Uncovering Criminal Networks From Crime Locations

Sarah White\textsuperscript{a}, Tobin Yehle\textsuperscript{b}, Hugo Serrano\textsuperscript{c}, Marcos Oliveira\textsuperscript{d} and Ronaldo Menezes\textsuperscript{e}

\textsuperscript{a}Colleges of Arts and Sciences, University of North Carolina at Chapel Hill; sarahw14@live.unc.edu
\textsuperscript{b}School of Computing, University of Utah; tobin.yehle@utah.edu
\textsuperscript{c}Computer Science Department, Florida Institute of Technology; hbarbosafilh2011@my.fit.edu
\textsuperscript{d}Computer Science Department, Florida Institute of Technology; marcosoliveirajun2013@my.fit.edu
\textsuperscript{e}Computer Science Department, Florida Institute of Technology; rmenezes@cs.fit.edu

ABSTRACT

Rising computation power allows for more sophisticated tools to identify patterns in criminal activity. New studies show criminals typically commit crimes in areas with which they are familiar, usually close to home. Using this information we proposed a new model based on networks built with links between crimes in close physical proximity. We showed the structure of the criminal world can be partially represented by this spatial network and we analyzed the centrality of nodes in the network to find patterns in the underlying structure of criminal activity.

Keywords: Network Science, Crime Mapping, Centrality, Borders, Community Detection

1. INTRODUCTION

Better tools to identify patterns in criminal activity can make law enforcement more effective and make cities safer. Heat maps have proven to be a useful tool for visualizing crime distributions, but they cannot reflect the social networks and patterns of mobility of criminals. Criminal activities rely heavily on different types of networks, such as a social network in gangs and the word of mouth flow of information regarding any type of illegal market. The presence of these relations suggests networks science can be a powerful tool for analyzing criminal activity. The existence of commercial tools for analyzing the structure of social networks of criminals shows the potential for a network based approach, such as IBM’s i2 Analyst’s Notebook.

We seek to explain crime phenomena in metropolitan areas of the US through the application of network science. We aim to show that there exists an underlying structure of criminal activity that can be represented by the spatial distribution of the crimes. Prior work shows crimes that happen close to each other are carried out by people with similar areas of familiarity [9]. Using this assumption we propose a new model in which networks are built with links between crimes in close physical proximity. We used data of crime incidents from US police department records to generate complex networks of criminal activity in Baltimore, MD, Los Angeles, CA, and Miami, FL. For each city, we have generated networks for multiple spatial distances, ranging from 0.1 miles to 3.2 miles, and various crime types.

To show that these networks capture real-world phenomena, we found borders between communities of crimes. We compared these borders to similar boundaries in demographic data and found statistically significant variations. This indicates the structure of the crime networks reflects real-world phenomena that can not be accounted for by demographic differences.

Finally we analyzed the centrality of regions in the networks. Centrality can be used to show how neighborhoods affect other neighborhoods in the network as well as how crime flows through cities over time. We used the closeness, betweenness, degree, and eigenvector centralities of regions in the networks to find patterns in the underlying structure of criminal activity.
2. BACKGROUND

2.1 Criminology

There exist two theories in criminology that are of interest to us. The routine activity theory argues that criminal activity occurs at the convergence of three things: a potential offender, a lack of guardianship or supervision, and a target [2]. The second theory of interest, the social disorganization theory, contends that criminal activity is the result of the social and physical environments of the neighborhood at hand [15]. For example, elevated crime rates are expected in a neighborhood that lacks a strong social community and access to resources. These two explanatory theories interest us because they seek to model crime phenomena using spatial and geological context.

The routine activity theory implies that crime is a convergence between criminal opportunity and a potential offender, where this convergence serves as a point in time and space. The opportunistic nature of this theory means that criminal activity typically occurs in the sphere of familiarity of the criminal. Crimes are committed as opportunities arise, which occur as the potential offender is carrying out their daily activity. A study by Levine and Lee [9] supported the notion that the likelihood of a potential offender committing an offense is inversely related to the distance the criminal is from their place of residence. Thus, criminals typically commit crimes within a short distance from their home. The sphere of familiarity of a criminal is specific to the individual, but areas of high traffic, such as downtown areas, lie within the sphere of familiarity of many individuals. Non-residential land uses are typically found to have higher traffic in comparison to residential areas, consequently such areas are witness to more crime. Street networks influence human mobility and therefore the mobility of potential offenders. It has been shown that the convergence of a potential offender with a criminal opportunity is much more likely to occur and be exploited on a street that is relatively accessible and frequently traveled [2].

Street networks are not only correlated with an increase in crime incidence but additionally have a relationship with the typical journey-to-crime length of an offender [9]. Roadways and public transportation link together different areas of a criminals sphere of familiarity and facilitate travel outside of a criminals immediate neighborhood. The type of crime can affect the journey-to-crime length. For example, it has been found that violent crime trips are shorter in length than property crime trips [9]. Another neighborhood-level factor that affects a criminals trip length is land use and metropolitan organization.

Metropolitan areas are typically organized by regions of different land use through the natural development of urban centers and unnaturally through zoning laws. These different land uses include: residential uses, commercial uses, and industrial uses. It is worth noting that the presence of types of crimes differs by land use. It has been found that neighborhoods with residential housing and no commercial businesses are perceived as safe and non-residential land uses are correlated with an increase in criminal activity [6]. Non-residential land use, such as shopping centers or public parks, coincides with an increase in foreign or non-residential presence. It has been determined that the presence of such strangers negatively impacts a neighborhood’s social structure. Both of our theories of interest serve to explain this finding in terms of the spatial context of the neighborhood. By the social disorganization theory, a break down in the social structure of a neighborhood causes the elevated crimes rates. On the other hand, the routine activity theory implies that the higher traffic associated with non-residential land use is the cause of such crime rates.

Crime mapping is a tool of crime analysis typically used by law enforcement. It provides context for spatial data analysis and has enabled the investigation of spatial behavior of crimes. Crime mapping has been utilized since the mid 1800’s although before recent developments in technology, crime mapping was a laborious and lengthy task. Desktop mapping is now commonplace and is comparatively very fast [8].

2.2 Network Science

Network science is the field of using complex networks as an avenue for understanding natural phenomena. The study of networks historically existed as a branch of mathematics known as graph theory. Graphs in their most basic form simply represent connections among objects. As the study of graphs evolved, the application of graphs to real phenomena evolved from addressing subjective relationships to modeling real occurrences using real data.
The recent past has seen dramatic changes in the field of network science due to developments in computing power and the gradual breakdown of boundaries between disciplines [1]. Advances in computing have lead to the computerization of data storage and collection. Computers now have the ability to house large database containing data of real network structures and to analyze networks with millions of nodes. These databases span many fields of study ranging from biological processes, to long-distance telephone calls, to crime records.

We choose to investigate criminal activity in the context of network science because network science enables the investigation of relationships between data elements rather than simply examining the individual pieces of data. We are using networks as a framework for uncovering what raw crime data cannot reveal about the underlying complex structure.

2.2.1 Network Properties

Before discussing network science in depth, it is important to define a few complex network properties. Unless otherwise noted, the following definitions are referenced from Boccaletti et al. [3].

The size of a graph $G$ is the number of nodes $N$ in the network. The shortest path length between two nodes is the minimum number of edges one must traverse to travel from one node to the other. If the edges have weights then it is the path with the lowest total weight between two vertices. The diameter is the maximum shortest path length. The average path length of $G$ is the average of all shortest paths from each node to all other nodes.

The degree of a node is the number of edges connected to it. If the edges have weights this attribute is known as strength, and is the sum of the weights of all edges attached to the node of interest. The degree distribution of a graph is a distribution function $P(k)$ that describes the spread in the number of edges connected with each node. The degree distribution serves as the most basic topological description of a graph.

A graph $G$ displays clustering if it upholds the transitive property. Transitivity implies the presence of triangles, meaning if node $a$ is connected to node $b$ and node $a$ is connected to node $c$ then it is highly likely that node $b$ is connected to node $c$. One method of quantifying the clustering in a graph $G$ is to measure the proportion of of triads in $G$ that form triangles. This is known as the transitivity measure $T$. Another measure of clustering is the clustering coefficient $C$. Given a graph $G$, let $G_i$ be a subgraph of $G$ containing node $i$ and node $i$’s respective $k_i$ neighbors. The clustering coefficient of a node $C_i$ is defined as

$$C_i := \frac{2e_i}{k_i(k_i-1)}$$

where $e_i$ is the number of edges in $G_i$ and $k_i$ is the number of neighbors of node $i$. The denominator is the maximum possible number of edges in $G_i$. The clustering coefficient of graph $G$ is the average of $C_i$ for all nodes $i$ in $G$.

Connectedness describes the way in which nodes of a graph are joined. A graph $G$ is completely connected if all nodes are connected to every other node in $G$. A fully connected graph is one in which it is always possible to make a path between two nodes. A disconnected graph is one for which this condition is not true. A The giant component of a graph $G$ is the largest connected subgraph of $G$.

The assortativity of a network is a measure of node connection correlation in which similar nodes are connected. The degree assortativity measures the probability that nodes of similar degrees are connected. Graphs display degree dissortativity if highly connected nodes tend to be connected to poorly connected nodes [11].

2.2.2 Centrality Measures

Node centrality quantifies the importance of a node within a network. Centrality was originally introduced as measure of the importance of an individual within a social network and many centrality measures reflect a sociological context [3].

The closeness centrality of a node is the inverse of the average path length from the node to all other nodes. Special algorithms must be considered for networks that are not fully connected. The closeness centrality of a node measures the ease of spreading information from the node of interest to all other nodes.
Betweenness centrality is a measure of the number of shortest paths that traverse through a node. A node with a high betweenness centrality is essential to the spread of information between different communities.

Degree centrality is the number of edges incident to a node. The degree centrality of a node represents the number of connections the node has. A node in a directed network has both an indegree and outdegree centrality.

Eigenvector centrality measures the influence of a node within a network, rather than solely the importance. The eigenvector centrality of a node is the relative global importance of the node’s connections. That is, connections to nodes of a higher global importance will contribute more to the node’s centrality than connections to nodes of less global value.

