Taming Near Repeat Calculation for Crime Analysis via Cohesive Subgraph Computing

Zhaoming Yin

Open Data Processing Platform Team,

Alibaba Cloud

Hangzhou, Zhejiang, China
zhaoming.yz@alibaba-inc.com

Xuan Shi
Department of Geosciences
University of Arkansas
Fayetteville, Washington County
xuanshi@uark.edu

Abstract—Near repeat (NR) is a well known phenomenon in crime analysis assuming that crime events exhibit correlations within a given time and space frame. Traditional NR calculation generates 2 event pairs if 2 events happened within a given space and time limit. When the number of events is large, however, NR calculation is time consuming and how these pairs are organized are not yet explored. In this paper, we designed a new approach to calculate clusters of NR events efficiently. To begin with, R-tree is utilized to index crime events, a single event is represented by a vertex whereas edges are constructed by range querying the vertex in R-tree, and a graph is formed. Cohesive subgraph approaches are applied to identify the event chains. k-clique, k-truss, kcore plus DBSCAN algorithms are implemented in sequence with respect to their varied range of ability to find cohesive subgraphs. Real world crime data in Chicago, New York and Washington DC are utilized to conduct experiments. The experiment confirmed that near repeat is a solid effect in real big crime data by conducting Mapreduce empowered knox tests. The performance of 4 different algorithms are validated, while the quality of the algorithms are gauged by the distribution of number of cohesive subgraphs and their clustering coefficients. The proposed framework is the first to process the real crime data of million record scale, and is the first to detect NR events with size of more than 2.

Keywords-Near Repeat, Graph Analysis

I. INTRODUCTION

In criminal research, it was found that when a crime incident takes place at a given geographical location, its neighboring areas would have a higher possibility of experiencing follow-up incidents in a short period of time [16], [25], [19], [8]. When the first incident occurs to a specific time, the follow-up incident at the same location and close to the initial time is a repeat, and the incidents both near the space and time of the initiator is a near-repeat. Such repeat and near-repeat phenomena have been found from burglaries and gun violence studies, and have important implication to dispatch police force in crime mitigation activities [16], [25], [21]. To prove the near-repeat effect, the classic way is to use the Knox test method [16], [22], [19]. The general idea of Knox test is to calculate the pairwise distance (in terms of space and time) between different crime events, and place

the event pair into different bins of a table, the residual value of each specific entry of the table are calculated to indicate how random these pairs are organized into the range. The issue with this method is, its time complexity is $O(n^2)$ (n is the number of crime events), when it's dealing with big real world data, it will take days if not months to finish the computing task.

When near-repeat research only considers the space-time interaction among every two incident, a complete spacetime event chain is more appropriate to differentiate such a scenario of separate space-time pairs [21]. For example, when three shooting events (A and B), (B and C), (A and C) comply with the near-repeat definition, a three event chain can be identified. In this case, it would be more meaningful to identify the correlation between multiple incidents rather than just two. Event chain analysis improves our understanding to the role of space and time among a series of shooting events, or other types of crime events. The significant existence of paired shooting events does not mean the significant existence of multiple shooting event chains in the same space-time context. Otherwise, all initiators or follow-up shooting events should be close to each other and form a spatial cluster in a city.

