follow the path of two major U.S. interstates that traverse the county.

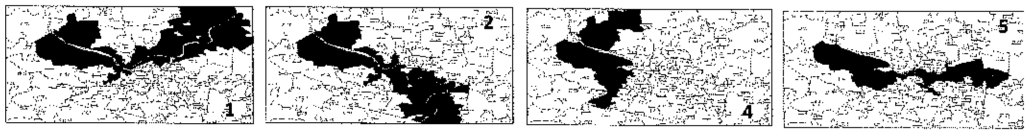

Figure 2.7: Outbreak regions used in the semi-synthetic tests. Regions #1 and #2 follow rivers and #4 and #5 follow interstates. #3 is the union of #1 and #2; #6 is the union of #4 and #5.

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 data set *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 [27]. 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. With the assumption that the non-injected data set is free of outbreaks that practitioners would have wanted to detect, this fraction represents the *false positive rate* that practitioners would have to accept in order to detect the outbreak on that particular day t. 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. A false positive rate of 1 per month is allowed, a level considered to be acceptable by many public health departments [26]. If no single day during a simulated outbreak has a proportion of false positives less than the rate r, then the outbreak would go undetected.

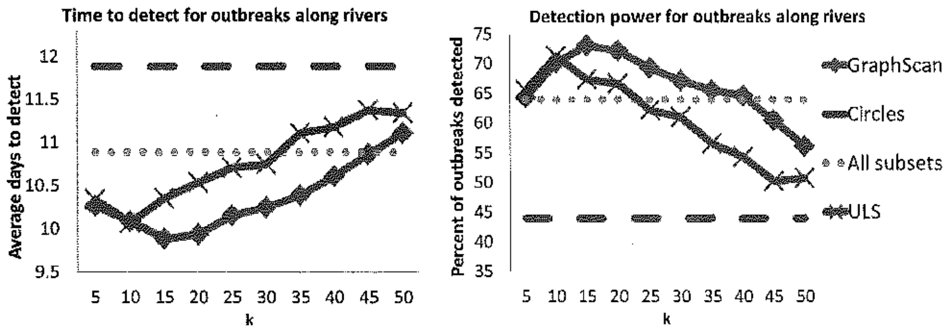

Figure 2.8: Detection time (average number of days to detect) and power at a fixed false positive rate of 1 per month for outbreaks along the rivers.

2.3.3 Detection Power Results

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 2.8 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 2.9 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.

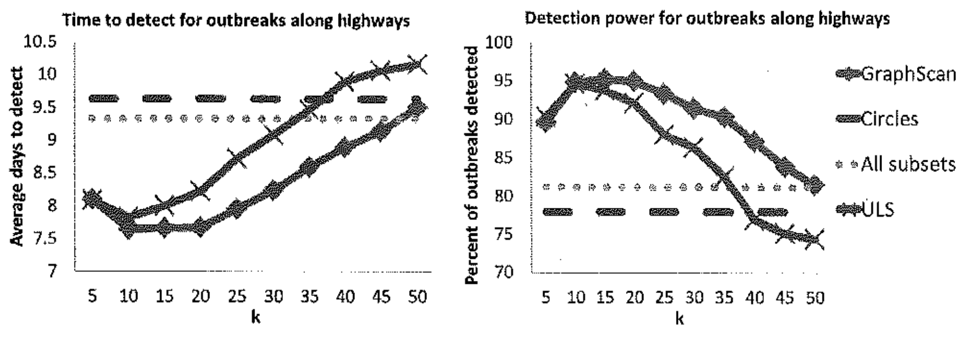

Figure 2.9: Detection time (average number of days to detect) and power at a fixed false positive rate of 1 per month for outbreaks along the highways.

2.4 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) [3] provided real-world data to teams tasked with placing *perfect* sensors to locate contaminants in the network of water pipes. The placement problem is an interesting one explored further in [5, 18]. This work focuses on the complementary problem of fusing data collected from *noisy* sensors assuming a given placement and network structure in order to identify which locations have been contaminated. Sensor fusion attempts to combine data from multiple distributed sensors in order to increase the detection power of the entire network [41].

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 [19] 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, $F_{EBB}(S) = \sum_{R_i \in S}(c_i \log(\frac{TPR}{FPR}) + (1-c_i)\log(\frac{1-TPR}{1-FPR}))$ 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 3 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 seconds. With no neighborhood constraints, GraphScan was able to process the entire 129 node network in 0.04 seconds.

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 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 2.10 presents the average

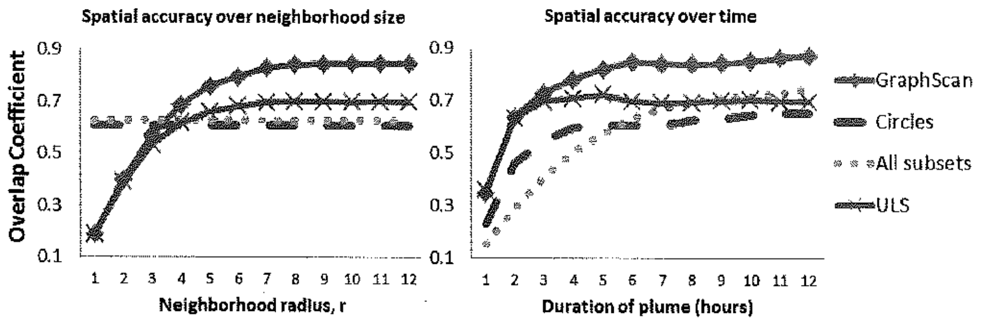

Figure 2.10: Spatial accuracy for contaminant plumes in a water distribution system. The left panel is accuracy as a function of neighborhood radius. The right panel is accuracy as a function of time since the beginning of the plume in hours.

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 hours 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 non-contaminated 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) in order 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.

2.5 Conclusions

This chapter 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 [19] and the Upper Level Set scan statistic proposed by Patil [35] 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, PA. 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.

Chapter 3

Additive Linear Time Subset Scanning and the Penalized Fast Subset Scan

This chapter introduces and formalizes a new property of scoring functions, Additive Linear-Time Subset Scanning (ALTSS), that allows incorporation of prior information about each data element's probability of inclusion in the highest scoring subset. This prior information may be interpreted as "soft" constraints on subset scanning methods. Certain types of hard constraints are possible to incorporate within the subset scanning framework: for example, the "fast localized scan" [29] enforces a hard constraint on spatial proximity by performing a separate, efficient search over the "local neighborhood" consisting of each spatial location and its $k - 1$ nearest neighbors. Similarly, GraphScan incorporates hard connectivity constraints by ruling out subsets that are disconnected in an assumed underlying graph structure, as outlined in Chapter 2. However, soft constraints (for example, a prior belief that some locations are more likely to be affected than others) cannot be easily incorporated. Given a score function satisfying the LTSS property, a penalized version of that score function is not guaranteed to satisfy LTSS, and thus can not efficiently identify the highest-scoring *penalized* subset. An example of this is provided at the end of Section 3.2.

In this chapter and the next, I highlight three contributions in the next two chapters that follow from the ALTSS property. The first is the Penalized Fast Subset Scan (PFSS) framework laid out in Section 3.3. PFSS is very general, enabling any element-specific priors to be incorporated into the search over subsets while maintaining computational efficiency and exactness.

The second contribution is an investigation of the connections between ALTSS and the Linear-Time Subset Scanning property (LTSS) [29]. More specifically, we show that scoring functions in the form of expectation-based scan statistics from the exponential family satisfy LTSS. This contribution extends LTSS, which was previously limited to the "separable" subfamily of the exponential family. Expectation-based scan statistics using the binomial and negative binomial distributions (which are not part of the separable subfamily) may now be efficiently optimized in their penalized and unpenalized forms.

The final contribution is a specific application of Penalized Fast Subset Scanning to spatial event detection, based on motivating examples from the fields of bioterrorism and disease surveillance. While the "fast localized scan" (subset scan with hard constraints on spatial proximity) has been shown to achieve high detection power and spatial accuracy in this setting [29], it does not take into account the spatial attributes of the locations beyond the "hard" proximity constraint of being one of the $k-1$ nearest neighbors of a center location, and considers each of the 2^k

subsets of the neighborhood equally likely.

Soft proximity constraints incorporate the prior expectation that locations closer to the center of an outbreak are more likely to be affected, thus rewarding spatial compactness and penalizing spatially dispersed clusters. We demonstrate that this approach increases both detection power and spatial accuracy as compared to the fast localized scan. Additionally, while fast localized scan achieves high performance for well-chosen values of the neighborhood size k, it performs worse than the standard, circular spatial scan [19] for badly chosen k. We demonstrate in Section 4.3 that incorporation of soft constraints enables our penalized version of the fast localized scan to be much more robust to the choice of k, while still guaranteeing that the most anomalous *penalized* subset of locations will be exactly and efficiently identified. This robustness to parameter selection is critical when a limited number of labeled training examples exist or when a public health surveillance system must be able to detect a wide range of possible outbreak types and threats.

3.1 Expectation-based Scan Statistics

We now review the use of expectation-based scan statistics [33] for spatial event detection. In the subset scanning framework, our goal is to identify a subset of the data $S \subseteq D$ that maximizes a score function $F(S)$. In the spatial event detection setting considered here, the dataset D consists of spatial time series data: observed counts x_i and expected counts μ_i at a set of spatial locations s_i $(i = 1 \ldots N)$ and possibly other parameters, such as the standard deviations σ_i. For example, a count x_i could represent the number of Emergency Department visits with respiratory complaints from a given zip code s_i on a given day. Likelihood ratio statistics have been commonly used as score functions [19, 33]. The log-likelihood ratio statistic is defined as $F(S) = \log(\Pr(D \mid H_1(S))/\Pr(D \mid H_0))$, where the alternative hypothesis $H_1(S)$ assumes an event occurring in region $S \subseteq \{s_1, s_2, \ldots, s_N\}$ and the null hypothesis H_0 assumes that no events are occurring. For the expectation-based scan statistics, the alternative hypothesis $H_1(S)$ assumes that counts x_i are drawn with mean $q\mu_i$ inside region S and mean μ_i outside region S, for some constant multiplicative factor $q > 1$ known as the *relative risk* or severity. We can then write the log-likelihood ratio for the expectation-based scan statistic as

$$F(S) = \max_{q>1} \sum_{s_i \in S} \left(\log \Pr(x_i \mid q\mu_i) - \log \Pr(x_i \mid \mu_i)\right). \tag{3.1}$$

A pivotal insight of our work is that for a *fixed* value of the relative risk q, the expectation-based scan statistics from the exponential family can be written as an *additive* set function over the data elements s_i contained in S. This insight leads to three useful consequences. First, additional penalty terms may be added *at the element level* (i.e., a bonus or penalty Δ_i for each element s_i) and the resulting penalized function will still be additive. Second, the highest scoring penalized subset can be efficiently identified by selecting only those data elements s_i making a positive contribution to the penalized scoring function. Finally, we show in Section 3.3.2 that only a small number of values of q must be considered, thus leading to efficient optimization of (penalized or unpenalized) score functions $F(S)$ over all $q > 1$.

3.2 The Additive Linear Time Subset Scanning Property

We now define the Additive Linear Time Subset Scanning (ALTSS) property. Informally, a score function $F(S)$ satisfies ALTSS if conditioning on the relative risk q allows the function to be written as an additive set function over the data elements s_i contained in S.

Definition 3.1. *For a given dataset D, the score function $F(S)$ satisfies the Additive Linear Time Subset Scanning (ALTSS) property if for all subsets $S \subseteq D$, we have $F(S) = \max_{q>1} F(S \mid q)$, where $F(S \mid q) = \sum_{s_i \in S} \lambda_i(q)$, and $\lambda_i(q)$ depends only on the given value of q, the observed count x_i, and expected count μ_i (and in some cases standard deviation σ_i) for element s_i.*

Theorem 3.1. *Expectation-based scan statistics from the (single parameter) exponential family satisfy the Additive Linear Time Subset Scanning property.*

Proof. Following the notation in [29], we write the distributions from the exponential family as $\log \Pr(x \mid \mu) = T(x)\theta(\mu) - \psi(\theta(\mu)) = T(x)\theta(\mu) - \mu\theta(\mu) + \phi(\mu)$, where $T(x)$ is the sufficient statistic, $\theta(\mu)$ is a function mapping the mean μ to the natural parameter θ, ψ is the log-partition function, and ϕ is the convex conjugate of ψ. Plugging this form of the exponential family into (3.1) gives

$$F(S) = \max_{q>1} \sum_{s_i \in S} \left(T(x_i)\left(\theta(q\mu_i) - \theta(\mu_i)\right) + \mu_i\theta(\mu_i) - q\mu_i\theta(q\mu_i) + \phi(q\mu_i) - \phi(\mu_i)\right). \tag{3.2}$$

Let $\lambda_i(q) = T(x_i)\left(\theta(q\mu_i) - \theta(\mu_i)\right) + \mu_i\theta(\mu_i) - q\mu_i\theta(q\mu_i) + \phi(q\mu_i) - \phi(\mu_i)$ and then $F(S)$ satisfies the ALTSS property. □

Table 3.1: Derivation of $\lambda_i(q)$ for expectation-based scan statistics in the exponential family.

Distribution	$\theta(q\mu_i)$	$\phi(q\mu_i)$	$\lambda_i(q)$
Poisson	$\log(q\mu_i)$	$q\mu_i \log(q\mu_i) - q\mu_i$	$x_i \log q + \mu_i(1-q)$
Gaussian	$\frac{q\mu_i}{\sigma_i^2}$	$\frac{(q\mu_i)^2}{2\sigma_i^2}$	$x_i\mu_i \frac{(q-1)}{\sigma_i^2} + \mu_i^2 \left(\frac{1-q^2}{2\sigma_i^2}\right)$
exponential	$-\frac{1}{q\mu_i}$	$-\log(q\mu_i)$	$\frac{x_i}{\mu_i}(1-\frac{1}{q}) - \log q$
binomial	$\log\left(\frac{q\mu_i}{n_i - q\mu_i}\right)$	$q\mu_i \log\left(\frac{q\mu_i}{n_i - q\mu_i}\right) +$ $n_i \log(n_i - q\mu_i)$	$x_i \log(q) + (n_i - x_i) \log\left(\frac{n_i - q\mu_i}{n_i - \mu_i}\right)$
negative binomial	$\log\left(\frac{q\mu_i}{r_i + q\mu_i}\right)$	$q\mu_i \log\left(\frac{q\mu_i}{r_i + q\mu_i}\right) -$ $r_i \log(r_i + q\mu_i)$	$x_i \log(q) + (r_i + x_i) \log\left(\frac{r_i + \mu_i}{r_i + q\mu_i}\right)$

Table 3.1 summarizes the derivation of $\lambda_i(q)$ for the expectation-based scan statistics in the exponential family.

An important consequence of scoring functions being written as additive functions over the data elements contained in the subset is that additional bonus or penalty terms Δ_i may be included for each data element s_i while maintaining the additive property.

Corollary 3.1. *Given a scoring function $F(S)$ that satisfies the ALTSS property, assume an additive bonus or penalty Δ_i for each $s_i \in S$. The resulting penalized score function, $F_{pen}(S) = F(S) + \sum_{s_i \in S} \Delta_i$, also satisfies ALTSS.*

Proof.

$$
\begin{aligned}
F_{pen}(S) &= F(S) + \sum_{s_i \in S} \Delta_i \\
&= \max_{q>1} F_{pen}(S \mid q) + \sum_{s_i \in S} \Delta_i \\
&= \max_{q>1} \sum_{s_i \in S} (\lambda_i(q) + \Delta_i) \\
&= \max_{q>1} \sum_{s_i \in S} \gamma_i(q)
\end{aligned}
$$

where $\gamma_i(q) = \lambda_i(q) + \Delta_i$ is referred to as the total contribution of data element s_i to the penalized scoring function for a fixed risk, q. Thus $F_{pen}(S) = \max_{q>1} F_{pen}(S \mid q)$, where $F_{pen}(S \mid q) = \sum_{s_i \in S} \gamma_i(q)$ is an additive set function. □

The Δ_i terms are assumed to be a function of the given data element s_i; they cannot depend on the entire subset S or the current value of q. We plan to investigate more sophisticated penalties in future work.

A second important consequence of scoring functions being written as additive functions is that the highest scoring subset for a fixed risk q can be easily identified.

Corollary 3.2. *For a fixed risk q, functions satisfying ALTSS can be efficiently optimized over all subsets $S \subseteq D$ by including all and only those data elements making a positive contribution to the scoring function, i.e., $s_i \in \arg\max_{S \subseteq D} F(S \mid q)$ if and only if $\gamma_i(q) = \lambda_i(q) + \Delta_i > 0$.*

The proof of Corollary 3.2 follows immediately from the fact that $F(S \mid q) = \sum_{s_i \in S} \gamma_i(q)$.

3.3 Penalized Fast Subset Scanning

Penalized Fast Subset Scanning (PFSS) is a novel method for scalable and accurate pattern detection which uses the Additive Linear-time Subset Scanning (ALTSS) property of commonly used scoring functions to incorporate prior information for each data element. This is in contrast to the Fast Subset Scanning method [29], which does not allow for additional terms to influence the subset's score and therefore considers each element equally likely to be included in the highest scoring *un*penalized subset. The first half of this section focuses on how the additional, element-specific terms are interpreted in the PFSS framework and the second half explains how the penalized scoring function may be exactly and efficiently optimized over all possible subsets.

3.3.1 Prior Log-odds Interpretation of Penalties Δ_i

We first show that the penalty terms Δ_i can be usefully interpreted as the prior log-odds that each data record s_i is affected. Let us assume a simple generative model where some subset of records $S_{true} \subseteq \{s_1, s_2, \ldots, s_N\}$ is affected, and each s_i is independently chosen to be included in S_{true} with prior probability p_i. We now consider the penalized score function $F_{pen}(S) = F(S) + \sum_{s_i \in S} \Delta_i$, where the log-likelihood ratio $F(S) = \Pr(D|H_1(S))/\Pr(D|H_0)$ and $\Delta_i = \log\left(p_i/(1-p_i)\right)$. Given the priors p_i, we show that this choice of Δ_i satisfies two useful properties: the highest-scoring penalized subset $S^* = \arg\max_S F_{pen}(S)$ minimizes the total probability of error, and is also a maximum a-posteriori (MAP) estimate of the true affected subset S_{true}.

