Assessment of advanced GIS tools for E-911 services

CSCI 5980 Fall 2012

Cole Kelleher & Benjamin Post
Motivation
The rise of location aware devices, GPS services, and analytic processing tools in geographic information systems has given emergency responders new tools to accurately determine where to direct resources and verify that those resources are being used as efficiently as possible. There is a large potential for application of additional GIS features to serve as advanced options to assist E-911 dispatchers and responders and thereby enhancing emergency response techniques, improving response time, and efficiently managing resources.

Goal
Identify value added GIS functions and processes for E-911 response teams and assess their implementation using a GIS software package, a web based client/server architecture, and location aware devices.

Assumptions
The information from an emergency call is collected via automated/manual methods and input into system. The server/client service where client requests relevant information from server will be designated to proper parties. E.g. Robberies will automatically forward to Police, fires forward to Fire Department, car accidents forward to police and medical…etc. A formal designation of incident assignments would need to be constructed and discussed by the entities involved. Location aware devices such as GPS receivers, smart-phones, laptops, etc would need to be available and used by the entities that wish to use these advanced GIS techniques.

Potential Advanced GIS Functionality
Conducting graph searches using Dijkstra’s algorithm to determine search area extent. (http://en.wikipedia.org/wiki/Dijkstra's_algorithm) Dijkstra’s algorithm is useful for search and rescue, crime suspect searches, and other time sensitive search applications. Often times, crime reports involve a suspect that is fleeing or on the move, therefore creating a time based model on possible suspect displacement from the crime scene given possible modes of transport would provide a baseline for an initial search extent as well as search extents as time passes.

By employing Voronoi diagrams incidents may be forwarded to the closest responder(s) for optimal emergency response. A dynamic refresh of officer/squad car locations would be needed due to patrols and movement of individual on duty officers. A primarily static location model for the use of Voronoi diagrams by fire departments and EMRs would suffice as an initial approach to those entities that do not conduct patrols. However, location tracking during response time would be useful in post emergency response analysis and future response improvements.

Geocoding coordinates to return meaningful street locations is another function of GIS software that may be used to assist in emergency response by locating the address at which the emergency is happening or has happened. Geocoding allows real world positions to be derived from a string of text depicting an address. A parcel base or spatial database can be queried based on the input address text string and the location of the address can be returned and displayed. After geocoding an address from an emergency call, other processes (Voronoi diagrams, Dijkstra’s algorithm, etc.) can be executed using that location to maximize the response for the emergency.
Project Features
In this section, we will discuss the details of the GIS functions described above that could be implemented to maximize emergency response.

Voronoi Polygon processing for optimal response time
This feature would maintain a Voronoi polygon set which is based on current location of emergency responders out in the field. The application would maintain 3 sets of polygon layers, which correspond to Police, Fire, and Emergency Medical Teams. Since Fire and EMT locations typically are static until tasked to respond to an appropriate call, the Voronoi diagrams for these two layers would be static and would only require updating upon the addition or deletion of a fire station or hospital.

The Voronoi polygon set for the Police is far more interesting because Police units are typically deployed and patrolling roads within their jurisdiction. This polygon set would need to be continuously updated to guarantee querying the set for the police unit which can respond the soonest to an incoming emergency call. To maintain this set, we are faced with the problem of quickly generating the polygon set from moving objects. This would require a fast operating algorithm which constructs Voronoi polygons. One such algorithm would be Fortune’s plane sweep algorithm, which runs in $O(n \log n)$ time.

This algorithm runs a sweep line either vertically or horizontally across the space to be considered. It is assumed that the points used to build the Voronoi polygons are in a sorted structure. After a sweep line has passed a point, the trailing beach line begins a parabola from the point to create a parabola around the point. When the parabolas from three separate points intersect, this intersection forms one vertex of a polygon within the set. This sweep continues until the line reaches the end of the area to be checked.

![Diagram of Voronoi polygons](image)

1. Fortune's sweep algorithm, just prior to intersection of 3 parabolas
2. Fortune's sweep algorithm, just after the intersections of three parabolas.