Enumerating event chains in a brute-force way would be extremely difficult because the time complexity grows exponentially. Nevertheless, we can abstract the problems of near repeat event chain detection by dividing it into two separate issues: 1) detecting near repeat pairs efficiently; 2) clustering or chaining near repeat pairs with high speed. To begin with, the most efficient way to avoid unnecessary pair wise computation of each crime event is to use an index to organize events, such that one only need to query its spatial-temporal adjacent events to generate event pairs; and R-tree [12] is a good choice. Once all event pairs are detected, they can be represented as a graph, and detecting chain of near repeat events can be modeled as cohesive subgraph enumeration problem [9]. Ideally, all events should have connection between all the others within a cohesive subgraph, and such a subgraph is a k-clique (k is the number of vertices in the subgraph) [15]; Nevertheless, in the real world, the graph is massive and approximation methods are more appropriate [18]. One of relaxed version requires that each edge in the subgraph should be in k-2 triangles, this is called k-truss [9]; Rather than triangle requirements, a k-core [4] only asks that each vertex in the subgraph has k degrees, and this restriction is even more relaxed. A variant of DBSCAN algorithm [5] can be applied to detect clusters of spatial-temporal data. The DBSCAN algorithm is the fastest among all the algorithms with complexity O(Vlog(V)). All the three alternatives to k-clique algorithm have polynomial complexity. In recent years, lots of advance had been made in the area of truss decomposition, regarding speed [23], variance of graph [14], and data streaming [13]. In this paper, we will discuss a framework to incorporate the methods of indexing crime events, computing event pairs and detecting event chains applying k-clique k-truss k-core and DBSCAN algorithms separately.

Organization. Section 2 formally defines the near repeat chain detection problem, gives the basic notations and describes the framework of our near repeat chain detection methods; Section 3 reports the experimental results using real world data; Conclusion is derived in the last section of this paper.

II. EVENT CHAIN CALCULATION THROUGH GRAPH ANALYTICS

A. Using graph to represent Crime Events

Suppose we use a vertex v to represent an event occurred in location (x, y) and at time t. And if two vertices v_1 and v_2 representing different events occurred within a given time and space constraint, an undirected edge (v_1, v_2) will be used to connect them. If there are V number of events, and E number of event pairs, the resulting vertices and edges form a graph G (In this article, we assume that there is only one undirected edge between any two vertices). If there exists a set of events with n number of events, each event is paired with every other n-1 events, we call this set of events an event chain. This event chain can be represented by a subgraph q in G such that each vertex in the q will have edges connecting every other vertices in g. Figure 1(a) shows an example of 8 crime events and 13 event pairs. In the figure, subgraph induced from vertices $\{1, 2, 3\}$ shows an example of 3-event chain which is a triangle, and subgraph induced from vertices $\{0, 1, 3, 4\}$ shows an example of 4event chain, which is a 4-clique.

The degree of a vertex v is defined as the number of edges connecting v. Take Figure 1(a) for example, the degree for vertex 1 is 5, and the degree for vertex 5 is 2. The definition of k-clique [6] is, each vertex in k-clique has degree of exactly k-1; Figure 1(b) shows a 4-clique subgraph. Similarly, k-core [4] is the subgraph in which each of its vertex has degree of no less than k. Figure 1(c) shows a 3-core subgraph, k-DBSCAN [5] is also a degree based cohesive subgraph, the method leverages k-degree

vertices to greedily expand clusters(we will discuss the detail later), and Figure 1(a) itself is the subgraph induced by 3-DBSCAN. As for an edge e in G, the number of triangles the edge is in is called the support of this edge. For instance, the support for edge (1,2) is 3, and the support for edge (2,7) is 1. k-truss [23] is the subgraph with each of its edge having support no less than k-2; Figure 1(d) shows a 3-truss induced from the original graph. The tightness of the connection in a cohesive subgraph is evaluated by the clustering coefficient [24]. If we use coe(g) to denote a graph's clustering coefficient, empirically we have $coe(g_{k-clique}) \geq coe(g_{k-truss}) \geq coe(g_{k-core})$ [23].

B. Near Repeat Event Chain Detection Algorithm

- 1) Algorithm description: Given a set of crime events, each event is represented by a coordinate x, y, and a time t of crime type p. We would transform the coordinate using UTM format [7]. The process of finding near repeat crime event chain can be formulated as the following two steps:
 - Create a graph based on the spatial temporal coordinates of a specific crime type; Since computing all pairs of events is expensive, we will build a R-tree [12] using 3 dimensional coordinates x, y and t. A vertex forms edges with its neighbors by specifying some query criteria in R-tree.
 - Based on the graph created at step 1, compute the cohesive subgraphs such as k-clique, k-core, k-DBSCAN, or k-truss etc. Optimization methods might be applied, for instance we can divide the graph into small graphs, if multiple connected components are detected [10].