First, when comparing the detected subset S^* and the true affected subset S_{true}, we wish to minimize both the probability of incorrectly including extra records (Type I error) and the probability of failing to detect truly affected records (Type II error). We show that the choice of $\Delta_i = \log\left(p_i/(1-p_i)\right)$ minimizes the sum of these two probabilities.

Theorem 3.2. *Let $\Delta_i = \log\left(p_i/(1-p_i)\right)$, where p_i is the prior probability that record $s_i \in S_{true}$. This choice of Δ_i minimizes the sum of the Type I and Type II error probabilities when comparing $S^* = \arg\max_S F_{pen}(S)$ and S_{true}.*

The proof of Theorem 3.2 is in Section 3.6.

Next, we show that $S^* = \arg\max_S F_{pen}(S)$ may be interpreted as the maximum a-posteriori (MAP) estimate of S_{true}.

Theorem 3.3. *Let* $\Delta_i = \log(p_i/(1-p_i))$*, where* p_i *is the prior probability that record* $s_i \in S_{true}$*. This choice of* Δ_i *makes* $S^* = \arg\max_S F_{pen}(S)$ *the maximum a-posteriori (MAP) estimate of the true affected subset* S_{true}*.*

Proof.

$$\begin{aligned}
\log \Pr(H_1(S) \mid D) &\propto \log \Pr(D \mid H_1(S)) + \log \Pr(H_1(S)) \\
&\propto F(S) + \log\left(\prod_{s_i \in S} p_i \prod_{s_i \notin S} (1-p_i)\right) \\
&= F(S) + \sum_{s_i \in S} (\log p_i - \log(1-p_i)) + \sum_{i=1}^{N} \log(1-p_i) \\
&= F(S) + \sum_{s_i \in S} \Delta_i - \sum_{i=1}^{N} \log(1+\exp(\Delta_i)) \\
&\propto F(S) + \sum_{s_i \in S} \Delta_i
\end{aligned}$$

where terms independent of S have been ignored. Thus choosing the subset S^* that maximizes $F_{pen}(S) = F(S) + \sum_{s_i \in S} \Delta_i$ also maximizes the posterior probability of $H_1(S)$ making S^* the MAP estimate of S_{true}. □

This Bayesian interpretation of the penalized maximum likelihood estimate should not be confused with the Bayesian and multivariate Bayesian spatial scan statistics [32, 30], which calculate marginal likelihoods and compute the total posterior probability that each subset S has been affected. While the Bayesian scan framework proposed in previous work has several benefits, including the ability to model and distinguish between multiple event types, it is limited to the assumption of Gamma-Poisson count data and cannot be easily generalized to other settings.

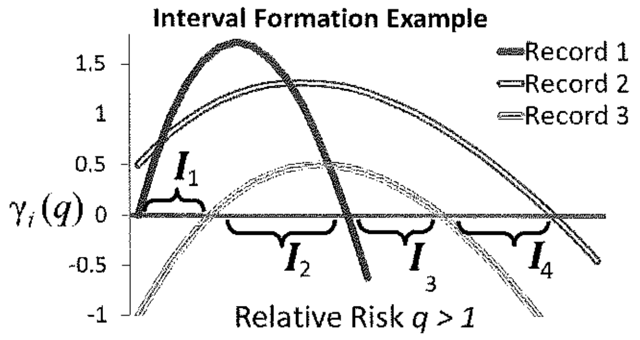

Figure 3.1: A three-record example of forming the $O(N)$ intervals needed to evaluate $F_{pen}(S\,|\,q)$. Throughout interval I_1, records 1 and 2 are making positive contributions and would be included in S_1^*. S_2^* would include all three records. S_3^* would include records 2 and 3, and S_4^* would include record 2 only. Further details: $x_1 = 130$, $\mu_1 = 110$, $\Delta_1 = 0$; $x_2 = 26$, $\mu_2 = 20$, $\Delta_2 = 0.5$; $x_3 = 40$, $\mu_3 = 30$, $\Delta_3 = -1$. $I_1 = [1, 1.132]$, $I_2 = [1.132, 1.3844]$, $I_3 = [1.3844, 1.557]$, and $I_4 = [1.557, 1.760]$.

3.3.2 Efficient Optimization of the Penalized Score Function

We now consider how the optimal *penalized* subset $S^* = \arg\max_{S \subseteq D} F_{pen}(S)$ can be efficiently computed. As noted above in Corollary 3.2, for a given value of the relative risk q, $F_{pen}(S \mid q)$ can be efficiently optimized over subsets by including all and only those data elements making a positive contribution to the penalized scoring function, i.e. those data elements with $\gamma_i(q) = \lambda_i(q) + \Delta_i > 0$. We now show that only linearly rather than exponentially many values of q must be considered:

Theorem 3.4. *The optimal subset* $S^* = \arg\max_S F_{pen}(S)$ *maximizing a penalized expectation based scan statistic from the exponential family may be found by evaluating only* $O(N)$ *subsets, where* N *is the total number of data elements.*

Proof. Let $\gamma_i(q) = \lambda_i(q) + \Delta_i$ as defined above, and assume that all Δ_i are independent of q. The first derivative $\gamma_i'(q) = \lambda_i'(q)$ has only one zero, obtained when $q = T(x_i)/\mu_i$ (the maximum likelihood estimate). Thus $\gamma_i(q)$ has at most two zeros. More precisely, we must have either a) there exists some q_i^{min} and q_i^{max} such that

$\gamma_i(q_i^{min}) = \gamma_i(q_i^{max}) = 0$ and $\gamma_i(q) > 0$ for all $q_i^{min} < q < q_i^{max}$, or b) for all q, $\gamma_i(q) \leq 0$. In the latter case, data element s_i will never be included in the highest-scoring penalized subset. Critically, we must consider at most $2N$ distinct values of q (this property also holds when restricting $q > 1$; see Figure 3.1 for an example using the penalized expectation-based Poisson scan statistic). We now sort these values of q (eliminating any duplicate q values) and let $I_1, \ldots, I_{2N}$ be the disjoint intervals formed by consecutive values of the sorted q. By construction, within each interval I_j, we have for each s_i that either $\gamma_i(q) > 0$ for all $q \in I_j$, in which case including s_i will increase the penalized score for all values of q in this interval, or $\gamma_i(q) < 0$ for all $q \in I_j$, in which case including s_i will decrease the penalized score for all values of q in this interval. Also, we note that if $q \notin \bigcup_{j=1}^{2N} I_j$, then $F_{pen}(S \mid q) \leq 0$ for all S, and hence we only need to evaluate the best subset for $q \in \bigcup_{j=1}^{2N} I_j$. We can write:

$$\begin{aligned} S^* &= \arg\max_S \max_{q>1} F_{pen}(S \mid q) \\ &= \arg\max_S \max_{q \in \bigcup_{j=1}^{2N} I_j} F_{pen}(S \mid q) \\ &= \arg\max_{j \in \{1,\ldots,2N\}} F_{pen}(S_j^*), \end{aligned}$$

where $S_j^* = \arg\max_S F_{pen}(S \mid q \in I_j)$. We can construct these sets efficiently as follows:

$$S_j^* = \{s_i : \gamma_i(q) > 0 \,\text{for all}\, q \in I_j\}$$

Note that S_j^* is the set of all elements that make positive contributions to the score $F_{pen}(S|q \in I_j)$ through $\gamma_i(q)$. Hence S_j^* is an optimal subset for any $q \in I_j$. Therefore we need to evaluate only $O(N)$ subsets (one for each interval) in order to find the optimal penalized subset S^*. $\square$

3.4 Relationship between ALTSS and LTSS

The Linear Time Subset Scanning property (LTSS) introduced in [29] enables exact and efficient optimization of *un*penalized score functions from the "separable" exponential family. In this section, we use insights from the Additive Linear Time Subset Scanning property (ALTSS) to expand on the LTSS property in two ways. First, we consider an alternative priority function which enables us to broaden the class of functions that satisfy LTSS to expectation-based scan statistics from the entire exponential family. Second, our PFSS framework introduced in Section 3.3 enables exact and efficient optimization of both penalized and unpenalized score functions, while LTSS applies only in the unpenalized case.

A function satisfies LTSS if and only if $\max_{S \subseteq D} F(S) = \max_{j=1...N} F(\{s_{(1)} \dots s_{(j)}\})$ where $s_{(j)}$ represents the j^{th} highest priority data element according to a provided priority function [29]. The highest scoring subset must be composed of the j highest priority data elements for some priority function $g(s_i)$ and some j between 1 and N. Neill (2012) defines a "separable" subfamily of the exponential family and proves that expectation-based scan statistics from the separable exponential family satisfy LTSS with the priority function $g(s_i) = \frac{x_i}{\mu_i}$. This ratio of observed counts to expected counts is also the maximum likelihood estimate of the relative risk, q, for the individual record s_i. This is referred to as q_i^{mle}.

The binomial distribution, while part of the exponential family, is not included in the separable exponential family. We show that the expectation-based binomial scan statistic cannot be efficiently optimized using the priority function q_i^{mle}. Consider a dataset with three elements $\{s_1, s_2, s_3\}$, where $(x_1, \mu_1, n_1) = (1500,\ 300,\ 4000)$; $(x_2, \mu_2, n_2) = (25,\ 8,\ 40)$; and $(x_3, \mu_3, n_3) = (12,\ 4,\ 40)$. The priority function $g(s_i) = \frac{x_i}{\mu_i} = q_i^{mle}$ suggests $\{s_1\}, \{s_1, s_2\}$, and $\{s_1, s_2, s_3\}$ as the three subsets to evaluate.

However, the highest scoring subset is $\{s_1, s_3\}$.

We now provide an alternative priority function that satisfies LTSS for unpenalized scoring functions from the single-parameter exponential family including the binomial and negative binomial distributions. Recall that for each record s_i there exists q_i^{min} and q_i^{max} such that $\lambda_i(q_i^{min}) = \lambda_i(q_i^{max}) = 0$. For unpenalized scoring functions ($\Delta_i = 0$), $q_i^{min} = 1$ for all i, while q_i^{max} is a function of the observed count x_i and expected count μ_i.

Theorem 3.5. *Unpenalized expectation-based scan statistics from the single-parameter exponential family satisfy the Linear Time Subset Scanning property with priority function $g(s_i) = q_i^{max}$, where q_i^{max} is the unique $q > 1$ such that $\lambda_i(q_i^{max}) = 0$.*

Proof. We denote the j^{th} highest priority record as $s_{(j)}$ with $s_{(1)}$ as the highest priority record and $s_{(N)}$ as the lowest. We write the priority of record $s_{(j)}$ as $g(s_{(j)}) = q_{(j)}^{max}$. Assume that the j^{th} priority record $s_{(j)}$ is included in the optimal subset S^*. It suffices to show that all higher priority records, $s_{(1)} \ldots s_{(j-1)}$, must also be included in S^*. By Theorem 3.1 we know that expectation-based scan statistics from the exponential family satisfy ALTSS and may be written as *additive functions* over the data elements contained in the subset for a fixed risk q. By Corollary 3.2 we know that if $s_{(j)} \in S^*$ then there exists a fixed relative risk q^*, where $q^* = \arg\max_{q>1} F(S \mid q)$, such that the j^{th} highest priority record is making a positive contribution at that risk, $\lambda_{(j)}(q^*) > 0$. Furthermore, we have $q_{(j)}^{min} = 1 < q^* < q_{(j)}^{max}$. Finally, consider any higher priority record $s_{(h)}$ and note that the priority ordering implies $q_{(h)}^{max} > q_{(j)}^{max}$. It follows that $s_{(h)}$ must also have $q_{(h)}^{min} = 1 < q^* < q_{(h)}^{max}$ which implies $\lambda_{(h)}(q^*) > 0$ and therefore $s_{(h)} \in S^*$. $\square$

In summary, using priority function $g(s_i) = q_i^{max}$, we have shown that inclusion

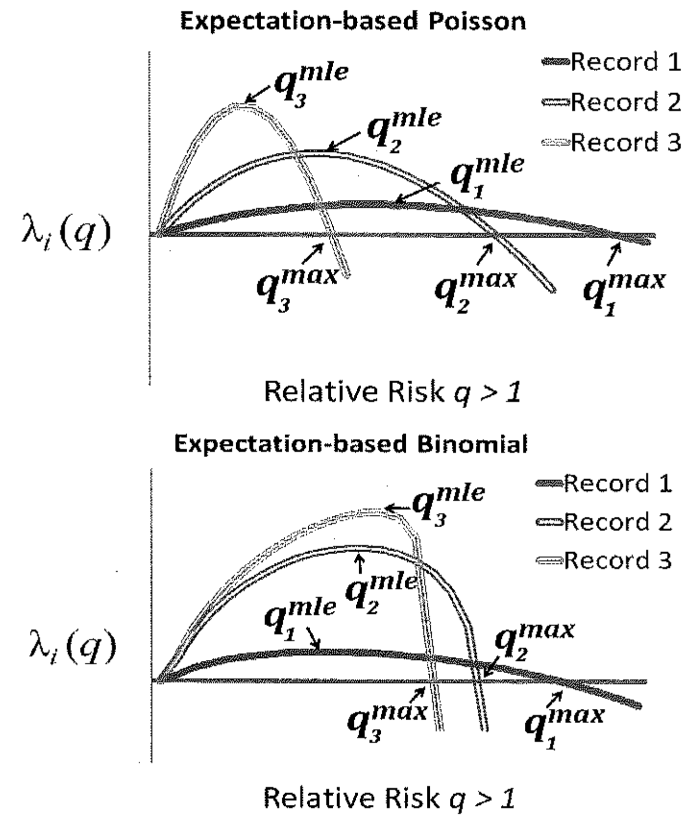

Figure 3.2: The top panel provides a 3-record example using the expectation-based Poisson scoring function which is a member of the "separable" exponential family. In this setting, both priority functions $g(s_i) = q_i^{mle}$ (introduced in Neill, 2012) and $g(s_i) = q_i^{max}$ (introduced in this work) satisfy LTSS. Note the same ordering produced by either function. Further details: $x_1 = 8$, $\mu_1 = 6$, $q_1^{mle} = 1.33$, $q_1^{max} = 1.74$; $x_2 = 35$, $\mu_2 = 28$, $q_2^{mle} = 1.25$, $q_2^{max} = 1.54$. $x_3 = 170$, $\mu_3 = 150$, $q_3^{mle} = 1.133$, $q_3^{max} = 1.28$; The bottom panel provides a 3-record example from the expectation-based binomial scoring function which is *not* a member of the "separable" exponential family. In this setting, the two priority functions result in different orderings. We prove in Theorem 3.5 that $g(s_i) = q_i^{max}$ is the correct priority ordering to satisfy LTSS for expectation-based scan statistics formed by distributions from the entire exponential family. Further details: $x_1 = 40$, $n_1 = 140$, $p_1 = 0.075$, $q_1^{mle} = 3.81$, $q_1^{max} = 7.95$; $x_2 = 125$, $n_2 = 190$, $p_2 = 0.15$, $q_2^{mle} = 4.39$, $q_2^{max} = 6.51$; $x_3 = 130$, $n_3 = 155$, $p_3 = 0.18$, $q_3^{mle} = 4.66$, $q_3^{max} = 5.555$.

of the j^{th} highest priority record in the highest scoring subset necessitates the inclusion of all higher priority records. Therefore, the optimal subset may be efficiently identified by sorting the records based on q_i^{max} and evaluating only the N subsets of the form $\{s_{(1)} \ldots s_{(j)}\}$ for $j = 1 \ldots N$. Figure 3.4 provides a visual comparison for the expectation-based Poisson and binomial scoring functions and the two priority functions q_i^{max} and q_i^{mle} discussed in this section.

Theorem 3.5 shows a connection between LTSS and ALTSS for unpenalized scoring functions. In contrast, we now provide a *penalized* scoring function that satisfies ALTSS but not LTSS. Consider maximizing the expectation-based Poisson scoring function with a penalty on subset size, $F_{pen}(S) = F_{EBP}(S) - |S|$. The unpenalized EBP scoring function satisfies the LTSS property, but including the size penalty violates LTSS, preventing the efficient optimization of the penalized scoring function over subsets of the data. Consider a dataset with three elements: $(x_1, \mu_1) = (5, 2)$, $(x_2, \mu_2) = (68, 55)$, and $(x_3, \mu_3) = (68, 55)$. Note that s_1 is the optimal penalized subset of $\{s_1, s_2\}$ so if $F_{pen}(S)$ satisfies LTSS, s_1 must be higher priority than s_2. However, the highest scoring penalized subset of $\{s_1, s_2, s_3\}$ is $\{s_2, s_3\}$. This implies that s_2 must be higher priority than s_1, which is a contradiction. Thus no priority function can exist for which $F_{pen}(S)$ satisfies the LTSS property.

This penalized scoring function *does* satisfy ALTSS: $F(S) = \max_{q>1} \sum_{s_i \in S}(\lambda_i(q) + \Delta_i)$ where $\lambda_i(q) = x_i \log q + \mu_i(1-q)$ from Table 3.1, and $\Delta_i = -1$ for all data elements s_i. This enables us to efficiently maximize the penalized scoring function in our new Penalized Fast Subset Scanning (PFSS) framework. Due to the Δ_i penalty terms, we no longer have $q_i^{min} = 1$ for all i. As shown in Theorem 3.3, this creates a *partitioning* over q instead of a *priority ordering* over q. The partitioning creates at most $2N$ intervals over the range of $q > 1$, and for each interval we need only to

Table 3.2: Summary of the LTSS and ALTSS comparisons.

Scoring functions	Priority function	Number of subsets to be evaluated	Notes
Separable exponential family with no penalty terms	a) $g(s_i) = \frac{x_i}{\mu_i}$ or b) $g(s_i) = q_i^{max}$	N	a) is from Neill, 2012. b) is proposed here.
Entire exponential family with no penalty terms	$g(s_i) = q_i^{max}$	N	We expand the class of scoring functions that satisfy LTSS.
Entire exponential family with penalty terms	No priority function satisfies LTSS.	$2N$	We introduce ALTSS to efficiently incorporate penalty terms.

consider the subset of records making a positive contribution to the scoring function. These $2N$ subsets are the only ones that must be evaluated to identify the highest scoring *penalized* subset in the PFSS framework. This partitioning of q intervals rather than use of a priority function differentiates the contributions from ALTSS in this work and LTSS in previous work. We conclude this section with Table 3.4 summarizing the comparison of LTSS and ALTSS.