2.2.3 Community Detection

Communities in networks are groups of nodes with dense intra-community connections and sparse connections between different communities. The degree to which communities are divided from other communities in the network can be measured by the modularity of the network. The modularity of a set of communities is the percentage of edges that fall within communities minus the percentage of edges that fall in the same communities in a randomly rewired network. If the rewiring process preserves the degree distribution of the network this can be simplified as

\[
M = \sum_{s=1}^{m} \left[ \frac{l_s}{L} - \left( \frac{d_s}{2L} \right)^2 \right]
\]

where \( m \) is the number of communities, \( l_s \) is the number of edges in community \( s \), \( L \) is the total number of edges in the network, and \( d_s \) is the sum of the degree of all vertices in community \( s \). It has been shown that the modularity of a set of communities has a resolution limit [5]. This limitation will be transferred to any algorithm that directly uses modularity to find communities. There are many methods for detecting communities with high modularity.

2.2.4 Topology of Real Networks

As previously mentioned, the study of network science originated in the context of graph theory. The simplest realization of a complex network is an Erdős-Rényi (ER) random graph. A graph \( G \) of \( N \) nodes generated by the ER model is constructed by connecting every pair of nodes with probability \( p \). The resulting graph has approximately \( pN(N-1)/2 \) edges [4]. This random graph model plays an important role in network science. Random graphs are often used to explore the properties of a typical graph.

The network structure behind complex real phenomena, such as social networks, the Internet, or biological processes, are not fundamentally random, rather such networks display organized underlying principles [3]. One such structure is the small world property. A small world network is a complex network that has a relatively short average shortest path length, despite a large number of nodes [3]. A small world is precisely defined as a graph \( G \) of \( N \) nodes such that

\[ L \sim \log(N) \]

where \( L \) is the typical distance between any two nodes.

This network property was first studied by Milgram [10] as a social experiment. In order to investigate the number of intermediate acquaintances between any two people, Milgram asked randomly selected people in Nebraska to forward letters to an unknown individual in Boston. The recipient was identified by only his name, occupation, and location and letters could only be sent to personal acquaintances. It was found that the number of intermediaries needed to reach the final recipient in Boston was much smaller than expected amount, at around six.

The small world property appears in social networks, as well as technological networks, such as the Internet, and biological processes. Also notably, ER random graphs have the small world property. Watts and Strogatz
discovered small worlds occurring in real phenomena tend to display clustering, unlike ER random graphs. While ER random graphs have a clustering coefficient of $C = p$, real networks typically have clustering coefficients notably larger than a random graph of the equivalent number of nodes and edges. The power grid of the Western United States is an example of a small world with generators, transformers, and substations serving as nodes and transmission lines serving as edges. The network contains $N = 4,941$ nodes and has an average degree of $\langle k \rangle = 2.67$. Watts and Strogatz found the average path length of the network to be approximately equal to a random graph of the same size and average degree, but the clustering coefficient was notably higher. The Watts and Strogatz random graph model generates graphs that uphold both of these characteristics, a small average path length $L$ and a high clustering coefficient $C$.

The limitation of the Watts and Strogatz model is that it produces a network that is relatively homogeneous. A homogeneous graph $G$ is one such that the majority of nodes are topologically equivalent. For example, nodes of an ER random graph have a degree similar to the average degree $\langle k \rangle$. Complex networks underlying real topology tend to be characterized by inhomogeneity. A study of documented food webs found that there existed a very short path length between from one node, or species, to another, where edges represented a predator-prey relationship [3]. Despite the presence of small world properties, food web networks possess a very inhomogeneous topology. The inhomogeneity most likely arises due to the different roles species play in the ecosystem, such as herbivore, carnivore, and decomposer [1].

The degree distribution of most real networks follow a power-law tail with exponents in the range $2 < \gamma < 3$. That is, the degree distribution

$$P(k) \sim k^{-\gamma}$$

with $\gamma$ between 2 and 3 [3].

Networks that display power-law tail degree distribution are called scale-free networks [1]. Scale-free networks have a highly inhomogeneous topology, with a small number of hubs, or nodes with a very high degree, and many nodes with a very low degree. Scale-free networks are indicative of real network structure arising in nature because they exhibit positive degree assortativity and are able to represent network growth [1]. Unlike in random graphs, the probability of an edge between two nodes in most real network structures is not independent of the nodes and their respective properties. Edges in scale-free networks act preferentially in respect to degree, meaning that the likelihood of a connection between two nodes depends on the nodes’ degrees. Most networks underlying real phenomena are witness to growth. Scale-free networks, such as the Internet, are able to support a continuous addition of new nodes.

3. APPROACH

3.1 Data Set

We have used data from Spot Crime. The data is a collection of police department records in the US and spans the time frame 2007 through 2010. The 2007 data set is under representative of the true amount of crime. It is likely that not all records from all police departments are accounted for in the 2007 data. Per each year, the data set is increasingly more rich, with our 2010 data being the most complete data set. The data relies on the reporting of police departments, so we do not get a perfect representation of the crime in a city. For example Miami hosted the Super Bowl in February 2010, and the number of reported crimes went down. It is possibly there were fewer crimes due to more policing because of the event, but it is equally likely the event caused many crimes to go unreported. There is no way for us to know the difference between these scenarios.

Each crime event in our data set is characterized by a description, address, geo-tag, type, date, and time. The types of crimes are arrest, assault, vandalism, burglary, theft, robbery, shooting, and other. Using the geotag of each crime we derived the ZIP Code Tabulation Areas (ZCTA) it belonged to.

We decided to exclusively use the data from Baltimore, MD, Los Angeles, CA, and Miami, FL to construct our model. We chose these three cities because they are large metropolitans, geographically different, culturally diverse, and have different demographics.
Figure 1: This is the process of collapsing four fully connected crimes in two regions. Crime 1 and Crime 2 are in region A, and Crime 3 and Crime 4 are in region B. The final weight of the edge between region A and region B is 4.

### 3.2 Building Networks

Building networks from our data requires us to decide what each vertex represents, and how these vertices should be connected. To construct edges we need to be able to measure the distance between every pair of crimes, but this can be computationally expensive for large networks. The spacial nature of our data allows us to optimize the construction of networks. We can further reduce the complexity of the network by allowing vertices to represent all crimes in an area, such as the ZCTA. This will also allow us to compare networks built from crimes happening at different times.

The networks we analyzed have vertices connected by an edge if crimes associated with the vertices occurred within a certain distance. We chose this method of linking crimes because criminals generally have a small area where they commit crimes, so crimes committed by the same, or similar people will be linked. We considered two methods of computing distance between two crimes, pure spherical geometry and a euclidean approximation. The euclidean approximation assumes latitude and longitude coordinate points are laid out on a plane, so it does not give good results over large distance, but for our networks we found the average error was on the order of $10^{-6}$ miles. The euclidean distance between two (latitude, longitude) pairs, $(\phi_1, \lambda_1)$ and $(\phi_2, \lambda_2)$, is given by

$$d = R \sqrt{\left(\cos \left(\frac{\phi_1 + \phi_2}{2}\right) (\lambda_1 - \lambda_2)\right)^2 + (\phi_1 - \phi_2)^2}$$

where $(\phi, \lambda)$ are measured in radians, and $R$ is the radius of the earth.

The choice of what each vertex defines the resolution of the network. In a network where vertices represent ZCTA, the structure within a ZCTA is lost, but for networks covering a large area this is a benefit. The loss of resolution not only simplifies results, but also greatly reduces computational cost of analysis. To build a lower resolution network we start by building a network where each vertex represents a single crime and then we collapse the crimes and their edges into a network with fewer connections. Edges between crimes in different regions are represented by a single edge with weight equal to the number of original edges. Edges between crimes in a single region are treated as self edges and removed. Figure 1 shows this process for four crimes collapsed into two regions.

Comparing every crime to every other crime is a very costly way to build networks, so we used a divide and conquer method for constructing spacial networks. By indexing each crime with a geohash we can very quickly find all crimes in a small box. We use the regular $O(n^2)$ algorithm for constructing graphs for the data in each
Figure 2: Four pre-built box networks are stitched into a single network. Only the crimes near the edge are considered for stitching.

small box. Stitching these small boxes together allows us to skip many of the $n^2$ comparisons needed for regular graph construction, instead only comparing vertices close to the edge of the box. Figure 2 shows an example of the stitching process.

3.3 Community Detection

We investigated and implemented a community detection method proposed by Guimera and Amaral [7] that uses a simulated annealing algorithm. The objective of the algorithm is to stochastically find the community partitions of the graph that maximize the graph’s modularity. The simulated annealing optimization technique allows the community detection method to bypass local maxima and produces stochastic communities. The stochastic output of the method is essential to our border analysis.

The simulated annealing algorithm has a computational temperature $T$. The algorithm is initialized with each node in its own community and a large $T$. During each iteration, a set of local and a set of global updates are proposed and accepted with probability

$$p = \begin{cases} 1 & \text{if } M_f \geq M_i \\ \exp \left( \frac{M_f - M_i}{T} \right) & \text{if } M_f < M_i \end{cases}$$

where $M_i$ is the modularity of the graph before the proposed updates and $M_f$ is the modularity of the graph with the proposed updates. Then the temperature is cooled and the process is repeated. The iterations continue until no improvement in modularity is observed for three consecutive iterations.

A local update is a movement of a randomly selected node to a new randomly selected community. During each iteration, $fS^2$ local updates are proposed, where $S$ is the number of nodes in the graph and $f$ is a constant between 0 and 1. These proposed updates are then accepted with probability $p$. Next $fS$ global updates are proposed. These updates are chosen at random to be a merge or a split. Merges are straightforward as there is only one possible outcome to merge two randomly selected communities. Splitting is more complex, and requires a non-deterministic algorithm. A nested simulated annealing algorithm that proposes only local updates is used to split a randomly selected module. The nested algorithm is initialized with a large $T$ and during each
iteration local updates are performed then the temperature is cooled. The split community is returned when the
temperature is lowered to that of the global system.

The community detection method proposed by Guimera and Amaral had a few shortcomings. The method
was not designed for graphs that had isolated components like our graphs. To account for this shortcoming we
introduced a penalty proportional to the number of components within a single community to the modularity
objective function, that is

\[
M = \sum_{s=1}^{m} \frac{1}{c_s} \left[ \frac{l_s}{L} - \left( \frac{d_s}{2L} \right)^{2} \right]
\]

where \( c_s \) is the number of components in community \( s \). Additionally, our implementation of the algorithm is
unusually slow.