The algorithm is described in Algorithm 1. The complexity of the algorithm could be divided into two steps. Suppose we have V events, and E event pairs. The complexity for the first graph generation process is dependent on data, and can not guarantee a worst case complexity, but its lower bound is O(V). The complexity of computing k-clique is NP-Hard [6], k-core is O(E) [4], k-truss is $O(E^{1.5})$ [23], k-DBSCAN is O(Vlog(V)) [11],

- 2) k-clique enumeration Revisited: Maximum clique problem is a widely researched area, and there are lots of papers on this topic, since the general algorithmic framework for clique enumeration algorithm is different from the other three algorithms, we will not spend too much effort on this theme.
- 3) k-core Computation Revisited: Algorithm 2 displays the skeleton of the k-core algorithm. Vertices are sorted by their degrees in ascending order and the criteria of k starts from 3. Vertices with degree less than k and their adjacent edges are removed from the graph G and the neighbor vertices of these removed vertices (we use nb(v) to denote neighbors of v) will update their degrees accordingly. Once there is no such vertex to be removed, the remaining graph will be placed in the k-core class T_k , and k will be incremented and the removing procedure will start again.

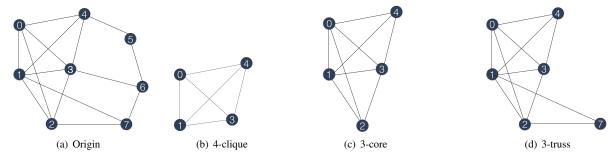


Figure 1. Example using graph to represent crime event pairs/chains.


```
Input: Set of crime events C, range criteria (r_x, r_y, r_t)
   Output: Set of near repeat event chains, T
1 Initialize R-tree R;
2 for v in C do
      R.insert(v);
4 end
5 for v in C do
       (x, y, t) = \text{coordinate of } v;
       S = \text{R.retrieve}([x \pm r_x, y \pm r_y, t \pm r_t]);
       for u in S do
8
           add edge (u, v) to G;
10
       end
11 end
12 T = \text{cohesive\_subgraph\_algo}(G);
13 return T;
```

Algorithm 2: K-CORE COMPUTATION.

```
Input: Graph G
   Output: k-core (k \ge 3) T
 1 k = 3, T_k = \emptyset;
 2 Compute deg(v) for v \in G;
 3 Sort all the vertices in ascending order by degree and
    place them in U;
 4 while \exists v \text{ in } U \text{ such that } deg(v) < k \text{ do}
       W = nb(v);
5
       for w in W do
 6
 7
           deg(w) - -;
 8
           deg(w) - -;
           Reorder w in U according to its new degree;
9
10
       Remove v and its adjacent edges from G;
11
       Remove v from U;
12
13 end
14 if Not all v in U are removed then
15
       T[k] = G;
       k++;
16
17
       goto step 4;
18 end
19 return T;
```

Algorithm 3: K-DBSCAN COMPUTATION.

```
Input: Graph G
   Output: k-DBSCAN (k \ge 3), T
1 k = 3, T_k = \emptyset;
2 compute deg(v) for v \in G;
3 Sort all the vertices in descending order by degree and
    place them in U;
4 while \exists v \text{ such that } deg(v) \geq k \text{ and }
    visited(v) == false \ do
     \operatorname{expand}(G, v, k);
6 end
   while \exists v \ such \ that \ visited(v) == false \ do
7
       Remove v and its adjacent edges from G;
       Remove v from U;
9
10 end
11 if Not all v in U are removed then
       T[k] = G;
12
13
       set all v in U as unvisited;
14
       goto step 4;
15
16 end
17 return T;
```

Algorithm 4: EXPAND PROCEDURE.

```
Input: Graph G, vertex v, k

1 W = nb(v);

2 for w in W do

3 | if visited(w) == false then

4 | visited(w) = true;

5 | if deg(w) \ge k then

6 | | expand(w);

7 | end

8 | end

9 end
```

The procedure continues until there is no vertex to be removed.