3.5 Conclusion

This chapter introduced and formalized the Additive Linear Time Subset Scanning (ALTSS) property, which allows exact and efficient optimization of *penalized* likelihood ratio scan statistics over all subsets of data elements. ALTSS is incorporated into a Penalized Fast Subset Scan (PFSS) framework which enables the scan statistics to be efficiently optimized with or without including additional, element-specific penalty terms. The critical insight is that the scoring function $F(S)$ may be written as an additive function, summing over all data elements $s_i \in S$, when conditioning on the relative risk q. This form provides two notable advantages.

First, additional terms may be added to the statistic to represent the prior log-odds of each data element, while maintaining the additive structure of the scoring function. Second, optimization of either the scan statistic $F(S \mid q)$ or the penalized scan statistic $F_{pen}(S \mid q)$ over subsets can be performed very efficiently, by including all and only those records making a positive contribution to the score. Moreover, only a small (linear rather than exponential) number of values of the relative risk q must be considered, making the computation of the highest scoring penalized subset $S^* = \arg\max_S \max_{q>1} F_{pen}(S \mid q)$ computationally tractable.

These additional element-specific penalty terms may be interpreted as the prior log-odds of a given record to be included in the highest scoring penalized subset. (If the alternative hypothesis $H_1(S)$ is true for some subset S, the highest scoring penalized subset can be interpreted as a maximum a posteriori estimate of the true affected subset S.) Critically, this extension opens up a wide range of probabilistically founded models to be incorporated into the subset scanning framework.

3.6 Proof of Theorem 3.2: Minimizing Error with Δ_i

We show that if we can correctly estimate the prior probability p_i for location s_i to be in the affected subset S_{true}, then setting $\Delta_i = \log\left(\frac{p_i}{1-p_i}\right)$ (the prior log-odds) minimizes the total probability of error, including both Type I errors (including location s_i in the detected subset S^* when $s_i \notin S_{true}$) and Type II errors (failing to include a location $s_i \in S_{true}$ in the detected subset S^*).

Assume $s_i \in S_{true}$ with probability p_i, and that we observe $x_i \sim \text{Dist}_1$ if $s_i \in S_{true}$ and $x_i \sim \text{Dist}_0$ if $s_i \notin S_{true}$. Moreover, assume that λ_i and Δ_i are the log-likelihood ratio and penalty for location s_i respectively, where:

$$\lambda_i = \log\left(\frac{p(x_i \mid \text{Dist}_1)}{p(x_i \mid \text{Dist}_0)}\right)$$

and Δ_i can be any real number. The (unconstrained) penalized subset scan will include s_i in the detected subset S^* if and only if $\lambda_i + \Delta_i > 0$. We now show that the total probability of error is minimized for $\Delta_i = \log\left(\frac{p_i}{1-p_i}\right)$:

$$\Pr(\text{error} \mid \Delta_i) = \Pr(s_i \in S^* \mid s_i \notin S_{true}, \Delta_i)\Pr(s_i \notin S_{true})$$

$$+\Pr(s_i \notin S^* \mid s_i \in S_{true}, \Delta_i)\Pr(s_i \in S_{true})$$

$$= (1-p_i)\Pr((\lambda_i + \Delta_i > 0) \mid s_i \notin S_{true}) + p_i\Pr((\lambda_i + \Delta_i < 0) \mid s_i \in S_{true})$$

$$= (1-p_i)(1 - \text{CDF}_0(-\Delta_i)) + p_i\text{CDF}_1(-\Delta_i),$$

where the cumulative density functions CDF_0 and CDF_1 are defined as follows:

$$\text{CDF}_0(z) = \int_{-\infty}^{z} p(\lambda_i = k \mid x_i \sim \text{Dist}_0)\, dk,$$

$$\text{CDF}_1(z) = \int_{-\infty}^{z} p(\lambda_i = k \mid x_i \sim \text{Dist}_1)\, dk,$$

and we also define the corresponding probability density functions PDF_0 and PDF_1:

$$\text{PDF}_0(z) = p(\lambda_i = z \mid x_i \sim \text{Dist}_0),$$

$$\text{PDF}_1(z) = p(\lambda_i = z \mid x_i \sim \text{Dist}_1).$$

Furthermore, we note the key property

$$\frac{\mathrm{PDF}_1(z)}{\mathrm{PDF}_0(z)} = \exp(z),$$

since $\mathrm{PDF}_1(z)$ and $\mathrm{PDF}_0(z)$ are respectively sums of $p(x_i)$ for all x_i with corresponding $\lambda_i = z$, and for each such x_i, we know that

$$\frac{p(x_i \mid \mathrm{Dist}_1)}{p(x_i \mid \mathrm{Dist}_0)} = \exp(z).$$

We proceed by setting the first derivative of Pr(error) equal to 0:

$$\frac{d\Pr(\text{error})}{d\Delta_i} = (1 - p_i)\mathrm{PDF}_0(-\Delta_i) - p_i\mathrm{PDF}_1(-\Delta_i)$$

$$= (1 - p_i - p_i \exp(-\Delta_i))\mathrm{PDF}_0(-\Delta_i) = 0.$$

This expression has a single zero at $\Delta_i = \log\left(p_i/(1-p_i)\right)$. The second derivative at this point is:

$$-(1 - p_i - p_i \exp(-\Delta_i))d\mathrm{PDF}_0(-\Delta_i) + p_i \exp(-\Delta_i)\mathrm{PDF}_0(-\Delta_i)$$

$$= p_i \exp(-\Delta_i)\mathrm{PDF}_0(-\Delta_i)$$

$$= \left(\frac{1}{1 + \exp(\Delta_i)}\right)\mathrm{PDF}_0(-\Delta_i) \geq 0,$$

so this is the value of Δ_i that minimizes the probability of error.

Chapter 4

Enforcing Soft Proximity Constraints with the Penalized Fast Subset Scan

This chapter's contribution is a specific application of the penalized fast subset scanning (PFSS) framework to spatial event detection, based on motivating examples from the fields of bio-terrorism and disease surveillance. While the "fast localized scan" (subset scan with hard constraints on spatial proximity) has been shown to achieve high detection power and spatial accuracy in this setting [29], it does not take into account the spatial attributes of the locations beyond the "hard" proximity constraint of being one of the $k-1$ nearest neighbors of a given center location, and considers each of the 2^k subsets of the local neighborhood equally likely.

Incorporating "soft" proximity constraints allows subset scanning methods to account for the prior expectation that locations closer to the center of an outbreak are more likely to be affected, thus rewarding spatial compactness and penalizing spatially dispersed clusters. This approach increases both detection power and spatial accuracy as compared to the fast localized scan. Additionally, while fast localized scan achieves high performance for well-chosen values of the neighborhood size k, it performs worse than the standard, circular spatial scan [19] for badly chosen k. The incorporation of soft constraints enables the penalized version of the fast localized scan to be much more robust to the choice of k, while still guaranteeing that the most anomalous *penalized* subset of locations will be exactly and efficiently identified.

4.1 PFSS with Soft Proximity Constraints

As noted above, the Fast Localized Scan [29] performs separate, computationally efficient searches over subsets for each local neighborhood (center location s_c and its $k-1$ nearest neighbors), thus enforcing hard constraints on spatial proximity.

Penalized Fast Subset Scanning with soft proximity constraints allows us to take additional spatial information into account, rewarding spatial compactness and penalizing sparse regions *within* a local neighborhood. To that end, when considering a local neighborhood S_{ck} with center location s_c and neighborhood size k, we define Δ_i for each location $s_i \in S_{ck}$ as: $\Delta_i = h\,(1-(2d_i/r))$, where d_i is the distance between location s_i and the center location s_c, r is the neighborhood radius (distance from s_c to its $(k-1)^{th}$ neighbor), and $0 \leq h \leq \infty$ is a constant representing the strength of the soft proximity constraint. Through the prior log-odds interpretation of Δ_i, discussed in Chapter 3, we may interpret h as assuming that the center location $(d_i = 0, \Delta_i = h)$ is $\exp(h)$ times as likely to be included in the affected subset as its $(k-1)^{th}$ neighbor $(d_i = r, \Delta_i = -h)$. Figure 4.1 shows the probability of inclusion for locations that are a distance d_i from the center, across various values of h. Note that for $h = 0$, PFSS reduces to the original FSS solution because the

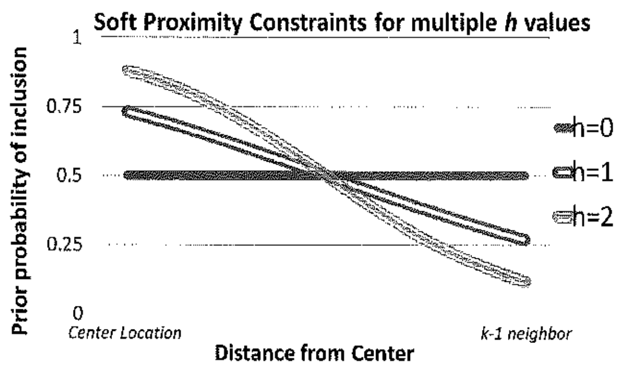

Figure 4.1: A location's prior probability of being included in the detected subset is based on both h and its distance from the center location. Note that the center location is assumed to be $\exp(h)$ times more likely to be included than the farthest ($k-1$ neighbor) location.

distance from the center location no longer influences the probability of inclusion, and thus all subsets of locations are considered equally likely. Incorporation of soft proximity constraints ($h > 0$) gives preference to more spatially compact clusters by rewarding locations that are closer to the center, while still considering all subsets within a given neighborhood. For very large h values, all locations with $d_i < r/2$ would have $\Delta_i >> 0$ and all locations with $d_i > r/2$ would have $\Delta_i << 0$, and thus PFSS reduces to the circular scan with fixed radius $r/2$.

Figure 4.2 illustrates the role of soft proximity constraint strength h when scanning through zip codes of Allegheny County, Pennsylvania, using the dataset described in Section 4.3 below. The three sets of zip codes are the highest scoring subsets for the *un*penalized FSS ($h = 0$), PFSS ($h = 1$), and PFSS ($h = 2$) for January 6th, 2005. Introducing the soft proximity constraint $h = 1$, as compared to the unpenalized $h = 0$ case, removes the spatially dispersed zip codes 15139, 15035, and 15148 while including zip codes 15211, 15219, and 15203, resulting in a more compact detection region. Increasing the strength to $h = 2$ results in an even more compact subset of zip codes by removing 15211 and 15203.

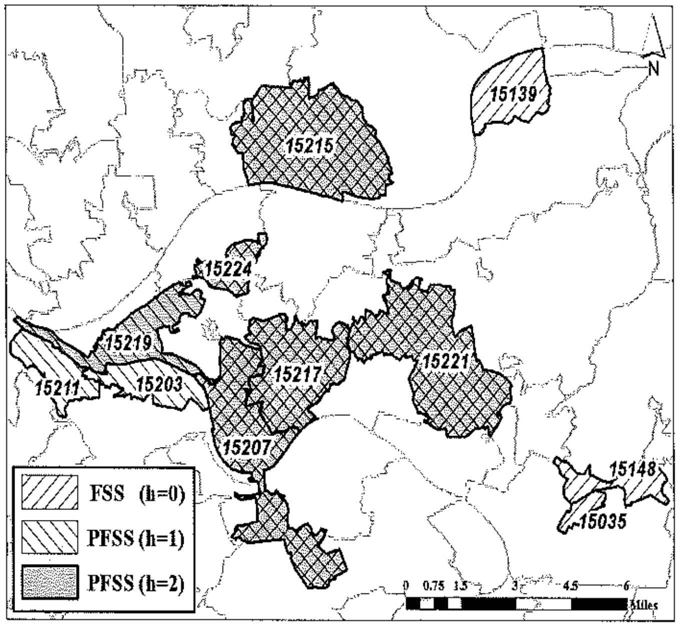

Figure 4.2: Three subsets of zip codes identified on January 6th, 2005 for various soft proximity constraint strengths.

We now present the PFSS algorithm with soft proximity constraints (Algorithm 4.1), which builds a local neighborhood of size k for each center location, then computes the penalties Δ_i and maximizes the penalized scoring function $F_{pen}(S)$ for each neighborhood. Then for each record within the neighborhood a Δ_i, q_i^{min}, and q_i^{max} are calculated. Recall from Chapter3 that q_i^{min} and q_i^{max} are the at most two risk values such that $\gamma_i(q_i^{min}) = \gamma_i(q_i^{max}) = 0$ where $\gamma_i(q) = \lambda_i(q) + \Delta_i$. In order to compare scores across different neighborhoods, we subtract the sum $\sum_{s_i \in S_{ck}} \log(1 + \exp(\Delta_i))$. This insures that $F_{pen}(S)$ is proportional to the log-posterior probability $\Pr(H_1(S) \mid D)$, and thus we maintain the interpretation of $S^* = \arg\max_S F_{pen}(S)$ as a MAP estimate (Theorem 3.3).

We conclude this section with a complexity analysis for Penalized Fast Subset Scanning. To find the optimal subset for a given neighborhood, we sort the at most

Algorithm 4.1 Penalized Fast Subset Scanning with soft proximity constraints.

```
1: for $c = 1 \dots N$ do
2:     Let $S_{ck}$ be center location $s_c$ and its $k-1$ nearest neighbors.
3:     for each $s_i \in S_{ck}$ do
4:         Compute $\Delta_i$, $q_i^{\min}$, and $q_i^{\max}$.
5:     end for
6:     $Q \leftarrow$ sort and remove duplicates($\{q_1^{\min}, q_1^{\max}, \dots, q_{2k}^{\min}, q_{2k}^{\max}\}$).
7:     If there exists any $q \in Q$ such that $q < 1$, exclude all $q < 1$ and add $q = 1$ to $Q$.
8:     $S \leftarrow \{\emptyset\}$.
9:     for $j = 1 \dots 2k$ do
10:        If $Q_j$ is a $q_i^{\min}$, then $S \leftarrow S \cup \{s_i\}$. If $Q_j$ is a $q_i^{\max}$, then $S \leftarrow S \setminus \{s_i\}$.
11:        Record $F_{pen}(S) = F(S) + \sum_{s_i \in S} \Delta_i$.
12:    end for
13:    Subtract $\sum_{s_i \in S_{ck}} \log(1 + \exp(\Delta_i))$ from $F_{pen}(S)$.
14: end for
15: Output the optimal subset $S^* = \arg\max_S F_{pen}(S)$.
```

$2k$ values of q, which is an $O(k \log k)$ operation, and step through the sorted values of q, which is an $O(k)$ operation. Over N neighborhoods, the total computational complexity of this algorithm is $O(Nk \log k)$. This assumes that the k-nearest neighbors have been pre-computed for each location, since this is a one-time operation; otherwise, computation of the k-nearest neighbors of each location can be done naively in $O(N^2 \log N)$ or more quickly using space-partitioning data structures. PFSS was able to identify the highest scoring penalized subset for a single day of our Emergency Department data described in Section 4.3 (with $N = 97$ locations) in 40-50 milliseconds for all values of $k = 5 \dots 50$, which is comparable to the runtimes of the original fast subset scan and the circular spatial scan.

4.2 Related Work

Penalized Fast Subset Scanning with soft proximity constraints combines penalized likelihood ratio statistics, spatial data, and subset scanning to increase detection power for irregularly shaped spatial clusters. The subset scanning approach is unique in separating this present work from methods that also use spatial information and attempt to optimize a penalized likelihood ratio statistic. For example, [45] penalizes non-connected search regions, while [10] and [20] compute the geometric regularity of the search region and penalize more elongated and irregularly-shaped clusters. More sophisticated methods combine geometric and non-connectivity penalties in a multi-objective framework [8]. However, most of these methods rely on a heuristic search to optimize the penalized scan statistic, which is computationally expensive and not guaranteed to identify the highest-scoring cluster, while Kulldorff et al. [20] limit their search to elliptical clusters, reducing detection power and spatial accuracy for any subsets that are not well-approximated by an ellipse. In contrast, our penalized fast subset scan approach is extremely computationally efficient and scalable while guaranteeing that the highest-scoring penalized subset will be found. It is also worth noting that the previously proposed methods focus on penalizing or rewarding properties of the region as a whole rather than individual data elements, while our penalties at the data-element level have a direct interpretation as the prior log-odds for each element's inclusion in the optimal subset (see Chapter ??). Either of these types of penalty could be preferable for a given application domain.

We note that the Additive Linear Time Subset Scanning property is distinct from prior work in submodular function optimization [34, 23], which has been used for sensor placement among many other applications. As shown by [29], the expectation-based Poisson statistic does not satisfy submodularity. Further, methods based on submodularity typically find approximate rather than exact solutions, while our approach is guaranteed to find the optimal subset that maximizes the penalized statistic.

4.3 Evaluation

We now provide a concrete example for the use of the Penalized Fast Subset Scanning method with soft proximity constraints in the public health surveillance domain. Emergency Department data from ten Allegheny County, Pennsylvania hospitals from January 1, 2004 to December 31, 2005 serves as the background data for both validation of the Penalized Fast Subset Scanning framework and a performance evaluation for detecting aerosolized anthrax bio-attacks. The background data was cleaned by removing records where the home zip code or admission date was missing or where the home zip code was outside Allegheny County. Through processing of each case's International Classification of Diseases (ICD-9) code and the free text in its "chief complaint" string, a count data set was created 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 data set had a daily mean of 44.0 cases, and a standard deviation of 12.1 cases. There were slight day-of-week trends, with counts peaking on Mondays, and seasonal trends, with counts peaking in February. The latitude and longitude coordinates of the centroid of each of the $N = 97$ zip codes formed the spatial component of the data set.

To validate the PFSS approach, we begin by examining a simple simulation that allows us to vary the size and spatial density of the affected region (i.e., subset of zip codes with additional counts injected into the background data) and thus understand the effects of these parameters on the relative performance of the competing methods.

We then evaluate the detection performance of PFSS using state-of-the-art dispersion models of an aerosolized anthrax release [16]. Both experiments and their results are discussed in their respective sections below. We use the expectation-based Poisson likelihood ratio statistic throughout, and compare three methods in each setting:

- Kulldorff's circular spatial scan statistic (Circles), which returns the highest scoring circular region, searching over all N distinct circles with neighborhood size k centered at the N locations [19].

- Fast Subset Scanning (FSS), which returns the highest scoring *un*penalized subset within a region consisting of a center location and its $k-1$ nearest neighbors for a fixed parameter k [29]. This can be considered a special case of the penalized fast subset scan with the strength of the soft proximity constraint $h = 0$.