We ultimately used the label propagation community detection algorithm because it is fast and stochastic. Label
propagation finds communities by assigning community labels to vertices matching the most common label of
their neighbors \([12]\). The algorithm starts by assigning an unique label to each vertex. In each iteration the
label of each vertex is assigned to be the most common label of its neighbors in the previous iteration, breaking
ties randomly. If the edges have weights it assigns the label connected by the highest weight. It has been shown
that this algorithm runs in \( O(m) \) time, where \( m \) is the number of edges in the graph \([12]\). This algorithm does
not directly use modularity to find the communities, but the resulting set of communities has high modularity.
Label propagation is both fast and stochastic which makes it a perfect choice for our chosen method of producing
borders between communities.

4. VALIDATION

4.1 Finding Borders

In order to analyze the borders between vertices in the networks we assigned each vertex an associated area.
The two resolutions of networks had different areas associated with each vertex. The vertices in the ZCTA level
networks contain all crimes in a single ZCTA. In this case we simply used the shape of the ZCTA each vertex
represents as the physical region for that vertex. For the crime level networks we used Voronoi maps. The
region associated with each crime is the Voronoi cell surrounding that crime, that is every possible coordinate
was assigned to the closest crime.

If the vertices in a network have an associated physical area then any communities in the network are a union
of the areas of each node belonging to the community. Each node belongs to a single community, so the areas of
all the communities completely cover the area of the network.

We are interested in not only the location of the borders between communities, but also the strength of these
borders. The use of a stochastic community detection algorithm makes it possible to measure the probability
of a border occurring as well as its physical position. This method of border detection was first implemented
by Thiemann et al. \([13]\). The borders for many runs of a community detection algorithm are overlaid, and
overlapping borders are combined. The weight \( w_{ij} \) of the border between two adjacent regions \( i \) and \( j \) is defined
as

\[
w_{ij} := \frac{1}{R} \sum_{r=1}^{R} \delta(c_{ri}, c_{rj})
\]

where \( R \) is the number of runs of community detection, \( c_{ri} \) is the community of region \( i \) in the \( r \)th run of
community detection, and \( \delta(a,b) \) is 1 if \( a \) and \( b \) are different community labels, and 0 otherwise. This results
in the weight of each border being the number of times the regions it divides appear in different communities
normalized to a maximum value of 1. Figure 3 shows how three runs of community detection would be combined
to find the weighted borders between regions.
We ran the label propagation algorithm 1000 times on the ZCTA level networks, and then aggregated the results using the method described above to find the weights of the borders between every adjacent ZCTA.

4.2 Clustering Census Data

In order to compare our generated borders against socio-economic and demographic borders inherent to the metropolitans we studied, we clustered ZCTA of the respective areas. The ZCTA were clustered based on features extracted from US Census data. We used data obtained from the American Community Survey for 2007-2011. We choose this time period as it was most similar to the time span of our crime records.

The generated clusters have no geographical relevance and rather and should instead be interpreted as a heat map. Each color in Figure 4 represents a different cluster. ZCTA with the same color share similar socio-economic and demographic properties.

The features we choose to cluster on are as follows: percent of population with a high school degree, joblessness, poverty rate, median income, percent of population receiving public assistance, percent of households that have moved in the past year, percent of properties that are vacant, percent of renter-occupied households, and percent of female-headed households. We choose to cluster on these nine variables because Willits et al. [15] showed, using principle component analysis, that they accounted for over 70% of the variance between neighborhoods at the block level.

To cluster the ZCTA using the nine variables as their respective features, we used hierarchical clustering. Hierarchical clustering produces a dendrogram that allows for parsing at different levels. This enables the creation of clusters at different resolutions, which is advantageous as we scale our crime networks to different geographical levels. Our hierarchical clustering used euclidean distance between the features of each ZCTA and was generated for average, complete, and single linkage.

In the interest of comparing the clustered ZCTA to the generated crime borders, we reduced the clusters to borders. A border exists between two ZCTA if the two ZCTA are geographically adjacent and not in the same cluster.

4.3 Comparing Borders

The ability to test the similarity of two sets of borders allows us to compare borders in our crime networks to borders in the demographic data. We generated one crime network for each month in four years, so it was not feasible to compare crime networks to each other as this would have required far too many comparisons to complete in a reasonable time frame. Comparing the crime borders to the borders in the demographic data quantifies the difference between crime networks and socio-economic zones. Comparing the results for different types of crimes, distance between crimes, and points in time, will show if the networks reflect a different underlying structures.

4.3.1 Cross Correlation

To compare different sets of borders for the same region we compare the absolute cross correlation of a network representation of the borders. A set of borders can be embedded in a network by representing the physical regions as vertices, and the borders between them as edges connecting adjacent regions. The edges have weights representing the strength of the borders between the regions.

We use the absolute cross correlation to measure the similarity of two border networks [13]. The absolute cross correlation is the normalized scalar product of the weights of edges in two networks. Cross correlation has an upper bound of 1 and a lower bound of 0 for non-negative weighted networks. For two border networks \( b \) and \( b' \), the cross correlation is defined as

\[
c(b, b') := \frac{\sum_{e \in E} b(e)b'(e)}{\sqrt{\sum_{e \in E} b(e)^2} \sqrt{\sum_{e \in E} b'(e)^2}}
\]
Figure 3: Combining runs of community detection to probabilistic borders.
Figure 4: Miami, FL ZCTA clustered using complete linkage.

Figure 5: Overlapping borders produce high correlation, while borders that cross, but do not overlap produce no correlation.
Figure 6: A single iteration of the randomization algorithm. We choose \( a \) to be the upper right blue borders. We choose a random \( w \in [0, 0.53] \), in this case we chose \( w = 0.33 \). We subtract 0.33 from \( a \), and add 0.33 to \( \bar{a} \) where \( b(e) \) is the weight of edge \( e \) in \( b \), and \( E \) is the set of edges in the two border networks. Overlapping borders produce high values, while borders in similar locations, but not overlapping produce low values as shown in Figure 5. Cross correlation is only a valid measure for two networks with exactly the same set of edges.

4.3.2 Normalization

Different weight distributions of networks produce vastly different cross correlations. This does not give meaningful results by itself [13]. In order to compare the cross correlation between networks with different distributions of weight we compare the cross correlation with the target network against the cross correlation with a random null model. This comparison gives a z-score we can use to compare the cross correlations of different sets of networks.

The border network represents associations between adjacent regions, so we cannot add or remove edges from the graph to randomize it. Furthermore cross correlation requires two networks with the same set of edges. Instead we will redistribute the weight of the borders to find a random network. We tried several methods of randomizing the borders, and settled on the iterative border redrawing method used by Thiemann et al. [13]. This method is a random process which iterative redraws borders until a sufficiently random set of borders are achieved. During each iteration choose a random vertex \( a \) in the border network. This vertex represents a physical area, and its neighbors in the network are the same as its physical neighbors. Select a subset \( r \) of the neighbors of \( a \). We will remove weight from these neighbors and add it to the remaining neighbors. Choose a random weight \( w \) between 0 and the smallest weight edge in \( a \). Subtract \( w \) from the weight of each border in \( a \), and add \( w \) to the weight of each border in \( \bar{a} \). Figure 6 shows an example of a single iteration of this algorithm.

The number of iterations the randomization algorithm needs to run is dependent on the number of edges in the border network it is based on [13]. Figure 7 shows how the cross correlation with the base border network changes over the number of iterations. The cross correlation levels off after the network is sufficiently random at around 16 times the number of edges network.

Using this set of randomized borders we can compute a z-score for each comparison. The resulting value can be used as a standalone measure of the similarity between the target borders and the base borders. The z-score for a border network \( t \) is defined as

\[
z_t(t) := \frac{c(h, t) - E(c(t, b_r))}{\text{std}(c(t, b_r))}
\]
Figure 7: The cross correlation between the original border network and the random border network tends to rapidly decrease and then level off as the weights of the borders become random.

Figure 8: The distribution of cross correlation between the target borders and random borders seeded from the base borders

where $E$ is the sample mean, std is the sample standard deviation, $b$ is the base network, and $b_r$ is the set of randomizations of the base network. Figure 8 shows an example of the distribution of cross correlations between the target borders and the randomized borders $c(t, b_r)$.

4.3.3 Analysis

We were interested in comparing the locations of the borders in the demographic data to the borders in the crime data. To ensure the comparison reflects the location, not the number of borders, we needed the demographic
data and the crime data to have a similar number of communities. To get borders in the demographic data of roughly the same resolution as the borders in the crime networks we cut the dendrogram of demographic clusters at four heights resulting in a similar number of communities in both data sets. The three methods of hierarchical clustering and the four levels we cut each dendrogram at left us with 12 sets of demographic borders to compare each network to. The clusters for single linkage tended to be much more sparse than the other two methods of clustering so we dropped all but the most fine resolution of borders for single linkage. This left us with nine sets of borders for each metropolitan area we analyzed.

The crime border networks often had fewer vertices than the demographic border networks due to certain ZCTA having no data. If no crimes appear in a region it cannot be meaningfully included in any community so it is ignored. In order to compare this incomplete border network to the complete border network of the demographic data we had to remove vertices from the demographic border network. This gave us two networks with identical edge sets we could compare using cross correlation.

We wished to compare many different sets of crime borders to each set of demographic borders, so we chose to generate random borders based on the demographic borders and reuse these for the comparison to each set of crime borders. For each of the 9 sets of demographic borders there were networks for each of four years (2007-2010), five distance parameters (0.1, 0.8, 1.6, 2.4, and 3.2 miles), and four types of crimes (all types, assault, burglary, and theft), for a total of 80 networks. This left us with 720 z-scores over a range of parameters for each city we were interested in.

Figure 9 highlights some of the different correlations between crime types, cities, and distance between crimes. For each of Miami, Los Angeles and Baltimore we saw more than a single standard deviation between other crime types for at least one of the five distances we analyzed. The fact that different crime types have statistically significant differences in correlation with the socio-economic boundaries in a city shows the structure of the crime networks are driven by different underlying phenomena. From this we concluded the structure of the crime networks could be used to find interesting patterns in the structure of criminal activity in a metropolitan area.

5. ANALYSIS

5.1 Experimental Design

We hypothesized that through the analysis of our network of crimes we could determine the complex structure underlying specific crime types, areas, and time spans. We used node centrality to analyze our networks because centrality conveys real world information that is useful to law enforcement efforts. For example, an area with high betweenness centrality in a theft crime network would be a good area to increase law enforcement because it serves as a bridge between communities.