4) k-DBSCAN Computation Revisited: k-DBSCAN is actually a density based clustering algorithm. In this paper we translate the k-DBSCAN algorithm into the equivalence of the cohesive subgraph algorithm. The algorithm is as Algorithm 3 shows. Starting from k=3, the algorithm find a vertex v with $deg(v) \ge k$ then expand it (as Algorithm 4

III. EXPERIMENTS

Algorithm 5: K-TRUSS COMPUTATION.

```
Input: Graph G
   Output: k-truss (k \ge 3), T
1 k = 3, T_k = \emptyset;
2 compute \sup(e) for e \in G;
3 Sort all the edges in ascending order of their support and
    place them in U;
  while \exists e \text{ in } U \text{ such that } sup(e) \leq (k-2) \text{ do}
       e = (u, v) with the lowest support;
       W = nb(u) \cap nb(v) ;
       for w in W do
7
           sup(u, w) - -;
8
           sup(v, w) - -;
           Reorder (u, w) and (v, w) according to their new
10
11
12
       Remove e from G;
       Remove e from U;
13
14 end
15 if Not all e in U are removed then
       T[k] = G;
16
17
       k++;
18
       goto step 4;
19 end
20 return T;
```

shows), every expansion will result in the vetices in the expansion being marked as visited. If there is no vertex to be expanded, we will remove all the vertices that are not visited from G. The procedure continues until there is no vertex to be removed.

5) k-truss Decomposition Revisited: Truss decomposition is firstly introduced in paper [9] to detect possible subgroups within a social network. It's pretty useful in community detection. New and efficient algorithms are introduced to compute truss efficienty [23]. The idea of the algorithm is to compute the support for each edge first. For each edge, the O(d) complexity algorithm for triangle enumeration will be applied, d is the larger degree of the two vertices forming an edge. In this paper we will use Compressed Sparse Row (CSR) [17] to store the edge array. The skeleton of the ktruss algorithm is shown on Algorithm 5. Then the edges are sorted in the ascending order by their support. To compute the k-truss, every edge with support less than k-2, along with its incident vertices will be removed. And the incident edges will update their support and their position in the edge array following similar methods k-core computation. After all edges that do not form a k-truss are removed, the remaining graph is consisted of k-trusses. And the value of k will be incremented, followed by the same edge removing steps until there are no edges left in the graph. In the algorithm, because the range of supports are already known, sorting can be done with O(E) complexity using bucket sort.

A. Data Sets

The data used in this research contains the real crime data onto New York (NYC), Washington D.C. (DC) and Chicago (CHI) retrieved from data.gov [1], [2], [3]. The general information about the data are displayed in Table I. We have removed the data that is not conformed to the right format (for instance, data that does not fall into the range in Table I), and combined the duplicated entries (for example, crime of the same type happens at the same time of the same location, see Table I #d). In general, all three data sets have crime numbers of million scale.

Since there are so many crime types of DC and CHI data (see Table I #t), we only choose the crime types of burglary (BUR), robbery (ROB) and theft (TFT) for detailed discussion. We selected the spatial-temporal range limit $r_x = r_y = 100(meters)$ with $r_t = 10(days)$, and the feature of graphs generated applying this criteria have the property as Table II shows. In the table, we use Floyd Warshal [10] algorithm to calculate all pairs shortest path of each clique, and use this information to infer diameter of each clique. As for the clustering coefficient [24], it's used to evaluate how densely these graphs are organized. In general burglary and robbery are sparse near repeat events in comparison to theft with respect to the number of vertices #V. which is also indicated by larger number of edges and connected components #CCand#E. The diameter and clustering coefficient feature also indicate that theft has larger clusters, and these clusters are more dense. Inferred from the table, the graphs in all data obey small world property because the diameters of the graphs are small [24].

B. Knox test with Mapreduce

Firstly, we prove the existence of near-repeat effect in real big data set by conducting a Knox test on the data set. We implement the knox test using Mapreduce framework on Amazon AWS EMR and store the input/output on S3. The program is written in python and run with Hadoop streaming [26] mode. For New York and Chicago theft data set, we used a cluster of 16 nodes, and for all the other data sets, we used cluster of 4 nodes (for budget reason).