- Penalized Fast Subset Scanning (PFSS) with soft proximity constraints, which returns the highest scoring *penalized* subset within a region consisting of a center location and its $k-1$ nearest neighbors for a fixed parameter k. The soft proximity constraints reward spatial compactness while penalizing sparse regions. We provide results for both weaker ($h = 1$) and stronger ($h = 2$) constraints. Choice of h is discussed below.

We provide two evaluation metrics for each of the competing methods: detection power (proportion of attacks or outbreaks detected) at a fixed false positive rate of 1 per year and spatial accuracy measured by the "overlap coefficient" between true and detected clusters. Overlap is a combination of precision and recall and requires two sets, S_{true} of affected locations and S^* of detected locations. Then the overlap coefficient is defined as: Overlap $= |S_{\text{true}} \bigcap S^*| / |S_{\text{true}} \bigcup S^*|$. An overlap coefficient of

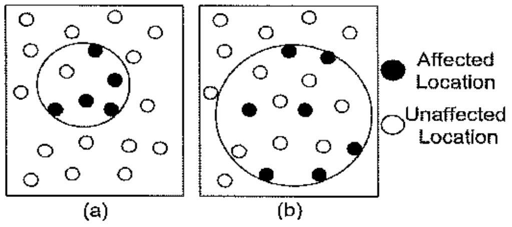

Figure 4.3: Two examples of inject density. Figure (a) is a region with a density of 5/6 and (b) is a region with density 7/14.

1 (or 100%) represents perfect precision and recall, while an overlap of 0 corresponds to disjoint sets S_{true} and S^*.

4.3.1 Validation on Simulated Outbreaks

We create a large set of simple simulated outbreaks for validation to compare the relative performance of Penalized Fast Subset Scanning (PFSS), Fast Subset Scanning (FSS), and the circular spatial scan (Circles) as a function of outbreak size, spatial density, and neighborhood size k. For each simulated outbreak, the simulator selects the affected subset of zip codes S_{true} uniformly at random (between 5 and 10 affected zip codes). Then Poisson($w_i\,|S_{\text{true}}|$) additional cases are injected into each location in S_{true}, where $w_i = c_i/(\sum_{s_j \in S_{\text{true}}} c_j)$ represents the relative "weight" of zip code s_i, proportional to the total number of cases in that zip code for the entire two years of Emergency Department data. The simulated outbreaks are categorized by spatial density, measured by the ratio of the number of affected locations to the total number of locations in the smallest circle that contains all affected locations and size, measured by the total number of affected locations. Figure 4.3 provides two examples. Results for 9 scenarios are provided; three categories of density (0.1-0.4 for "low", 0.4-0.7 for "medium", and 0.7-1.0 for "high") and three categories of size based on the number of affected zip codes (5-6 for "small", 7-8 for "medium", and

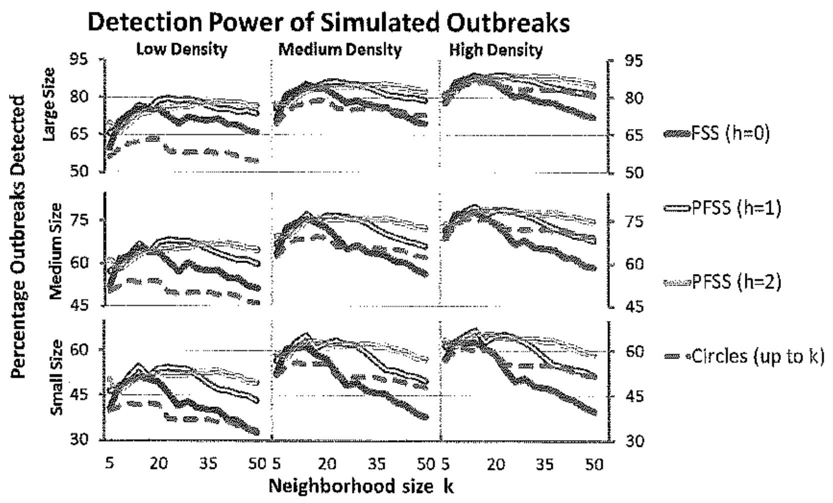

Figure 4.4: Comparison of detection power for multiple methods at a fixed false positive rate of 1 per year. Each panel represents different outbreak spatial density and size. Neighborhood sizes from $k = 5 \ldots 50$ are provided within each panel.

9-10 for "large"). This stratification by size and density is done for evaluation purposes and is independent from the generative model assumed by the soft proximity constraints in the Penalized Fast Subset Scanning method described above.

Figures 4.4 and 4.5 have the same layout with spatial density increasing between panels from left to right and outbreak size increasing between panels from bottom to top. Neighborhood size k increases within a panel. The lower left panel of few, highly-dispersed affected zip codes represents the most difficult detection scenario while the upper right panel of many, highly-compact affected zip codes reflects the easiest scenario.

Figure 4.4 provides a comparison of detection power (proportion of outbreaks detected at 1 false positive per year). First, as expected, the overall performance for all methods increases with the number and spatial density of the affected zip codes. Second, we note the poor performance of the spatial scan statistic (Circles) for the low density outbreaks. This is due to only scanning over circular regions, which results in much lower detection power for irregularly shaped clusters. The spatial scan statistic performs comparably in the high density outbreaks which are compact and close to circular in shape. Third, we examine the effect that the neighborhood size k has on the methods and note that the detection power of FSS is heavily influenced by the choice of k. The influence of k is more pronounced in outbreaks composed of few, compact affected zip codes (lower right panel). In contrast, we note that the detection power of PFSS remains strong for a wide range of neighborhood sizes, densities, and numbers of affected locations. Despite the lack of spatial structure in the low density outbreaks, the penalized methods (which reward spatially compact subsets) outperform the un-penalized method, FSS. We attribute this strong performance to PFSS's robustness to noise in the background data, increasing overall detection power. For large values of k, FSS is more likely to give high scores to spatially dispersed subsets in the background data, increasing the threshold needed to detect the simulated events, while PFSS will only identify such spurious clusters if they happen to be spatially localized.

Figure 4.5 provides a comparison of spatial accuracy. We note that larger, more spatially compact outbreaks result in higher spatial accuracy for all methods. The circular spatial scan statistic consistently underperforms FSS and PFSS, particularly for the low density clusters. It tended to return overly large circular regions with high recall but low precision, resulting in a low overlap coefficient. The robustness of the PFSS methods is shown again for the low density outbreaks. Although low density injects have a relative lack of the spatial structure that PFSS is designed to reward, the ability of PFSS to penalize sparse regions increases spatial precision while maintaining reasonably high recall, resulting in spatial accuracy that is comparable to FSS. This robustness to parameter selection is critical when a limited number of

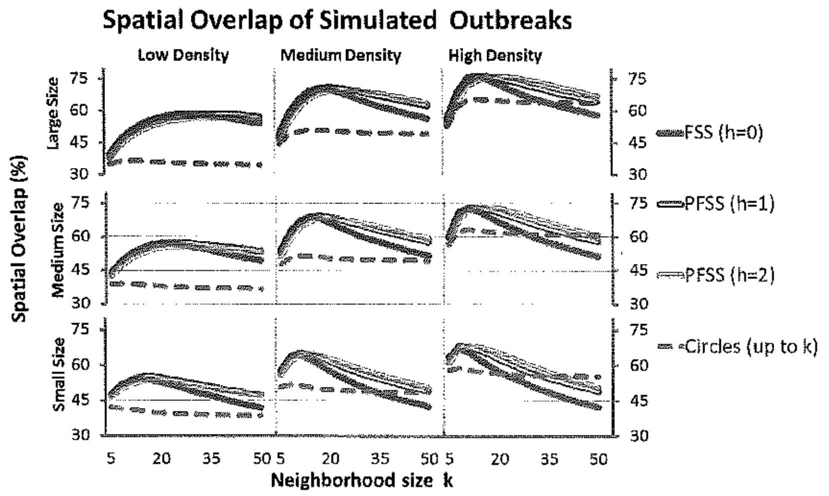

Figure 4.5: Comparison of spatial accuracy (overlap coefficient) for multiple methods. Each panel represents different outbreak spatial density and size. Neighborhood sizes from $k = 5 \ldots 50$ are provided within each panel.

labeled training examples exist or when a disease surveillance system must be able to detect a wide range of possible outbreak types and threats.

4.3.2 Evaluation on BARD Anthrax Attacks

The anthrax attacks are based on a state-of-the-art, highly realistic simulation of an aerosolized anthrax release, the Bayesian Aerosol Release Detector (BARD) simulator [16]. BARD uses a combination of a dispersion model (to determine which areas will be affected and how many spores people in these areas will be exposed to), an infection model (to determine who will become ill with anthrax and visit their local Emergency Department), and a visit delay model to calculate the probability of the observed Emergency Department visit counts over a spatial region. These complex simulations take into account weather data when creating the affected zip codes, S_{true}, and demographic information when calculating the number of additional Emergency Department cases within each affected zip code. The weather patterns

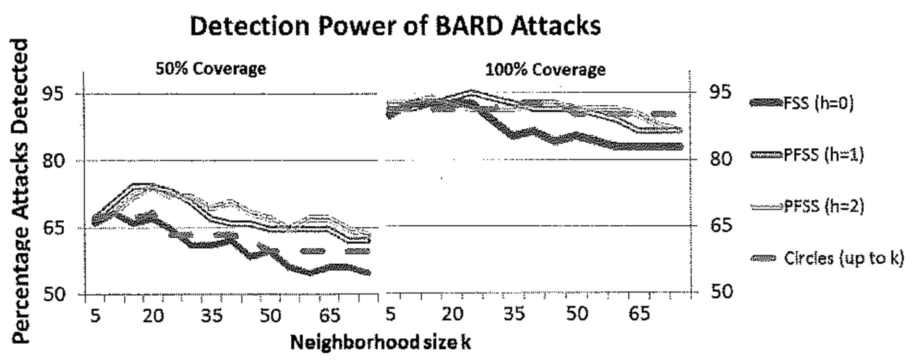

Figure 4.6: Comparison of detection power for multiple methods on simulated anthrax bio-attacks at a fixed false positive rate of 1 per year, for 50% and 100% coverage scenarios respectively. Neighborhood sizes from $k = 5 \ldots 75$ are provided within each panel.

are modeled with Gaussian plumes, resulting in elongated, non-circular regions of affected zip codes. Wind direction, wind speed, and atmospheric stability all influence the shape and size of the affected area. Although the simulator produces data for a 10-day period after the spores are released, we simplify the temporal component by using only the data from the midpoint (day 5) of the simulation.

For evaluation purposes, we consider two coverage scenarios. In the 100% coverage case, we assume that all of the anthrax victims present at an Emergency Department with a functioning bio-surveillance program and thus are appropriately accounted for. This assumption is extremely optimistic, so we also provide a possibly more realistic 50% coverage case where half of the population of anthrax victims seek medical attention from institutions that do not collect or share this type of data, thus reducing the strength of the outbreak signal accordingly and creating a more difficult detection problem.

Figure 4.6 provides a comparison of detection power for anthrax attacks. Intuitively, the optimistic 100% coverage scenario has higher detection rates for all methods. In the more difficult 50% coverage setting, the penalized scoring functions

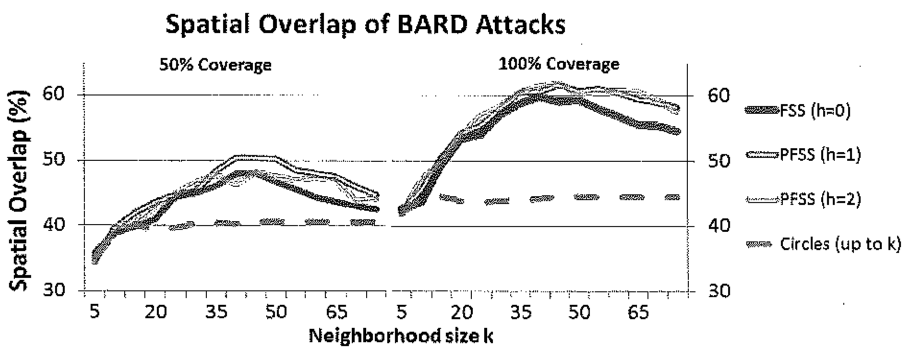

Figure 4.7: Comparison of spatial accuracy (overlap coefficient) for multiple methods on simulated anthrax bio-attacks, for 50% and 100% coverage scenarios respectively. Neighborhood sizes from $k = 5 \ldots 75$ are provided within each panel.

show higher detection rates and greater robustness to the choice of neighborhood size parameter, k. The unpenalized FSS method struggles for improperly chosen k even in the easier 100% coverage scenario.

Figure 4.7 provides a comparison of spatial accuracy for anthrax attacks. The strong performance of the subset scanning methods, PFSS and FSS, compared to the circular spatial scan is due to the elongated, non-circular regions (based on assumed, randomly generated wind direction and speed) of affected zip codes produced by the BARD simulation. The performance of 'Circles' is similar in the 100% and 50% coverage scenarios, suggesting that it is limited by the geometry of the circular spatial scan. Subset scanning methods (both penalized and unpenalized) are able to identify subsets of zip codes within a circular region and therefore have much higher spatial accuracy when detecting irregularly shaped clusters.

Figure 4.8 demonstrates PFSS's robustness to the choice of the proximity constraint strength, h, by comparing average detection power of the anthrax bio-attacks (averaged over neighborhood sizes $k = 5, 10, \ldots, 75$) for varying $h = 0 \ldots 7$. For this analysis, the 82 BARD-simulated anthrax attacks were split into separate training and test groups. The black cross represents the value of h that maximized average

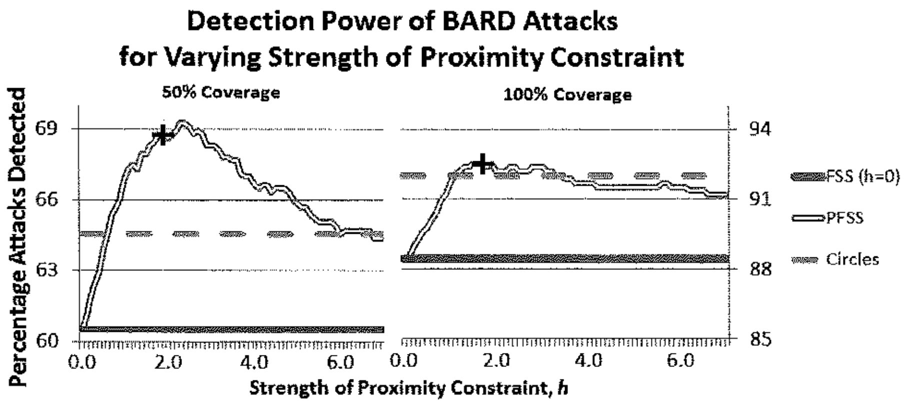

Figure 4.8: Comparison of detection power (averaged over all neighborhood sizes) for multiple methods on simulated bio-attacks, for 50% and 100% coverage scenarios. Soft proximity constraint strengths from $h = 0 \ldots 7$ are provided within each panel. The black marker represents the h that maximized average detection power for a separate training data set. Note different y-axis scale.

detection power for the training data set, while performance results are shown only for the separate test data set.

In both coverage scenarios, we note the strong performance of PFSS as compared to FSS for all values of $h = 0 \ldots 7$. This increased performance is a combination of the robustness of PFSS to choice of h and the sensitivity of FSS to poorly chosen neighborhood size, k. The circular scan is also robust to neighborhood size k and therefore performs comparably to PFSS in the 100% coverage scenario. We note that near-optimal values of h can be learned from a small number of labeled training examples. The learned $h = 1.8$ and $h = 1.7$, for 50% and 100% scenarios respectively, out-performed circles and FSS when evaluated on held out test data.

4.4 Conclusion

This chapter provided a concrete example and implementation of the Penalized Fast Subset Scan introduced in Chapter 3. "Soft" constraints on spatial proximity (i.e., for a given local neighborhood under consideration, locations closer to the center are assumed to be more likely to have been affected) were enforced in the subset scanning framework. This method was applied to the task of detecting anthrax bio-attacks, comparing its detection power and spatial accuracy to the current state of the art. PFSS with soft proximity constraints demonstrated strong results, outperforming the traditional, circular spatial scan statistic [19] and the unpenalized Fast Subset Scan (FSS) [29] in both detection power and spatial accuracy. Compared to Fast Subset Scan, PFSS showed remarkable robustness to selection of the neighborhood size k, and this robustness extended even to low density outbreaks designed to challenge the use of soft proximity constraints.

The PFSS framework with soft constraints introduced a parameter h for the strength of the spatial proximity constraint. The extreme cases of $h = 0$ and $h \to \infty$ correspond to the unpenalized FSS and a fixed-radius circular scan respectively. This work showed that near-optimal values of h can be learned from a small number of labeled training examples ($\sim$40). Additionally, PFSS demonstrated robustness to the choice of h, outperforming FSS for all values $h = 0 \ldots 7$.

Soft proximity constraints serve as one example of many different applications that can take advantage of including additional prior information in the subset scanning framework. Chapter 5 provides a more sophisticated example of incorporating prior information through temporal consistency constraints.

Chapter 5

Enforcing Soft Temporal Consistency Constraints with the Dynamic Subset Scan

Chapter 3 introduced the Additive Linear Time Subset scanning property which allows prior information to be incorporated into the subset scanning framework while maintaining computational efficiency and exactness. The resulting Penalized Fast Subset Scan was put in practice in Chapter 4 to enforce "soft proximity" constraints in the subset scan.

The next two chapters introduce the "Dynamic Subset Scan". Dynamic Subset Scanning is composed of two pieces. The first component, detailed in this chapter, is *temporal consistency constraints* which reward spatial subsets that are temporally consistent with each other, in order to detect dynamic clusters that change the affected region over time. The second component, explained in Chapter 6, enforces hard connectivity constraints on top of the soft temporal consistency constraints through the Additive Graphscan algorithm. Additionally, evaluation results of the Dynamic Subset Scan are provided in Chapter 6.

