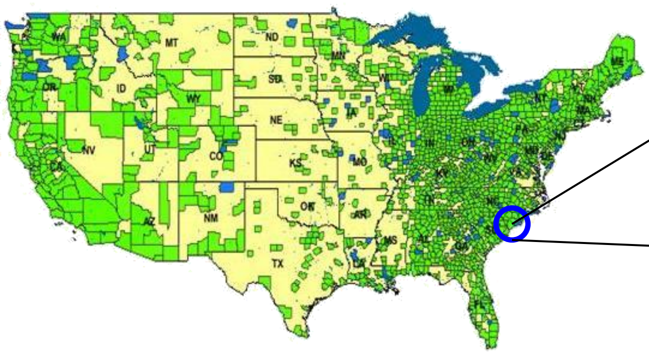
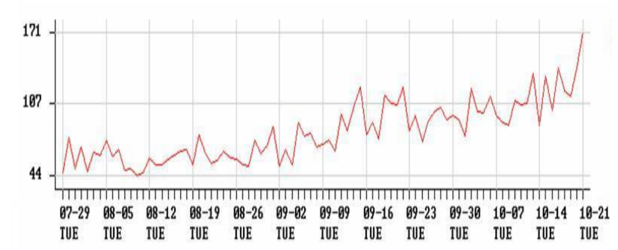
Scalable Detection of Anomalous Patterns with Connectivity Constraints

Skyler Speakman, Ed McFowland III, Daniel B. Neill

Event and Pattern Detection Lab
H.J. Heinz III College
Carnegie Mellon University
This work was partially supported by NSF grants IIS-0916345, IIS-0911032, and IIS-0953330



Daily health data from thousands of hospitals and pharmacies nationwide



Time series of counts c_i^t for each zip code s_i

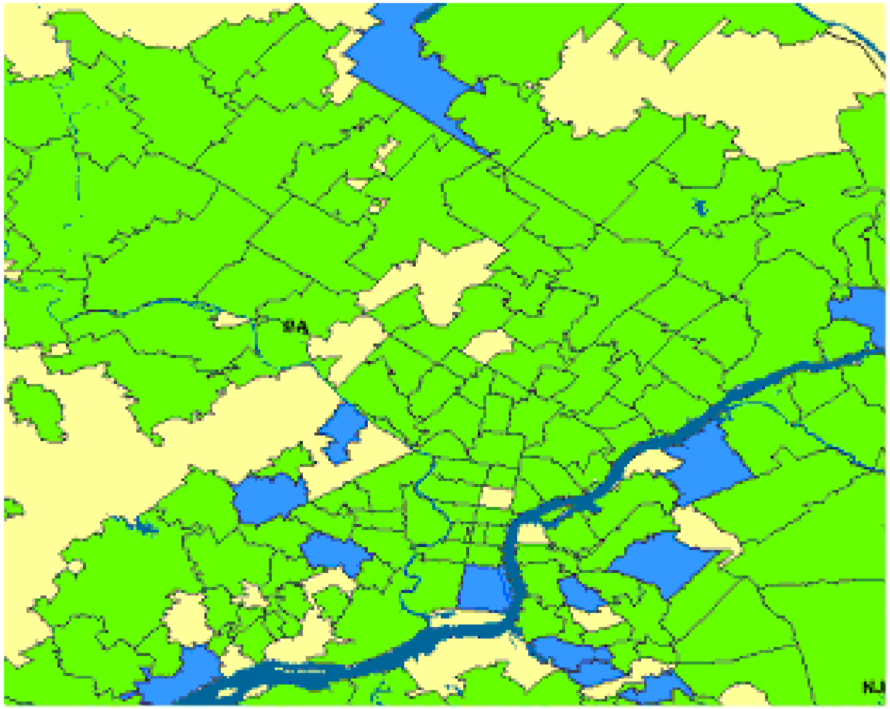
Use this data to detect **anomalous patterns**

Detect any emerging events (i.e. outbreaks of disease)

Pinpoint the affected areas

Biosurveillance

(Kulldorff, 1997; Neill and Moore, 2005)



Scan over multiple regions to detect where counts are higher than expected

Aggregate the individual counts from each location within a region

$$C = \sum_{S} c_i^t \text{ and } B = \sum_{S} b_i^t$$

Determine *anomalousness* of region with a scoring function

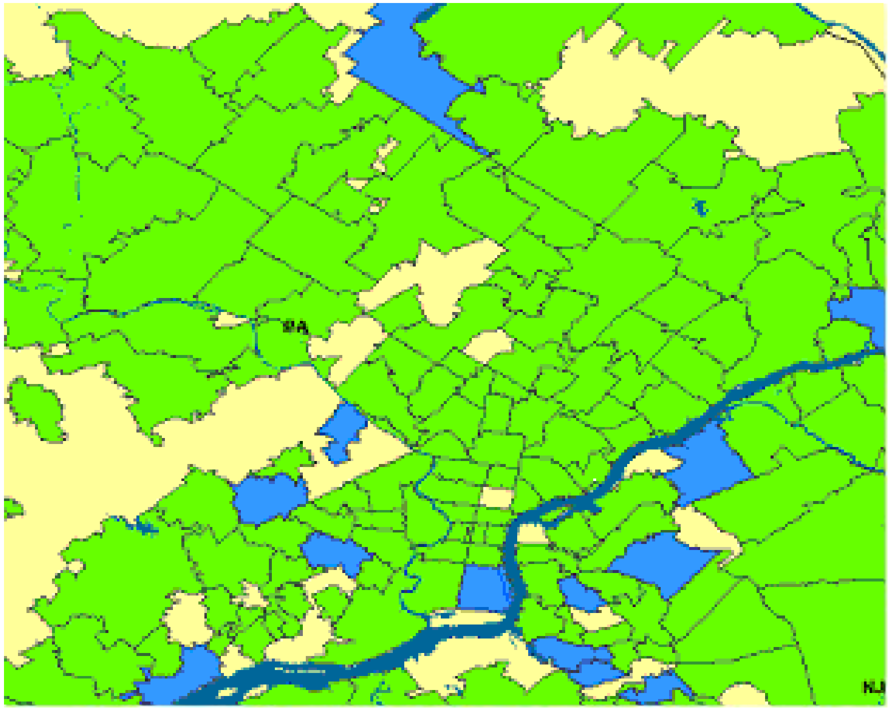
$$F(S) = \frac{\Pr(Data \mid H_1(S))}{\Pr(Data \mid H_0)}$$

$$F(S) = \left(\frac{C}{B}\right)^C e^{B-C}$$

Expectation-Based Scan Statistics

Expectation-Based Scan Statistics

(Kulldorff, 1997; Neill and Moore, 2005)



Scan over multiple regions to detect where counts are higher than expected

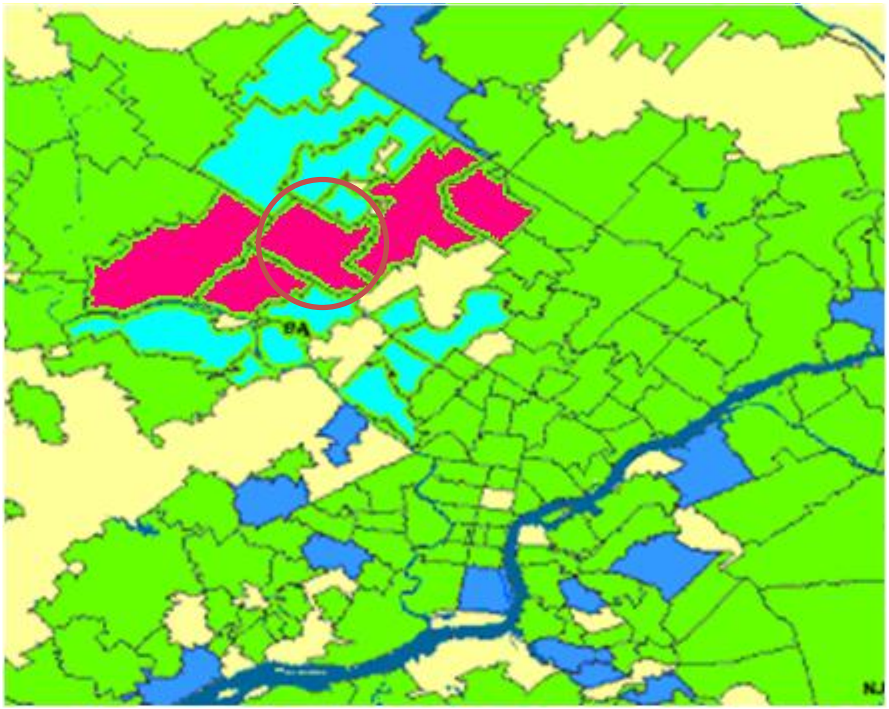
Aggregate the individual counts from each location within a region