In the context of our generated crime networks, the centrality of a node represents the role of its spatial location within the metropolitan. A crime node with a high closeness centrality is in a location that is physically close to all other crime locations. A crime node with a high degree centrality is in location that has dense criminal activity. A crime node with a high betweenness centrality exists in a location that serves as a bridge between two areas of criminal activity. The location of a crime node with a high eigenvector centrality is near many crimes that are also near many crimes. In other words, the crime node is well connected to other well connected crime nodes. The centrality of ZCTA nodes has analogous context to that of crime nodes. Ultimately, node centrality serves as a measure of node importance.

Our analysis focused on the eigenvector centrality measure. Eigenvector centrality correlates with degree centrality which allows us to incorporate more information in our findings without increasing the dimensions of outputted data. Additionally, the eigenvector centrality of a node is both a local and global network measure [3].

We designed two experiments through which we aimed to answer the following two questions:

Q1 - How does the spatial distribution of crime change over time?
Figure 9: As the distance between associated crimes changes the correlation with the demographic borders also changes. The varying patterns of this change for different crime types shows the crime networks have different structure for different types of crime.
Q2 - How can we use the spatial distribution of crimes, and specifically the relative importance of different areas, to estimate the role neighboring areas?

Each experiment is explained below in detail and is followed by its respective results. Given the volume of data computed for each experiment, analysis was impossible on all data dimensions simultaneously. For each experiment, the results are first addressed on a higher level and then interesting lower level results are discussed.

5.2 Temporal Centrality

5.2.1 Experimental Setup

To answer Q1 we used centrality as a tool for analyzing networks over time. Thus we must use networks of a regional resolution. Crimes are defined as a moment in time and therefore networks built using data from different periods of time will have a unique set of nodes. Working with nodes of a higher regional resolution, such as ZCTA, enables us to investigate the centrality of nodes over time periods at different time intervals.

We recorded the centrality of the ZCTA nodes in Baltimore, Miami, and Los Angeles using four different centrality measures: betweenness, degree, eigenvector, and closeness. We considered four different crime networks: all crimes, assault, theft, and burglary. Additionally, we considered networks of five different distances between crimes: 0.1 miles, 0.8 miles, 1.6 miles, 2.4 miles, and 3.2 miles. Centralities were recorded for month long intervals over the four years of our data. We chose month long intervals as opposed to a shorter or longer interval, in order to have enough data to perform meaningful analysis, while producing a sequence of networks that were fairly similar.

5.2.2 Results

In this section we analyze the spatial distribution of crimes over time for each crime type, distance parameter, and metropolitan area.

Centrality differs over time, between metropolitan areas, and between crime types as shown in Figure 10. Additionally, there exist differences in the information conveyed by each measure of node centrality. The behavior of the mean eigenvector centrality is the inverse of that of the mean degree and betweenness centrality. When the degree and betweenness centrality rise over time, the eigenvector centrality falls. This is particularly evident in Los Angeles. The mean degree centrality describes the average density of crimes in the vicinity of a ZCTA. The mean betweenness centrality describes the number of shortest paths that path through a ZCTA. The mean eigenvector centrality describes the average global importance of a ZCTA. Thus an increase in mean degree and betweenness centrality coincided with a decrease in mean eigenvector centrality must indicate an increase in the presence of boundary area crimes, or crimes that occur between areas of dense crime. In other words, crimes are occurring in areas where they were not previously, that is criminal activity is spreading. This phenomena is observed sporadically in Baltimore and Miami and dramatically in Los Angeles.

We can deduce that theft is more widespread than assault and burglary in Los Angeles because the range of its closeness distribution is shorter as shown in Figure 11. We can also deduce that despite being widespread, theft has areas of very dense activity because its degree range is much larger than that of assault and burglary. From theft’s betweenness centrality distribution we can conclude that there exist a small number of ZCTA that connect dense areas of theft activity. Assault is widespread within the Los Angeles metropolitan area but very geographically sparse. We can infer this due to its medium length closeness centrality range distribution and extremely short degree centrality range. Burglary is less widespread than assault and theft because it has the largest closeness centrality distribution range. Compared to theft, burglary in Los Angeles is less spatially dense. The volume of ZCTA with a high eigenvector centrality for burglary indicate that spatial locations are of more influence to burglary activity than assault and theft.

Centrality distributions also differ between distance parameters. Though these differences are in a large majority due to differences in the number of edges present in the network. A larger distance means the presence of more edges because there exist a greater number of crimes in a 3.2 mile radius than a 0.1 mile radius. Differences in centrality are indicative of this change of distance, particularly the degree centrality. Despite the fact the range
Figure 10: The mean closeness, eigenvector, degree, and betweenness centrality over time for each metropolitan area for 2008-2010. The crime types by color are stacks rather than cumulative measures. Results from networks of different distance parameters are averaged. Since a network with a distance parameter of 3.2 miles contains all edges present in networks with a distance parameter less than 3.2 miles, this plot is indicative only of networks with a distance parameter of 3.2 miles.
Figure 11: The distribution of the closeness, degree, eigenvector, and betweenness centrality for assault, theft, and burglary in Los Angeles. The majority of ZCTA are of very little spatial importance because the abundance of ZCTA have a centrality close to 0. Miami and Baltimore display similar centrality distributions across these three crimes.

The differences in spread between different distance parameters can indicate the presence of an underlying criminal structure that is distance dependent, such as a localize drug market.

**Burglary in Santa Monica, CA**

Through analysis of the eigenvector centrality of the burglary networks in Los Angeles, we came to the conclusion that there must exist a strong social network or crime ring underlying burglaries in Santa Monica during mid-2009 through 2010. Figure 12 and Figure 13 show the eigenvector centrality for a burglary network in Los Angeles for two different time periods with a 3.2 mile distance parameter. Before June 2009, the eigenvector centrality is less concentrated and is centered to the south of central Los Angeles. From June 2009 to December 2010, the eigenvector centrality is extremely concentrated in Santa Monica. This indicates a dramatic change in the spatial layout of burglaries in the Los Angeles metropolitan area.

Figure 14 is a plot of the eigenvector centrality for the second time period for various distance parameters. It indicates the presence of a communication or word-of-mouth flow of information since communication can be maintained in the real world on all scales of physical distance.

While burglaries occurred in Santa Monica during the first time period, during the second time period the ZCTA in Santa Monica were the most well connected within the spatial distribution of burglaries. Furthermore, there was a decrease in the number of burglaries between the time periods. Since the eigenvector centrality is a function of node degree, the shift indicates an increase in the density of burglaries in Santa Monica beginning in June 2009 in conjunction with an decrease in burglaries and an increase in Santa Monica’s neighborhood influence. Furthermore, the concentration of eigenvector centrality and its lack of change over the year and a half long period indicates that this burglary phenomenon did not spread to surrounding areas but rather shifted around the city. Thus we can deduce that a strong social network was behind the burglaries in Santa Monica during the second time period.

Figure 12: The average eigenvector centrality for Los Angeles burglaries with a distance parameter of 3.2 miles between January 2008 and May 2009. The high eigenvector centrality in Central Los Angeles is expected.

5.3 Time-Shifted Centrality Correlation

5.3.1 Experimental Setup

To answer Q2 we used a similar strategy as the previous experiment. The eigenvector centrality of ZCTA nodes in Baltimore, Miami, and Los Angeles were recorded for networks of four different crime types: all crime types, assault, theft, and burglary. Additionally, we considered networks of five different distances between crimes (0.1, 0.8, 1.6, 2.4, and 3.2 miles). We recorded centralities for month long intervals for the year 2010. We chose to utilize only the eigenvector centrality because the eigenvector centrality can additionally be interpreted as a measure of influence a node has on its neighbors within a network.

We computed the correlation between a node’s temporal centrality for the node’s first and second neighbors. The first neighbors of a ZCTA node are the nodes that are physically adjacent to the node, while the second neighbors of a node are nodes that are physically adjacent to the node’s neighbors. This is illustrated in Figure 15. The correlation was measured using the Pearson product-moment correlation coefficient. It is a measure of the linear relationship between two data sets. The output is between -1 and 1 (inclusive) where the higher magnitude indicates a strong negative or positive correlation respectively. An output of 0 indicates no correlation.

We can conduct three different types of correlation analysis between nodes: in-time, off-time by one month, and off-time by two months. Let node $a$ and node $b$ be neighbors. An in-time correlation means that node $a$’s centrality at month $t$ was compared to node $b$’s centrality at month $t$, an off-time by one month correlation means that node $a$’s centrality at month $t$ was compared to node $b$’s centrality at month $t + 1$, and an off-time by two month correlation means that node $a$’s centrality at month $t$ was compared to node $b$’s centrality at month $t + 2$. We will refer to each off-time correlation as one month correlation and two month correlation, respectively.
Eigenvector Centrality of Los Angeles Burglary Network: June 2009 - Dec. 2010

Figure 13: The average eigenvector centrality for Los Angeles burglaries with a distance parameter of 3.2 miles between June 2009 and December 2010. The highly concentrated eigenvector centrality in Santa Monica is a drastic shift from the previous time period.

5.3.2 Results

In this section we analyze the spatial distribution of crimes, and specifically the relative importance of different areas, to estimate the role neighboring areas for each crime type, distance parameter, and metropolitan area.

By the nature of eigenvector centrality, we expected the ZCTA to have positive in-time correlations as shown in Figure 16 and Figure 17. The eigenvector centrality of a node is a function of the connectedness of the node’s neighbors. Therefore neighboring ZCTA will have similar eigenvector centralities at each point in time, resulting in a positive eigenvector centrality correlation. The abundance of ZCTA with a negative in-time correlation are ZCTA for which little to no crimes are reported.

Both the one month correlation distributions and two month correlation distributions of the area are fairly negative. This means that given a ZCTA with an increasing eigenvector centrality at $t$, its neighbors will typically have a decreasing eigenvector centrality at $t + 1$ and $t + 2$. These are differences we generally regardless of area, crime type, or linkage distance. While the differences between crime types in Figure 16 are subtle, they are important. The distributions of the burglary correlations have slight bulges in between 0 and 0.5. These bulges are also faintly present in the theft correlation distributions. This indicates that networks of these two respective property crimes have a larger abundance of positive correlations than assault and all crime networks.

In Figure 17, the absence of a general pattern between the differing link distances indicates that the different distances are indicative of underlying criminal structure that is resolution dependent. That is, small scale criminal activity (0.1 miles) differs between Baltimore, Los Angeles, and Miami.
Figure 14: The average eigenvector centrality for Los Angeles burglaries with a distance parameter of 0.1 miles, 1.6 miles, and 2.4 miles between June 2009 and December 2010. The strong eigenvector centralities are present in solely Santa Monica regardless of the distance.