The motivating example comes from the field of public health; with a focus on source-tracing, tracking, and predicting contaminant plumes in a water distribution system. Creating sensor networks for detecting deliberate or accidental contamination of these systems has been a popular research domain following the terror attacks of September 11, 2001. The "Battle of the Water Sensor Networks" (BWSN) [3] provided real-world data to teams tasked with placing *perfect* sensors to quickly detect contaminants and limit the amount of contaminated water consumed by the population. The placement problem is an interesting one explored further in [5, 18]. This current work focuses on the complementary problem of fusing data collected from *noisy* sensors assuming a given placement. Sensor fusion attempts to combine data from multiple distributed sensors in order to increase the detection power of the entire network [41].

The simulation proceeds by modeling simple, binary sensors at each pipe junction (graph node) in the system with a fixed false positive rate (e.g., FPR $= 0.1$) and true positive rate (e.g., TPR $= 0.9$). An additional assumption is that each sensor operates independently of the others in the network. The simulations use the network structure and plumes provided in the BWSN data to generate sensor readings over the course of 12 one-hour intervals.

The task is then: Given (1) the graph structure (pipe network), (2) false positive and true positive rates of the sensors, and (3) independent observations from the sensors over time, the method(s) must provide: (A) hour-to-hour *tracking* of which pipe nodes have been affected by the plume on the current and recent past time steps, (B) *source-tracing* to determine which node(s) in the system spawned the contaminant, and (C) *predictions* of which nodes are likely to be affected in future time steps. Corresponding evaluation metrics include: (A) spatial-temporal overlap coefficient between the true and detected subsets of nodes over time in the past, (B) spatial overlap coefficient between the true and identified subsets of source nodes, and (C) spatial overlap between future time steps predicted by the tracking method and the true subset of nodes in the future. These metrics are defined below.

This is not the first work to apply spatial or subset scan statistics to contamination early warning systems. Koch and McKenna [17] used Kulldorff's spatial scan [19] to detect statistically significant circular clusters of anomalous activity. They used properties of the pipe network to create a distance metric based on travel time between sensing nodes in order to define their "circles". However, they were not able to enforce connectivity constraints and take advantage of the topology of the network. Through the Additive GraphScan algorithm detailed in Chapter 6, we are able to search over connected subsets of the pipe network to find anomalous connected subgraphs. Berry et al. [4] have also considered the detection power of a network of imperfect sensors, showing that it is worth deploying a sensor network even when individual sensors have low detection probability. However, their experiments did not allow for sensors with false positives, making the detection and source tracing problems much easier to solve as compared to the more difficult scenario considered here.

Spatial scan statistics attempt to identify regions of interest or "hot spots". This is achieved by maximizing a scoring function $F(S)$, typically defined as the likelihood ratio $F(S) = \frac{\Pr(\text{Data} \mid H_1(S))}{\Pr(\text{Data} \mid H_0)}$, over spatial regions S. In this expression $H_1(S)$ assumes increased activity in region S, and H_0 assumes regular behavior. This work monitors binary sensors s_i each producing $x_i \sim$ Bernoulli (FPR) "triggers" under H_0

or $x_i \sim$ Bernoulli (TPR) "triggers" for $H_1(S)$ containing node s_i. This makes the expectation-based binomial scan statistic [19] a logical choice.

Spatial-*temporal* scan statistics incorporate the time dimension. It is standard to aggregate this temporal information over a time window w so that $x_i = \sum_{t=1...w} x_i^t$. Once the temporal information has been aggregated for each window $w = 1 \dots W$, maximizing the spatial-temporal scan statistic for that window proceeds identically to the regular spatial scan statistic; it then maximizes over all window sizes from 1 to W. However, an inherent assumption in this aggregation of temporal information is that the affected spatial-temporal subset *does not change* over time. Therefore, this approach will be referred to as the *Static* scan method throughout this chapter and the next.

The fundamental goal of this work is to relax this strong assumption on the spatial-temporal structure in order to increase the power to detect *dynamic* patterns that change the affected region over time. One simple approach is to optimize each of the w time steps independently. This allows for each time step t to identify an entirely different spatial region, but does not allow the sharing of information between time steps, possibly reducing detection power. This approach is referred to as the *Independent* scan method throughout this chapter and the next.

As a compromise between Static and Independent methods, we propose the *Dynamic Subset Scan* which enforces temporal consistency constraints to allow temporally adjacent time steps to share information forward and backward in time. As demonstrated below, this flexibility increases power to track and predict dynamic patterns while scaling to the size of real-world networks.

These soft temporal consistency constraints enforced between temporally adjacent time steps should not be confused with the binary metric of temporal connectedness (i.e. two spatial subsets are temporally connected as long as they contain one node in common). The temporal connectedness approach to dynamic pattern detection is explored further in [6].

Other contributions in the field have also focused on the source-tracing task. "k-effectors" [22] use an information propagation model and dynamic programming to identify the original k nodes that produced a cascade in tree graphs. Net-sleuth [36] uses a susceptible-infected model along with minimum description length (MDL) to identify the original culprits. Critically, both of these methods require labeled data (affected and unaffected nodes) at later timesteps in order to source-trace the origins of the pattern at earlier timesteps.

This chapter has three sequential objectives. The first is to demonstrate how the expectation-based binomial (EBB) scoring function may incorporate additional constraints while remaining straightforward to optimize over all possible subsets S (i.e., we show that EBB can be written as an additive function over the data elements $s_i \in S$). These derivations are specific examples of Theorem 3.1 and Corollaries 3.1 and 3.2 applied to the expectation-based binomial scoring function.

Second is to provide the formal definition of temporal consistency constraints based on a probabilistic generative model that incorporates both forward and backward temporal consistency. We consider both the homogenous case, in which the propensity to propagate the pattern is the same across all edges in the network, and the heterogeneous case, in which these probabilities can vary across edges. The chapter concludes with a description of the iterative optimization process that "lines up" the spatial-temporal region according to the provided temporal consistency constraints.

5.1 Additive Scoring Function and Additional Terms

Conditioned on the false and true positive rates (FPR, TPR) of the sensors, the expectation-based binomial (EBB) statistic can be written as an additive set function over the data elements $s_i \in S$. This is an important feature for two reasons originally explained in Corollaries 3.1 and 3.2. First, additive functions are easy to optimize over all possible subsets. Without connectivity constraints, the score function $F(S)$ can be optimized over subsets of records by simply including all records making a positive contribution and excluding the rest. Determining the "most positive" *connected* subset is more complicated, and is covered in Chapter 6. Second, additive functions allow for additional penalty terms Δ_i to be included at the element level while the total *penalized* scoring function remains additive and thus amenable to efficient optimization.

Theorem 1. *The expectation-based binomial statistic may be written as* $F(S) = \sum_{s_i \in S} \lambda_i$, *where* λ_i *depends only on the binary sensor response* c_i *for sensor* s_i *(i.e., whether that sensor triggers or not) as well as the false and true positive rates of the sensors in general.*

Proof. The log-likelihood ratio form of the EBB scan statistic can be written as follows:

$$\begin{aligned}
F(S) &= \log \frac{\Pr(\text{Data}|H_1(S))}{\Pr(\text{Data}|H_0)} \\
&= \log \frac{\prod_{s_i \in S} \Pr(c_i \sim \text{Bernoulli}(\text{TPR}))}{\prod_{s_i \in S} \Pr(c_i \sim \text{Bernoulli}(\text{FPR}))} \\
&= \log \prod_{s_i \in S} \frac{(\text{TPR})^{c_i}(1-\text{TPR})^{1-c_i}}{(\text{FPR})^{c_i}(1-\text{FPR})^{1-c_i}} \\
&= \sum_{s_i \in S} \left[c_i \log\left(\frac{\text{TPR}}{\text{FPR}}\right) + (1-c_i)\log\left(\frac{1-\text{TPR}}{1-\text{FPR}}\right)\right]
\end{aligned}$$

Then $\lambda_i = c_i \log\left(\frac{\text{TPR}}{\text{FPR}}\right) + (1 - c_i)\log\left(\frac{1-\text{TPR}}{1-\text{FPR}}\right)$.

□

Next, assume a bonus or penalty Δ_i for each $s_i \in S$. These can easily be incorporated into the score function. Define:

$$F_{pen}(S) = F(S) + \sum_{s_i \in S} \Delta_i = \sum_{s_i \in S} (\lambda_i + \Delta_i) = \sum_{s_i \in S} \gamma_i.$$

$F_{pen}(S)$ is a penalized form of the EBB scan statistic that is still additive over the data elements s_i. Note that the Δ_i terms are assumed to be a function of only the given data element s_i; they cannot depend on the entire subset S. This is a limitation of the current work and will be investigated in extensions to more sophisticated penalties in future work.

5.2 Derivation of Δ_i^t for Temporal Consistency

The following subsections derive the formulas for Δ_i^t that correspond to two generative models for temporal consistency. In the first case, we assume "node homogeneity". This assumes that a node's propensity to propagate the plume to its neighbors is the same across all nodes and all neighbors. In the latter case, we allow this influence to vary from node to node and neighbor to neighbor. We refer to this method as "node heterogeneity". The homogenous nodes model was originally developed in [39].

In both cases, we emphasize the importance of forward and backward temporal consistency. The intuitive role of Δ_i^t is that it must *simultaneously* make the current subset, S^t, appear likely to have been generated from the past, S^{t-1}, and able to generate the future, S^{t+1}, thus conveying temporal consistency information both forwards and backwards in time. As discussed in Section 5.3, we will jointly optimize the detected subsets S^t for each time step t by an iterative approach that optimizes each time step, t, enforcing consistency with the previous and next time steps, until convergence.

5.2.1 Node Homogeneity

In the homogeneous case, we allow the prior-log odds for node i to be influenced by two sources: whether node i itself was included in the previous optimal subset and the proportion of node $i's$ neighbors included in the previous optimal subset. We define the following terms. Let p_i^t be the prior probability that data element s_i will be contained in the detected subset S^t on time step t. Let x_i^t be 1 if data element s_i is included in S^t, and 0 otherwise. Let n_i^t be the number of neighbors of s_i that are included in S^t, and let k_i be the degree of node s_i. Then the generative model of event propagation, which incorporates temporal consistency constraints under the homogeneity assumption, is defined as:

$$\log\left(\frac{p_i^t}{1-p_i^t}\right) = \beta_0 + \beta_1 x_i^{t-1} + \beta_2 \frac{n_i^{t-1}}{k_i}. \tag{5.1}$$

As a concrete example of the interpretation of this model, assume $\beta_0 = -1.5$, $\beta_1 = 5$, and $\beta_2 = 0$. Then, if a node is included in the previous detected subset, S^{t-1}, it has a 97% prior probability of being included in the current detected subset, S^t. If it was not included in the previous subset, then it only has an 18% probability of being included in the current subset. When $\beta_2 > 0$, the proportion of neighbors j included in S^{t-1} will further influence the prior probability of s_i being included in the current subset.

We now compute the total impact Δ_i^t of including x_i^t on the overall penalized log-likelihood ratio score $F(S)$, as compared to the score $F(S \setminus x_i^t)$ when x_i^t is excluded. Given the log-linear model of p_i^t above, we have:

Consider Δ_i^t as the total impact of including x_i^t on the overall penalized log-likelihood ratio score $F(S)$, where $S = S_1 \bigcup \ldots S_t \bigcup \ldots \bigcup S_w$, as compared to the score $F(S \setminus x_i^t)$ when x_i^t is excluded. We note that the inclusion or exclusion of a step t affects both the log-likelihood ratio score of S^t, conditioned on S^{t-1}, and the log-likelihood ratio score of S^{t+1}, conditioned on S^t. The log-linear model of p_i^t above provides:

$$\begin{aligned}\Delta_i^t = \left(\log(p_i^t) - \log(1 - p_i^t)\right) &+ \sum_{j \in S^{t+1}} \left(\log(p_j^{t+1} \mid x_i^t) - \log(p_j^{t+1} \mid \bar{x}_i^t)\right) \\ &+ \sum_{j \notin S^{t+1}} \left(\log(1 - p_j^{t+1} \mid x_i^t) - \log(1 - p_j^{t+1} \mid \bar{x}_i^t)\right) .\end{aligned} \tag{5.2}$$

In equation (5.2), the initial difference results from the prior probability of x_i^t, conditioned on x_i^{t-1} and its number of included neighbors n_i^{t-1} from the *previous* time step. This difference can be calculated directly from the model:

$$\log(p_i^t) - \log(1 - p_i^t) = \beta_0 + \beta_1 x_i^{t-1} + \beta_2 \frac{n_i^{t-1}}{k_i}. \tag{5.3}$$

The two sums in (5.2) account for the fact that including x_i^t changes the prior probabilities of x_i^{t+1} and its neighbors n_i^{t+1} for the *next* time step. These sums can be rewritten as:

$$
\begin{aligned}
&\sum_{j \in S^{t+1}} \left(\log(p_j^{t+1} \mid x_i^t) - \log(p_j^{t+1} \mid \bar{x}_i^t)\right) + \sum_{j \notin S^{t+1}} \left(\log(1 - p_j^{t+1} \mid x_i^t) - \log(1 - p_j^{t+1} \mid \bar{x}_i^t)\right) \\
&= \sum_{j \in S^{t+1}} (\beta_0 + \beta_1 x_j^t + \beta_2 \frac{n_j^t}{k_j} \mid x_i^t) - \sum_j f(\beta_0 + \beta_1 x_j^t + \beta_2 \frac{n_j^t}{k_j} \mid x_i^t) \\
&- \sum_{j \in S^{t+1}} (\beta_0 + \beta_1 x_j^t + \beta_2 \frac{n_j^t}{k_j} \mid \bar{x}_i^t) + \sum_j f(\beta_0 + \beta_1 x_j^t + \beta_2 \frac{n_j^t}{k_j} \mid \bar{x}_i^t),
\end{aligned}
\tag{5.4}
$$

where the function $f(x) = \log(1 + \exp(x))$. Next, note that the contributions to equation (5.4) are equal to 0 for all nodes j except for node i and its neighbors. For $j = i$, the corresponding terms in (5.4) simplify to:

$$
\beta_1 x_i^{t+1} + f\left(\beta_0 + \beta_2 \frac{n_i^t}{k_i}\right) - f\left(\beta_0 + \beta_1 + \beta_2 \frac{n_i^t}{k_i}\right). \tag{5.5}
$$

For each neighbor j of i, the corresponding terms in (5.4) simplify to:

$$
\beta_2 \left(\frac{x_j^{t+1}}{k_j}\right) + f\left(\beta_0 + \beta_1 x_j^t + \beta_2 \frac{n_j^t}{k_j}\right) - f\left(\beta_0 + \beta_1 x_j^t + \beta_2 \frac{n_j^t + 1}{k_j}\right). \tag{5.6}
$$

Adding the contributions of equations (5.3), (5.5), and (5.6) provides:

$$
\begin{aligned}
\Delta_i^t &= \beta_0 + \beta_1 \left(x_i^{t-1} + x_i^{t+1}\right) + \beta_2 \left(\frac{n_i^{t-1}}{k_i} + \sum_{j \in S^{t+1}} \frac{1}{k_j}\right) \\
&+ f\left(\beta_0 + \beta_2 \frac{n_i^t}{k_i}\right) - f\left(\beta_0 + \beta_1 + \beta_2 \frac{n_i^t}{k_i}\right) \\
&+ \sum_j f\left(\beta_0 + \beta_1 x_j^t + \beta_2 \frac{n_j^t}{k_j}\right) - \sum_j f\left(\beta_0 + \beta_1 x_j^t + \beta_2 \frac{n_j^t + 1}{k_j}\right),
\end{aligned}
\tag{5.7}
$$

where the sums are taken over all neighbors j of i. In the special case of $\beta_2 = 0$, equation (5.7) simplifies to:

$$\Delta_i^t = \beta_0 + \beta_1 \left(x_i^{t-1} + x_i^{t+1}\right) + f\left(\beta_0\right) - f\left(\beta_0 + \beta_1\right). \tag{5.8}$$

Note that, when $\beta_2 = 0$, Δ_i^t can be computed exactly. When $\beta_2 \neq 0$, Δ_i^t must be approximated. We do so by assuming that $\beta_0 + \beta_2 \ll 0$ and $\beta_0 + \beta_1 \gg 0$. Noting that $f(x) \approx 0$ when $x \ll 0$, and $f(x) \approx x$ when $x \gg 0$, provides:

$$\begin{aligned}\Delta_i^t \approx {} & \beta_0 + \beta_1 \left(x_i^{t-1} + x_i^{t+1}\right) + \beta_2 \left(\frac{n_i^{t-1}}{k_i} + \sum_{j \in S^{t+1}} \frac{1}{k_j}\right) - \left(\beta_0 + \beta_1 + \beta_2 \frac{n_i^t}{k_i}\right) + \\ & \sum_{j \in S^t} \left(\beta_0 + \beta_1 + \beta_2 \frac{n_j^t}{k_j}\right) - \sum_{j \in S^t} \left(\beta_0 + \beta_1 + \beta_2 \frac{n_j^t + 1}{k_j}\right) \\ = {} & \beta_1 \left(x_i^{t-1} + x_i^{t+1} - 1\right) + \beta_2 \left(\frac{n_i^{t-1}}{k_i} + \sum_{j \in S^{t+1}} \frac{1}{k_j} - \sum_{j \in S^t} \left(\frac{1}{k_i} + \frac{1}{k_j}\right)\right).\end{aligned} \tag{5.9}$$

However, equation (5.9) assumes knowledge of which other elements are contained in S^t, and this information would not be known in advance. Thus the final sum over $j \in S^t$ is further approximated with half the corresponding sum over all neighbors j of i:

$$\Delta_i^t \approx \beta_1 \left(x_i^{t-1} + x_i^{t+1} - 1\right) + \beta_2 \left(\frac{n_i^{t-1}}{k_i} + \sum_{j \in S^{t+1}} \frac{1}{k_j} - \frac{1}{2} \sum_{j} \left(\frac{1}{k_i} + \frac{1}{k_j}\right)\right). \tag{5.10}$$

5.2.2 Node Heterogeneity

We now provide the derivation of Δ_i^t without the homogeneity assumption. This allows the contribution to Δ_i^t in temporal consistency to vary by node and by neighbor. Intuitively, this recognizes asymmetric relationships among nodes: node i may regularly propagate to neighbor j, but neighbor j may not regularly spread to i.

This type of heterogeneous relationship cannot be captured under the stricter homogeneity assumption. Nevertheless, it is necessary for more realistic scenarios such as a water distribution system where water flow (and this the spread of potential contamination events) may occur in a single direction.