Circles

Choose a center location s_c and its k nearest neighbors

Find the circle that maximizes the score function of the aggregated counts and baselines

(Kulldorff, 1997; Neill and Moore, 2005)



Power to Detect

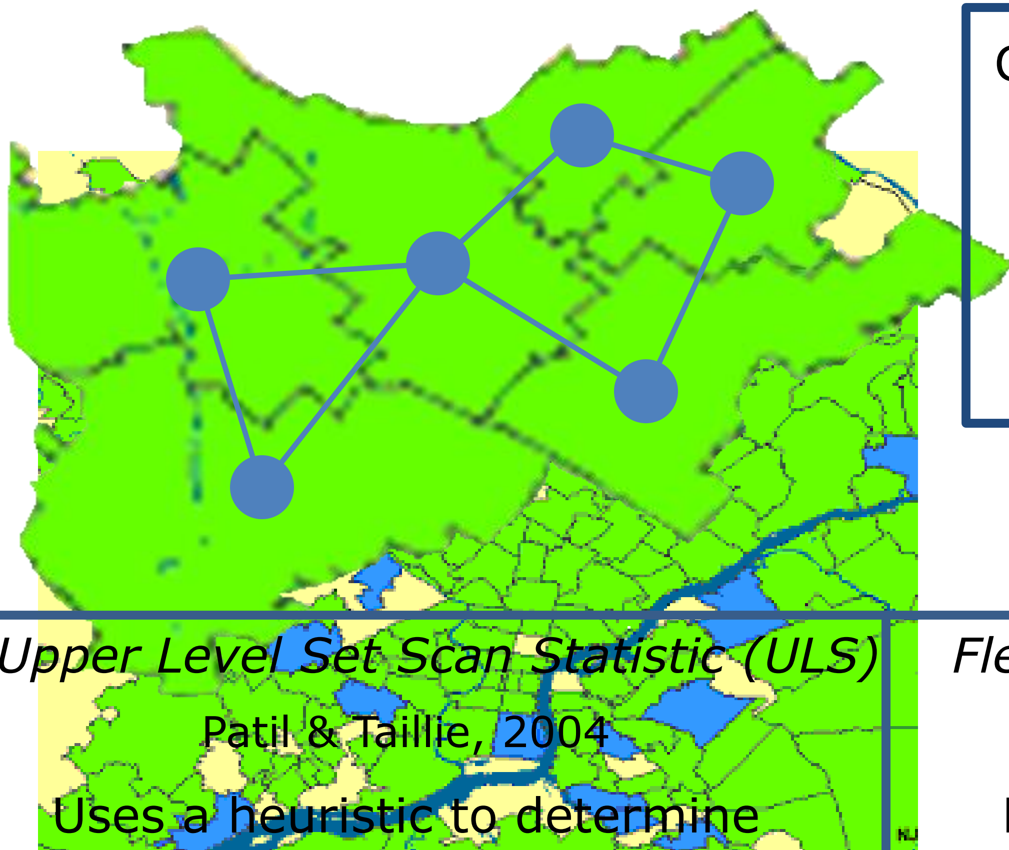
Circles are useful for detecting tightly clustered outbreaks

However, they lose power to detect abnormally shaped clusters

Affected locations

Un-affected locations contributing to region score

Expectation-Based Scan Statistics



Create an adjacency graph of the locations and score ***connected subsets***

Increase power to detect non-circular clusters

Upper Level Set Scan Statistic (ULS)

Patil & Taillie, 2004

Uses a heuristic to determine high scoring connected subsets
Is not guaranteed to find the highest scoring connected subset

Flexible Scan statistic (FlexScan)

Tango & Takahashi, 2005

Naively scores all connected subsets
Infeasible for regions of >30 locations

Connectivity Constraints

(Neill, 2008)

PROBLEM:	The number of subsets grows exponentially with the size of the region 2^N

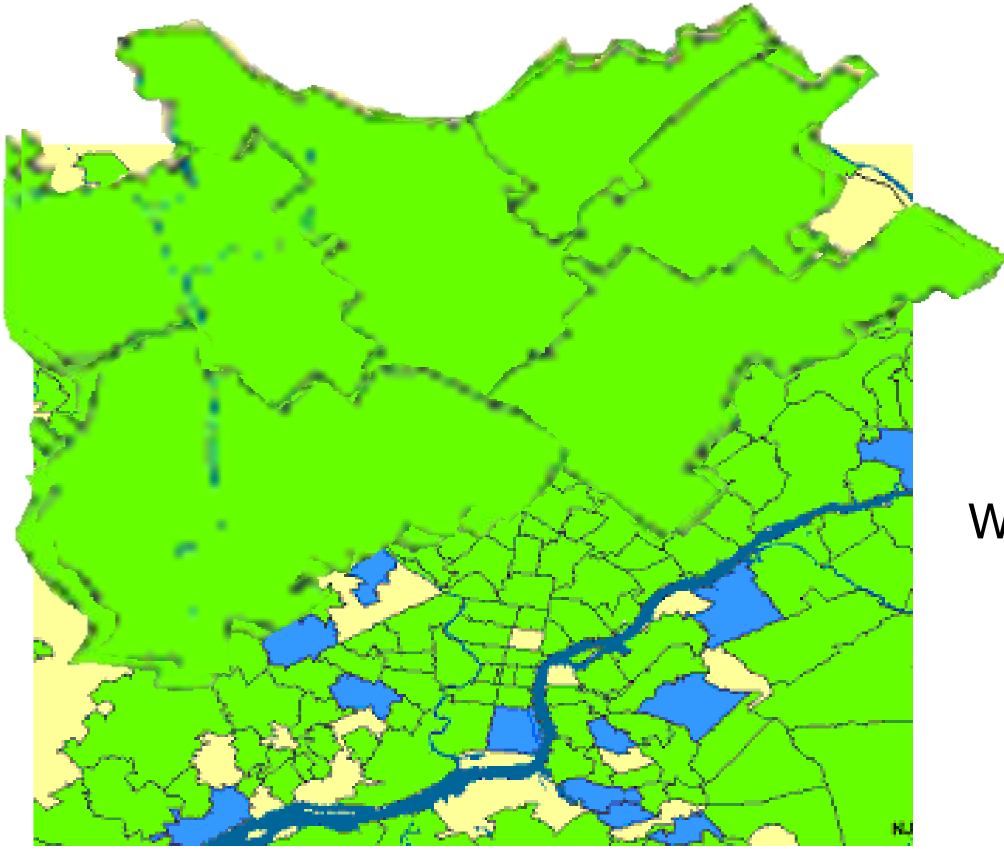
This makes it computationally infeasible for regions with more than ~30 locations

SOLUTION:	Exploit a property of scoring functions to rule out subsets that cannot obtain the highest score

This reduction in the search space allows for exact and efficient calculation of the highest scoring ***unconstrained* subset**

EXTENSION:	Use this same property for exact and efficient calculation of the highest scoring ***connected subset***

Subset Scanning



(Neill, 2008)