3. Voronoi polygons following Fortune's sweep algorithm.
The Fortune algorithm from its Wikipedia entry
(http://en.wikipedia.org/wiki/Fortune%27s_algorithm)

let *(z) be the transformation *(z) = (z_x, z_y + d(z)), where d(z) is a parabola with minimum at z
let T be the "beach line"
let \( R_p \) be the region covered by site p.
let \( C_{pq} \) be the boundary ray between sites \( P \) and \( Q \).
let \( P_1, P_2, \ldots, P_m \) be the sites with minimal \( y \)-coordinate, ordered by \( x \)-coordinate
\[ Q \leftarrow S - P_1, P_2, \ldots, P_m \]
create initial vertical boundary rays \( C^0_{p_1.p_2}, C^0_{p_2.p_3}, \ldots, C^0_{p_{m-1}.p_m} \)
\[ T \leftarrow *(R_{p_1}), C^0_{p_1.p_2}, *(R_{p_2}), C^0_{p_2.p_3}, \ldots, *(R_{p_{m-1}}), C^0_{p_{m-1}.p_m}, *(R_{p_m}) \]
while not IsEmpty(Q) do
\[ p \leftarrow \text{DeleteMin}(Q) \]
    case \( P \) of
    \( P \) is a site in \( *(V) \):
      find the occurrence of a region *\( (R_q) \) in \( T \) containing \( P \),
      bracketed by \( C_{rq} \) on the left and \( C_{qs} \) on the right
      create new boundary rays \( C^-_{pq} \) and \( C^+_{pq} \) with bases \( P \)
      replace *\( (R_q) \) with *\( (R_q), C^-_{pq}, *(R_p), C^+_{pq}, *(R_q) \) in \( T \)
      delete from \( Q \) any intersection between \( C_{rq} \) and \( C_{qs} \)
      insert into \( Q \) any intersection between \( C_{rq} \) and \( C^-_{pq} \)
      insert into \( Q \) any intersection between \( C^+_{pq} \) and \( C_{qs} \)
    \( P \) is a Voronoi vertex in \( *(V) \):
      let \( P \) be the intersection of \( C_{qr} \) on the left and \( C_{rs} \) on the right
      let \( C_{uq} \) be the left neighbor of \( C_{qr} \) and
      let \( C_{sv} \) be the right neighbor of \( C_{rs} \) in \( T \)
      create a new boundary ray \( C^0_{qs} \) if \( q_y = s_y \),
      or create \( C^+_{qs} \) if \( P \) is right of the higher of \( q \) and \( s \),
      otherwise create \( C^-_{qs} \)
      replace \( C_{qr}, *(R_r), C_{rs} \) with newly created \( C_{qs} \) in \( T \)
      delete from \( Q \) any intersection between \( C_{uq} \) and \( C_{qr} \)
      delete from \( Q \) any intersection between \( C_{rs} \) and \( C_{sv} \)
      insert into \( Q \) any intersection between \( C_{uq} \) and \( C_{qs} \)
      insert into \( Q \) any intersection between \( C_{qs} \) and \( C_{sv} \)
record \( p \) as the summit of \( C_{qr} \) and \( C_{rs} \) and the base of \( C_{qs} \).
output the boundary segments \( C_{qr} \) and \( C_{rs} \).
endcase
endwhile
output the remaining boundary rays in \( T \).

Other sites with information about Fortune's algorithm:

  - Contains implementations in ActionScript 3, JavaScript, and C++
  - Implementation of Fortune's algorithm in C#
- [http://www.skynet.ie/~sos/mapviewer/docs/Voronoi_Diagram_Notes_1.pdf](http://www.skynet.ie/~sos/mapviewer/docs/Voronoi_Diagram_Notes_1.pdf)
  - More reading on Fortune's algorithm

**Voronoi-based Network Nearest Neighbor (VNNN)**

This approach could be explored to further refine the optimal response time for emergency calls. This concept comes from Mohammed Kolahdouzan and Cyrus Shahabi in the Department of Computer Science at the University of Southern California. Their approach uses first order Voronoi diagrams based on points of interest to partition road networks into their respective Voronoi polygons. Then they compute the distances within the polygon and the distances across other polygons and store this information with the Voronoi polygon.