The computational time is recorded and is shown on Table III. To the best of our knowledge, the previous knox test research on crime data are orders of magnitudes smaller than our data. Since the complexity of knox test is $O(n^2)$, there is no way to compare the timing of these results against the previous experiments, hence in this paper, we claim that our method can finish the knox test within reasonable time from less than an hour to approximately 10 hours using big real world data.

To construct the knox test table [22], we have set the distance step as 100 meters and the time step as 14 days. The knox test result is shown on Figure 2 using heatmap.

Table I

GENERAL INFORMATION OF DATA. THE GRANUNARITY OF TIME IS IN DAY(S), AND THE GRANULARITY OF SPACE IS IN METERS. #T MEANS NUMBER OF CRIME TYPES, #D MEANS NUMBER OF DUPLICATED EVENTS. #EVENTS ARE THE NUMBER OF EVENTS AFTER COMBINING DUPLICATIONS.

name	earliest	latest	min x	max x	min y	max y	#events	#t	#d
NY	2006/06/04	2015/12/31	134239	1067186	121080	7220451	1123221	7	29k
DC	1978/01/01	2015/12/31	4840550	18915876	777144	8480189	2130867	43	89k
CHI	2001/01/01	2015/12/31	1092706	1205119	1813894	1951610	3102758	35	54k

Table II

GENERAL INFORMATION OF GRAPHS. #V IS THE NUMBER OF VERTICES, #E IS THE NUMBER OF EDGES, #CC IS THE NUMBER OF CONNECTED COMPONENTS(WE DO NOT COUNT THE ISOLATED VERTICES AS CC), D_AVG AND D_VAR ARE THE MEAN AND VARIANCE OF THE DIAMETERS OF CONNECTED COMPONENTS; C_AVG AND C_VAR ARE THE MEAN AND VARIANCE OF CLUSTERING COEFFICENT OF CONNECTED COMPONENTS.

	#V	#E	#CC	d_avg	d_var	c_avg	c_var		
NY									
BUR	187k	112k	24k	1.25	0.64	0.12	0.064		
ROB	198k	152k	27k	1.34	1.38	0.13	0.068		
TFT	421k	1.5m	55k	1.77	6.66	0.20	0.089		
DC									
BUR	156k	54k	13k	1.26	0.80	0.10	0.054		
ROB	54k	32k	6k	1.40	1.30	0.138	0.069		
TFT	344k	1.1m	33k	1.75	7.78	0.17	0.080		
СНІ									
BUR	197k	118k	29k	1.29	0.56	0.12	0.060		
ROB	124k	68k	14k	1.34	0.96	0.11	0.058		
TFT	650k	3.4m	89k	1.84	7.95	0.19	0.081		

It's obvious from the heatmap that all three crime types in three cities exhibit near repeat effect.

Table III COMPUTATIONAL TIME (IN SECONDS) FOR KNOX TEST USING MAPREDUCE.

City	BUR	ROB	TFT
NY	13320 (4 nodes)	13980 (4 nodes)	13560 (16 nodes)
DC	9240 (4 nodes)	1440 (4 nodes)	34320 (4 nodes)
CHI	14340 (4 nodes)	6000 (4 nodes)	24060 (16 nodes)

C. Near repeat chain detection

We implement the k-core, k-DBSCAN and k-truss algorithm using C++, and gcc compiler with the c++-11 features enabled; and the code is freely available in github with package name OPTKIT. To build the spatial-temporal index with R-tree package, we use the open source implementation of [12]. As for the k-clique, and the graph properties, we use boost graph library (BGL) [20].

1) Computational Time: The computational time is divided into 7 parts, including the time to 1) load the data, which includes parsing and reading spatial-temporal coordinates of csv format; 2) build R-tree and edges based

on querying the R-tree, 3) separate edges based on the connected component computation using BGL; 4-6) implement the k-truss, k-core and k-DBSCAN algorithms; and 7) calculate k-cliques.