We wish to maximize a scoring function

$$F(S) = F\left(\sum_{s_i \in S} c_i, \sum_{s_i \in S} b_i\right)$$

over all possible subsets, S

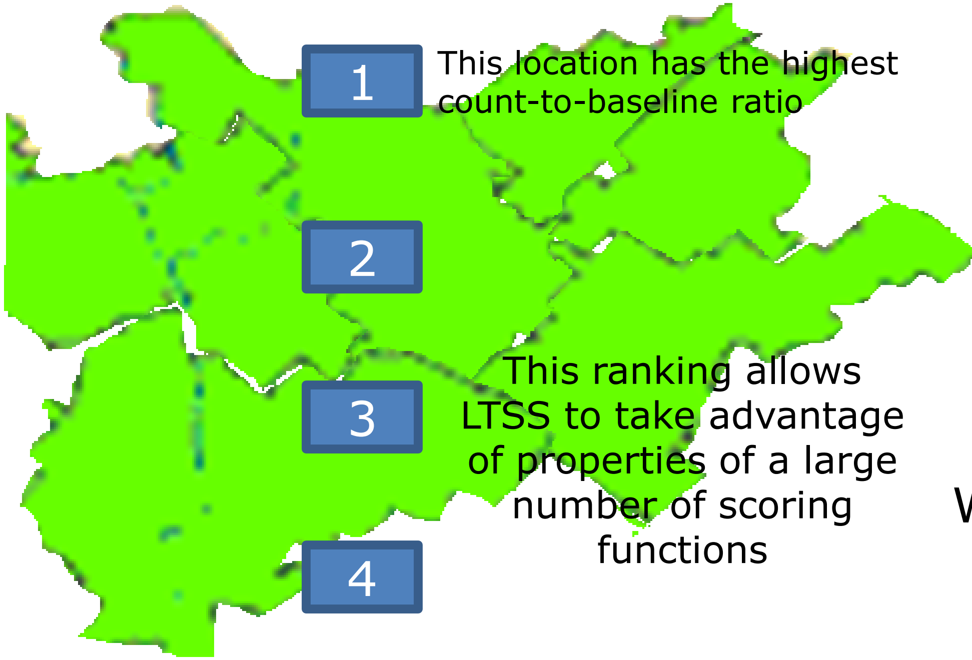
We sort the locations according to a priority function

For example,

$$G(s_i) = \frac{c_i}{b_i}$$

Works for expectation-based Poisson (EBP)

Linear Time Subset Scanning



1 This location has the highest count-to-baseline ratio

2

3 This ranking allows LTSS to take advantage of properties of a large number of scoring functions

4

5

•
•
•

N This location has the lowest count-to-baseline ratio

(Neill, 2008)

We wish to maximize a scoring function

$$F(S) = F\left(\sum_{s_i \in S} c_i, \sum_{s_i \in S} b_i\right)$$

over all possible subsets, S

We sort the locations according to a relevance criteria

For example,

$$G(s_i) = \frac{c_i}{b_i}$$

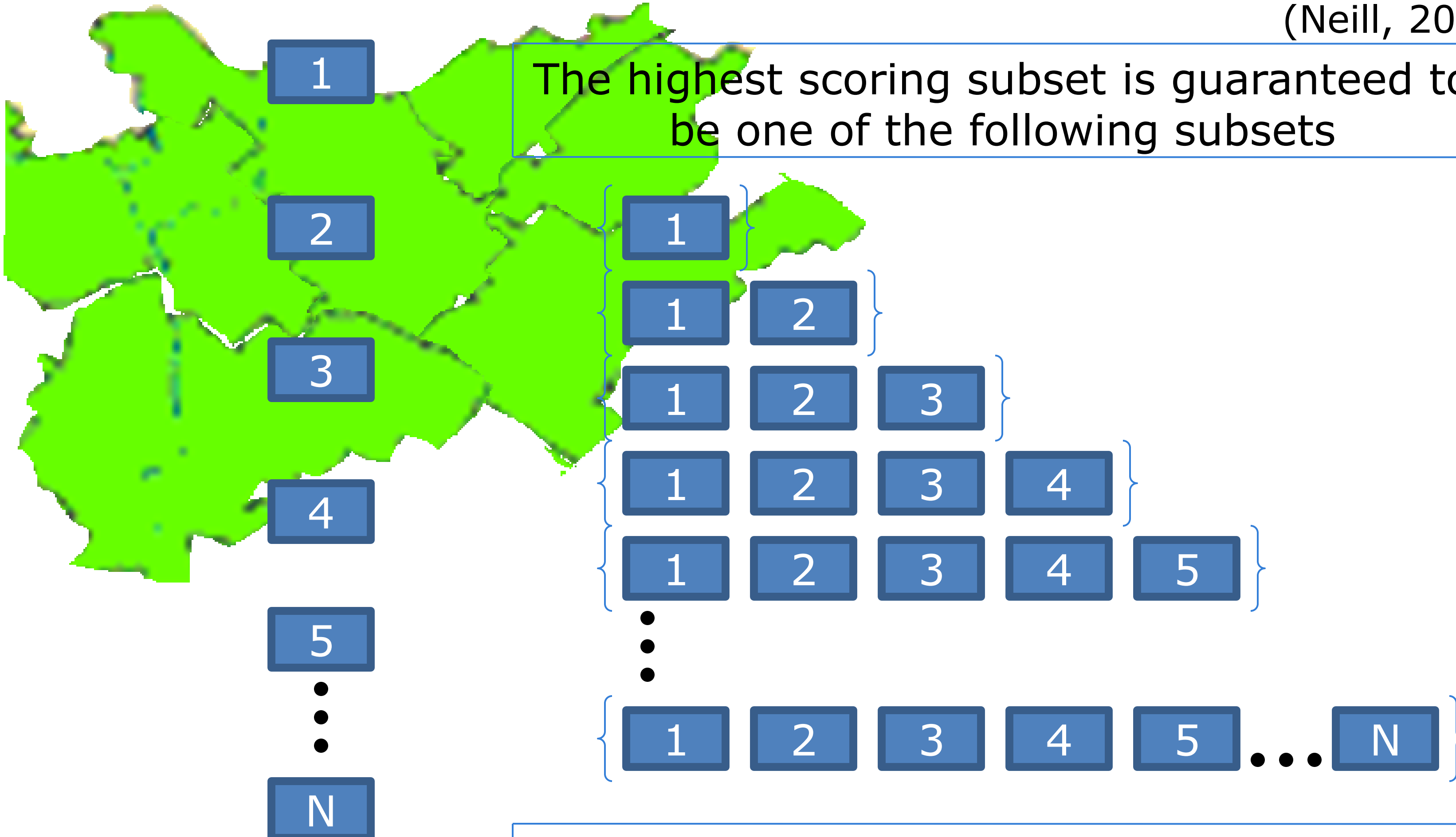
works for Expectation-based Poisson (EBP)

Linear Time Subset Scanning

Linear Time Subset Scanning

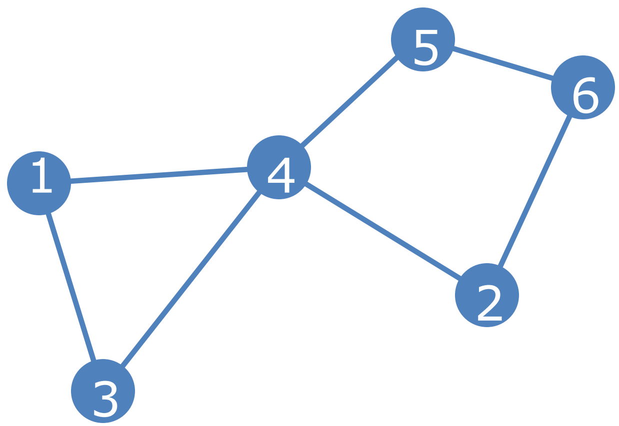
(Neill, 2008)

The highest scoring subset is guaranteed to be one of the following subsets



Decreases the search space from 2^N to N

LTSS with Connectivity Constraints



Use property of LTSS to reduce the search space and rule out a large number of connected subsets

Rank the locations according to priority function

Remove subsets that are guaranteed to be suboptimal

GraphScan Logic:

If location $s_{(k)}$ is contained in the optimal subset S*
and if removing $s_{(k)}$ does not disconnect the subgraph,
any higher priority **neighbor** of $s_{(k)}$ will also be contained in S*.

