Hierarchical Pathfinding by Density-Based Neighborhood Clustering

Conner Hinman

Department of Computer Science

Georgia State University

Atlanta, Georgia, United States

c@nnerhinman.com

*Abstract*—In this paper the problem of pathfinding is presented in the context of Vehicle Guidance Systems (VGS), and a solution is detailed based on density-based clustering techniques. Density-based clustering on Global Positioning System (GPS) data creates a reasonable partition of the problem into a hierarchy, each layer of which, a different traditional approach is better suited for. Traditional approaches are introduced, their ideal use-cases detailed, and thus the case is made for the hierarchical approach. Approaches for navigating within clusters, between clusters, and ultimately toward a destination are described. The use-cases for which our cluster-based technique is intended are described, and the benefit over traditional methods is documented.

Keywords—clustering; pathfinding; GPS; VGS; DBSCAN; convex hull; Djikstra; shortest path; A-star; heuristic; graph

# Introduction

The Global Positioning System (GPS) is a large, space-based navigation system, which provides a system of spatial coordinates, made using satellite detection, for use in spatial identification and navigation. Pathfinding is the process of determining programmatically an acceptable path, from a specified source to a specified destination, in a graph-like structure. In this paper, the problem of graphical pathfinding is explored in the context of Vehicle Guidance Systems (VGS). An optimization method is proposed, in which density-based clustering is used to reduce the complexity of the process by properly separating the problem into hierarchical subsections, each better solved by a different pathfinding algorithm.

The problem of pathfinding, as it relates to GPS navigation, is solved using search algorithms that navigate their way through a highly populated, and structurally variable graph of intersections (nodes) and the roadways that connect them (edges). Common approaches to this problem generally fall in to one of the two following categories: greedy uninformed approaches, and heuristic-based approaches. The uninformed approaches, the most popular of which is Djikstra’s algorithm, search the graph on limited information. Djikstra’s algorithm is computationally expensive, but produces accurate results. Heuristic-based approaches seek to improve upon the speed of typical search algorithms by sacrificing a bit of accuracy. They do so by adding their own cost estimate to the provided weights, based on secondary information. The most common heuristic-based algorithm for this problem setting is the A-Star algorithm. It uses an estimation of the distance to the destination, for instance the straight-line distance, as its heuristic value, and enjoys great efficiency with little loss of accuracy. Each algorithm has its own ideal scenario, and its own pitfalls.

Solving the problem with Djikstra’s algorithm achieves great accuracy, but does so by greedily exploring immediate paths, resulting in a high number of computations. As the graph becomes dense, and the level of interconnectivity increases, Djikstra’s begins to make more suboptimal choices. For high numbers of vertices () and edges (), the greedy method becomes prohibitive. However, solving the problem with heuristic approaches is not without its own pitfalls. In graphs with large variation among edge weights (ie dense areas separated by long distance), A-Star can make poor decisions regarding edges with large weight, and suffer from poor performance.

Our method employs clustering, specifically density-based clustering, using the DBSCAN algorithm, to separate graphs into hierarchical subsections, within which each algorithm is more suited. The remainder of this paper explores the motivations for such an approach, and details the specifics of its implementation and performance

# Pathfinding for VGS Systems

Global road systems are complex and have incredibly varied types of structure, so neither approach to the problem functions best in all scenarios. The graph representing these road systems consists of streets and highways represented by edges, and their intersections represented by vertices. The structure of roadways is such that there are many dense, highly interconnected areas, within which distances and thus edge costs are short. These dense areas can be thought of as cities or neighborhoods. Within them, there are greater numbers of edges and vertices, compared to the areas that surround them. Connecting the dense areas, which will be referred to as neighborhoods, there are comparatively long edges, though there are much fewer of them. Navigation within neighborhoods consists of making numerous decisions, each of which adds little to the cost of the search alone. Navigation between neighborhoods incurs much greater cost with each move, and thus the accuracy of the decision matters much more.

Regarding the graph as a collection of dense neighborhoods, with sprawling interconnections, the problem of navigation can be divided into two problems: navigating within neighborhoods, a problem suited for A-Star, and navigating between neighborhoods, a problem for which the accuracy of Djikstra’s is necessary.

# Djikstra’s shortest path

The general approach to this problem is to use, Djikstra’s shortest path algorithm, or some variation of it, for each new destination. In experimentation using real road networks, Djikstra’s algorithm has proved to outperform very consistently [1]. Djikstra’s algorithm is referred to as an uninformed search and employs a greedy best-first search, meaning it traverses the edge with the smallest weight at each step. The explorative nature of the algorithm makes for accurate results, with consistently optimal path outputs. However, that same nature makes it run rather slowly in densely connected graphs with many vertices. At each step, the algorithm has little concept of its destination, and so it can make many superfluous steps on its way.

