A Conjecture for Optimizing the Shortest Path Problem in Graphs

Washington State University

Abstract

For the problem of finding the shortest path in the sparse graph, an algorithm for improving the efficiency of the query by advancing and merging adjacent points in the graph is explored.

Keywords: Shortest path

A Conjecture for Optimizing the Shortest Path Problem in Graphs

Introduction

The shortest path search algorithm of the graph is used in many ways. The most typical example map program. However, as the scale of the graph continues to increase, the number of nodes and edges increases, and the amount of computation for path search increases significantly. Such problems can significantly lead to the problem that such search requirements can be summarized as searching for the shortest path in the sparse graph. Because the data of such graphs does not change frequently, there is a possibility to preprocess the data in the graph to obtain higher performance in the search. If performance can be significantly improved by such optimization, then similar searches can be made easier on devices with limited performance such as mobile devices. And performance is better when offline or when you need to re-route frequently.

Observing the characteristics of such sparse graphs, in most cases, the nodes have only a small number of edges, and the distances to the surrounding connections are very close, and only a few are distant edges. Then, these adjacent points are calculated as one node, and only considering these adjacent points at the destination and the departure place should be able to reduce a large amount of calculation. Similar to the route planned from New York to Seattle in the map, we can calculate the point group in the figure, such as the point of Chicago, as a point, and do not need to calculate every point in the city of Chicago. In order to achieve this, an algorithm is needed to implement the process of merging multiple points into one point and to ensure that the search results are correct.

Related Work

The shortest path search algorithm is one of the basic contents of computer algorithms, especially in the process of data mining today. Different methods for solving different requirements have also been proposed by scholars. Chow E (2005) points out that use the A* algorithm to optimize the shortest path search of graphs in “A Graph Search Heuristic for Shortest Distance Paths”. (Chow E. 2005) According to Rattigan et al. (2007), “Graph clustering with network structure indices ", the graph clustering algorithm can be used to estimate the shortest path. (Rattigan et al.,2007) These things are for all types of graphs, and there are some optimizations for specific situations. The article “Shortest path approximate algorithm for complex network analysis” by Tang et al., describes a valuation algorithm for large-scale networks. It is mentioned that the value distribution of edges in large-scale networks is concentrated, so it is possible to cluster with multiple high-weight points. However, the graph data used to describe the map is basically unchanged, and such search graphs can be further optimized to obtain accurate values.

Algorithm

This article envisages the following optimization steps:

First set the distance threshold D, used to determine whether the current point and the next point score are grouped into a group. It can be a constant or a variable. According to my code test, the dynamic quantity distinguishes better performance, but it makes the subgraph size difficult to control. The specific search algorithm is free to choose, using Dijkstra's algorithm in my own test program. The algorithm is as follows:

Original input G (V, E), where V is the points set and E is the edges set

(1) visit every point that is not accessed in the graph and mark the visited point as an access lab.

(2) For each current visit point, create a new graph G'. Assume that the current point is A.

(3) Breadth searches its neighbors and mark them visited. Assume that the current neighbor is B. the edge connect A and B is E (A, B). If the distance is less than or equal to D, then remove this B point from G, add B to G'. and remove E (A, B) from G, add E (A, B) into G'. If the distance is greater than d, add a pair of "virtual point" X to each of G and G', to make this edge. Then add E (A, X) which distance equal to 0 into V', and add E (B, X) into V', the distance of that edge is equal to E (A, B). Remove E (A, B) from G.

(4) Calculating the shortest path length d_min between each virtual node in V', the add marked E (X, X') with distance equal to d_min into G.

(5) Repeat step 3 until there is no point. If G' is empty except virtual point, delete two virtual points and point the edge of V to the virtual point to the current point, re-add the current point to G, delete V'. If G' has other points besides the virtual point, calculating every shortest path length d_min between each virtual node in V', the add marked E (X, X') with distance equal to d_min into G.

Example: Figure 1 shows part of a raw graph G, after pre-processing with D = 10, the part of G graph will become to Figure 2. a and Figure 2.b. The G graph in Figure 2.a stores the shortest path information in Figure 2.b. Figure 2. a and Figure 2. b 's c points are one-to-one correspondence