We represent groups of subsets as a string of 0's, 1's, and ?'s

Priority Ranking	1	2	3	4	5	6
Bit String	1	0	0	1	?	?

The above bit string represents 4 possible subsets:
{1,4} {1,4,5} {1,4,6} {1,4,5,6}

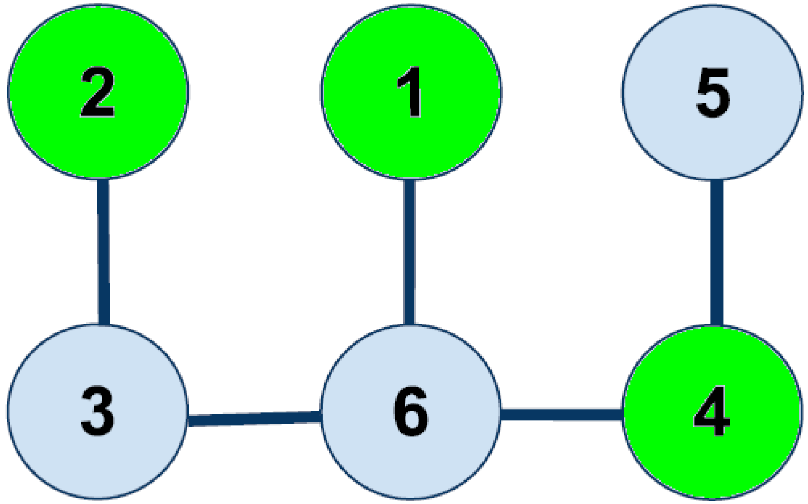
A Naïve approach would search all 2^N subsets and is computationally infeasible

1	2	3	4	5	6
?	?	?	?	?	?

1	2	3	4	5	6
1	?	?	?	?	?
0	?	?	?	?	?

1	2	3	4	5	6
1	1	?	?	?	?
1	0	?	?	?	?
0	?	?	?	?	?

GraphScan Algorithm

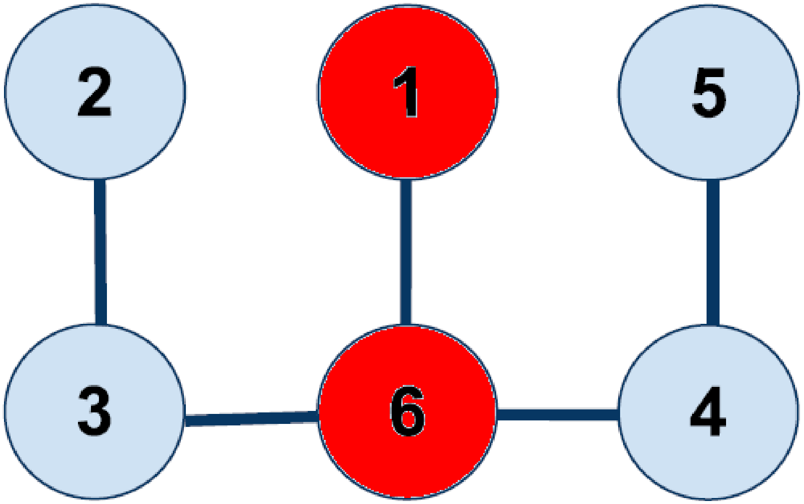


Seed nodes have higher priority than all of their neighbors

We can rule out bit strings whose highest priority node is not a seed node

	1	2	3	4	5	6
S_1	1	?	?	?	?	?
S_2	0	1	?	?	?	?
~~**S_3**~~	~~0~~	~~0~~	~~1~~	~~?~~	~~?~~	~~?~~
S_4	0	0	0	1	?	?
~~**S_5**~~	~~0~~	~~0~~	~~0~~	~~0~~	~~1~~	~~?~~
~~**S_6**~~	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~1~~

GraphScan Algorithm



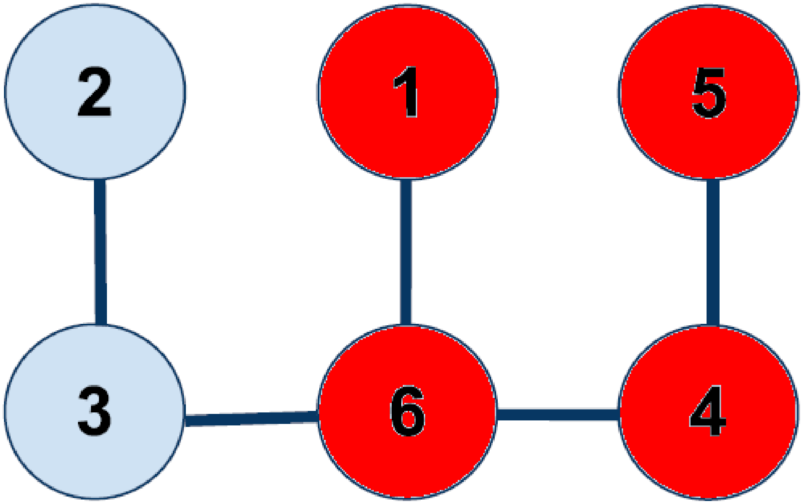
Seed nodes have higher priority than all of their neighbors

We can rule out bit strings whose highest priority node is not a seed node

If we rule out a high priority node, we can also rule out all of its lower priority neighbors...

	1	**2**	**3**	**4**	**5**	**6**
S_1	1	?	?	?	?	?
S_2	0	1	?	?	?	**0**
~~**S_3**~~	~~0~~	~~0~~	~~1~~	~~?~~	~~?~~	~~?~~
S_4	0	0	0	1	?	**0**
~~**S_5**~~	~~0~~	~~0~~	~~0~~	~~0~~	~~1~~	~~?~~
~~**S_6**~~	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~1~~

GraphScan Algorithm



Seed nodes have higher priority than all of their neighbors

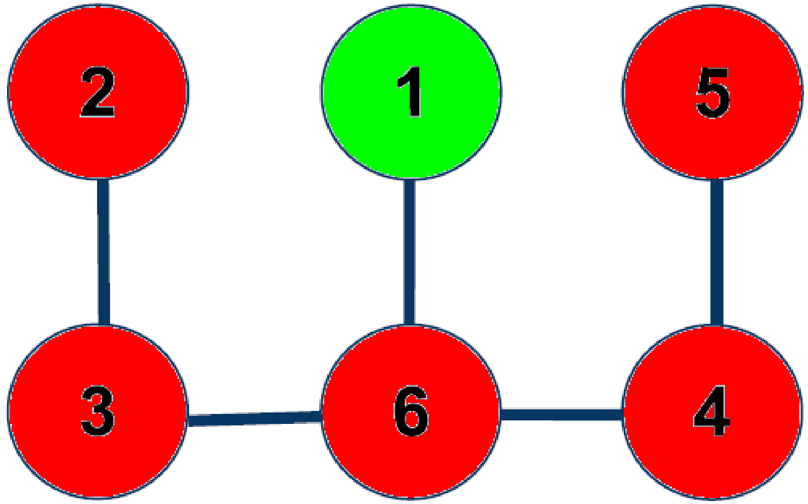
We can rule out bit strings whose highest priority node is not a seed node

	1	2	3	4	5	6
S_1	1	?	?	?	?	?
S_2	0	1	?	**0**	**0**	0
~~S_3~~	~~0~~	~~0~~	~~1~~	~~?~~	~~?~~	~~?~~
S_4	0	0	0	1	?	0
~~S_5~~	~~0~~	~~0~~	~~0~~	~~0~~	~~1~~	~~?~~
~~S_6~~	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~1~~

If we rule out a high priority node, we can also rule out all of its lower priority neighbors…