Table IV

RESULTS FOR COMPUTATIONAL TIME. LOAD IS THE TIME FOR LOADING SPATIAL-TEMPORAL INFORMATION IN CSV FORMAT, R-TREE IS THE TIME FOR BUILDING R-TREE INDEX, EDGES IS THE TIME TO BUILD EDGES BASED ON R-TREE, CC IS THE TIME TO CALCULATE CONNECTED COMPONENTS, TRUSS IS THE TIME TO CALCULATE K-TRUSS, CORE IS THE TIME TO CALCULATE K-TRUSS, CORE IS THE TIME TO CALCULATE K-DBSCAN IS THE TIME TO CALCULATE K-DBSCAN, AND BGL IS THE TIME FOR K-CLIQUE CALCULATION USING BOOST GRAPH LIBRARY.

	load	R-tree	edges	CC	truss	core	dbscan	BGL
			NY					
BUR ROB TFT	0.54 0.62 1.24	0.50 0.51 1.10	0.15 0.21 1.06	0.11 0.13 0.80	6.15 7.05 10.55	1.53 1.97 4.05	4.30 4.16 4.71	2.14 2.00 302.79
			DC					
BUR ROB TFT	0.47 0.15 1.07	0.43 0.13 0.96	0.17 0.05 0.59	0.05 0.03 0.44	4.97 0.86 7.87	0.91 0.28 1.87	1.59 0.32 2.37	0.83 0.44 87.31
			CHI					
BUR ROB TFT	0.62 0.39 1.10	0.54 0.33 1.69	0.21 0.13 4.33	0.12 0.07 2.76	10.08 6.09 21.20	1.65 1.06 7.82	2.6 2.11 10.93	1.67 0.97 487.92

Table V NUMBER OF K-CLIQUE DETECTED.

	3	4	5	6	7	8	9	≥ 10			
	NY										
BUR ROB	4117 4867	935 1364	304 491	87 195	33 88	11 46	5 24	16 40			
		l	DC								
BUR ROB	1770 1073	432 314	133 118	56 45	26 20	11 12	6 8	4 8			
	CHI										
BUR ROB	4941 2338	1039 600	223 207	41 66	10 34	5 12	1 2	0 4			

Time results is excerpt on Table IV, it shows that no matter the size of the data, the dominant computational time is spent on the cohesive subgraph calculation. It is observed that when the graph is small and less dense, it takes less time to utilize BGL to compute graph properties. In case the graph

Figure 2. Knox test results. Each test is represented by a 4×10 colored matrix, the row step is 14 days, and the column step is 100 meters. The color of the entry is plotted by the residual value of that range.

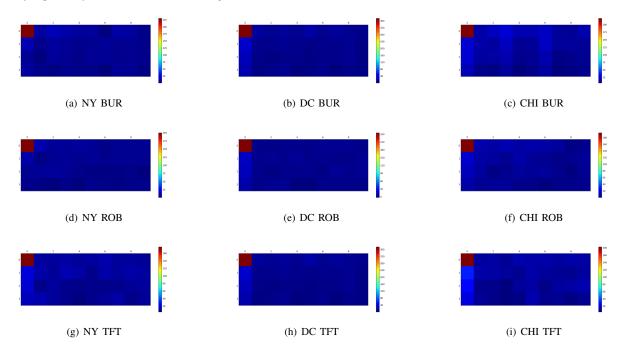


Table VI

	3	4	5	6	7	8	9	≥ 10		
			NY							
BUR ROB TFT	1336 1732 5227	1295 1944 7467	803 1144 5787	307 624 4452	163 351 3124	60 233 2272	35 177 1824	94 321 8661		
			DC							
BUR ROB TFT	641 415 2980	664 496 4090	306 285 2934	193 159 2415	114 87 1490	66 42 1153	52 33 951	34 68 6855		
СНІ										
BUR ROB TET	2092 481 11223	1729 315 12904	649 153 8549	195 105 6081	40 62 4329	33 6 3047	13 18 2234	0 0 15028		

Table VII NUMBER OF K-CORES DETECTED. NUMBER OF K-DBSCAN DETECTED.