![Original graph and Network Voronoi diagram](image)

*Figure 4 - Converting the original graph to a network Voronoi diagram based on locations P1, P2, P3*
Their algorithm aims to fulfill the following requirements:

1. Use road networks to obtain distances between objects.
2. Support real-time nearest neighbor queries for moving objects.
3. Scalable to support large networks.
4. Independent of object location density and distribution.
5. Able to handle database updates, inserts and deletes of information.
6. Capable of handling constraining parameters such as direction and distance.

Once the data has been precomputed and stored, nearest neighbor queries on the network can be performed as follows. The first object is located by simply identifying the Voronoi polygon which contains the querying agent. Then using the pre-computed intra and inter distances of the Voronoi polygon, we are able to determine the other nearest neighbors which can be considered.

In their approach, the authors were considering the case of restaurants and other businesses a driver might be interested in knowing about while driving down the road. One important consideration to implementing this approach would be to consider the dataset used in generating the Voronoi polygon. The requirements for rural areas could be based on cellular towers and might be far different than an urban based approach, which could be based on centroids of city blocks. Of course, Voronoi diagrams could be created for every known piece of geographic data, since incidents usually occur at known locations.

**Reverse Nearest Neighbor Queries**

Reverse nearest neighbor queries are complementary to nearest neighbor queries in that they find the set of database points which are the nearest neighbor to the query location. However, it is important to note that if location B is a nearest neighbor of location A, it cannot be implied that location B is a reverse nearest neighbor of A.

We think it would be possible to use reverse nearest neighbor queries to discriminate emergency personnel (i.e. law enforcement patrols) responses for two separate emergency incidents which are spatially co-located. If we limited the query response to one patrol vehicle per incident, we could then automatically assign pairings based on reverse nearest neighbor queries. In the following illustration, the blue squares 1 and 2 represent patrol vehicles, while the red circles A and B represent emergency incidents.
If we performed nearest neighbor queries for response to incidents A and B, then both queries would return patrol vehicle 1. But if we performed a reverse nearest neighbor query for both incidents, only one incident can claim to be closest to patrol vehicle 1. Using Euclidean distance as an example, patrol vehicle 1 would be the reverse nearest neighbor of incident A. That would then make patrol vehicle 2 the reverse nearest neighbor of incident B.

![Figure 5 - Reverse nearest neighbors using Euclidean distance](image)

However, our application would depend on using a road network to correctly calculate the reverse nearest neighbor. Figure 6 demonstrates how a road network could change the query responses. In this scenario, vehicle 2 would become the reverse nearest neighbor of incident A and vehicle 1 would become the reverse nearest neighbor of incident B.

![Figure 6 - Reverse nearest neighbor using networks](image)

**Dijkstra's Algorithm for finding all paths for a given time interval**

Dijkstra's Algorithm was primarily designed to find the shortest path between two points. However, it can be modified to allow the search to run until all paths are discovered for a given time interval. It works by specifying a start position and a given time interval for travel time. By operating on a road network represented as a weighted graph where each graph segment uses the following weight: road speeds/segment distance = time, we can determine how far a vehicle can travel in any direction given the time interval.

When Dijkstra's algorithm finishes its search, it will have a list of nodes stored in memory which will have a time value assigned. This time value represents the lowest total amount of time
required to reach each node from the start point, which is important to note given there might be multiple paths to each node depending on the direction of the search. When providing visualization of the search results, there are several options to consider, but most visualizations involve a green to yellow to red color scheme. Green represents nodes easily reached given the time frame and red represents nodes that are the furthest nodes reachable given the time constraint. This color scheme can be applied to point features representing the nodes, or using contour lines to represent time intervals. We envision that Dijkstra's algorithm could be used to determine search and cordon locations for situations such as fleeing suspects, Amber Alerts, and search and rescue for missing persons.