Proceeding forward, we will follow the format in Section 5.2.1. This allows for easier comparison of the two derivations. For clarification, β_{ji} is the influence that neighbor j has on node i and β_{ij} is the influence that node i has on neighbor j. We will discusee how β_{ji} and β_{ij} can be learned from labeled training data in Chapter 6.

With heterogenous nodes our generative model is

$$\log\left(\frac{p_i^t}{1-p_i^t}\right) = \beta_{0i} + \beta_{ii}x_i^{t-1} + \sum_j \beta_{ji}x_j^{t-1} \tag{5.11}$$

where the summation is performed over all of node i's neighbors, j.

Similar to the derivation in Section 5.2.1, we must calculate the change in the overall penalized log-likelihood ratio score $F(S)$ resulting from the impact of including x_i^t.

$$\begin{aligned}\Delta_i^t = &\left(\log(p_i^t) - \log(1-p_i^t)\right)\\ &+ \sum_{j\in S^{t+1}} \left(\log(p_j^{t+1} \mid x_i^t) - \log(p_j^{t+1} \mid \bar{x}_i^t)\right)\\ &+ \sum_{j\notin S^{t+1}} \left(\log(1-p_j^{t+1} \mid x_i^t) - \log(1-p_j^{t+1} \mid \bar{x}_i^t)\right).\end{aligned} \tag{5.12}$$

The first term in (5.12) is the "forward" temporal consistency constraint and is described by the heterogeneous generative model.

$$\log(p_i^t) - \log(1-p_i^t) = \beta_{0i} + \beta_{ii} x_i^{t-1} + \sum_j \beta_{ji} x_j^{t-1} \tag{5.13}$$

The latter two terms in (5.12) are the "backward" temporal consistency constraints and require further attention.

$$\begin{aligned}
&\sum_{J\in S^{t+1}} \left(\log(p_J^{t+1} \mid x_i^t) - \log(p_J^{t+1} \mid \bar{x_i^t})\right) \\
&+ \sum_{J\notin S^{t+1}} \left(\log(1-p_J^{t+1} \mid x_i^t) - \log(1-p_J^{t+1} \mid \bar{x_i^t})\right) \\
&= \sum_{J\in S^{t+1}} (\beta_{0J} + \beta_{JJ} x_J^t + \sum_j \beta_{jJ} x_j^t \mid x_i^t) \\
&- \sum_J f(\beta_{0J} + \beta_{JJ} x_J^t + \sum_j \beta_{jJ} x_j^t \mid x_i^t) \\
&- \sum_{J\in S^{t+1}} (\beta_{0J} + \beta_{JJ} x_J^t + \sum_j \beta_{jJ} x_j^t \mid \bar{x_i^t}) \\
&+ \sum_J f(\beta_{0J} + \beta_{JJ} x_J^t + \sum_j \beta_{jJ} x_j^t \mid \bar{x_i^t}),
\end{aligned} \tag{5.14}$$

where the function $f(x)$ remains $\log(1+\exp(x))$ as in Section 5.2.1.

In the specific case when $J = i$ i.e, the effect of including the node i on the next time step x_i^{t+1}, the terms in (5.14) simplify to

$$\begin{aligned}
&\left((\beta_{0i} + \beta_{ii} + \sum_j \beta_{ji} x_j^t) - (\beta_{0i} + \sum_j \beta_{ji} x_j^t)\right) x_i^{t+1} \\
&- f(\beta_{0i} + \beta_{ii} + \sum_j \beta_{ji} x_j^t) + f(\beta_{0i} + \sum_j \beta_{ji} x_j^t) \\
&= \beta_{ii} x_i^{t+1} + f(\beta_{0i} + \sum_j \beta_{ji} x_j^t) - f(\beta_{0i} + \beta_{ii} + \sum_j \beta_{ji} x_j^t)
\end{aligned} \tag{5.15}$$

The remaining non-zero terms from (5.14) capture the influence of node i's neighbors in backward temporal consistency. For each neighbor j of i, we have the impact of including node x_i^t on x_j^{t+1} equal to the following:

$$\begin{aligned}
&\left((\beta_{0j} + \beta_{jj}x_j^t + \beta_{ij} + \sum_{j'\neq i}\beta_{j'j}x_{j'}^t) - (\beta_{0j} + \beta_{jj}x_j^t + \sum_{j'\neq i}\beta_{j'j}x_{j'}^t)\right) x_j^{t+1} \\
&-f(\beta_{0j} + \beta_{jj}x_j^t + \beta_{ij} + \sum_{j'\neq i}\beta_{j'j}x_{j'}^t) + f(\beta_{0j} + \beta_{jj}x_j^t + \sum_{j'\neq i}\beta_{j'j}x_{j'}^t) \qquad (5.16) \\
&=\beta_{ij}x_j^{t+1} + f(\beta_{0j} + \beta_{jj}x_j^t + \sum_{j'\neq i}\beta_{j'j}x_{j'}^t) - f(\beta_{0j} + \beta_{jj}x_j^t + \beta_{ij} + \sum_{j'\neq i}\beta_{j'j}x_{j'}^t)
\end{aligned}$$

Note that j' represents the neighbors of neighbor j with careful exclusion to avoid double counting node i in the summation.

Combining the results from (5.13), (5.15), and (5.16) we arrive at:

$$\begin{aligned}
\Delta_i^t =& \beta_{0i} + \beta_{ii}(x_i^{t-1} + x_i^{t+1}) + \sum_j \beta_{ji}x_j^{t-1} + \sum_j \beta_{ij}x_j^{t+1} \\
&+ f(\beta_{0i} + \sum_j \beta_{ji}x_j^t) - f(\beta_{0i} + \beta_{ii} + \sum_j \beta_{ji}x_j^t) \\
&+ \sum_j f(\beta_{0j} + \beta_{jj}x_j^t + \sum_{j'\neq i}\beta_{j'j}x_{j'}^t) - \sum_j f(\beta_{0j} + \beta_{jj}x_j^t + \beta_{ij} + \sum_{j'\neq i}\beta_{j'j}x_{j'}^t).
\end{aligned} \qquad (5.17)$$

We conclude by making two further simplifying assumptions. The last two sums in (5.17) require information on the other nodes included in S^t in order to obtain the Δ_i^t for a given node i; but that information is unknown at this point. Therefore we approximate with half of the corresponding sums as above. This approximation results in a $-\frac{1}{2}\beta_{ij}$ term. Similarly, the middle two terms in (5.17) may be approximated with a *lower bound* of $-\beta_{ii}$. The true sum of those terms is unknown but falls between $-\beta_{ii}$ and 0.

These approximations provide our final estimate of the heterogeneous influence on the penalized log-likelihood ratio score $F(S)$ when including node i in S^t:

$$\Delta_i^t \approx \beta_{0i} + \beta_{ii}(x_i^{t-1} + x_i^{t+1} - 1) + \sum_j \left(\beta_{ji}x_j^{t-1} + \beta_{ij}\left(x_j^{t+1} - \frac{1}{2}\right)\right). \qquad (5.18)$$

Chapter 6 will compare and contrast the performance of homogeneous and heterogeneous temporal consistency constraints in more detail.

5.3 Iterative Convergence

The previous section provided definitions and interpretations for $F_{pen}(S) = \sum_{s_i^t \in S}(\lambda_i^t + \Delta_i^t)$ for the EBB scan statistic with temporal consistency constraints. However, recall that the values of Δ_i^t for a given time step t depend on the detected subsets at $t-1$ and $t+1$. To resolve this issue, the Dynamic Subset Scan uses an iterative method that converges to a (local) optimum. To better approach the global optimum, multiple restarts and simulated annealing (which gradually increases the strength of the Δ_i^t from 0 to their full values) are wrapped around steps (3)-(13) in Algorithm 5.1.

5.4 Conclusion

This work introduced the Dynamic Subset Scan for detecting *dynamic* patterns that change the affected subset over time. It developed *temporal consistency constraints* that may be enforced on temporally adjacent, spatial subsets. These constraints are a fruitful compromise between traditional spatial-temporal scan statistics that do not allow the detected region to change over time (Static) and the other extreme where temporal information is ignored (Independent). The key insight to enforcing

Algorithm 5.1 Iterative convergence to local optimum for Dynamic Subset Scan (without multiple restarts or simulated annealing)

```
1: for window duration $w$ from 1 to $max\ window\ W$ do
2:   Initialize each of the $w$ spatial subsets independently (i.e., separately compute
     the highest scoring subsets $S^t$ for each time step $t$, assuming $\Delta_i^t = 0$ for all $s_i$).
3:   repeat
4:     Randomly select a time step $t$ that is not flagged as "Checked". Copy current
       spatial subset $S^t$.
5:     Compute $\Delta_i^t$ for each node $s_i$ given subsets $S^{t-1}$ and $S^{t+1}$, using equation
       (5.8) or (5.10).
6:     Compute new optimal subset $S'^t$ for time step $t$ using $\Delta_i^t$. Without con-
       nectivity constraints, simply include all positive contributions $\lambda_i^t + \Delta_i^t$; with
       connectivity constraints, call Additive GraphScan (see Chapter 6).
7:     if new subset $S'^t$ does not improve penalized log-likelihood ratio of spatial-
       temporal subset $S$ then
8:       Revert to $S^t$ and mark time step $t$ as "Checked".
9:     end if
10:    if new subset $S'^t$ does improve penalized log-likelihood ratio of spatial-
       temporal subset $S$ then
11:      Replace $S^t$ with $S'^t$ and remove "Checked" flags from time steps $t-1, t+1$,
         and $t$.
12:    end if
13:  until no further changes improve penalized log-likelihood ratio of spatial-
     temporal subset $S$, i.e., all time steps have been flagged as "Checked".
14: end for
15: Return the highest scoring spatial-temporal subset $S_w^*$.
```

temporal consistency constraints is recognizing that the expectation-based binomial scoring function may be written as an additive function over the data records (see Chapter 3 for details on Additive Linear Time Subset Scanning). This allows for additional terms (constraints) to be included in the penalized log likelihood ratio while remaining efficient to optimize. Critically, these temporal consistency constraints were derived to allow temporal information to be shared both forward and backward in time. We provided two different forms of temporal consistency constraints. The more restrictive homogeneity constraint assumes that the propensity to propagate is the same across all nodes and neighbors. The heterogeneous derivation relaxes this assumption and allows the influence from neighbors to vary.

In Chapter 6, we will add hard constraints on graph connectivity to the Dynamic Subset Scan framework developed here, thus enabling Dynamic Subset Scan to be applied to detect events spreading through a graph or network structure. We will also present a concrete example of the use of this method, applied to the tracking, source-tracking, and prediction of spreading contamination in water distribution networks.

Chapter 6

Enforcing Soft Temporal Consistency and Hard Connectivity Constraints

Many complex data sets containing emerging events or patterns are commonly represented in a known and fixed graph structure. Examples of this include water pipelines, transportation routes, power grids, and supply chains in general. While other recent work [25, 15] has focused on learning graph structure, here we assume a given graph structure and wish to detect which nodes are currently affected, by observing data produced at the nodes of the graph on each time step.

Chapters 3 and 5 outlined how the expectation-based binomial (EBB) scoring function (among others) may be penalized with additional constraints $F_{pen}(S) = \sum_{s_i \in S}(\lambda_i + \Delta_i)$ while remaining an additive set function over the data elements $s_i \in S$. Optimizing additive functions *without* connectivity constraints is very straightforward and consists of including all records with positive contributions $(\lambda_i + \Delta_i > 0)$

and excluding the rest (see Corollary 3.2). Enforcing hard connectivity constraints on additive functions (i.e. determining the "most positive" connected subset) is an interesting and difficult problem. For example, not all nodes making positive contributions will be included in a high-scoring connected subset because they are likely disconnected in the underlying graph structure. Also, a high-scoring connected subset may include a node with a negative contribution in order to connect two positive nodes.

The GraphScan algorithm [37] presented in Chapter 2 exactly identifies the highest scoring connected subset for any scoring function that satisfies the Linear-Time Subset Scanning (LTSS) property [29]. It is trivially shown that additive functions satisfy LTSS, and therefore GraphScan could be used to determine the highest scoring ("most positive", in the case of an additive scoring function) connected subset. However, GraphScan is designed to optimize over more complex scoring functions; most importantly, its computation time is exponential in the graph size and therefore it does not scale well in this setting. We would like to apply our Dynamic Scan approach to datasets with thousands or tens of thousands of nodes, and GraphScan cannot handle these cases. Therefore, *Additive GraphScan* is proposed as an efficient heuristic alternative to GraphScan which can be used to identify high-scoring (most positive) connected subsets in a given graph structure with real-valued weights at each node. This work builds on a foundation provided in [39].

We show below that Additive GraphScan achieves very good approximations to the true highest scoring subset with high probability; moreover, since Additive GraphScan is used within the iterative convergence approach described in Chapter 5, it is less important that it converges to the true highest scoring subset for a given time step as long as it continues to increase the score of the spatial-temporal subset

$S = \bigcup_t S^t$.

6.1 Additive GraphScan Algorithm

Additive GraphScan was developed to solve the following problem. Given a graph $G = (V, E)$ and a real-valued weight $w(n)$ for each node $n \in V$, we must compute $\max_{S:\text{S connected in G}} \sum_{n \in S} w(n)$. We note that this is a variant of the prize-collecting Steiner tree problem. Minimum description length (MDL) has also been used as a heuristic for a similar type of problem [2].

The optimization process is a key step in the iterative convergence framework introduced in Section 5.3. In particular, given the subsets S^{t-1} and S^{t+1}, we can compute the total contribution $\gamma_i^t = \lambda_i^t + \Delta_i^t$, where λ_i^t is the log-likelihood ratio for each node and Δ_i^t reflects temporal consistency. The highest scoring connected subgraph is $\max_{S:\text{S connected in G}} \sum_{s_i \in S} \gamma_i^t$.

Additive GraphScan makes use of the following notation. $w(n)$ is the real-valued weight of node n. A path p is any connected subgraph of nodes. $w(p)$ is the sum of weights for every node in the path. $g(p)$ is the *gain* that would result from merging path p into a single node. It is the difference between the weight of the resulting merged node and the highest weighted node in the path. Identifying and merging paths with positive gains is an integral part of Additive GraphScan. $g(n, p^*)$ is the gain that would result from merging two paths together. The first path, p^*, is a previously identified path of interest with positive gain. The second path is the shortest path between node n and any point along path p^*. $g(n, p^*)$ is the difference between the weight of the resulting merged paths and $\max(w(n), w(p^*))$.

$pw(n)$ is the *pathweight* of a node used when calculating single source, shortest paths traversing through the node. Note the difference between the weight of a node

$w(n)$ (which may be positive or negative and is used in the *gain* calculations above) and the pathweight of a node $pw(n)$ (which is non-negative and used in shortest path calculations). Pathweights of positive nodes are set to 0, reflecting no penalty (or reward) for traversing positive nodes while identifying shortest paths. Pathweights for negative nodes with no positive neighbors are $-w(n)$. Pathweights for negative nodes with positive neighbors have

$$pw(n) = -\min\left(0, w(n) + \sum_{\text{pos neighbors}, n_i} \frac{w(n_i)}{\text{degree}(n_i)}\right).$$

These positive weights may be thought as uniformly "diffusing" over their negative neighbors and then using this altered weight as the pathweight for negative nodes with positive neighbors. In the case where a large positive node overwhelms its negative neighbor, the negative neighbor's pathweight is set to 0.

Finally, $s(n_a, n_b, n_c)$ determines a fourth node, n_s in the graph as a *Steiner* point for n_a, n_b, and n_c. A Steiner point in this setting is a node that forms the shortest interconnect between the three provided nodes using the pathweights of the graph. $s(n_a, n_b, n_c)$ returns the shortest interconnecting path formed between the three nodes going through n_s.

Some basic pre-processing may be applied to the graph before running Additive GraphScan. For example, any positive node with a positive neighbor may be merged together into a larger, single positive node (adding their weights) and repeated until no further merges exist. Also, any negative nodes with degree of 1 or less may be recursively removed because these are guaranteed to not be included in a high scoring connected subset. Lastly, any negative node with at least two positive neighbors may be merged into a single node if the resulting merged node has a higher weight than any individual positive neighbor. Additive GraphScan can then be applied to the pre-processed graph. Additive GraphScan scales as $O(kN^2) = O(N^{2.5})$, dominated by steps (3) and (5) as denoted in Algorithm 6.1.

Algorithm 6.1 Additive GraphScan

```
1: while positive gain path merges exist do
2:     Identify top-$k$ positive nodes where $k = \sqrt{N}$.
3:     Compute path weights $pw(n)$ for all nodes and create single-source shortest
       paths from each top-$k$ node.
4:     Compute $g(p)$ for each shortest path $p$ between top-$k$ pairs. Determine highest
       gain path $p^*$ and record endpoints as $n_a$ and $n_b$.
5:     Compute $g(n_i, p^*)$ for each remaining top-$k$ node, $n_i$. Determine highest gain
       node for $p^*$ and record as $n_c$. If no positive gain exists between $p^*$ and any $n_i$,
       then merge $p^*$ and restart.
6:     Form new path $p^{**}$ as the union of $p^*$ and the path connecting $p^*$ to $n_c$.
7:     Compute $s(n_a, n_b, n_c)$. Compare $w(s(n_a, n_b, n_c))$ and $w(p^{**})$. Merge the one
       with higher weight.
8: end while
9: The highest weight merged node is returned as the most positive connected subset
   found by Additive GraphScan. Note that this node may need to be "unpacked"
   to determine the contents in the original graph form.
```

6.1.1 Additive GraphScan Example

This section concludes by applying Additive GraphScan to a sample pre-processed graph found in Figure 6.1. The most positive connected subgraph consists of nodes $\{0, 1, 6, 3, 4, 8, 9\}$ where node 6 is the Steiner point used to connect nodes 0, 4, and 9. Additive GraphScan correctly identifies this subgraph even though node 6 is not on the shortest paths connecting nodes 0 and 4 or nodes 4 and 9. A key insight into the strong performance of Additive GraphScan is delaying path merges while searching for a potential Steiner point. Begin at step (2:) Nodes 0, 4, and 9 are identified as the top-k nodes. (3:) Dijkstra's algorithm is called on nodes 0, 4,

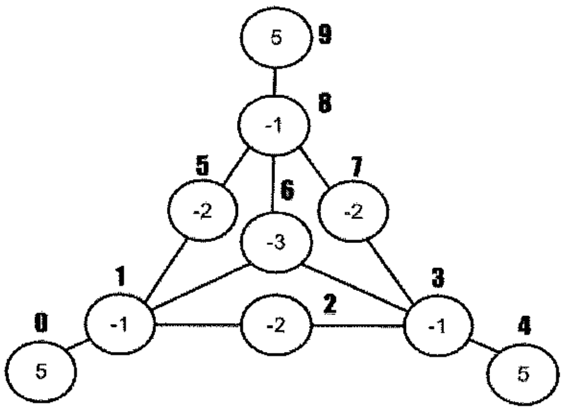

Figure 6.1: An example graph to demonstrate the Additive GraphScan algorithm. The large bolded numbers are node identifiers and the small numbers within each node are the nodes' corresponding weights. The most positive subgraph consists of nodes $\{0, 1, 6, 3, 4, 8, 9\}$ and is correctly identified by Additive GraphScan.

and 9 providing single-source shortest path information from each of them. (4:) The shortest path from node 0 to node 4, $p^* = \{0, 1, 2, 3, 4\}$, has highest gain of $g(p^*) = (5-1-2-1+5)-5 = +1$. Because a positive gain path was found between nodes $n_a = 0$ and $n_b = 4$, Additive GraphScan continues searching for a third node, n_c. (5,6:) Node $n_c = 9$ is found with $p^{**} = \{0, 1, 2, 3, 4, 7, 8, 9\}$ and $w(p^{**}) = 8$. (7:) Calculate a Steiner point for nodes 0, 4, and 9 and note that node 6 forms the shortest interconnect between these three points. This interconnect is formed by the nodes $\{0, 1, 6, 3, 4, 8, 9\}$ and $w(s(0, 4, 9)) = 5-1-3-1+5-1+5 = 9$. Because $w(s(0, 4, 9)) > w(p^{**})$ the Steiner interconnect $s(0, 4, 9)$ is condensed into a single node with weight 9. After this merge, no more positive gain path merges exist and the loop exits. (9:) The highest scoring connected subset is then $\{0, 1, 6, 3, 4, 8, 9\}$. Notice that greedily merging either p^* or p^{**} would have resulted in a sub-optimal merge. Delaying these merges while searching for a potential Steiner point is a key insight into the strong performance of Additive GraphScan.