The explorative nature of Djikstra’s algorithm results in a time complexity of in the worst case. This is typically improved upon in practice by using a minimum priority queue. Other contemporary solutions employ clever tricks such as preprocessing the data by building adjacency matrices [1], a costly process. Though the algorithm is necessarily sequential, some level of parallelism can be achieved by offloading some of the priority queue operations to an auxiliary processor [2].

Though there are approaches for supplementing Djikstra’s algorithm, typically offering some constant speedup, the greedy nature of it is just not ideal in dense graphs. To improve runtimes, heuristic approaches perform informed searches, which sacrifice little accuracy, and greatly outperform in dense graphs.

# Heuristic approaches

A heuristic approach to the search problem, is one that uses some superficial cost predictions, to augment the given edge weights. The typical choice in the VGS scenario is the A\* algorithm, because it uses as a heuristic the estimated distance to the destination. A common choice is the straight-line distance between the node and the destination. A-Star is essentially a modification of Djikstra’s algorithm. It is a best first algorithm, and greedily chooses the best choice, but it does so by augmenting edge weights with the heuristic value. It achieves great performance improvements over Djikstra’s algorithm.

However, the inherent lack of accuracy in heuristic approaches can result in very suboptimal behavior for certain structures of graph. With the augmentation of the edge weights, it is possible for A\* to make incredibly suboptimal decisions. If A\* traverses a long edge to a neighborhood closer to the destination, but the neighborhood is not connected to the destination, without foresight the algorithm will never escape that neighborhood. Since A\* stops when it reaches its destination, but makes possibly suboptimal decisions along its way, poor decisions for the paths between neighborhoods, make for terribly inefficient paths.

Hierarchical heuristic methods, such as the Hierarchical A-Star algorithm (HAS) [3] seek to solve this problem by creating hierarchical spatial abstractions. Each level of the hierarchy may have different implementations. The hierarchical method is a good insight and allows algorithms to avoid things at one point in the execution that they may have explored earlier. For instance, a hierarchical method would not explore the inner structure of a neighborhood which contains neither the source nor destination node, but a straightforward A\* search might. Some contemporary methods, such as HAS [3], design their hierarchy through manual observation of the data, relying on humans to describe the structure. This is a tedious and inefficient solution, and is better handled using automatic clustering techniques.

# Clustering for hierarchy

On first observation of a map, a human might analyze it by finding logical structures within it, and looking for the relations between structures. That behavior might manifest in noticing relatively dense areas, relatively interconnected areas, or any sort of defining characteristics that allow the map to be decomposed into a combination of less complex structures. For each of these manifestations, there is an automatic analog that can be realized through clustering.

A number of graph clustering techniques exist, the majority of which use the idea of interconnection as a measure of similarity. Others may use the distance between two connected vertices, such as an algorithm that uses the shortest path overlap [4] implicitly using Djikstra’s algorithm to determine the distance. Interconnection between vertices is certainly a useful metric for graph clustering, but in the case of VGS pathfinding, it turns out to be more than is necessary, and constantly computing Djikstra’s algorithm defeats the purpose of clustering the data in the first place. In the context of GPS, data using the interconnections between nodes is overkill, as interconnectivity is implied by proximity. When considering the use-cases of Djikstra’s algorithm and A-Star, the purpose of clustering is clearly to cluster regions of great density together, for use with A-Star, leaving inter-cluster travel to Djikstra’s. This implies that we can greatly reduce the size of the clustering problem by disregarding the graph structure, and simply clustering based on spatial density. The algorithm best suited for this job is the DBSCAN algorithm.

DBSCAN works by accepting some predefined distance value and clustering together points that are packed closely enough together. The algorithm accepts all points, and classifies them as core points, which are central to the clusters, member points, which belong to clusters, or noise points which belong to no cluster. The result for GPS data is that vertices are clustered into neighborhoods of nodes with little distance between them. In the typical DBSCAN, noise points are discarded, but in our scenario, it makes more sense to keep noise points and classify them as their own clusters. The end result is a collection of interconnected clusters, within which there is either a single node, or a collection of minimally distanced nodes. This is the neighborhood model, for which the use-cases for the disparate A-Star algorithm and Djikstra’s algorithm, are separated and their boundaries well defined. Djikstra’s can avoid the prohibitively large search spaces of dense urban areas, where it in practice, fails to compute paths in an acceptable time window [5]. A-Star can avoid the potential flummox of computing an unacceptable path by making an incorrect choice, with regards to an edge of high cost. DBSCAN makes more sense than other non-graph clustering methods, such as K-Means, because it doesn’t require a predefined number of clusters and thus works in an unsupervised environment. Manual observation of the graph is avoided.

DBSCAN has a runtime complexity of , and only must be run once, to improve computation time on future iterations of pathfinding within the same system. It functions as an incredibly reasonable preprocessing method, for this particular domain. Now the problem consists of navigating within clusters, and navigating between clusters.