	3	4	5	6	7	8	9	≥ 10
			NY					
BUR ROB TFT	782 1 1	509 1068 3894	211 707 3391	101 389 2744	41 243 1967	24 148 1473	47 110 1187	0 224 5193
			DC					
BUR ROB TFT	1 415 2980	362 496 4090	194 285 2934	124 159 2415	77 87 1490	45 42 1153	31 33 951	29 68 6855
			CHI					
BUR ROB TFT	6 481 57	944 315 6071	396 153 4676	112 105 3547	24 62 2630	20 6 1859	7 18 1400	0 0 8608

size is expanding fast, with larger clusters, it takes huge amount of time to get the k-clique result, Consequently, the advantage of approximate cohesive subgraph computation will be obvious. It seems our implementation is slower in the small graph case. But in general, the less cohesive requirement of the results, the less time it takes to compute the result.

2) Results comparison: Table VI, VII, VIII, V shows the distribution of the number of the cohesive subgraphs. Figure 3 shows the clustering coefficient of the cohesive subgraphs detected using different methods on different data. We exclude the results of k-clique because the cluster coefficient is always 1. Although there are some variations of the results, we can in general conclude that k-truss is better than k-DBSCAN which is better than k-core method. It is also worth noting that in many results, when it comes to the cohesive subgraphs of large k (approximately k > 10), subgraphs detected by k-DBSCAN and k-core algorithm have very stable clustering coefficients, which might indicate some specific and stable graph patterns detected by these algorithms when k is large.

IV. CONCLUSION

In this paper, we explore to identify efficient algorithms to derive near repeat event chains. By representing crime events

data - core_coe - dbscan_coe - truss_coe (a) NY BUR (b) DC BUR (c) CHI BUR data — core_coe — dbscan_coe — truss_coe data — core_coe — dbscan_coe — truss_coe data - core_coe - dbscan_coe - truss_coe (d) NY ROB (e) DC ROB (f) CHI ROB core_coe — dbscan_coe — truss_coe data — core_coe — dbscan_coe — truss_coe (g) NY TFT (h) DC TFT (i) CHI TFT

Figure 3. Cluster coefficient of cohesive subgraphs detected on different data ((x axis is k, and y axis is the cluster coefficient value)).

Table VIII NUMBER OF K-TRUSS DETECTED.

	3	4	5	6	7	8	9	≥ 10
			NY					
BUR ROB TFT	4988 6153 20091	1217 1885 9628	401 738 5448	120 340 3420	48 166 2217	16 87 1538	8 48 1153	69 109 4412
			DC					
BUR ROB TFT	2178 1359 10564	600 436 5025	206 168 2871	94 77 1922	43 36 1349	17 18 1037	8 12 796	6 11 4180
			CHI					
BUR ROB TFT	5851 2886 32345	1227 811 14418	253 290 7815	51 101 4912	12 40 3299	5 16 2410	1 5 1885	0 4 9683

into a graph enabled by R-tree indexing, the near repeat crime chains can be derived through cohesive subgraph analysis. 4 different cohesive subgraph analysis methods are implemented using AWS resources and compared with respect to time and quality. The proposed solution has never been applied in the prior works on crime analysis and will have broader impact in this research front in the future.