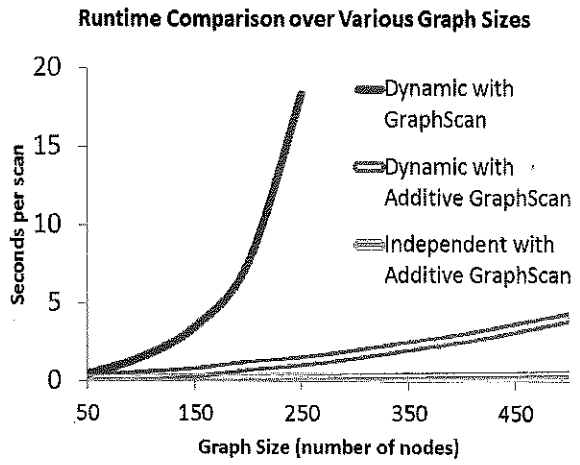

Figure 6.2: Runtime comparisons for the Dynamic Subset Scan with GraphScan and Additive GraphScan as the optimization algorithm. Independent Scan with Additive GraphScan is also shown.

6.2 Comparison of Additive GraphScan vs. GraphScan

This section compares the fast heuristic, Additive GraphScan, to the slower, but exact, GraphScan algorithm. First, a runtime analysis is presented comparing the two optimization algorithms. The much larger "network 2" provided in the Battle of the Water Sensor Networks [3] is used to create connected subgraphs of various sizes from 50 to 500 nodes from the network. The graphs are processed with three different scans: Dynamic Subset Scan with GraphScan, Dynamic Subset Scan with Additive GraphScan, and Independent with Additive GraphScan. The average runtime for each method is reported and are shown in Figure 6.2.

GraphScan begins to struggle with graph sizes of 250 nodes while Additive GraphScan quickly scans graphs of 500 nodes in approximately 4.1 seconds. Independent with Additive GraphScan processed the entire 12,000+ node "Network 2" in 221 seconds while Dynamic with Additive GraphScan required 1830 seconds (approximately a half hour). This difference represents the additional calls to Additive GraphScan required by Dynamic Subset Scan to "align" the individual spatial subsets according to the temporal consistency constraints.

The comparison of Additive GraphScan and GraphScan is concluded by analyzing the scores of the spatial-temporal subsets identified by the scanning methods using both Additive GraphScan and GraphScan. The approximation ratio results compare the highest-scoring subsets found by Additive GraphScan and GraphScan as a percentage averaged over 2000 simulations. Table 6.1 provides detailed information for the approximation ratios. The ratios over 100% in the Dynamic cases reflect the noise in the iterative convergence process outlined above in Section 5.3. To be clear, Additive GraphScan is not identifying a higher scoring subgraph than GraphScan *for an individual time slice.* However, the local optimum after the iterative convergence of Additive GraphScan-based optimizations at each step may have a higher score than the local optimum reached with GraphScan-based optimizations at each time step. The Static and Independent methods do not use this iterative convergence process to identify the highest scoring spatial-temporal region and may reflect a more direct comparison between the performance of Additive GraphScan and GraphScan. The ratio does not fall below 98.4%, indicating that Additive Graphscan is providing a huge speed increase with minimal loss of accuracy compared to scan statistics using GraphScan.

Table 6.1: Approximation ratios comparing scores for Additive GraphScan and GraphScan for multiple methods and FPR and TPR.

method	Background(0.1)	Injects(0.9)	Background(0.2)	Injects(0.8)
Static	100.00%	100.00%	99.15%	99.83%
Independent	100.00%	99.99%	98.48%	99.65%
Dynamic (hom)	100.59%	101.44%	99.70%	100.50%

6.3 Tracking, Source-Tracing, and Predicting Contaminant Plumes

This section evaluates the tracking, source-tracing, and prediction abilities of the Dynamic Subset Scan under both homogenous and heterogeneous assumptions introduced in Sections 5.2.1 and 5.2.2. The 129-node "Network 1" from the Battle of the Water Sensor Networks [3] served as the test bed for these evaluations. Simulations were performed with sensors at FPR $= 0.1$ and TPR $= 0.9$. The results below are averaged over 500 contaminant plumes simulated for 12, one-hour intervals.

Comparisons are made for four different spatial-temporal scan statistics:

- *Static* scan does not allow the detected spatial region to change over time.
- *Independent* scan allows the detected spatial region to change over time but does not share temporal information between time steps. This implies that $\Delta_i^t = 0$ for all nodes s_i and time steps t.
- *Dynamic (Hom)* scan allows the detected spatial region to change over time and uses *homogeneous* temporal consistency constraints to "align" the individual time steps. See Section 5.2.1 for details.
- *Dynamic (Het)* scan allows the detected spatial region to change over time and

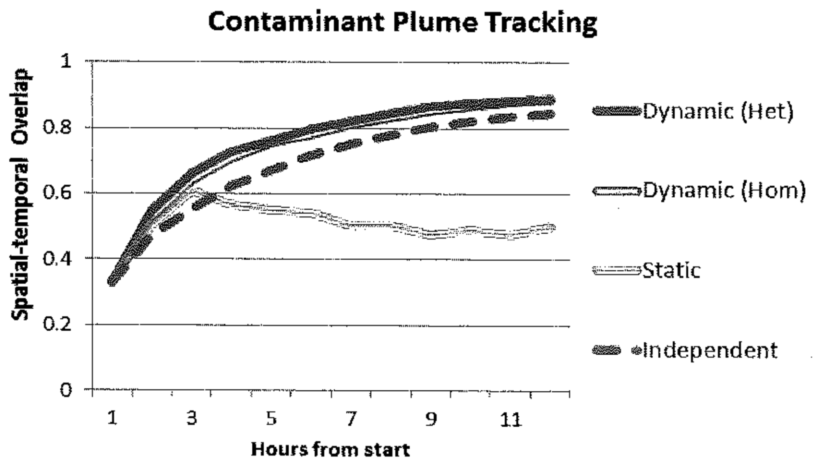

Figure 6.3: Contaminant tracking results for four competing methods. This graph summarizes tracking ability by reporting the spatial-temporal overlap of the detected and affected subsets over the course of 12 hours.

uses *heterogeneous* temporal consistency constraints to "align" the individual time steps. See Section 5.2.2 for details.

Each of these methods have graph connectivity constraints enforced through Additive GraphScan. Simpler versions of the scans without connectivity constraints were evaluated and had much lower performance (not shown here) for tracking, source-tracing, and prediction tasks.

The $\beta_0 \ldots \beta_2$ parameters for Dynamic (hom) were set using a grid search on a separate 500 plume training set. The parameter values that maximized spatial-temporal overlap in the training data are $\beta_0 = -0.9$, $\beta_1 = 5.2$, and $\beta_2 = 1.4$. For Dynamic (het), the β_{0i}, β_{ii}, and β_{ji} terms were learned through fitting the logistic regression model for each node i on a separate 500 plume training set.

6.3.1 Tracking Results

Figure 6.3 summarizes the methods' tracking ability over the duration of a spreading contaminant plume (12 hours). A scan statistic's tracking ability may be summarized through *spatial-temporal overlap.* Spatial-temporal overlap is a combination of precision and recall applied to spatial-temporal subsets. A measure of 1.0 corresponds to perfect agreement between the affected and detected spatial-temporal regions, while 0.0 means the affected and detected regions are disjoint. For two spatial-temporal subsets, *Affected* and *Detected,* the overlap is defined as: $\frac{|\text{Affected} \cap \text{Detected}|}{|\text{Affected} \cup \text{Detected}|}$. See Figure 6.4 for details.

Figure 6.3 demonstrates that Static struggles to accurately track the dynamic plumes over time. However, we also provide another measure of tracking ability in Figure 6.5. Figure 6.5 provides three "snapshots" in time at hours 6, 9, and 12 into the plume. These snapshots report spatial overlap over time rather than spatial-temporal overlap. The peculiar shape of Static's tracking in Figure 6.5 is a result of attempting to capture a dynamic pattern using a static region. Referencing the 9^{th} hour snapshot as an example (middle panel), we see that Static's overlap with the plume peaked 4 hours earlier at $t = 5$. For hours $1 - 4$, the region detected by Static is too large. However, that same region is too small for the plume as it continues to grow in hours $6 - 9$.

The Independent and Dynamic methods do not suffer this problem because they allow the detected subset to change at each step. Additionally, we see a small bonus to the Dynamic methods' tracking ability due how they incorporate temporal consistency constraints.

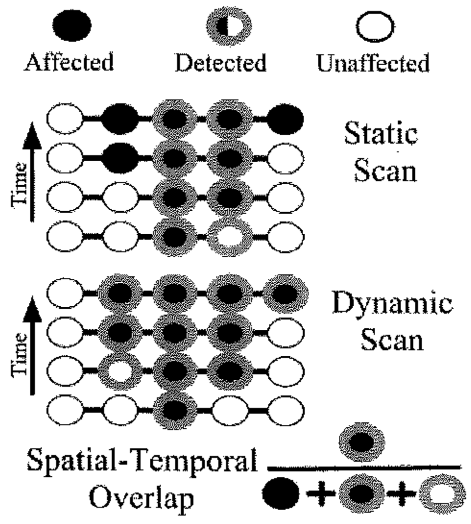

Figure 6.4: This figure demonstrates the calculation of spatial-temporal overlap for plume tracking. A plume spreads through a simple 5-node line graph over the course of four time steps. Affected nodes turn from white to shaded as the contaminant spreads. The Static scan method is constrained to keep the exact same detected spatial region throughout the event duration. Hence, it may fail to capture the plume at later time steps. Dynamic Scan allows the detected spatial region to change at each time step, tracking the plume as it spreads. Due to connectivity constraints, both methods must return a connected subgraph as the detected spatial region at each time step. Spatial-temporal overlap is penalized for both false positives and false negatives. A measure of 1.0 corresponds to perfect agreement between the affected and detected spatial-temporal regions, while 0.0 means that the affected and detected regions are disjoint.

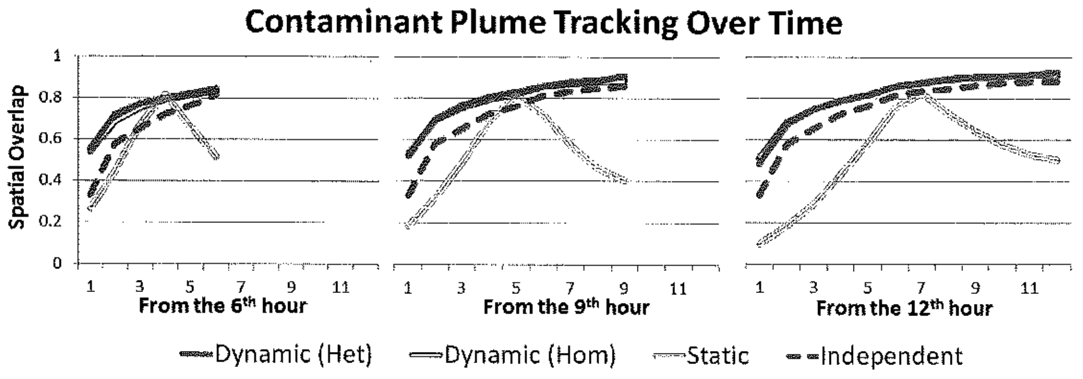

Figure 6.5: Contaminant tracking results for four competing methods. This graph represents tracking ability by reporting the spatial overlap of past detected and affected subsets at the 6^{th}, 9^{th}, and 12^{th} hours of the contaminant plume.

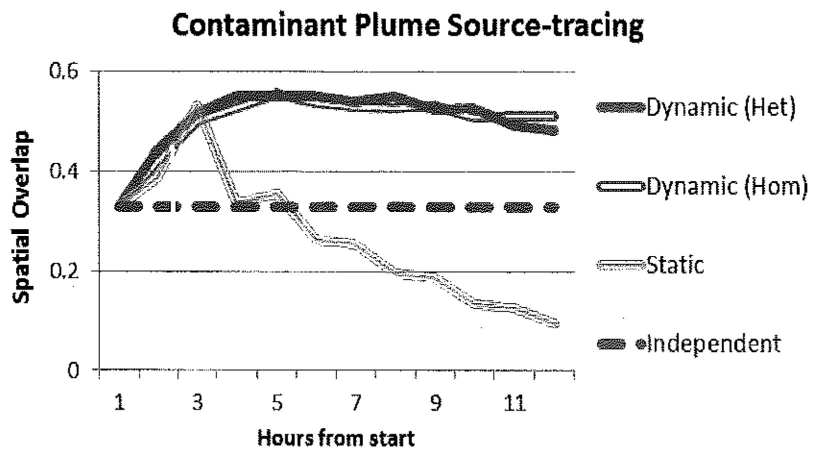

Figure 6.6: Source-tracing results for four competing methods. Source-tracing is measured by reporting the spatial overlap between the earliest spatial region of the detected subset and the original affected node(s) at the start of the plume.

6.3.2 Source-tracing Results

Figure 6.6 reports the methods' ability to identify *where* the contaminant originated over the duration of the plume (12 hours). This is measured through purely spatial overlap between the earliest time step in the detected region and the source node(s) of the plume. Note that it is possible for a quickly spreading plume to affect multiple nodes within the first hour. In such cases, all of these nodes are treated as source nodes.

The source-tracing results clearly demonstrate the advantage of sharing information between time steps during the optimization process. The key to the Dynamic Subset Scan's success for source-tracing is the *backwards* flow of temporal consistency information allowed in our model. The Dynamic methods are able to change the detected subset for previous time steps based on new, more current data. This gives it superior source-tracing abilities throughout the duration of the plume.

We do not see a substantial difference between Dynamic (hom) and Dynamic (het)

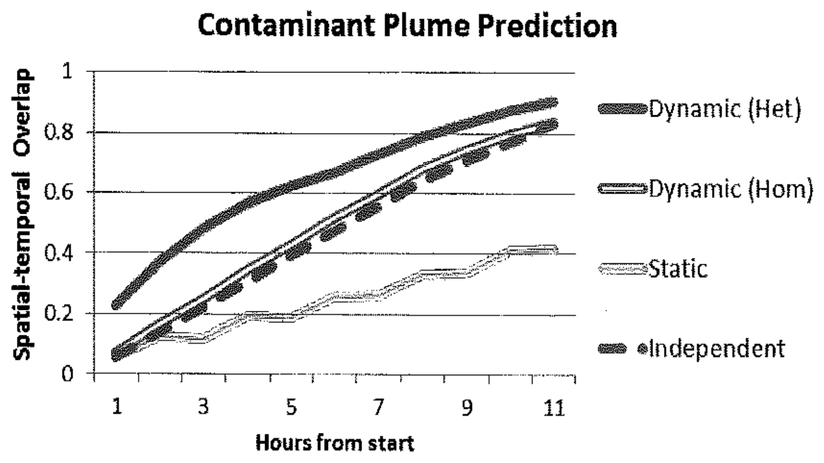

Figure 6.7: Contaminant prediction results for four competing methods. This graph summarizes prediction ability by reporting the spatial-temporal overlap of the predicted and affected subsets over the course of 12 hours.

in this scenario, suggesting that the simpler homogeneous assumption is sufficient for high source-tracing performance.

Static's ability to identify the source nodes actually *decreases* over the course of the contamination event as more information is gathered. The typically large regions returned by the Static method later in a contaminant plume harm its ability to accurately identify the source of the contaminant.

Finally, we note that the Independent method does not share any temporal information and therefore the subset identified at the first time step of a plume does not change as new information is gathered on the plume. Again, homogeneous and heterogeneous variants of the Dynamic method achieve similar performance on the tracking task.