Figure 15: The navy ZCTA’s first neighbors are shown in teal and second neighbors are shown in pale yellow.
Figure 16: The in-time, one month, and two month centrality correlation distributions are shown for all crimes, assault, burglary, and theft. This shows that the in-time correlations always had a positive center while the off-time correlations always have a negative center.

**Assault and Burglary in Baltimore, MD**

The correlation between a ZCTA’s centrality and its neighbor’s centrality is a single number between -1 and 1. Thus each ZCTA has multiple correlations between all of its neighbors (both first and second). The time shift correlation plots quantify the relation of a ZCTA with its neighbors as the average correlation between it and all of its neighbors. The magnitude of this average serves as a measure of a ZCTA’s influence. Note that the correlation measure of a ZCTA is not indicative of the ZCTA’s actual eigenvector centrality. For example, a ZCTA with a very high centrality correlation could have a very low eigenvector centrality.

We came to the conclusion that there exists a complex structure underlying burglaries in downtown Baltimore with a cyclic nature. This structure is only visible at low resolutions due to the density of the population in the area. Figure 18 shows the one month and two month correlation for assault and burglary in Baltimore for different distance parameters. While burglary is an organized and social crime, assault crimes for the most part are personally motivated and acted on in an individual manner. Assault is shown next to burglary for the purpose of comparison.

In Figure 18 it is apparent that most ZCTA of burglary networks have a strong negative correlation with their neighbors. This means that given a ZCTA with an increasing network influence, its neighbors will have a decreasing influence in the following month. As the distance parameter is increased, the ZCTA in downtown Baltimore and surrounding areas become less correlated. This indicates that there exist a burglary structure within downtown Baltimore (which has a dense population density) that is lost as the resolution, or distance parameter is increased. We see the strong correlation in downtown Baltimore and surrounding areas disappear as
**Figure 17:** The in-time, one month, and two month centrality correlation distributions are shown for distance parameters of 0.1 miles, 1.6 miles, and 3.2 miles for each Metropolitan area. This shows that the in-time correlations always had a positive center while the off-time correlations always have a negative center.

**Figure 18:** The one month and two month centrality correlation between ZCTA in Baltimore with various distance parameters. The correlations are shown for assault and burglary. The color of the ZCTA shows this average correlation. The blue ZCTA have a positive correlation with their neighbors, the brown ZCTA have a negative correlation with their neighbors, the white ZCTA have no correlation, and the there is no data for the gray ZCTA.
the distance increases because burglaries within the presumed structure become connected to burglaries outside of the structure. It is also likely that a separate less dense burglary structure exist to the south of downtown Baltimore. Contrarily, the assault networks seem to show no correlation patterns as the distance parameter is increased.

Burglary surprisingly has a positive two month correlation, while most areas and crime types have a negative two month correlation. This means that for a distance parameter of 0.1 miles and a ZCTA with an increasing network influence at month \( t \), its neighbor will have an increasing influence at month \( t + 2 \). Thus we can conclude that the structure underlying downtown Baltimore and surrounding areas is geographically cyclic nature. This underlying burglary structure does not target the same neighborhoods regularly, rather they let neighborhoods 'rest' in between targeting them. Again, this phenomena disappears as the distance parameter is increased. Thus it is only on a small scale that the targets are cyclic. Contrarily, the ZCTA to the south of downtown Baltimore become more positively correlated as the distance increases. Findings like these could aid law enforcement efforts by providing better criminal forecasting insight.

6. CONCLUSIONS

The reliance of criminal activities on different types of networks makes network science an obvious choice for analyzing criminal activity. Using assumptions about the spatial distribution of crimes, we proposed a new model in which networks are built with links between crimes in close physical proximity. We showed that there exists an underlying structure of criminal activity that can be represented by the spatial distribution of the crimes. The model we propose allows for the construction of criminal networks without the need for gathering extra data about individual criminals or patterns of crimes. No additional work is required to gain insights available from this new model because the data we used to construct the networks is already a part of law enforcement bookkeeping. The growth of computational resources available to law enforcement agencies allows for more complex analysis than the heatmaps that have been useful in the past.

For Baltimore, MD, Los Angeles, CA, and Miami, FL, we generated networks for multiple spatial distances, ranging from 0.1 miles to 3.2 miles, and various crime types. We showed that these networks capture real-world phenomena, by comparing borders between communities of crimes. We compared these borders to similar boundaries in demographic data and found statistically significant variations, which indicates the structure of the crime networks is reflecting real-world phenomena.

Finally we analyzed the centrality of ZIP Code Tabulation Areas (ZCTA) in the networks. We used the centrality to show how neighborhoods affect other neighborhoods in the network as well as how crime flows through cities over time. We used the closeness, betweenness, degree, and eigenvector centralities of ZCTA in the networks to find these patterns in the underlying structure of crimes in metropolitan areas.

On a broad level, future directions include implementing our network model on different scales, such as county level networks and block level networks. An additional avenue of research is analyzing data from other metropolitan areas around the world. One shortcoming we considered with our model was data records that were incon siderate with observed levels of crime. Investigating the sensitivity of the model to missing data would expand the possible applications of our model.

The ease of gathering data combined with the power of our new network based model provides a promising avenue for uncovering patterns in criminal activity.

ACKNOWLEDGMENTS

The authors acknowledge support from National Science Foundation (NSF) grant No. 1263011. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF.
References


APPENDIX A. GEONET DOCUMENTATION

A.1 Package geonet

A.2 Modules

- `analyze_networks` *(Section A.4, p. 26)*
- `box_networks` *(Section A.7, p. 27)*
- `comm_sa` *(Section A.10, p. 29)*
- `community_detection` *(Section A.13, p. 32)*
- `compare_borders` *(Section A.16, p. 39)*
- `database` *(Section A.19, p. 41)*
- `multithreading`: Contains some convenience methods for multithreading jobs. *(Section A.22, p. 42)*
- `network_creation` *(Section A.26, p. 45)*
- `plotting` *(Section A.29, p. 47)*
- `save_networks` *(Section A.32, p. 48)*
- `verify_cross_correlation` *(Section A.35, p. 53)*
- `zip_clustering` *(Section A.38, p. 54)*

A.3 Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>package</em></td>
<td>Value: None</td>
</tr>
</tbody>
</table>

A.4 Module geonet.analyze_networks

**Author:** Sarah White

A.5 Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>analyze_graphs()</code></td>
<td>Run analysis on saved networks.</td>
</tr>
<tr>
<td><code>get_dynamic_modularity(path, filename, algorithm)</code></td>
<td></td>
</tr>
<tr>
<td><code>get_dynamic_node_centrality(path, filename, node_zipcode, measure)</code></td>
<td></td>
</tr>
<tr>
<td><code>plot_edge_weight(crime_types, distance, year)</code></td>
<td>Plots four graphs on a single plot.</td>
</tr>
<tr>
<td><code>plot_four(g1, g2, g3, g4, title)</code></td>
<td></td>
</tr>
<tr>
<td><code>mult_graphs(t_list, title)</code></td>
<td>Plots the same graph indicative of four different centrality measures.</td>
</tr>
<tr>
<td><code>centrality_corr(path)</code></td>
<td></td>
</tr>
<tr>
<td><code>centrality_corr_neighbors_single(zipcode, path)</code></td>
<td>Find the relationship between node of given zipcode at t0 and node’s neighbors at t1</td>
</tr>
</tbody>
</table>
centrality_corr_neighbors_multiple(zipcode, path)
Find the relationship between node of given zipcode at t0 and node’s neighbors at t1

A.6 Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>logger</td>
<td>Value: logging.getLogger(<em>name</em>)</td>
</tr>
</tbody>
</table>

A.7 Module geonet.box_networks

Author: Tobin Yehle

A.8 Functions

get_client()

Gets a new instance of the pymongo client object.

get_bounds(city)

Finds a set of bounding coordinates using the shapes of the zip codes for a city.

:param city: The name of a city in the cities.json file
:return: The bounds as a tuple of (left, bottom, right, top)

Examples
--------
>>> left, bottom, right, top = get_bounds('miami')
distance_crime_network_by_box(distance, city, box_size=20, limits=None, crime_limit=100000.0)

Builds a network for a city by merging many smaller networks.

The module level flag, '_multiprocess' determines if the construction of the network will happen in different process or in serial on a single process.

:param distance: The maximum distance between two connected crimes.
:param city: The name of the city to find crimes in. There should be an entry in cities.json for the given name.
:param box_size: The approximate length of a side of the sub-networks in miles. This will be rounded so there is an integer number of boxes of the same size.
:param limits: Any additional limits to be passed to the crime_window function. These may include dates or crime types, eg: {'type': 'Theft'}
:param crime_limit: An approximate limit on the number of crimes in the network. Very large networks may make some systems run out of memory. If that happens this value can be lowered. This limit is used to define a chance of any crime being included in the network, thus the exact number of crimes in the final network is not known until the network is complete.
:return: A network of crimes for the given area and distance with approximately the given number of nodes.

Examples
--------
>>> g = distance_crime_network_by_box(1.6, 'miami', ...
... limits={'type': 'Assault'})

stitch_boxes(networks, distance)

Turns box networks into a list of rows.

:param networks: A map<(x, y), network>, where (x, y) is the index of the box.
:param distance: The maximum distance between linked crimes.
:return: A list of row networks. Each row is composed of all boxes with the same y index. The list of rows is ordered from bottom to top. i.e: the index of a row in the list is the same as its y index in the input map.

get_box_networks(distance, bottom, left, width, height, num_box_x, num_box_y, limits=None, keep_frac=1)

Gets a number of networks contained in boxes.

:param distance: The maximum distance between connected nodes
:param bottom: The bottom coordinate of the group of boxes
:param left: The left coordinate of the group of boxes
:param width: The total width of the group of boxes
:param height: The total height of the group of boxes
:param num_box_x: The number of boxes in a row
:param num_box_y: The number of boxes in a column
:param limits: Any restrictions on database output eg: {'type': 'Theft'}
:return: A map from the (x,y) index of a box to the network of crimes in the box.
**stitch_rows(rows, distance, row_overlap)**

Stitches a number of rows into a single network.

It is assumed the rows are ordered bottom to top in the given list. This method does log(N) parallel row merges to stitch the network.

