# TJHSST Senior Research Project
# Random Map Generation and Pathfinding with Dynamic Point Costs
# 2006-2007

Olex Ponomarenko

June 11, 2007

## 1 Abstract

An abstract representation of a map was used to incorporate realistic conditions not currently in place in commercial path finding programs. As the project is focused on the back-end of random graph generation and searching, it can be useful for other projects as an educational tool about efficiency and memory usage, as well as more sophisticated frontend systems. The project incorporates more realistic aspects of traffic movement such as traffic light and stop sign delays in order to provide realistic delay calculation when traversing a map. Other factors such as whether one is turning left or right are also considered in delay calculation. Traffic light intersections are weighed in favor of larger roads, meaning smaller roads tend to have less green-light time on average than larger ones intersection with smaller roads. Exits are placed only in realistic locations, meaning a tiny driveway won.t have its own exit onto I-95.

The map is generated using a sophisticated random graph generator, which was also developed as part of this project. The graph generator works by assigning random location a higher density value, making roads stemming from that location more likely to be larger roads . interstate and local highways, so on. Residential areas are properly structured with a road architecture that resembles real life, and intersections are placed with realistic

delay mechanisms (traffic lights, stop signs, different kinds of exits, etc). The entire project is also designed to be easily testable, flexible, and scalable. All of the parts of the project, from the shell to the final heuristic were be designed and created with a large-scale problem in mind. The Java display is more of a testing and debug tool, and the vast majority of the work is done on the back-end, with little visual representation. The python code for generating the graph and searching is also meant to be a tool for future projects, perhaps on learning heuristics and similar endeavors.

# 2 Introduction

The problem we are facing with the commercial programs is that they often ignore delays caused by stop signs and traffic lights on roads. For short distances, this often results in overly-complicated paths that are slower than a longer but simpler path. This project is aimed at creating a primarily back-end program which will incorporate such realistic features and provide funtionality to randomly create and search through maps with such attributes. Both the magnitute of improvement as well as analysis of the applicability of dynamic point costs on the large scale will be included.

The graph generator and the backend of the program were created using the Python computer language. This language is very easy to use, very feature-rich, and fast enough for the scale of that I'm working with. The program started out from very basic maps, without speed limits and path costs, and advanced on to more complex ones with added features such as intersections, traffic lights, and stop signs. The solutions were checked manually in the early stages, and later using slower but guaranteed-correct searching techniques, ensuring that the heuristic evaluation worked well. In addition, a Java program was also created to display the randomly generated graph and solutions in order to avoid obvious errors and provide visual results. However, the Python code includes all of the features of the program besides display, and the display is not needed to run and interact with the important parts of the code. This results in a program that can be incorporated into bigger projects easily.

As resource use is also an important part of the project - it is a major limitation to including dynamic point costs within existing map pathfinding programs and algorithms - algorithms will be optimized for fastest results to see if this project is feasable on a large scale. Everything was looked over

multiple times to make sure no extra data is passed around between methods, and that no unnecessary lists, maps, and other structures are created. Several common optimization techniques as well as heuristic aids were used to improve processing time with additions to data size.

# 3 Background

This topic has obviously been covered in commercial programs such as google maps and mapquest. Most such programs, however, only consider the maximum speed limit when searching for the fastest possible path. Some of the other important factors of travel speed, such as delays caused by traffic lights and stop signs, are ignored. There isn't a whole lot of research out there on complex graph traversal and path finding - few consider the real world application in maps and travel and instead focus on applications in problem-solving programs and genetic algorithms.

Consider, for instance, an intersection of a fairly small community road with a larger state road. Both are sizable, and do not have any stop signs, nor are they highways with a whole lot of exits. The intersection has traffic lights, but the road sizes are different - one is clearly larger than the other, and traffic lights are favored toward the larger road, giving them a faster pass-through time than to those on the smaller road. Another level of complexity is achieved with directionals. Turning right, on average, is very quick, and turning left, especially from the smaller onto the larger road, is slower. Google Maps and Yahoo account for the distance that you travel, but do not count such delays as traffic lights.

# 4 Structure of the Program

## 4.1 Random Graph Generator

While it is important that the program considers realistic obstacles when searching for the quickest path through a map, it is just as important that the map that the program is using is believable and realistic. The random graph generator created for the program is fairly effective at achieving both of these requirements. While it does not properly account for population density and create concentric highway structure like in real life, the structure makes sense: there are no abrupt ends to larger roads, and there are plenty of

smaller roads connecting to the highways, and there are differently-weighed locations connected by larger roads. An inner city can also be simulated by increasing certain parameters.

The graph generator creates a map, comprised of the two auxillary classes together in a structure of dictionaries (also known as maps). This allows O(1) access time to both connecting locations and the roads that connect to them, allowing the heuristic to perform its function effectively (also O(1), for both the control and the sophisticated heuristic). In certain cases, only subsections of roads are stored in the map, which have references to their "parent" road. This avoids considering some curved roads as right or left turns and allows for intersections of non-perpendicular roads to be accounted for correctly.