7. Travel time search represented with point features

8. Travel time search represented with contour lines.

```plaintext
function Dijkstra(Graph, source):
    for each vertex v in Graph:
        dist[v] := infinity ;
        previous[v] := undefined ;
    end for
    dist[source] := 0 ;
    Q := the set of all nodes in Graph ;
    while Q is not empty:
        u := vertex in Q with smallest distance in dist[] ;
        remove u from Q ;
        if dist[u] = infinity:
            break ;
        end if
        for each neighbor v of u:
            alt := dist[u] + dist_between(u, v) ;
            if alt < dist[v]:
                dist[v] := alt ;
                previous[v] := u ;
                decrease-key v in Q;
            end if
        end for
    end while
return dist;
```

Integration of GIS functions
Integrating the GIS functions described above is essential to maximizing the efficiency of emergency response. Ideally, the implementation of these functions would be as follows. The initial input would come from the dispatcher who receives the emergency call. The type of emergency, time of the incident, address, and any other pertinent information such as suspect mobility and description would be discerned by the dispatcher. If the gps location of the caller is available from their phone, those coordinates would be recorded in case that location is needed for the input or for future reference. All of this information would be entered into the system by the dispatcher.
As soon as the inputs are entered situation specific processes would be executed. Initially the geocoding of the address provided and/or the location of the emergency call will be derived to provide an accurate location of the emergency situation. Then the Fortune Algorithm will be run using Voronoi diagrams to determine the closest hospital, fire station, or on duty police officers depending on the emergency type. Those closest to the emergency would then be automatically relayed the information entered into the system by the dispatcher.

In the event of a fleeing crime suspect, Dijkstra's algorithm will be executed immediately after an accurate determination of the crime location. This process will be run according to the suspect’s assumed mode of transportation (rate of possible displacement) from the location and time of the incident relative to possible transportation routes. The output from this process is then transmitted to those officers closest to the incident using a standardized visualization which depicts the maximum distance the suspect could have traveled since the time of the incident. This would aid those officers near the incident who will be in pursuit of the suspect.

For non-crime related emergency, Dijkstra’s algorithm may be used to compute the most time efficient route to the location of the emergency for responders. When a victim may need to be hospitalized, the most time efficient route to the hospital may be calculated and transmitted to the responders. Automatically computing these routes and other possible routes immediately after the emergency call is received by the dispatcher would provide on-demand route support to increase the efficiency of emergency response.

Tracking the emergency responders using location aware devices would allow for the adjustment and recalculation of the output that is delivered to the responders. Also, an assessment of these processes, this type response system, and the responders themselves can be made with the location data. Then improvements can be made in areas that did not work well and areas that did work well can be acknowledged.
Example of basic system workflow.
**Project Implementation**
Implementing the GIS functionality described above will require real world testing, substantial infrastructure, and further research into the initial inputs, user interface, and evaluation methods. Multifaceted system structures such as this take time to implement and perfect as there are many different variables in the system. The various emergency response entities would also need to customize this type of system to meet their own needs for effective use.

**Future Development**
Further development and research into the integration and implementation of these processes is absolutely necessary. Assessing entity specific inputs, responses, deliverables, and visualizations will take a substantial amount of research. A voice recognition and parsing system could accompany the dispatcher input in the initial input stage to increase the on-demand processing time. However, voice recognition systems will need further development to be fully implemented. The automation of all of these processes is also something that will take time to develop along with the collection of the required datasets for route planning and perimeter delineation. Standards for visualization and output delivery will be required to normalize interpretations across emergency entities as well as facilitate each entities needs.

**Challenges and Reflections**
The primary challenge for this project is its inability to be tested and implemented in a controlled environment. Real world implementation is really the only way to test and evaluate this type of multifaceted system. Tailoring, automating, and standardizing the output of these processes for each emergency entity is also a hurdle that will take a large amount of time and effort to complete, all of which would most likely need to be reevaluated after the initial implementation. Infrastructure is also a challenge. Servers, software, and devices need to be available and the people using this infrastructure need to be trained and able to use it efficiently. Conducting case studies and test scenarios in the real world would have and will be important to the future of this system implementation.
Project Sources:

**Tacit Information**

**System for Mobile GPS positioning and tracking:**
Most likely an app that already exists and that can be modified.
Available across a variety of platforms. (iPhone, Android, etc.)
Private, secure monitoring system with specific subscribed users.