:param rows: The list of rows to stitch
:param distance: The distance to connect edges
:param row_overlap: The maximum distance into another network needed to check for edges. This value is in degrees.
:return: A single unified network

**stitch_two_rows(base, other, distance, overlap)**

Called in parallel to stitch rows together and return the result.

**get_box_network(width, height, x, y, gl_bottom, gl_left, limits, distance, keep_frac=1)**

Constructs a single network of all crimes in a box.

:param width: The width of a box in degrees.
:param height: The height of a box in degrees.
:param x: The x index of the box to construct.
:param y: The y index of the box to construct.
:param gl_bottom: The latitude of the bottom of the first box.
:param gl_left: The longitude of the right side of the first box.
:param limits: Any additional limits on the crimes used to construct the network, eg. `{type: 'Theft'}`.
:param distance: The maximum distance between two associated crimes.
:return: A network composed of all crimes in the box and provided limits. The network has the top, bottom, left, and right attributes set to the bounding coordinates of the box.

**make_row_network(network_row, distance, row_number)**

Stitches a number of box networks into a row network.

:param network_row: A list of box networks ordered from left to right.
:param distance: The maximum distance between two linked nodes.
:param row_number: The row index. This is used to rename the logger for more comprehensible output if run in parallel.
:return: A single network representing the whole row of box networks

**stitch_networks(base, other, distance, overlap, overlap_axis, base_edge)**

Stitches two networks together.

:param base: The base network. All nodes and edges from the other network will be added to this one, and the nodes will be stitched with any additional edges that may be required.
:param other: The network to add to the base network. This method does not cause any side effects in this network.
:param distance: The maximum distance between two connected nodes.
:param overlap: The length of overlap between the two networks in which nodes could be connected.
:param overlap_axis: The axis on which the stitch is taking place. Should be one of either 'latitude' or 'longitude'.
:param base_edge: The edge.
:return:

**A.9 Variables**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>logger</td>
<td>Value: <code>logging.getLogger(_name_)</code></td>
</tr>
</tbody>
</table>

**A.10 Module geonet.comm_sa**

**Author:** Sarah White
A.11 Functions

```python
community_sa(g, mod_calc, t0=0.00025, C=0.75, f=0.5)
```

Partitions the graph using the SA community detection algorithm proposed in Guimera and Amaral’s publication.

Assumptions made when implementing the algorithm include randomly selecting the node \( n \) for which to locally modify and using a 50% to determine whether a global split or merge is proposed. The algorithm follows the detection algorithm exactly. For each \( T \), \( f \times S^2 \) local changes are made.

Parameters
----------
- \( g \): igraph.Graph
  - The graph of interest.
- \( mod\_calc \): lambda
  - Function indicating which modularity measure to use.
- \( T0 \): float
  - The initial temperature. The default is \( 2.5 \times 10^{-4} \), as proposed in Brockman’s supplemental materials.
- \( c \): float
  - The cooling factor. The default is \( c = 0.75 \), as proposed in Brockman’s supplemental materials.
- \( f \): float
  - The proportional of changes made. The default is 0, as proposed in Brockman’s paper.

Returns
-------
- igraph.VertexClustering
  - Returns a clustering of the vertex set of the graph.

References
---------
R. Guimera, L. Amaral

Examples
--------
```python
>>> mod_calc = lambda p, g: modularity(p, g)
>>> parts = community_sa(g, mod_calc, f=0.65)
>>> type(parts)
igraph.clustering.VertexClustering
```
modularity_weights(p_list, g)

Calculates the modularity of a partition of g.

Uses the constraints implied in the publication as well as constraints necessary to partition a graph that is disconnected. The given constraints include: returning a modularity of one to partitions of a single module containing a single isolated node, returning a modularity of zero to partitions of multiple isolated modules, incrementing the modularity by 1/# of modules for a module that contains a single isolated node. Otherwise, the modularity is calculated as proposed in equation 1. This modularity calculation takes into respect the weights of the edges.

Parameters
-----------
p: list
    Membership list of interest.
g: igraph.Graph
    Graph of interest.

Returns
-------
m: float
    Modularity of partition. The value will be between 0 and 1.

References
---------
R. Guimera, L. Amaral
modularity(p_list, g)

Calculates the modularity of a partition of g without accounting for edge weight.

Uses the constraints implied in the publication as well as constraints necessary to partition a graph that is disconnected. The given constraints include: returning a modularity of one to partitions of a single module containing a single isolated node, returning a modularity of zero to partitions of multiple isolated modules, incrementing the modularity by 1/# of modules for a module that contains a single isolated node. Otherwise, the modularity is calculated as proposed in equation 1.

Note: This modularity calculation is much faster than modularity_weights but is less accurate.

Parameters
----------
p: list
    Membership list of interest.
g: igraph.Graph
    Graph of interest.

Returns
-------
m: float
    Modularity of partition. The value will be between 0 and 1.

References
---------
R. Guimera, L. Amaral

A.12 Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>logger</td>
<td>Value: logging.getLogger(<em>name</em>)</td>
</tr>
<tr>
<td><em>package</em></td>
<td>Value: 'geonet'</td>
</tr>
</tbody>
</table>

A.13 Module geonet.community_detection

Author: Tobin Yehle
### A.14 Functions

<table>
<thead>
<tr>
<th>ensure_folder(file_path)</th>
</tr>
</thead>
</table>

Ensures the folder structure exists for a file.

:param file_path: The path to the file to ensure.

Examples
--------
```python
>>> import json
>>> test = {'first': 10, 'second': 'foo'}
>>> path = 'data/testing/foobar.json'
>>> ensure_folder(path)
>>> json.dump(test, open(path, 'w'))
```

<table>
<thead>
<tr>
<th>load_network(path, filename)</th>
</tr>
</thead>
</table>

Loads cached network for the filesystem.

:param path: The base path to the network. This will contain all information about the type of the network.
:param filename: The filename of the network. This is usually the dates of the data used to build the network.
:return: an igraph.Graph representation of the network.

Examples
--------
```python
>>> path = 'data/lake_wobegon/distance/1.6/crime'
>>> filename = '2010'
>>> network = load_network(path, filename)
```
add_regions(g, path, filename, region_type)

Adds regions to each node in a network.

The regions are stored as shapely.Polygon objects as the cell attribute in each vertex of the network. If no regions can be found on disk the create_and_add function is used to make new ones. These are then saved to disk for future use.

:param g: The network to add regions to.
:param path: The path to the base folder of the network of interest.
:param filename: The filename of the network of interest.
:param region_type: The type of region to add to the graph. Should be one of: 'voronoi' or 'zip'.

Examples
--------

```python
>>> path = 'data/testing'
>>> name = 'test'
>>> g = load_network(path, name)
>>> add_regions(g, path, name, 'zip')
>>> 'cell' in g.vs.attributes()
True
```

layout_position(g)

Sets the coordinates of each node in a graph using the latitude and longitude attributes.

:param g: The graph to layout

Examples
--------

```python
>>> import igraph
>>> g = igraph.Graph.Read('test.graphml')
>>> layout_position(g)
>>> 'x' and 'y' in g.vs.attributes()
True
```
get_bounds($g$)

Gets a bounding polygon from the zipcodes in a graph.

The bounds of the graph is the union of the areas of all the zipcodes in the graph.

:param $g$: The graph to find the bounds of. Each node in the graph must have a zipcode attribute.

Examples
--------
>>> import igraph
>>> g = igraph.Graph.Read('test.graphml')
>>> bounds = get_bounds(g)
>>> bounds.is_valid
True

create_voronoi_regions($g$, $bounds$)

Adds a 'cell' attribute to each node containing that node’s Voronoi region.

Uses 'x' and 'y' attributes of the nodes to calculate the positions of the Voronoi cells.

:param $g$: The graph to add cells to
:param $bounds$: The physical bounds of the graph. This is used to determine how large unbounded Voronoi cells should be.

Examples
--------
>>> import igraph
>>> g = igraph.Graph.Read('test.graphml')
>>> bounds = get_bounds(g)
>>> create_voronoi_regions(g, bounds)
>>> 'cell' in g.vs.attributes()
True
**clip_cells(g, bounds)**

Clips all the cells in a graph to a bounding polygon.

:param g: The graph containing nodes with cells to clip. Each vertex should have a 'cell' attribute representing the area of influence of the cell.

:param bounds: The bounds to clip the cells to. This should be a polygon covering all the area cells could potentially be in.

**Examples**

```
>>> import igraph
>>> g = igraph.Graph.Read('test.graphml')
>>> bounds = get_bounds(g)
>>> create_voronoi_regions(g, bounds)
>>> clip_cells(g, bounds)
```

**get_communities(g, n, path, filename, algorithm='label_propagation')**

Gets a number of igraph.VertexClustering objects.

These objects are loaded from file if possible, otherwise they are found using the given algorithm.

:param g: The graph to find communities in.
:param n: The number of communities to find.
:param path: The path to the base folder for the graph.
:param filename: The filename of the graph to use.
:param algorithm: The name of the clustering algorithm to use.

The filename and path arguments are used to find clusters stored on disk. Any new clusters are stored along with the ones already present for future use.

:return: A list of VertexClustering objects

**Examples**

```
>>> path = 'data/testing'
>>> filename = 'test1'
>>> g = load_network(path, filename)
>>> comms = get_communities(g, 10, path, filename, algorithm='random_walk')
>>> len(comms)
10
```
get_adjacency_network\((g, \text{path}, \text{filename}, \text{region\_type})\)

Gets a network representing the physical adjacency of another network.

:param \(g\): The network to use as a base. The vertices of this network must have a cell attribute. If two cells have an intersection of non-zero length then they are considered adjacent.
:param \(\text{path}\): The base path to the network. The algorithm uses this path to cache temporary results.
:param \(\text{filename}\): The filename of the network. Also used for caching.
:param \(\text{region\_type}\): The type of regions contained in the cell attribute of the base network. This is also used for caching.

:return: An igraph.Graph object. All vertex attributes of the base network and the new network should be the same. Any attributes that cannot be written to a file by igraph (except cell) may not be present.

Examples
-------
```python
>>> path = 'data/testing'
>>> region_type = 'zip'
>>> g = load_network(path, filename)
>>> add_regions(g, path, filename, region_type)
>>> adj = get_adjacency_network(g, path, filename, region_type)
>>> adj.vcount() == g.vcount()
True
```

get_voronoi_adjacency\((g)\)

Gets the adjacency network for a graph by regenerating the Voronoi cells for each node.