Rigorous testing (1000 maps of 800x800 pixels) was conducted to check for errors with the program, as well as peer review of numerous such maps for realism and consistency.

## 4.2 Auxillary Classes

In terms of structure, there are two main component classes to the random graph generator: Location and Road. The Location class is used to represent both locations such as certain buildings and intersections between two or more roads. This allows for flexibility both in terms of searching and optimizing the heuristic. A user can select to go from an intersection to another intersection, rather than simply from one building to another. The heuristic can exploit this similarity between buildings and intersections and provide faster results. Aside from its connecting roads, a location also provides its weight and a single int for the number of intersections for the heuristic.

The Road class represents all kinds of roads that are out there. Whether it is a small residential road or a state highway, the Road class is used. Intersections of roads are realistically represented. Highways do not have exits onto small roads, traffic lights are favored to the larger road, and the amount of smaller roads is greater than the number of highways. Aside from the locations between which the road is situated, it also provides its size (from which the kinds of intersections one can encounter can be extracted), the number of intersections (used by the sophisticated heuristic), and refereces to subsections of the road between intersections.

## 4.3 Display

There is also a Java-based display that was developed for this project, which displays the randomly-generated graph as well as the latest search through the map. There were simple storage classes built for the Java program that corresponded to roads and locations in the python code. All data was passed using a single text file. Graphics2D and java.swing graphics were used to create the image.

The Java section of the project, however, is completely separate from the python code, and is in no way necessary to use the back-end code. This avoids the middle-man if another project is to use my code. If someone decided to apply a learning heuristic to my code, they would only need to work either within the eixisting Python code or in communication with the Python code, and wouldn't have to worry about the clunky Java display and input.

## 4.4 Searching

Two varieties of searching were done, both using the A* algorithm. There was a "control" heuristic and evaluation that did not account for point costs. In other words, it simply used distance divided by speed limit to determine the cost between locations. The more sophisticated heuristic had additional cost added to roads where a left turn had to be made, or strings of road with lots of disadvantageous intersections.

Admittedly, there is no certainty that Google and MapQuest do not use point costs. The most prevailing evidence against them having point costs comes out when getting directions from a location on a small street to a very near location (300 feet perhaps), but one that is on a different, larger road, onto which you have to take a painstakingly long left turn. After finding a number of examples, google has consistently responded with answers such as 5 seconds. In a case right next to my house, there is a wait time of at least 30-40 seconds on average, as the traffic light is heavily favored toward the other road. These kinds of anecdotal examples as well as the sheer size and scope of such a feature are the primary evidence that Google as well as other programs simply use the distance times the speed limit for the evaluation of cost, and disregard point costs.

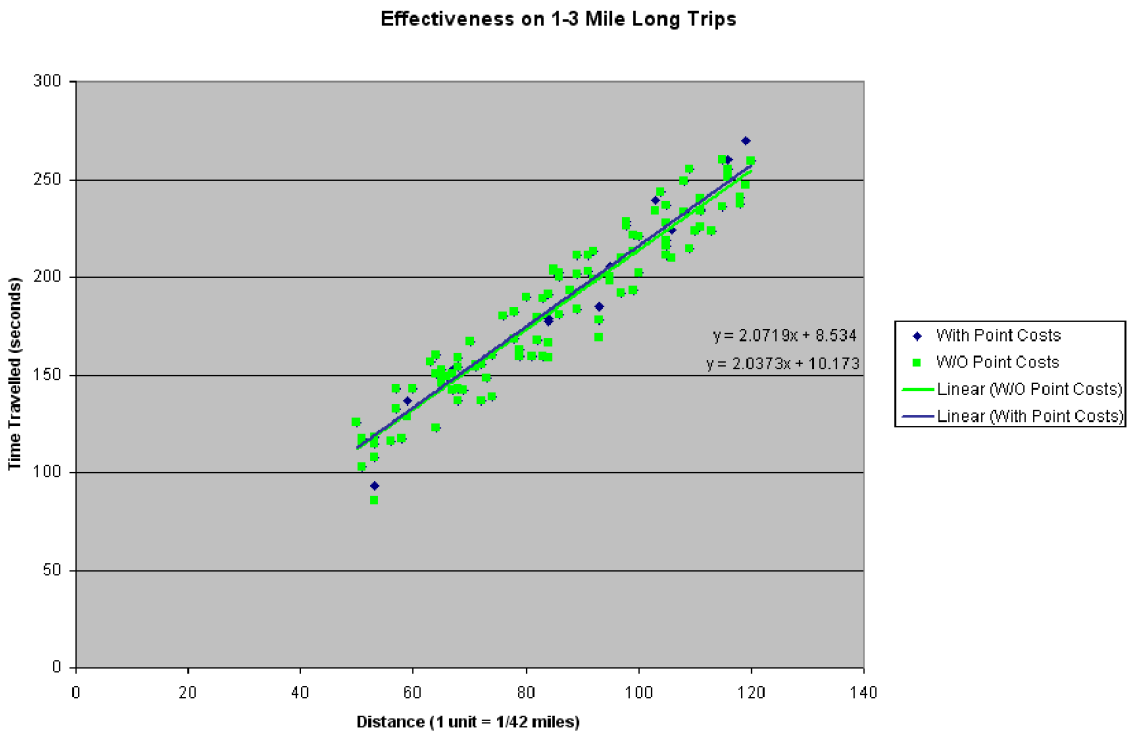


Figure 1: A portion of the data collected during the project. Note the minuteness and consistentency in travel time improvements when using the point-costs heuristic while searching for the fastest path. The equations for the trend lines are shown.