…and any additional nodes that are disconnected when these nodes are ruled out

GraphScan Algorithm



	1	2	3	4	5	6
S_1	1	?	?	?	?	?
S_2	0	1	?	0	0	0
~~S_3~~	~~0~~	~~0~~	~~1~~	~~?~~	~~?~~	~~?~~
S_4	0	0	0	1	?	0
~~S_5~~	~~0~~	~~0~~	~~0~~	~~0~~	~~1~~	~~?~~
~~S_6~~	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~1~~

Propagation of bit strings:

Pull off $\mathbf{S_1}$ and consider the two cases of including or excluding node 2

Including node 2 implies including nodes 3 and 6

$\mathbf{S_{1a}}$: 1 1 1 ? ? 1

Excluding node 2 implies excluding nodes 3, 4, 5, and 6

$\mathbf{S_{1b}}$: 1 0 0 0 0 0

GraphScan Algorithm

We can improve GraphScan's performance by taking advantage of *unconstrained LTSS directly*

Let UBound($\mathbf{S_j}$) = Highest possible score of any *unconstrained* subset in $\mathbf{S_j}$

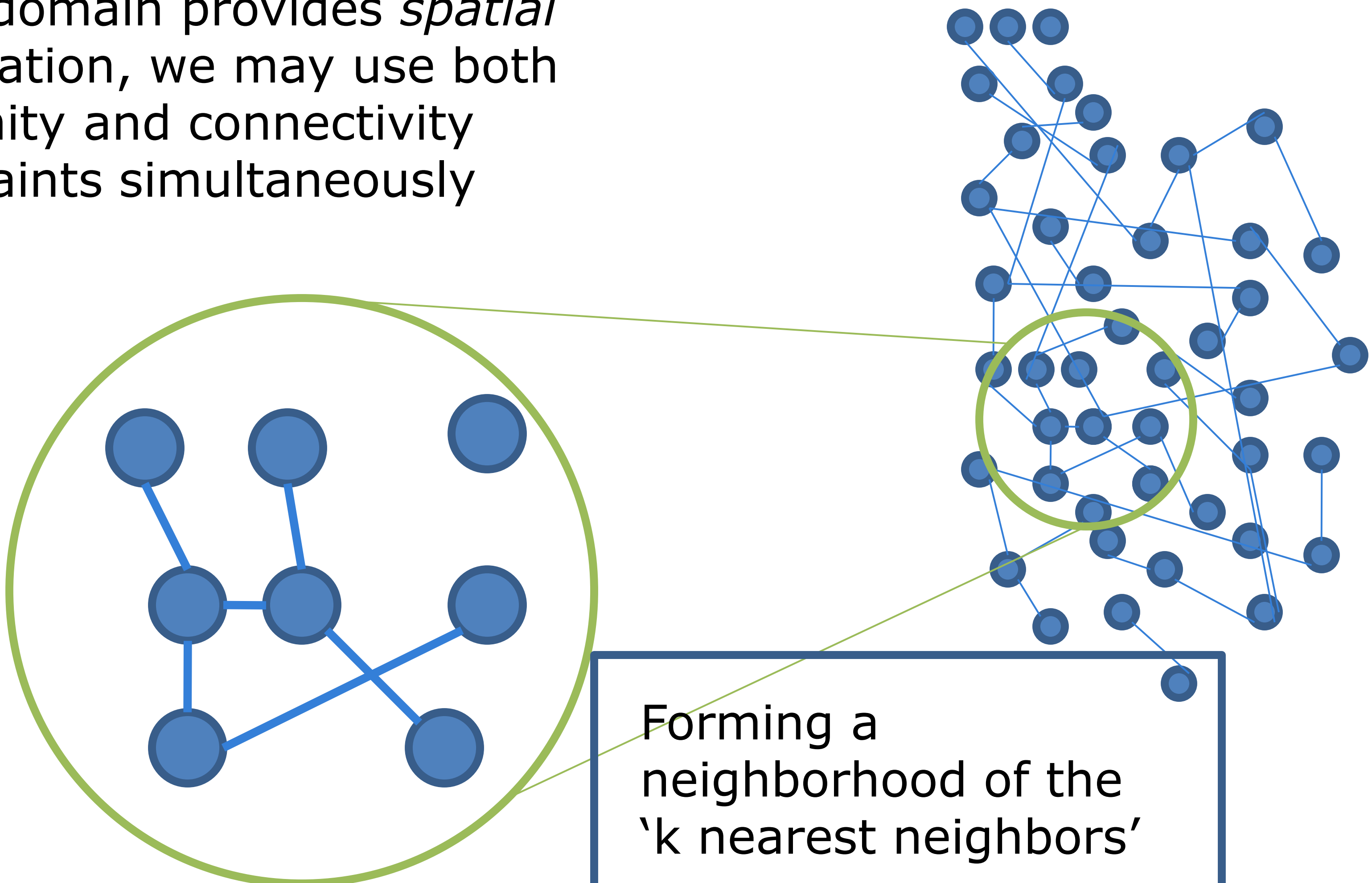
If UBound($\mathbf{S_j}$) ≤ Current best
Stop Processing $\mathbf{S_j}$

$\mathbf{S_j}$ guaranteed to not contain highest scoring subset

Branch and Bounding

Proximity Constraints

If the domain provides *spatial* information, we may use both proximity and connectivity constraints simultaneously