Some cells considered adjacent may not be touching after the cells are clipped.

:param \(g\): The graph to find the adjacency of. The graph must have the latitude and longitude attributes for each vertex.

:return: A network where adjacent cells are connected by an edge.

find_border_weight\((\text{comms\_runs}, a, b)\)

Finds the weight of the border between two nodes.

Finds the number of times the two given nodes appear in different communities. This is the weight of the border between the two nodes, given they share a border.

:param \(\text{comms\_runs}\): The list of clusters to search.
:param \(a\): The first node.
:param \(b\): The second node.

:return: The weight of the border between the given nodes.
get_border_network(path, filename, region_type, algorithm, iterations)

Finds a network representing the borders between communities.

:param path: The base path to the network of crimes.
:param filename: The filename of the network of crimes.
:param region_type: The type of regions around each vertex.
:param algorithm: The community detection algorithm to use.
:param iterations: The number of runs of the community detection algorithm.
:return: An ‘igraph.Graph‘ object where the weights of edges between two vertices represent the strength of a border between them.

Examples
--------
>>> path = 'data/testing'
>>> filename = 'test'
>>> bn = get_border_network(path, filename, 'voronoi', ...
... 'label_propagation', 30)
>>> bn.write_graphml('{}/borders/{}.graphml'.format(path, filename))

crime_to_zip_borders(crime_borders, zip_adjacency)

Produces a set of zip code level borders as close as possible to the given crime borders.

This is used to compare the crime level borders to the zip code level borders in the census data.

Vertices in ‘crime_borders‘ and ‘zip_adjacency‘ must have a zipcode attribute. Edges in ‘crime_borders‘ must have a weight attribute.

:param crime_borders: A set of crime borders to work from. :param zip_adjacency: The adjacency of the desired zip code borders. :return: A structurally similar network to given adjacency network, but with edge weights corresponding to the strength of borders between two zip codes.
saveBorders(path, filename, region_type, iterations, algorithm)

Saves a shapefile containing the borders found a network.
Finds a border network containing the information about any borders.
Uses the cell attribute of the vertices and the weight attribute of the
edges to build the shapes of the borders between communities.

:param path: The base path to the network of crimes.
:param filename: The filename of the crimes network.
:param region_type: The type of region surrounding each vertex.
:param iterations: The number of iterations of the community detection
algorithm.
:param algorithm: The community detection algorithm to use.

Examples
--------
```python
>>> import plotting
>>> path = 'data/testing'
>>> filename = 'test'
>>> iterations = 30
>>> saveBorders(path, filename, 'zip', iterations, 'random_walk')
>>> fig = plotting.getBorderFig('{}borders/{}{}'.format(path, ...
... filename, iterations))
>>> fig.savefig('test.svg')
```

A.15 Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>logger</td>
<td>Value: logging.getLogger(<strong>name</strong>)</td>
</tr>
</tbody>
</table>

A.16 Module geonet.compareBorders

Author: Tobin Yehle
A.17 Functions

**randomize_borders_conserve_weight**(*b, iterations=-1, choose_prob=0.5*)

Randomizes the borders in a border network, conserving the overall weight of all edges in the network.

This changes the given network into a random border network.

:param b: The border network to randomize.
:param iterations: The number of iterations to run the randomization process. If this is -1 (the default) then the process will run for the number of edges in the network.
:param choose_prob: The probability of choosing a border to subtract weight from.

Examples
--------

```python
>>> import igraph
>>> r = igraph.Graph.Read('borders.graphml')
>>> b = igraph.Graph.Read('borders.graphml')
>>> randomize_borders_conserve_weight(r)
>>> cc = get_absolute_cross_correlation(b, r)
```

**randomize_borders**(*b, iterations=-1*)

Randomizes the borders in a border network.

This changes the given network into a random border network.

:param b: The border network to randomize.
:param iterations: The number of iterations to run the randomization process. If this is -1 (the default) then the process will run for the number of edges in the network.

Examples
--------

```python
>>> import igraph
>>> r = igraph.Graph.Read('borders.graphml')
>>> b = igraph.Graph.Read('borders.graphml')
>>> randomize_borders(r)
>>> cc = get_absolute_cross_correlation(b, r)
```
get_absolute_cross_correlation(a, b)

Finds the absolute cross correlation between two networks with the same structure.

The two given networks must have nodes with a zipcode attribute. The zip codes of the source and targets of all edges in a must match those in b.

:return: The absolute cross correlation between a and b.

Examples
--------
>>> import igraph
>>> a = igraph.Graph.Read('crimes.graphml')
>>> b = igraph.Graph.Read('census.graphml')
>>> cc = get_absolute_cross_correlation(a, b)

get_z_score(base, random, target)

get_z_scores(area, clustering, level, crime_types, distances, node_types, filenames, algorithms, iterations_list, randomizations=10)

A.18 Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>logger</td>
<td>Value: logging.getLogger(<strong>name</strong>)</td>
</tr>
</tbody>
</table>

A.19 Module geonet.database

Author: Sarah White

A.20 Functions

str2date(s)

date2str(d)
crime_window(start_date=None, end_date=None, zipcodes=None, crime_types=None, max_size=None)

Return all crimes within the given limits.

Parameters
----------
start_date, end_date: datetime
    The output will contain only crimes within the given window.
zipcodes: list
    Output will only contain crimes in the given zipcodes.
crime_types: list
    The results will contain only crimes of the given type.
max_size: int
    Limit the total number of crimes fetched from the database.

Returns
-------
crimes : list
    A list of crimes, where each crimes is a dictionary

Examples
--------
>>> crimes = crime_window(max_size=2)
>>> len(crimes)
2

>>> from datetime import datetime
>>> crimes = crime_window(end_date=datetime(2009, 2, 15),
... start_date=datetime(2009, 2, 13))
>>> len(crimes)
21669

normalize_data(crime_list)

Normalizes the format of crimes from the database

:param crime_list: A list or Cursor containing crimes from the database
:return: A list containing the normalized data

zip_box(minlat, minlon, maxlat, maxlon)

A.21 Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>client</td>
<td>Value: MongoClient('163.118.78.22', 27017)</td>
</tr>
<tr>
<td>db</td>
<td>Value: client ['crimes']</td>
</tr>
<tr>
<td>crimes</td>
<td>Value: db.crimes</td>
</tr>
<tr>
<td>geom</td>
<td>Value: db.geometry</td>
</tr>
</tbody>
</table>

A.22 Module geonet.multithreading

Contains some convenience methods for multithreading jobs.
### A.23 Functions

#### combinations(**kwargs)

Creates a list of dictionaries with all combinations of the kwargs.

- **param** kwargs: Each argument should be a list of possible values.
- **return**: A list of dictionaries. The keys are the argument names, and the values store every permutation of the given values.

**Examples**

```python
>>> names = combinations(first_name=['Tom', 'Jim'],
... last_name=['Smith', 'Glenn'])
>>> len(names)
4
>>> {'first_name': 'Jim', 'last_name': 'Smith'} in names
True
```

#### map_kwargs(func, items, failsafe=False)

Map a function over a list of keyword arguments

- **param** func: The function to apply.
- **param** items: A list of dictionaries containing keyword arguments to the function.
- **return**: A list of the results of the function

**Examples**

```python
>>> def f(first_name, last_name):
...     return last_name+', '+first_name
... >>> names = combinations(first_name=['Tom', 'Jim'],
... last_name=['Smith', 'Glenn'])
... >>> all_names = map_kwargs(f, names)
... >>> len(all_names)
4
... >>> 'Smith, Tom' in all_names
True
```
**lock_file_handle**(handle)

Locks a file for synchronous activities.

This will block the current thread until the lock can be acquired.

:param handle: A file handle object pointing to the file to be locked.

Examples
--------

```python
def lock_file_handle(h):
    try:
        print(json.load(h))
    finally:
        h.close()
```

**unlock_file_handle**(handle)

Unlocks a file after finishing synchronous activities.

This file handle object must be the same one that was previously locked.

:param handle: The file handle to be unlocked

Examples
--------

```python
def unlock_file_handle(h):
    h.close()
```

---

### A.24 Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>logger</td>
<td>Value: <code>logging.getLogger(_name_)</code></td>
</tr>
<tr>
<td><em>package</em></td>
<td>Value: <code>'geonet'</code></td>
</tr>
</tbody>
</table>

### A.25 Class Worker

```python
def object
    geonet.multithreading.Worker
```

A pickleable object to store the function to perform.
A.25.1 Methods

```python
__init__(self, do_work, failsafe=False)
```
Make an object that know how to do the work we want done. :param do_work: A function that does the work we want done. :param failsafe: If true wraps calls to do_work in a try catch to prevent every job from dying if one of them encounters an error. Any thread that error out will return false, and log the error.
 Overrides: object.__init__

```python
__call__(self, args)
```
Do the work this worker was build for. :param args: A dictionary of keyword arguments for the work function :return: The result of the work function

Inherited from object

```python
__delattr__, __format__, __getattribute__, __hash__, __new__, __reduce__, __reduce_ex__, __repr__, __setattr__, __sizeof__, __str__, __subclasshook__
```

A.25.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inherited from object</td>
<td></td>
</tr>
<tr>
<td><strong>class</strong></td>
<td></td>
</tr>
</tbody>
</table>

A.26 Module geonet.network_creation

Author: Tobin Yehle

A.27 Functions

```python
reduce_to_zip_graph(crime_graph)
```
Builds a zip graph based on the relations in the given crime graph.

Warning: Any graph given to this method may change. Save a copy first!

:param crime_graph: The crime graph to use as a seed for the zipcode graph. This graph’s nodes must have a zipcode attribute and the edges must have a weight attribute. :return: The contracted graph. The weight attribute of the edges contains the sum of the weights from the original. Each node retains a zipcode, latitude, longitude, description, and type attribute. The description attribute is the number of contracted nodes in that vertex.

```python
distance_graph(crime_list, distance, node_type)
```

```python
distance_zip_graph(crime_list, distance)
```

```python
distance_crime_graph(crime_list, distance)
```

```python
within_distance(a, b, distance)
```
get_graph(attribute_list, is_associated, get_id=None, combination_rules=None, add_index=False)

Builds a graph from a list of crimes with edges and nodes based on the given rules.

The generated graph uses the data from `attribute_list` to generate nodes. An edge between two nodes has a weight equal to the number of associated crimes in the two nodes.