# 5 Results and Conclusion

The program proved most useful in small-scale situations simulating cities, where you might have to make a turn every few blocks. The difference between right and left turns in this case became apparent. In most test cases the speed-limit-only search took the same road as the search with point costs, and resulted in an the same path anywhere from 72% to 92% of the time. Coupled with the fact that an improved seach was sometimes along the lines of $< 5$ seconds' improvement, the effective improvement in travel time was low. Over the range of 0-16 mile range, the point-cost algorithm made improvements in just 7% of the paths, with an average improvement of 4.6% in the better paths. This results in only a .3% effective improvement over a more conventional searching algorithm. On smaller data sets, especially on searches between 1 and 3 miles long (right about where multiple path choices appear, but still fairly small scale), the program performed somewhat better. 12.4% of all searches were improved, with an average of 7.2% improvement amongst the paths that did improve. This, however, still only amounts to a .89% overall improvement over the control algorithm. As expected, when scaling up the delay (which in effect decreases the pixel distance / simulated distance ratio), my search performs better and better than a search that doesn't account for intersections. This action, however, decreases the realism of the generated maps by a similar factor.

It is important, however, to also look at the resources used. Had google the choice to improve all of their inner-city searches by 3-5% at the cost of a 3-5% increase in data and processing time, it is likely that they would go for it. The size of this data, however, is much larger. The program developed for the project, although not perfect, used almost 20% more bytes per location and road in order to make the heuristic and dynamic point cost calculation possible, and an additional 30-40% locations (Although large-scale projects already have locations set up for some sections of road and most intersections, and would not suffer an increase in locations). The heuristic calculation time was also greatly increased - almost a 35% difference in select cases, which would be impractical in a large-scale environment, even though it's on the same order of magnitude.

While this area of pathfinding may be impractical now, the fact that there is room for improvement is important to note. When there are discrepancies of 10-12%, however rare, between the fastest travel path and the path returned by a currently widely used search, there is definitely some place

where the search fails. While this project fails to deliver a viable solution to the problems in realistic map pathfinding, it does provide insights into the possibilities and limitations of the algorithms and methods used.

# 6 Discussion

The addition of traffic obstacles to path finding techniques in maps has a greater effect on small-scale applications such as cities. When it comes to cross-country road trips where one simply drives on the highway for a great majority of the trip, the program will not be very valuable. It is designed to study different road patterns for commuters with several choices for roads. It is valuable in figuring out whether to take the extra couple miles and use a highway versus using a smaller, direct road with a lot of intersections. Here there are some limitations to the project conducted. Since I did not have any real-life data and mapping, and instead used a randomly generated graph, no matter how good my algorithm for creating the map was, it is distinguishable from real maps. It is likely that results would be better with a real map, as they tend to have more intersections and a much more grid-like structure in many places, which is where point costs become more relevant.

Further research into real-life traffic patterns and delay caused by different kinds of traffic obstacles would be the obvious next step for the program. The better the approximations for such delays, the better the program will be at pointing the user in the right direction. Another extension could be incorporating this structure into a traffic simulation with a dynamic search, perhaps even showing real-time fastest paths as traffic load changes. Combined with a network of live empyrical data feeds, this can be expanded to be a great real-time searching tool. For a large scale implementation, however, porting the program to C and other faster languages would be needed, as Python is not meant to handle such large pieces of data and processing.

# References


[1] T. Cain, "Practical Optimizations for A* Path Generation", A.I. Game Programming Wisdom, Charles River Media, 2002.

[2] P. C. Chen, Y. K. Hwang, "SANDROSA Dynamic Graph Search Algorithm for Motion Planning",

IEEE Transactions on Robotics and Automation, Vol. 14, No. 3, June 1998.

[3] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein, "Graph Traversal and Pathfinding Algorithms", Introduction to Algorithms, Second Edition, MIT Press, 2001.

[4] G. Johnson, "Avoiding Dynamic Obstacles and Hazards", A.I. Game Programming Wisdom 2, Charles River Media, 2004.

[5] "Roadway Extent, Characteristics, and Performance", *U.S. Department of Transportation Federal Highway Administration*. Retrieved February 22, 2007, from http://www.fhwa.dot.gov/policy/ohim/hs03/re.htm

[6] S. J. Russell, P. Norvig, Artificial Intelligence: A Modern Approach, pp. 97-104, Prentice Hall, 2002.

[7] B. Salomon, M. Garber, M. C. Lin, D. Manocha, "Interactive Navigation in Complex Environments Using Path Planning", *University of North Carolina Department of Computer Science*. Retrieved January 9, 2007, from http://gamma.cs.unc.edu/Navigation/navpath.pdf.