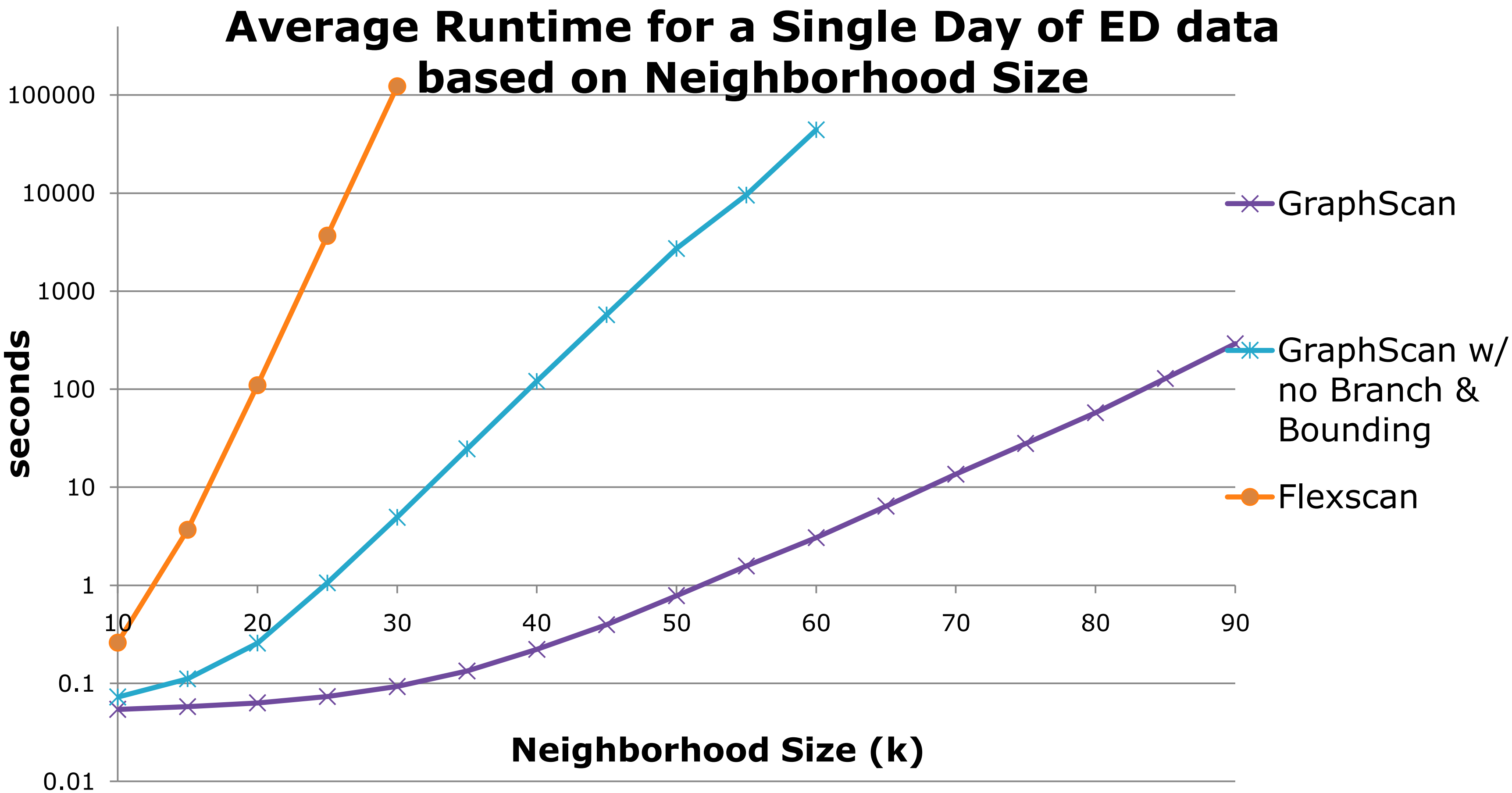


Two years of admissions from 10 different Allegheny County Emergency Departments

The patient's home zip code is used to tally the counts at each location (node)

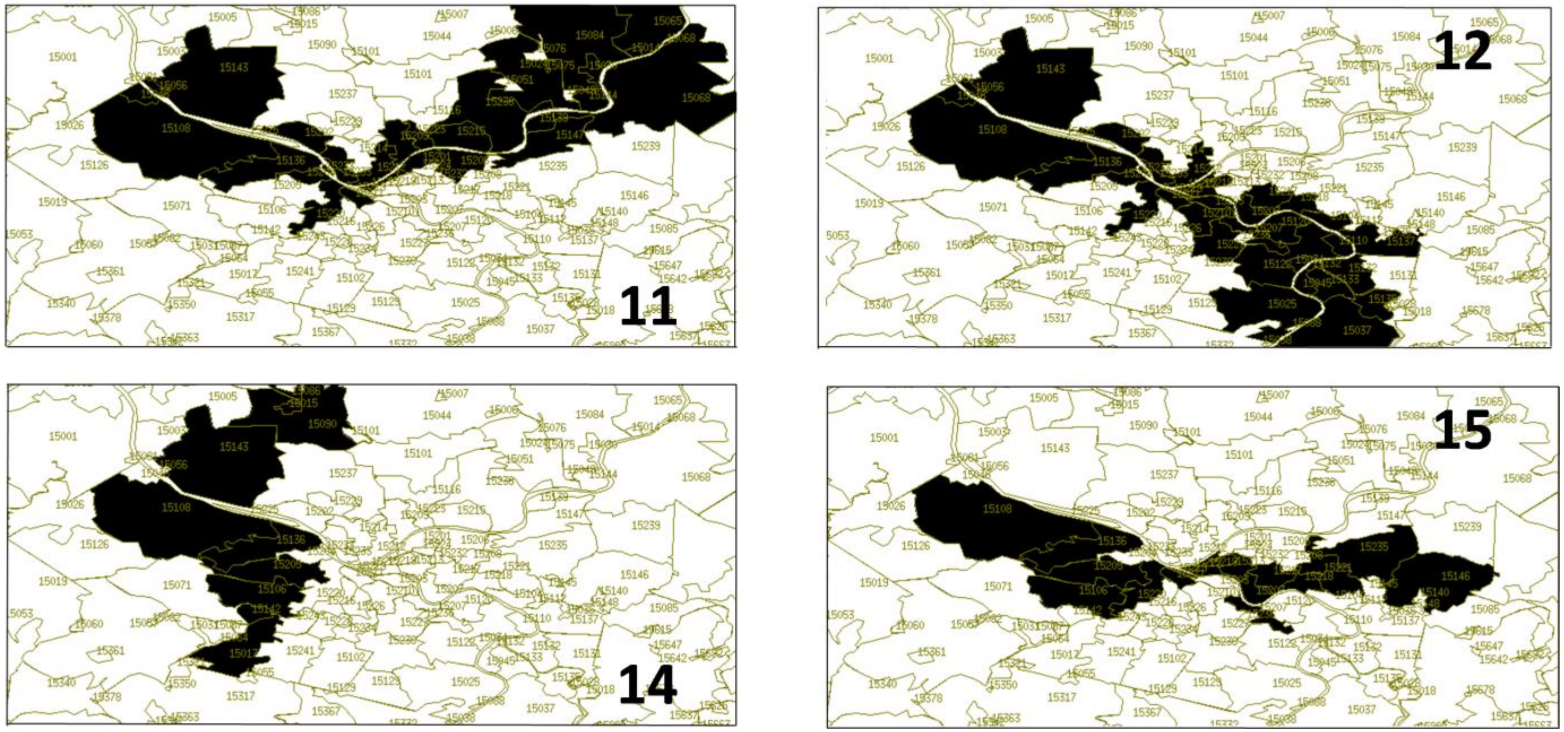
Only consider patients from within Allegheny County