REFERENCES

- [1] "City of chicago data portal," https://data.cityofchicago. org/Public-Safety/Crimes-2001-to-present/ijzp-q8t2/data, accessed: 2016-10-22.
- [2] "District of columbia open data," http://opendata.dc.gov/datasets?q=crime&sort_by=relevance, accessed: 2016-10-22.
- [3] "Historical new york city crime data," http://www.nyc.gov/ html/nypd/html/analysis_and_planning/historical_nyc_crime_ data.shtml, accessed: 2016-10-22.
- [4] V. Batagelj and M. Zaversnik, "An o (m) algorithm for cores decomposition of networks," arXiv preprint cs/0310049, 2003.
- [5] D. Birant and A. Kut, "St-dbscan: An algorithm for clustering spatial-temporal data," *Data & Knowledge Engineering*, vol. 60, no. 1, pp. 208–221, 2007.
- [6] I. M. Bomze, M. Budinich, P. M. Pardalos, and M. Pelillo, "The maximum clique problem," in *Handbook of combinatorial optimization*. Springer, 1999, pp. 1–74.
- [7] M. F. Buchroithner and R. Pfahlbusch, "Geodetic grids in authoritative maps—new findings about the origin of the utm grid," *Cartography and Geographic Information Science*, pp. 1–15, 2016.
- [8] S. P. Chainey and B. F. A. Silva, "Examining the extent of repeat and near repeat victimisation of domestic burglaries in belo horizonte, brazil," *Crime Science*, vol. 5, no. 1, p. 1, 2016.
- [9] J. Cohen, "Trusses: Cohesive subgraphs for social network analysis," *National Security Agency Technical Report*, p. 16, 2008.
- [10] T. H. Cormen, Introduction to algorithms. MIT press, 2009.

- [11] J. Gan and Y. Tao, "Dbscan revisited," in ACM SIGMOD International Conference, 2015.
- [12] A. Guttman, *R-trees: a dynamic index structure for spatial searching.* ACM, 1984, vol. 14, no. 2.
- [13] X. Huang, H. Cheng, L. Qin, W. Tian, and J. X. Yu, "Querying k-truss community in large and dynamic graphs," in *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*. ACM, 2014, pp. 1311–1322.
- [14] X. Huang, W. Lu, and L. V. Lakshmanan, "Truss decomposition of probabilistic graphs: Semantics and algorithms."
- [15] J. Konc and D. Janezic, "An improved branch and bound algorithm for the maximum clique problem," proteins, vol. 4, no. 5, 2007.
- [16] J. H. Ratcliffe and G. F. Rengert, "Near-repeat patterns in philadelphia shootings," *Security Journal*, vol. 21, no. 1-2, pp. 58–76, 2008.
- [17] Y. Saad, "Sparskit: A basic tool kit for sparse matrix computations," 1990.
- [18] N. Satish, N. Sundaram, M. M. A. Patwary, J. Seo, J. Park, M. A. Hassaan, S. Sengupta, Z. Yin, and P. Dubey, "Navigating the maze of graph analytics frameworks using massive graph datasets," in *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*. ACM, 2014, pp. 979–990.
- [19] M. B. Short, M. R. Dorsogna, P. Brantingham, and G. E. Tita, "Measuring and modeling repeat and near-repeat burglary effects," *Journal of Quantitative Criminology*, vol. 25, no. 3, pp. 325–339, 2009.
- [20] J. G. Siek, L.-Q. Lee, and A. Lumsdaine, Boost Graph Library: User Guide and Reference Manual, The. Pearson Education, 2001.
- [21] M. Townsley, "Near repeat burglary chains: describing the physical and network properties of a network of close burglary pairs," in *Crime Hot Spots: behavioral, computation,* and mathematical models symposium, vol. 1, no. 31, 2007, p. 2007.
- [22] M. Townsley, R. Homel, and J. Chaseling, "Infectious burglaries. a test of the near repeat hypothesis," *British Journal of Criminology*, vol. 43, no. 3, pp. 615–633, 2003.
- [23] J. Wang and J. Cheng, "Truss decomposition in massive networks," *Proceedings of the VLDB Endowment*, vol. 5, no. 9, pp. 812–823, 2012.
- [24] D. J. Watts and S. H. Strogatz, "Collective dynamics of 'small-world' networks," *nature*, vol. 393, no. 6684, pp. 440– 442, 1998.
- [25] W. Wells, L. Wu, and X. Ye, "Patterns of near-repeat gun assaults in houston," *Journal of Research in Crime and Delinquency*, p. 0022427810397946, 2011.
- [26] T. White, "Hadoop: The definitive guide," *Oreilly Media Inc Gravenstein Highway North*, vol. 215, no. 11, pp. 1 4, 2010.