6.3.3 Prediction Results

Figure 6.7 summarizes the prediction ability of the competing methods by reporting the spatial-temporal overlap of the predicted and affected subsets over the course of

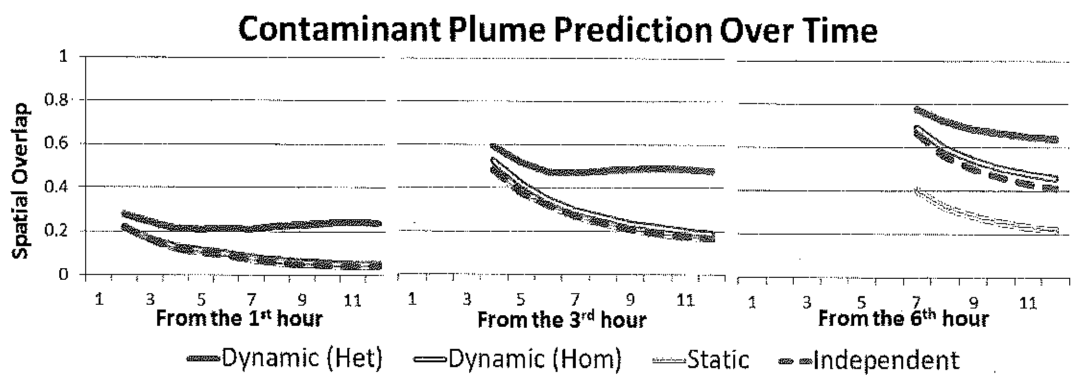

Figure 6.8: Contaminant prediction results for four competing methods. This graph represents prediction ability by reporting the spatial overlap of future predicted and affected subsets at the 1^{st}, 3^{rd}, and 6^{th} hours of the contaminant plume.

11 hours (there is no prediction on the 12^{th} hour). Note that the predicted subsets are carreid foreward to the 12^{th}; as such, prediction from the 1^{st} hour (with up to 11 hours of look-ahead) is an extremely difficult task. Similar to the tracking results, we also provide prediction results at three "snapshots" in time at hours 1, 3, and 6 in Figure 6.8. These snapshots report spatial overlap over time rather than spatial-temporal overlap.

For the Static method, prediction is simply extending the spatial subset forward in time. This results in poor prediction power because the static region that maximizes the scoring function in the past is constrained to capture the early stages of the plume rather than predicting how the plume will grow over time. This results in a prediction with relatively high precision but extremely poor recall due to the increasing size of the plume over time.

For Independent, only the spatial subset of the *most recent time step* is extended into the future. This results in increased prediction power compared to Static because the prediction is not influenced by past plume behavior.

For the Dynamic methods, prediction results are created by implementing their respective *forward temporal consistency* constraints from the most recent time step. Critically, this allows the predicted subsets to change going forward in time. In particular, the temporal consistency constraints with node heterogeneity allows the predicted subsets to change intelligently over time. This results in the overall strongest prediction performance of the four methods under consideration.

6.4 Conclusion

The last two chapters introduced the Dynamic Subset Scan for source-tracing, tracking, and predicting *dynamic* patterns that change the affected subset over time. This novel extension of the well-known spatial and subset scan statistics is composed of two main contributions. First is the incorporation of *temporal consistency constraints* that may be enforced on temporally adjacent, spatial subsets. These constraints were introduced in Chapter 5.

The second novel contribution is the Additive GraphScan algorithm, which allows the Dynamic Subset Scan to enforce both soft temporal consistency constraints *and* hard connectivity constraints while scaling to large, real world networks. Additive GraphScan is a fast, heuristic alternative to GraphScan. However, the results demonstrate an approximation ratio of over 99%, suggesting a very small sacrifice for dramatic gains in speed and scalability.

The Dynamic Subset Scan was evaluated on data provided through the "Battle of the Water Sensor Networks" [3]. Dynamic scan succeeded in source-tracing, tracking, and predicting these events more accurately compared to other competing methods. The gains were due to Dynamic Scan's constrained flexibility: competing methods either failed to capture the dynamics of the spreading plume (Static) or were susceptible to over-fitting from lack of constraints (Independent). We considered two forms of temporal consistency constraints in the Dynamic Subset Scan. Under the homogeneous nodes assumption, all of a node's neighbors contributed equally to the prior probability of that node becoming contaminated. This assumption was sufficient in the source-tracing and tracking tasks. In the prediction task, however, it is important for the detection method to identify key nodes and edges in the network as this extension allows the predicted subsets to intelligently expand over time. Therefore, the heterogeneous nodes version of temporal consistency constraints was introduced, and our experimental results demonstrate that this approach significantly increases the prediction power of the Dynamic Subset Scan over the previous homogeneous implementation.

In conclusion, relaxing constraints on spatial-temporal region shape must be done carefully. Strict temporal constraints work well when the affected subset of the data does not change over time. However, removing them completely in order to track dynamic patterns leads to ignoring valuable temporal information. Dynamic Subset Scan with temporal consistency and connectivity constraints provides a scalable solution for future work in dynamic pattern detection in graph-based or sensor network data.

Chapter 7

Conclusions and Possible Extensions

The two overarching goals of this thesis are 1) appropriately evaluating a subset's anomalousness through probabilistically founded scoring functions and 2) developing algorithms to efficiently scan over subsets of the data, finding those subsets which maximize the scoring functions subject to various constraints. Achieving these goals resulted in significant contributions to pattern detection through subset scanning in three broad categories: development of scalable methods, introduction of novel, probabilistic-based theory, and experimental results in multiple domains of application related to public health. This concluding chapter summarizes these contributions and highlights some possible extensions for future work.

Chapter 2 introduced the GraphScan algorithm which enforces connectivity constraints on the subset scan. GraphScan efficiently identifies anomalous *connected* subsets of locations and achieves speed improvements of 450,000x faster than the state-of-the-art. Despite this large speed increase, we proved that GraphScan is guaranteed to identify the highest scoring connected subset. GraphScan was applied in the public health domain by scanning through Emergency Room respiratory complaints and showed high detection power for simulated outbreaks along rivers and highways. These irregularly shaped outbreaks were connected in a simple, underlying ZIP code adjacency graph (i.e. if two ZIP codes shared a boundary they were connected). Further work in this field could create more sophisticated graph structures for GraphScan to optimize over. These novel graphs could add additional edges such as public transportation routes and remove some adjacency-based edges that do not reflect a societal connection.

Chapter 3 introduced and formalized the Additive Linear Time Subset Scanning (ALTSS) property. This property of commonly used scoring functions allows prior information to be efficiently incorporated into the subset scan. We demonstrated that expectation-based scan statistics from the exponential family may be written as *additive* set functions when conditioning on the relative risk, q. Additive functions satisfy two simple and important corollaries. First, these functions are easy to optimize by including only the data elements making a positive contribution to the scoring function. Second, additional element-specific terms may be introduced to the scan statistic while remaining an additive set function. We proceeded to prove that only linearly many subsets need to be scanned in order to identify the highest scoring *penalized* subset. Finally, these penalty terms may be viewed as the prior log-odds for the data element to be included in the highest scoring subset. This assumption provides two easily interpretable results in the subset scanning framework. The highest-scoring penalized subset $S^* = \arg\max_S F_{pen}(S)$ minimizes the total probability of error, and is also a maximum a-posteriori (MAP) estimate of the true affected subset S_{true}.

Penalized Fast Subset Scanning (PFSS) is the practical implementation of the properties of ALTSS. Chapter 4 provided one possible interpretation of these penalty terms as "soft" proximity constraints. Soft proximity constraints penalize spatially dispersed clusters by basing the prior probability of inclusion for a ZIP code on its distance from the center of the outbreak. We demonstrated that these constraints increase the detection power and spatial accuracy of simulated aerosolized Anthrax attacks over a populated area.

Chapters 3 and 4 provide the basis for many possible extensions. First, PFSS is very general and readily adapted to a variety of more sophisticated prior log-odds models. However, the more intriguing extension of this work is to develop *subset-level* penalty terms. Currently, the ALTSS property and the resulting PFSS algorithm assume that the penalty terms are element-specific and cannot depend on other individual records or the subset in general. For example, the soft proximity constraints of Chapter 4 are based on the distance from an individual location to a fixed center. We are currently not able to penalize a location based on its distance to the closest included location because that penalty term is no longer element-specific. Efficiently optimizing over subsets constrained by subset-level penalty terms will drastically increase the type of constraints that may be used in the subset scanning framework.

Chapters 5 and 6 introduced Dynamic Subset Scanning. The Dynamic Subset Scan was designed to increase the detection power for dynamic patterns that change the affected subset over time. Chapter 5 builds on the ALTSS property by developing a generative model for the prior log-odds of a record to be included in the subset at time step t based on the optimal subset at time step $t-1$. These *temporal consistency* constraints are derived to allow temporal information to be shared both forward and backward in time. The Dynamic Subset Scan enforces both temporal consistency constraints and connectivity constraints and Chapter 6 detailed how connectivity constraints may be enforced on additive functions through Additive GraphScan. Additive GraphScan is a heuristic alternative to the original GraphScan algorithm introduced in Chapter 2. The Dynamic Scan was used to track, source-trace, and predict contaminant plumes spreading through a water distribution system equipped with noisy binary sensors.

One possible extension of the Dynamic Subset Scan is to recognize that subset scanning may not be the best method for the prediction task of dynamic patterns. Some domains may benefit from knowing the individual probabilities that a given location will be affected at time $t + 1$ (as compared to the most anomalous subset at that time) and it is unclear how to generate these probabilities for a look-ahead window beyond $t + 1$.

A more sophisticated extension of the Dynamic Subset Scan is to consider the ramifications of a *dynamic graph* that changes edges over time. This scenario applies to our complex food supply chain and may be used to quickly identify food-borne contaminants that potentially spread from farms to whole-sale distributors to local grocers and ultimately to consumers. Finally, this type of extension could also be used in contact-tracing scenarios for identifying the spread of a contagious disease spread by proximity or physical contact.

Bibliography

[1] D. Agarwal, A. McGregor, J. M. Phillips, S. Venkatasubramanian, and Z. Zhu. Spatial scan statistics: approximations and performance study. In *Proc. 12th ACM SIGKDD Intl. Conf. on Knowledge Discovery and Data Mining*, pages 24–33, 2006.

[2] L. Akoglu, D. H. Chau, C. Faloutsos, N. Tatti, H. Tong, and J. Vreeken. Mining connection pathways for marked nodes in large graphs. In *SDM*, pages 37–45, 2013.

[3] Avi Ostfeld et al. The battle of water sensor networks: A design challenge for engineers and algorithms. *Journal of Water Resources Planning and Management*, 134(6):556–568, 2008.

[4] J. Berry, R. D. Carr, W. Hart, V. J. Leung, C. A. Phillips, and J. P. Watson. Designing contamination warning systems for municipal water networks using imperfect sensors. *Journal of Water Resources Planning and Management*, 135(4):253–263, 2009.

[5] J. Berry, L. Fleischer, W. Hart, and C. Phillips. Sensor placement in municipal water networks. *J. Water*, 131:237–243, 2003.

[6] P. Bogdanov, C. Faloutsos, M. Mongiovì, E. E. Papalexakis, R. Ranca, and A. K. Singh. Netspot: Spotting significant anomalous regions on dynamic networks. In *Proceedings of the 13th SIAM International Conference on Data Mining, May 2-4, 2013. Austin, Texas, USA.*, pages 28–36, 2013.

[7] D. L. Buckeridge, H. S. Burkom, A. W. Moore, J. A. Pavlin, P. N. Cutchis, and W. R. Hogan. Evaluation of syndromic surveillance systems: development of an epidemic simulation model. *Morbidity and Mortality Weekly Report*, 53 (Supplement):137–143, 2004.

[8] A. L. Cancado, A. R. Duarte, L. H. Duczmal, S. J. Ferreira, C. M. Fonseca, and E. C. Gontijo. Penalized likelihood and multi-objective spatial scans for the detection and inference of irregular clusters. *International Journal of Health Geographics*, 9:55, 2010.

[9] L. Duczmal and R. Assuncao. A simulated annealing strategy for the detection of arbitrary shaped spatial clusters. *Computational Statistics and Data Analysis*, 45:269–286, 2004.

[10] L. Duczmal, A. L. Cancado, R. H. Takahashi, and L. F. Bessegato. A genetic algorithm for irregularly shaped spatial scan statistics. *Computational Statistics and Data Analysis*, 52:43–52, 2007.

[11] P. Erdos and A. Renyi. On random graphs i. *Publicationes Mathematicae*, 1959.

[12] M. Ester, H. Kriegel, J. S, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. pages 226–231. AAAI Press, 1996.

[13] G. W. Flake, S. Lawrence, and C. L. Giles. Efficient identification of web communities. In *In Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 150–160. ACM Press, 2000.

[14] J. Friedman and N. Fisher. Bump hunting in high-dimensional data. *Statistics and Computing*, 9(2):123–143, 1999.

[15] M. Gomez-Rodriguez, J. Leskovec, and A. Krause. Inferring networks of diffusion and influence. In *Proc. 16th ACM SIGKDD Conf. on Knowledge Discovery and Data Mining*, pages 1019–1028, 2010.

[16] W. R. Hogan, G. F. Cooper, G. L. Wallstrom, M. M. Wagner, and J.-M. Depinay. The bayesian aerosol release detector: An algorithm for detecting and characterizing outbreaks caused by an atmospheric release of bacillus anthracis. *Stat Med*, 26:5225–5252, 2007.

[17] M. W. Koch and S. A. Mckenna. Distributed sensor fusion in water quality event detection. *Journal of Water Resources Planning and Management*, 137:10–19, 2011.

[18] A. Krause, J. Leskovec, C. Guestrin, J. VanBriesen, and C. Faloutsos. Efficient sensor placement optimization for securing large water distribution networks. *Journal of Water Resources Planning and Management*, 134(6):516–526, November 2008.

[19] M. Kulldorff. A spatial scan statistic. *Communications in Statistics: Theory and Methods*, 26(6):1481–1496, 1997.

[20] M. Kulldorff, L. Huang, L. Pickle, and L. Ducmzal. An elliptic spatial scan statistic. *Statistics in Medicine*, 25:3929–3943, 2006.

[21] A. H. Land and A. G. Doig. An automatic method of solving discrete programming problems. *Econometrica*, 28(3):497–520, 1960.

[22] T. Lappas, E. Terzi, D. Gunopulos, and H. Mannila. Finding effectors in social networks. In *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '10, pages 1059–1068, New York, NY, USA, 2010. ACM.

[23] J. Leskovec, A. Krause, C. Guestrin, C. Faloutsos, J. VanBriesen, and N. Glance. Cost-effective outbreak detection in networks. In *Proc. 13th Intl. Conf. on Knowledge Discovery and Data Mining*, 2007.

[24] E. McFowland, S. Speakman, and D. B. Neill. Fast generalized subset scan for anomalous pattern detection. *Journal of Machine Learning Research*, 14:1533–1561, 2013.

[25] S. Myers and J. Leskovec. On the convexity of latent social network inference. In *Advances in Neural Information Processing Systems 23*, pages 1741–1749. 2010.

[26] D. B. Neill. Detection of spatial and spatio-temporal clusters. Technical Report CMU-CS-06-142, Ph.D. thesis, Carnegie Mellon University, School of Computer Science, 2006.

[27] D. B. Neill. An empirical comparison of spatial scan statistics for outbreak detection. *International Journal of Health Geographics*, 8:20, 2009.

[28] D. B. Neill. Expectation-based scan statistics for monitoring spatial time series data. *International Journal of Forecasting*, 25:498–517, 2009.

[29] D. B. Neill. Fast subset scan for spatial pattern detection. *Journal of the Royal Statistical Society (Series B: Statistical Methodology)*, 74(2):337–360, 2012.

[30] D. B. Neill and G. F. Cooper. A multivariate Bayesian scan statistic for early event detection and characterization. *Machine Learning*, 79:261–282, 2010.

[31] D. B. Neill and A. W. Moore. Rapid detection of significant spatial clusters. In *Proc. 10th ACM SIGKDD Conf. on Knowledge Discovery and Data Mining*, pages 256–265, 2004.

[32] D. B. Neill, A. W. Moore, and G. F. Cooper. A Bayesian spatial scan statistic. In *Advances in Neural Information Processing Systems 18*, pages 1003–1010, 2006.

[33] D. B. Neill, A. W. Moore, M. R. Sabhnani, and K. Daniel. Detection of emerging space-time clusters. In *Proc. 11th ACM SIGKDD Conf. on Knowledge Discovery and Data Mining*, 2005.

[34] G. Nemhauser, L. Wolsey, and M. Fisher. An analysis of the approximations for maximizing submodular set functions. *Mathematical Programming*, 14:265–294, 1978.

[35] G. P. Patil and C. Taillie. Upper level set scan statistic for detecting arbitrarily shaped hotspots. *Envir. Ecol. Stat.*, 11:183–197, 2004.

[36] B. A. Prakash, J. Vreeken, and C. Faloutsos. Spotting culprits in epidemics: How many and which ones? In M. J. Zaki, A. Siebes, J. X. Yu, B. Goethals, G. I. Webb, and X. Wu, editors, *ICDM*, pages 11–20. IEEE Computer Society, 2012.

[37] S. Speakman, E. McFowland, and D. B. Neill. Scalable detection of anomalous patterns with connectivity constraints. *Journal of Computational and Graphical Statistics*, 2014 10.1080/10618600.2014.960926.

[38] S. Speakman, S. Somanchi, E. Mcfowland, and D. B. Neill. Penalized fast subset scanning. *Submitted for review.*

[39] S. Speakman, Y. Zhang, and D. B. Neill. Dynamic pattern detection with temporal consistency and connectivity constraints. In *Proc. 13th IEEE International Conference on Data Mining*, 2013.

[40] T. Tango and K. Takahashi. A flexibly shaped spatial scan statistic for detecting clusters. *International Journal of Health Geographics*, 4:11, 2005.

[41] R. Viswanathan and P. K. Varshney. Distributed detection with multiple sensors: Part I Fundamentals. *Proceedings of the IEEE*, pages 54–63, 1997.

[42] G. L. Wallstrom, M. M. Wagner, and W. R. Hogan. High-fidelity injection detectability experiments: a tool for evaluation of syndromic surveillance systems. *Morbidity and Mortality Weekly Report*, 54 (Supplement):85–91, 2005.

[43] B. Wang, J. M. Phillips, R. Schrieber, D. Wilkinson, N. Mishra, and R. Tarjan. Spatial scan statistics for graph clustering. In *Proc. 8th SIAM Intl. Conf. on Data Mining*, pages 727–738, 2008.

[44] M. Wu, X. Song, C. Jermaine, S. Ranka, and J. Gums. A LRT framework for fast spatial anomaly detection. In *Proc. 15th ACM SIGKDD Intl. Conf. on Knowledge Discovery and Data Mining*, pages 887–896, 2009.

[45] N. Yiannakoulias, R. J. Rosychuk, and J. Hodgson. Adaptations for finding irregularly shaped disease clusters. *International Journal of Health Geographics*, 6:28, 2005.