**Interface for displaying spatial data and positions:**
Maximize the map area within the interface.
Available across a variety of platforms. Will probably have to use javascript for compatibility.
Employ some attribute data access.

**Storage and software:**
We will likely need some server storage to record data from the participants and push data out to users.
A database system may be needed for archiving location records for future retrieval.
Use of ArcMap or another GIS for processing functions if needed.
Geoprocessing functions will need to be scripted using some time of programming language.
(python?)

**Codified Information**

**CrimeReports: https://www.crimereports.com/**
CrimeReports claims to be the largest and most comprehensive crime mapping network in the world. It is a web mapping interface that employs the Google Maps API and maps a variety of crime types across the United States, Canada, and the United Kingdom. CrimeReports is a near real-time source of official crime data from a large number of law enforcement agencies. There is also a free downloadable mobile version of CrimeReports for the iPhone, but none for the Android user market. The user can also sign up for crime alerts and view crime data in different ways. A temporal aspect has been added to the mapping interface to allow the user to view different types of crimes over time. However, analysis tools are not included in the interface to analyze the crime data and its spatial distributions. Users are also able to report crime incidences within the mapping interface with the click of a button. The intermingling of official crime data and crowd-sourced crime data does present the possibility for misuse and abuse of their service.

**Real Time Crime Center:**
The New York City Police Foundation has developed a Real Time Crime Center. This innovative center was developed to incorporate advanced technological concepts and methods to fight crime and assist the New York Police Department in the Twenty-First Century. It was the first Real Time Crime Center in the United States and is equipped with the hardware and software to not only store, retrieve, and analyze billions of records but also provide up to date real-time information for officers and detectives when crimes take place. Using satellite imagery and GPS, the center is able to map and record the NYPD’s resources throughout the city of New York. The Real Time Crime Center has also developed a 911 call location mapping service which records the location of the emergency calls within the city to analyze how personnel and resources may be used to efficiently deal with each situation. This comprehensive system has been used to solve a variety of crimes within New York City.

CrimeStat is a free stand-alone statistics package that is used to analyze crime data distributions and can be implemented into most GIS packages. The advantage of CrimeStat is that it performs a multitude of spatial analysis functions for crime data such as distribution or centrographic measures, spatial autocorrelation, hot spot detection and analysis, interpolation, space-time analysis, journey to crime analysis, and demand modeling. CrimeStat is not only a useful for law enforcement; it can be applied to traffic analysis, urban planning, sociology, and even forestry. In the future (post 2009), CrimeStat will have a GUI interface and also be consistent with .NET framework. Additional spatial analysis tools will also be added to CrimeStat for enhanced crime data analysis techniques.

Geographic Dynamics, Visualization and Modeling, M. Yuan, page 358-362

Geographic Dynamics is a highly complex concept that is multiscale and multidimensional with a variety of processes that operate at multiple spatiotemporal scales in multiple thematic domains. With so many processes acting in geographic dynamics, modeling such integrated processes becomes a very daunting task. However, geographic dynamics can be a powerful way to gain useful insights into geographic processes. Animated cartography, geovisualization and geocomputational models have all been used to visualize and display geographic phenomena and processes that happen on temporal scales. A large amount of varying algorithms, models, and relationships contribute to each attempt to express geographic dynamics however, the applications of geographic dynamics are extremely useful. Many of these applications are present in our everyday lives whether we realize it or not. Weather and climate modeling is a great example of geographic dynamics that is used around the world every day. Other applications such as land cover change, human activity (ex. Crime events), and other biological, geological, and tectonic models are key for the monitoring and interpretation of the world around us.

Other sources:
Fortune’s Algorithm - http://en.wikipedia.org/wiki/Fortune%27s_algorithm