**Parameters**

 attribute_list : list of dict
    The list of crimes used to build the graph. Each crime is represented as a dictionary of attributes and their values.

 is-associated : (dict, dict -> bool or int)
    This function takes two crimes represented as dictionaries and determines if they should be associated. In addition to the attributes of the crimes, the dictionaries passed to this function also contain the index the crimes appeared in the crime list. *ie. The first crime in the list would be passed as* 
    ```
    {'type': 'Theft', 'index': 0}
    ```

 get_id : (dict -> id)
    This function should take a crime as a dictionary and return a comparable id object. Any two crimes that have the same id object will be represented in a single vertex.

 combination_rules : dict<string,(list<val> -> val)>
    This dictionary defines the rules used to collapse many vertices with the same id into a single vertex. Keys should be the same as the keys in the attribute_list, and the values should be functions taking a list of values from the crime list and returning a single value for use in the final graph.

 add_index : boolean
    Specifies if the index is needed in the association function.

**Returns**

 graph : igraph.Graph
    A graph constructed using the data in `attribute_list` and the rules defined by the other parameters.

**Notes**

The complexity of the algorithm is $O(N^2 \times k)$ where $N$ is the number of elements in the attribute_list, and $k$ is the complexity of `is_associated`.

**Examples**

```python
>>> from database import crime_window
>>> cs = crime_window(max_size=30)
>>> cs = sorted(cs, key=lambda c: c['date'])
>>> first = lambda x: x[0]
>>> seq_g = get_graph(cs,
...                   lambda a, b: abs(a['index'] - b['index']) is 1,
...                   lambda c: c['type'],
...                   {'type': first},
...                   add_index=True)
>>> type(seq_g)
<class 'igraph.Graph'>
>>> mean = lambda x: sum(x)/float(len(x))
>>> dist_g = get_graph(cs,
...                     lambda a, b: within_distance(a, b, 100),
...                     lambda c: c['type'],
...                     {'type': first},
...                     {'latitude': mean, 'longitude': mean})
>>> type(dist_g)
<class 'igraph.Graph'>
```
A.28 Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>logger</td>
<td>Value: <code>logging.getLogger(_name_)</code></td>
</tr>
<tr>
<td>client</td>
<td>Value: <code>MongoClient('163.118.78.22', 27017)</code></td>
</tr>
<tr>
<td>db</td>
<td>Value: <code>Database(MongoClient('163.118.78.22', 27017), u'crimes')</code></td>
</tr>
<tr>
<td>crimes</td>
<td>Value: <code>Collection(Database(MongoClient('163.118.78.22', 27017), ...</code></td>
</tr>
<tr>
<td>zipcodes</td>
<td>Value: <code>Collection(Database(MongoClient('163.118.78.22', 27017), ...</code></td>
</tr>
<tr>
<td><strong>package</strong></td>
<td>Value: <code>'geonet'</code></td>
</tr>
</tbody>
</table>

A.29 Module `geonet.plotting`

Author: Tobin Yehle

A.30 Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>get_region_fig(path, filename, region_type)</code></td>
<td>Gets a figure with a plot of the regions of a network.</td>
</tr>
<tr>
<td></td>
<td>:param path: The base path for the network :param filename: The filename of the network :return: The figure with the plot in it</td>
</tr>
<tr>
<td><code>get_cluster_fig(g, clustering)</code></td>
<td>Gets a figure with a plot of the regions of a network.</td>
</tr>
<tr>
<td></td>
<td>:param g: The graph with cells in it :param clustering: The VertexClustering object to plot :return: The figure with the plot in it</td>
</tr>
<tr>
<td><code>get_adjacency_fig(path, filename, region_type)</code></td>
<td>Gets a figure showing the adjacency network.</td>
</tr>
<tr>
<td><code>get_network_fig(path, filename, region_type)</code></td>
<td>Get a figure showing a network on a map.</td>
</tr>
<tr>
<td><code>get_border_fig(path, filename, region_type, algorithm, iterations)</code></td>
<td>Plots the borders in a shapefile.</td>
</tr>
<tr>
<td></td>
<td>:return: A figure with the borders</td>
</tr>
</tbody>
</table>
get_census_fig(city, linkage, lev)

Plots zip code clusters in a shapefile.

:return: A figure with the clusters colored accordingly.

get_censusBorders_fig(city, borders_path, region_type, algorithm, filename, iterations, linkage, lev)

Plots network borderse from a shapefile overlaying zip code clusters from a shapefile.

:return: A figure with the clusters colored accordingly and overlaying crime borders.

get_map(left, right, top, bottom, ax, pad=.05)

Gets a Basemap instance for the given area.

The continents and oceans are filled in, and the coastline is drawn.

:param ax: The axes to draw on. :param pad: The percentage of extra space on the edges of the map.

add_graph_to_map(g, m, c='r')

Plots a network on a map. :param g: The network to plot :param m: The map to plot on :param c: The color of the vertices of the network

shade_regions_with_data(m, ax, path, filename, region_type)

Shades the all regions in a network on a map.

This is used to show where we have data, and where we do not. :param m: The map to draw on :param ax: The axes to draw on :param path: The base path of the network :param filename: The filename of the network :param region_type: The region type to fill

A.31 Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>logger</td>
<td>Value: logging.getLogger(name)</td>
</tr>
</tbody>
</table>

A.32 Module geonet.save_networks

Author: Sarah White
A.33 Functions

**save_graph**(\(g, \text{file_path}\))

Saves a graph, creating the directory first if it does not exist.

:param \(g\): The graph to save.
:param \(\text{file_path}\)\: The path to write the graph to.

Examples
--------
>>> import igraph
>>> g = igraph.Graph.Full(7)
>>> path = 'data/testing/test.graphml'
>>> save_graph(g, path)
>>> os.path.exists(path)
True

**collapse_all_to_zip()**

Creates zip code networks out of all the crimes networks available.

Will not overwrite previously existing zip code networks.

Examples
--------
>>> collapse_all_to_zip()

**date_string**(\(d\))

Gets a string representing a date, but not time.

This is used to generate readable dates for use in file names.

:param \(d\): A datetime object
:return: A string representing the date

Examples
--------
>>> from datetime import datetime
>>> date_string(datetime(2010, 12, 14, 5, 31, 02))
'2010-12-14'
get.crime.name(crime.types)

Converting a list of crime types into a readable file name.

'None' is converted to 'all'

:param crime.types: The list of crime types. Each type is a string.
:return: A single string representing the list of types.

Examples
--------
>>> get.crime.name(None)
'all'
>>> get.crime.name(['Theft', 'Burglary'])
'theft-burglary'
save_dynamic_distance_delta_graph(initial, final, delta_name, area_name, distance, node_type, crime_types=None)

Creates graphs per each unit of delta time given a window of crime.

A crime window is created using the relevant given parameters. For each increment of delta between time initial and time final, a graph of the crime window for relevant times is saved as a .graphml file using a unique file name.

Parameters
----------
initial: datetime.datetime
    Initial time of crimes used to retrieve crime window.
final: datetime.datetime
    Final time of crimes used to retrieve crime window.
delta: datetime.timedelta
    Time difference of interest.
area_name:
    String indicating the name of the city find crimes in.
distance: float
    Maximum distance between linked crimes.
node_type: String
    What each node in the network represents. Should be one of 'zip' or 'crime'.
crime_types: list
    An optional additional parameter passed to 'crime_window'

Returns
-------
.graphml
    Multiple .graphml files with unique names indicative of the time delta at hand.

Examples
--------
>>> from datetime import datetime, timedelta
>>> initial = datetime(2010, 1, 1)
>>> final = datetime(2010, 1, 8)
>>> delta = timedelta(days=1)
>>> save_dynamic_distance_delta_graph(initial, final, delta, ...
...     'baltimore', 1.6, 'zip')
save_dynamic_distance_month_graph(years, area_name, distance, node_type, crime_types=None)

Saves a number of networks, where each network represents a month.

Each network is saved to disk at a location that matches the parameters used to construct it.

:param years: The time frame of networks to create. Should be a list of integers.
:param area_name: The name of the area to search for crimes. This should be a valid entry in cities.json
:param distance: The maximum distance between two connected crimes.
:param node_type: What each node represents. The is passed on to network_creation.distance_graph.
:param crime_types: The types of crimes to include in the network.

Examples
--------
>>> save_dynamic_distance_month_graph([2008, 2009], 'miami', 1.6, ... 'crime', ['Theft'])
save_dynamic_distance_year_graph(years, area_name, distance, node_type,
crime_types=None)

Saves a number of networks, where each network represents a year.

Each network is saved to disk at a location that matches the parameters
used to construct it. This method uses the box network method to make
network creation faster. This means each network may not contain all
the crimes available for that period of time.

:param years: The time frame of networks to create. Should be a list of
integers.
:param area_name: The name of the area to search for crimes. This
should be a valid entry in cities.json
:param distance: The maximum distance between two connected crimes.
:param node_type: What each node represents. The is passed on to
network_creation.distance_graph.
:param crime_types: The types of crimes to include in the network.

Examples
--------
>>> save_dynamic_distance_year_graph([2008, 2009], 'miami', 1.6,
... 'crime', ['Theft'])

year_files(years)

month_files(years, months=range(1,13))

week_files(start, stop)

A.34 Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>logger</td>
<td>Value: logging.getLogger(<em>name</em>)</td>
</tr>
</tbody>
</table>

A.35 Module geonet.verify_cross_correlation

Author: Tobin Yehle

A.36 Functions

correlation_progression(g, iterations, conserve_weight=False)

A.37 Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>logger</td>
<td>Value: logging.getLogger(<em>name</em>)</td>
</tr>
</tbody>
</table>
A.38 Module geonet.zip_clustering

Author: Sarah White

A.39 Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>retrieve_census(area_name, file_names=[‘aff_download/ACS_12_5YR_DP02_with_ann.csv’,’aff_downloa..., cols=[[33,237,241,321],[37,247,297,289,513],[13,185]])</td>
<td>Reads census data from files for variables of interest. Outputs a dictionary of features by zip code and a dictionary of variables.</td>
</tr>
<tr>
<td>cluster_zips(area_features, linkage, t, return_dist=False)</td>
<td>Clusters zip codes using a hierachial method with euclidean distance and the inputted feature vector.</td>
</tr>
<tr>
<td>census_cluster_graph(area_name, linkage, t)</td>
<td></td>
</tr>
<tr>
<td>census_cluster_plot(area_name, linkage, t)</td>
<td></td>
</tr>
</tbody>
</table>