# Inter-cluster navigation

Now that the graph has been preprocessed to determine clusters, a method for navigating between the clusters is necessary. The ideal method is to determine the peripheral nodes with the least cost, connecting each pair of clusters. A small bit of intuition worth considering is that no neighborhood-like structure would have peripheral nodes, around which all paths toward the center passed. Most paths between neighborhoods will pass through one of the nodes that comprise the outside border of the neighborhood. To determine the true shortest path, would require comparison of every point pair between two clusters. This is too computationally expensive and once again, defeats the purpose of the preprocessing. Using that bit of intuition it is safe to assume that we can reduce the points considered to the points on the peripheries of the clusters.

An algorithm called the Convex Hull helps solve this problem. Convex Hull takes a set of points in a graph, and selects the subset, which can form a convex polygon, encompassing all other points in the cluster, such that no point falls outside of the polygon. This extra bit of preprocessing does a good job of reducing future complexity, and does so with a time complexity of only for each cluster, amounting to for the entire map.

Determining the shortest route between clusters is now reduced to determining the smallest edge weight between each pair of clusters, which can be done using Quicksort, which runs in , for each cluster pair, where is the number of edges connecting the clusters. Results are stored in a lookup table.

With the shortest connecting weights between each cluster pair determined, ideal paths, which traverse through multiple clusters, are still left undiscovered. A cluster may never know if it has the ability of reaching a cluster to which it is not immediately connected. At this level of abstraction, where the number of vertices and edges to consider is greatly reduced, is where Djikstra’s algorithm can thrive, and its accuracy is needed. In a final preprocessing step, Djikstra’s algorithm is run, to determine the true shortest path between clusters, and the values in the lookup table are replaced.

The complexity of the data which Djikstra’s algorithm considers is greatly reduced, alleviating runtime concerns, and the type of problem left for A-Star to consider, is one for which it is perfectly suited. Djikstra’s algorithm runs only once, and need not be recomputed for every new destination. The concise sequence of our pathfinding process follows.

# Algorithm sequence and comparison

The full execution steps of our process are as follows:

1. Obtain GPS data, and store in a graph , of vertices , and edges .
2. Run DBSCAN on the coordinates, to cluster dense neighborhoods: .
3. Run Convex Hull for each cluster to determine the peripheral points: .
4. Determine the shortest edge weight connecting each cluster pair, using quicksort: , where n is the number of points on the cluster border. will be in practice.
5. Compute Djikstra’s to find the shortest path between all cluster pairs: , where n is the number of clusters, and is significantly less than |V|.
6. For each iteration that follows, determine the source and destination clusters. Look up the cluster path, and for each cluster in between, navigate to the peripheral vertex, which connects to the next cluster in sequence, using A-Star: , where is the average degree of the vertices, and is the depth of the eventual solution.

For just a single iteration, the combined computational complexities of the preprocessing techniques make for a similar runtime to the straightforward Djikstra approach. For each iteration that follows, the benefits of A-Star begin to produce great outperformance, as Djikstra’s algorithm is continually recomputed.

# Conclusion

In conclusion, different search algorithms are better suited for the problem of pathfinding in different situations. Contemporary approaches typically seek to exploit Djikstra’s algorithm and improve upon it marginally with slight modifications such as multiple source parallelization, computation offloading, and preprocessing. But, Djikstra’s solution is not ideal in certain scenarios, where graph complexity is high, and speed is crucial. Where, Djikstra’s algorithm does poorly, heuristic approaches, like A-Star, thrive, and vice versa. The problem of pathfinding by VGS provides for a textbook example of a split of these algorithms’ ideal use-cases, due to the nature of roadway infrastructure, and thus GPS graph data. An ideal method for splitting the problem as such is a preprocessing by clustering. We have detailed how a density-based clustering method for pathfinding might be implemented, outlined our specific implementation, and made the case for its use in practice.

##### References

1. Lixia Zhang, Fuquan Pan, and Fuyuan Wang, “Study and Realization of Vehicle Guidance System Based on GIS and GPS”. IEEE International Conference on Service Operations and Logistics, and Informatics, 2008. IEEE/SOLI 2008.
2. Jason Loew, Dmitry Ponomarev, Patrick H. Madden, “Customized Architectures for Faster Route Finding in GPS-Based Navigation Systems”. 2010 IEEE 8th Symposium on Application Specific Processors (SASP).
3. Haifeng Wang, Jiawei Zhou, Guifeng Zheng, Yun Liang, “HAS: Hierarchical A-Star algorithm for big map navigation in special areas”. 2014 5th International Conference on Digital Home (ICDH).
4. G.R. Jagadeesh, T. Srikanthan, Senior Member, IEEE, and K.H. Quek, “Heuristic Techniques for Accelerating Hierarchical Routing on Road Networks”. IEEE Transactions on Intelligent Transportation Systems, Vol.3, No. 4, December 2002.
5. Waqas Nawaz, “Traversal Optimizations and Analysis for Large Graph Clustering”. 2015 31st IEEE International Conference on Data Engineering Workshops (ICDEW)