Evaluation: Emergency Department Data

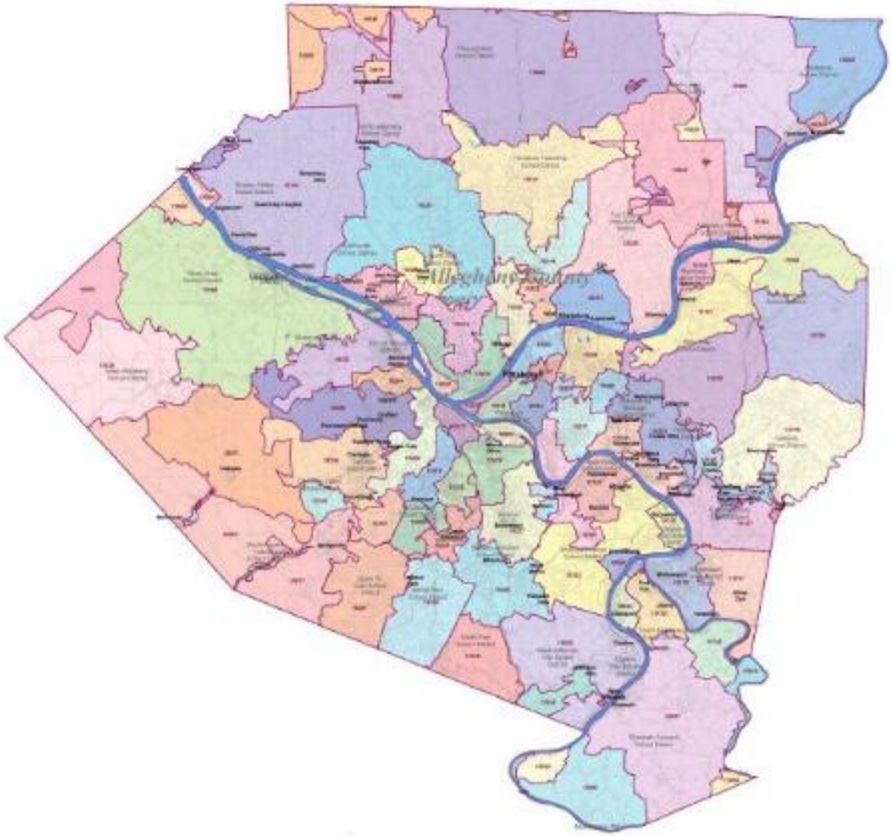


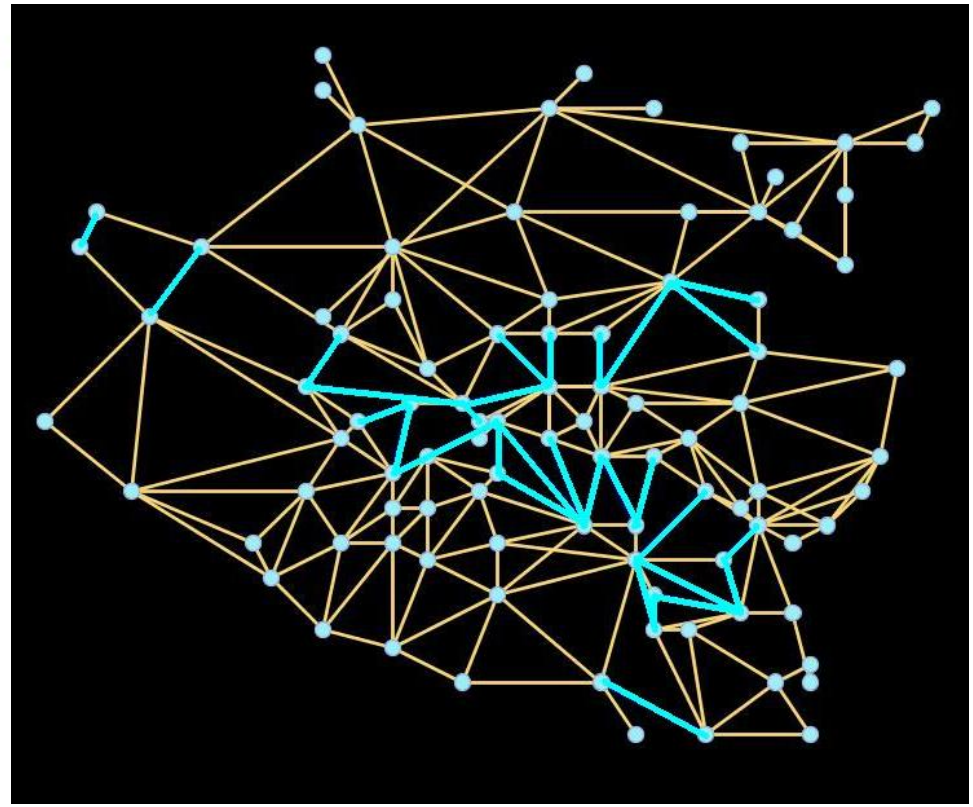
GraphScan is still worst-case exponential

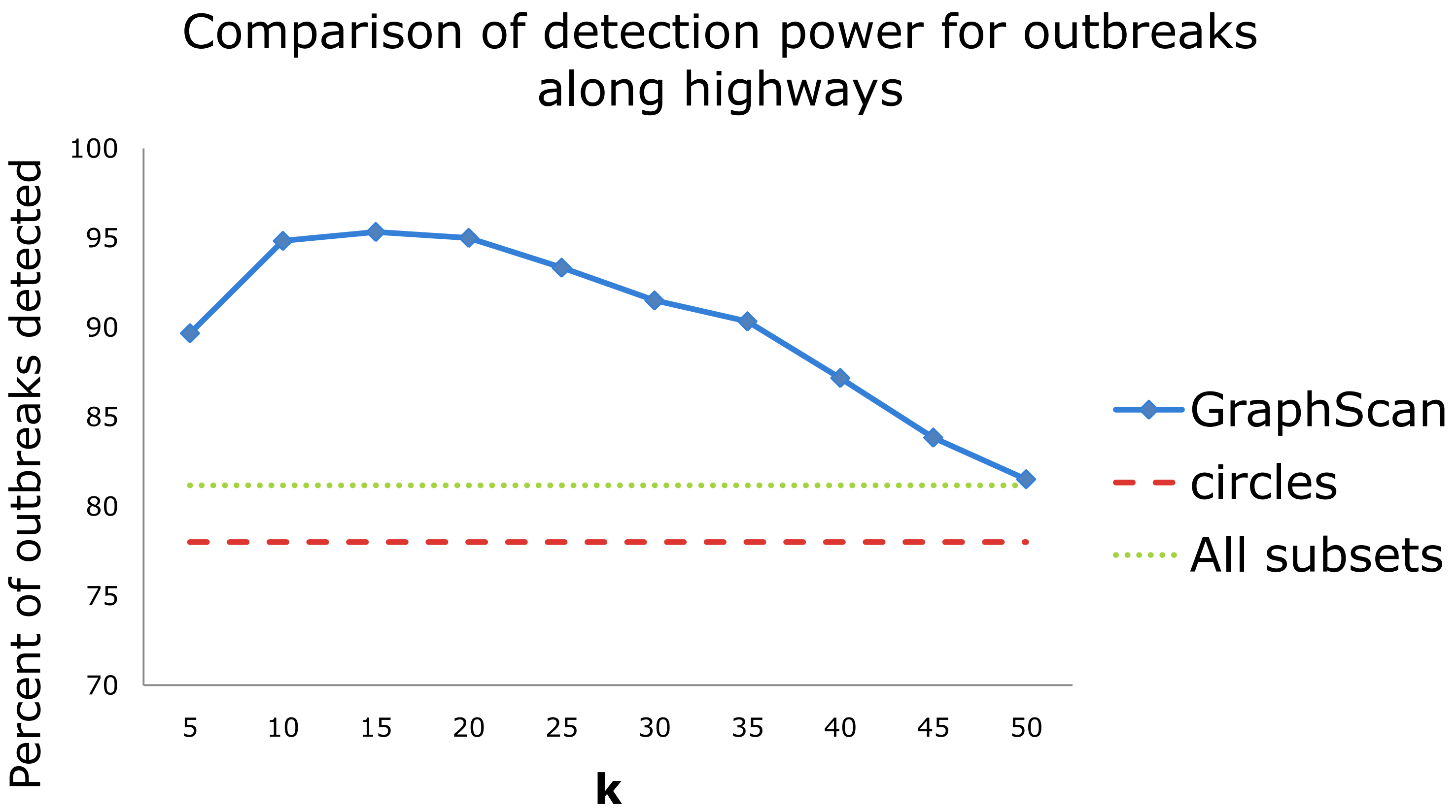
Evaluation: Run times



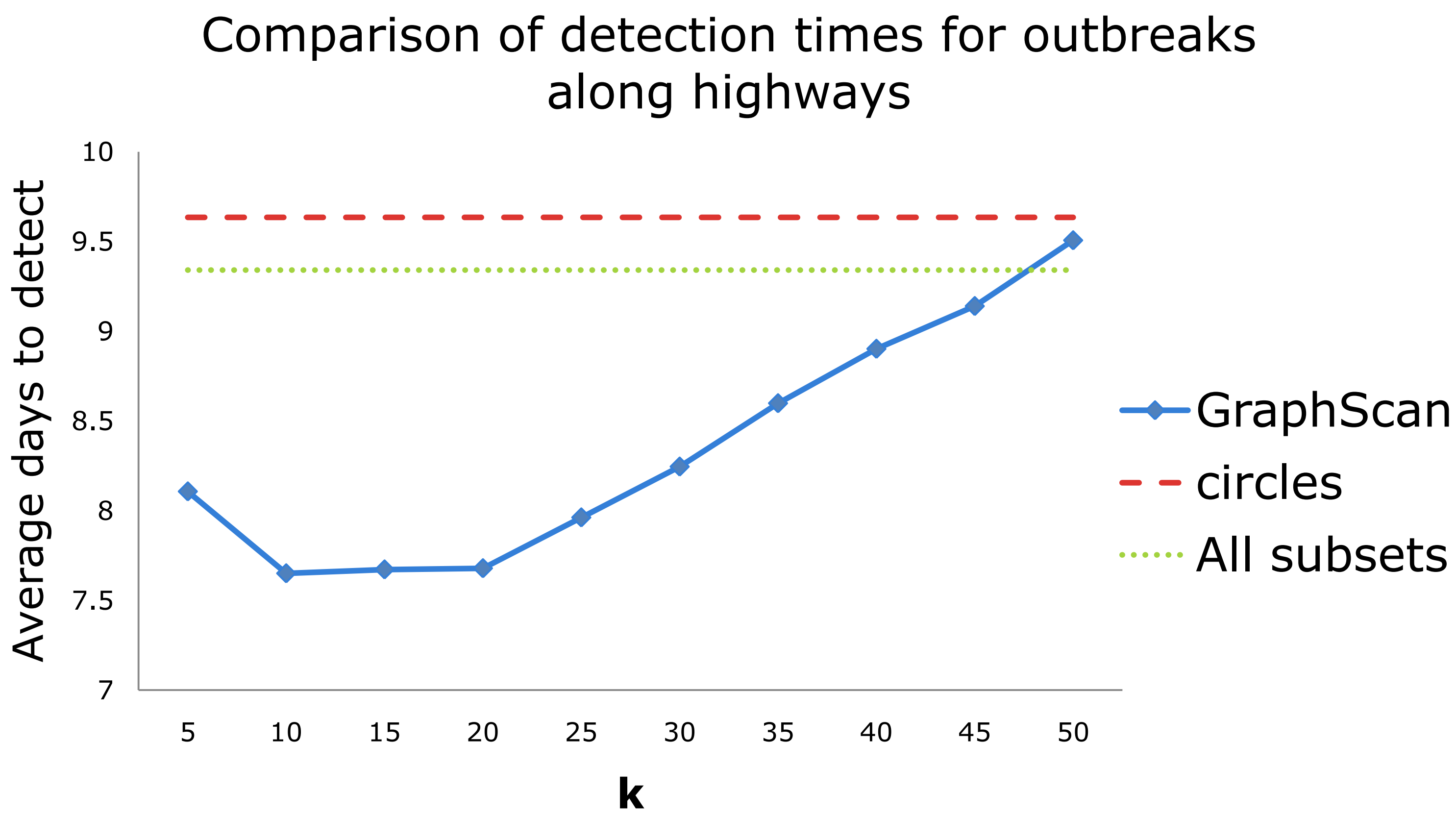
Evaluation: Injects





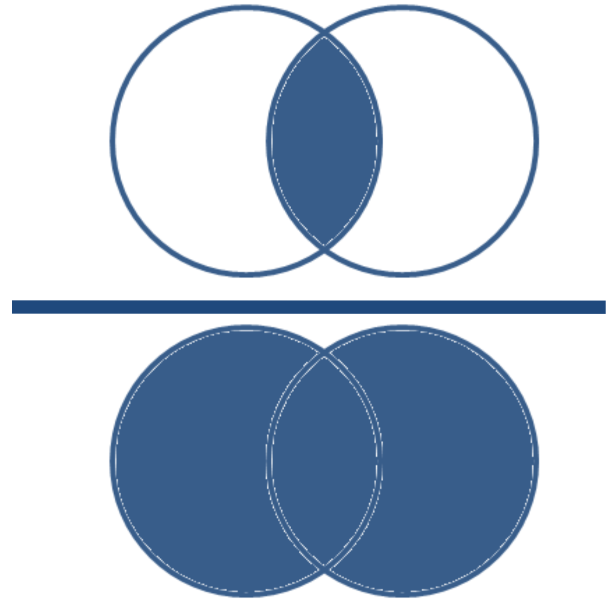


Results: Detection Power



Results: Detection Time

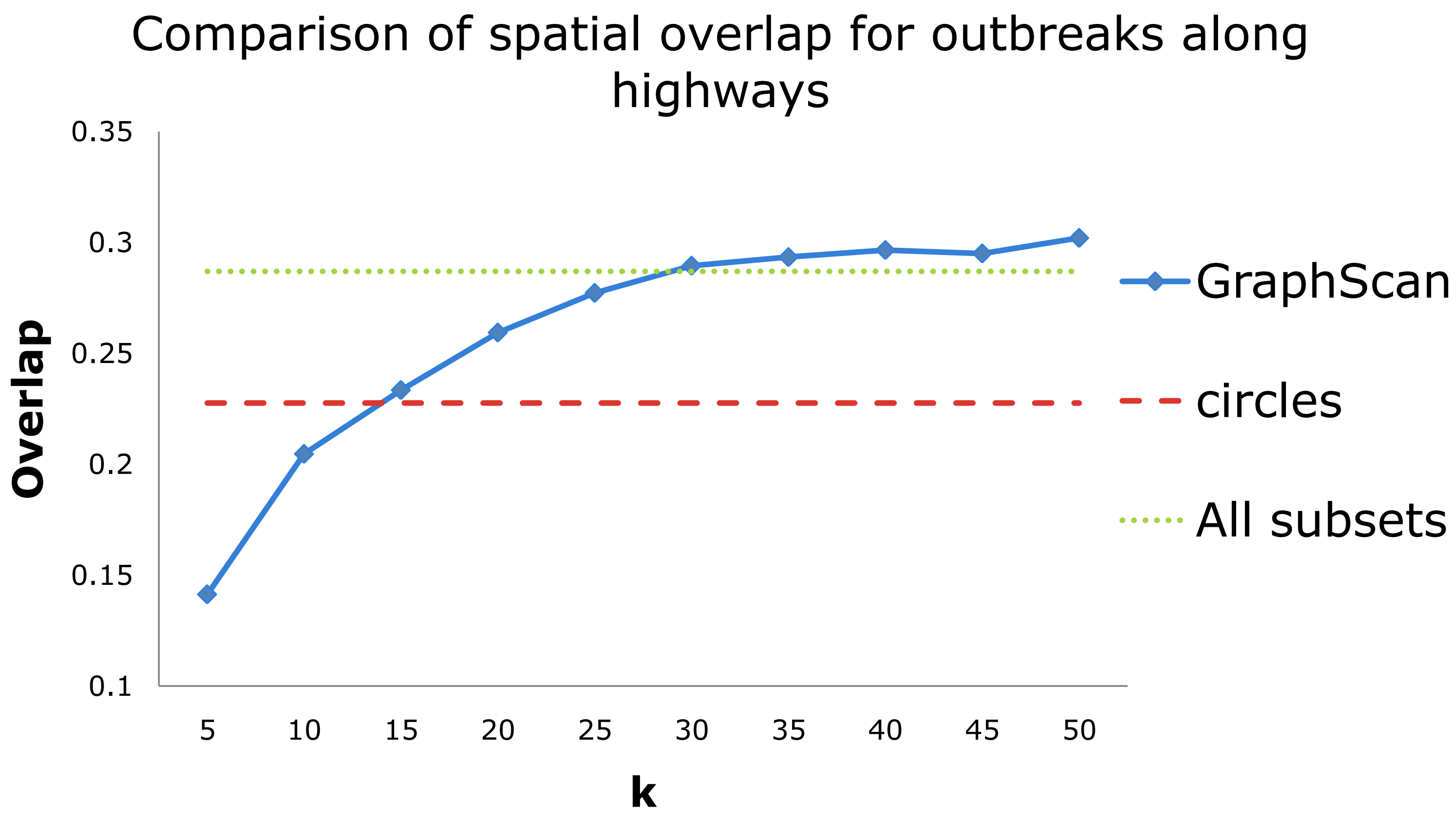
$$Overlap = \frac{A \cap B}{A \cup B} =$$



$Overlap = 1$ Perfect Match

$Overlap = 0$ Completely Disjoint

Results: Spatial Overlap



Results: Spatial Overlap

This work provides…

Theoretical framework for ruling out connected subsets that are provably suboptimal according to the LTSS property

Practical implementation of LTSS with connectivity constraints through the GraphScan Algorithm

GraphScan has shown…

Extremely large speed improvements over FlexScan, while still guaranteeing to identify the highest scoring connected subset

Using connected subsets can increase detection power for irregularly shaped disease clusters

Conclusions

Speeding up GraphScan

Exponentially many multiple paths between nodes represent a significant bottleneck

Better handling of this will allow us to scale to even larger graphs

Non-spatial Applications

Cell phone and SMS network:

Early results have shown we can detect the most active connected group of 'texters' in a graph of ~300 users in 16 minutes

Dynamic Graphs

Allowing edges (or nodes) to enter and leave the graph over time

Apply GraphScan to more complex settings such as supply and transportation networks

Current & Future Work

Thank you

Questions and Comments