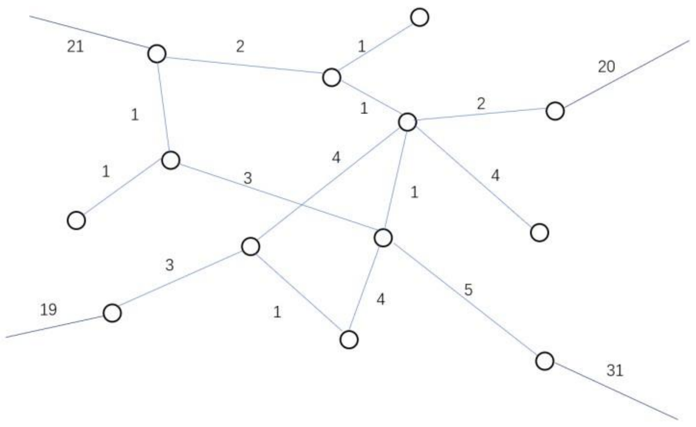
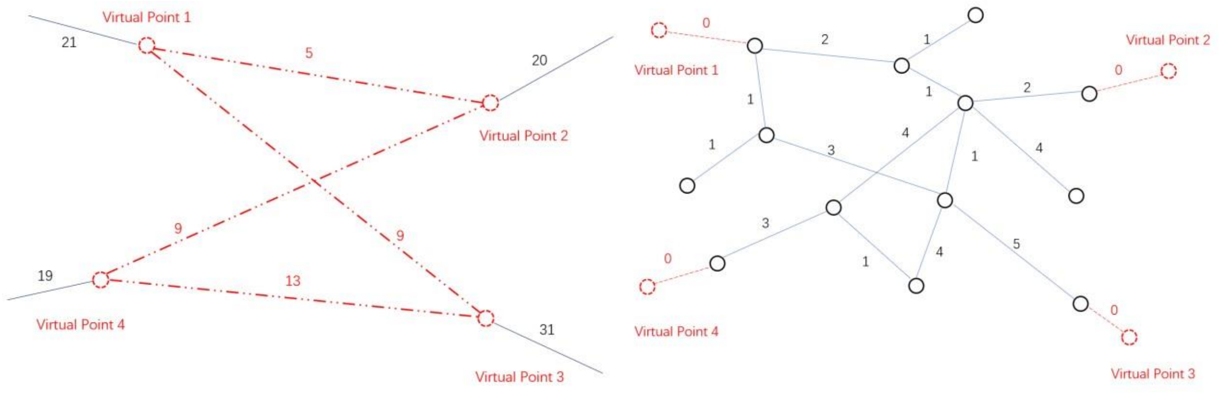


Figure 1. Example Raw Graph G



(a)Pre-processed Graph G (b) Pre-processed Graph G’

Figure 2. Example Pre-processed Graph

After the pre-processing is completed, a total map G is generated, and the “virtual points” of the plurality of sub-graphs G' respectively point to the corresponding virtual points of the total map G. Search algorithm is as follows:

Assume search from A to B.

(1) If two points don't belong to a submap, search in G and use marked edge when searching. If the result includes marked edges, search for the corresponding submap to replace that edge in the result.

(2) If point A belongs to a subgraph, then search the shortest path from A to each virtual point in the same subgraph and find the path from B to the corresponding submap point in the general graph G, and the path is calculated and taken. Calculate the distance of the combined route to find the shortest.

(3) If both points belong to two subgraphs, the shortest path between the two points in the subgraph G' is calculated. And calculate the distance between the two points in G' to the virtual point and the distance of the virtual point in the G map to the non-virtual path. Comparing the path within the subgraph with the outer path of the subgraph, the shorter one is the result.

For example, suggest a problem is search shortest path from A in Figure 2.b to B in Figure 2.a. The solution is finding shortest way from A to every “virtual point” in Figure 2.b and find every path from point B to “virtual point” in G, these “virtual point” are same as destination points in G’. Then calculate every A- “virtual point”-B path distance, the shortest one is what we what.

Evaluation

The difficulty with this algorithm is how to determine the threshold distance. In other words, how to judge whether the neighbor of a certain point is in the same subgraph with the current point. Because of the different densities of different regions, using a constant threshold makes the subgraph distribution more like a fixed-grid grid, which is less effective. So, a dynamic distance is more suitable for the current problem. The dynamic distance should be related to the overall distance distribution of the graph. Too small distance value will produce too many subgraphs, which will not help the optimization but will increase the amount of calculation. Too large distance value will make the subgraph include too many points, making the search close to optimization. Limited to my ability, I did not find a suitable mathematical method to calculate the appropriate distance threshold, so I can only analyze this from the perspective of complexity

the complexity can be optimized to $O\ (V\ lg\ V + E)$ by the Fibonacci heap. (Cormen,2009, p.662) The optimal distance algorithm obtained from the formula can adjust the average complexity of the processed main and subgraphs to approximate. So, the algorithm complexity proposed in this paper is as follows:

E' is the average number of virtual edges that each subgraph needs to add, and C is the average number of virtual points that each subgraph needs to add.

which means that if one graph can be divided into multiples, there is a suitable N value to make the total computation of multiple subgraphs is less than the computation of the raw graph. At the same time, it can be analyzed that if the distance value is not appropriate, the additional virtual point and virtual edge and the computational cost of switching between the graphs will make the overall algorithm need more time.

References

Last Name, F. M. (Year). Article Title. *Journal Title*, Pages From - To.

Last Name, F. M. (Year). *Book Title*. City Name: Publisher Name.

Chow, E. (2005). A Graph Search Heuristic for Shortest Distance Paths. *Presented At: AAAI-*

05: The Twentieth National Conference on Artificial Intelligence, Presented at: AAAI-05: The Twentieth National Conference on Artificial Intelligence, 2005.

Rattigan, M., Maier, M., & Jensen, D. (2007). Graph clustering with network structure indices. *Proceedings of the 24th International Conference on Machine Learning, 227*, pages 783-790.doi: 10.1145/1273496.1273595

Tang, Jin-Tao, Wang, Ting, & Wang, Ji. (2011). Shortest Path Approximate Algorithm for Complex Network Analysis. *Journal of Software*,*22*(10), pages 2279-2290.

Cormen, T. (2009). *Introduction to algorithms* (Third ed.). Cambridge, Mass.: MIT Press. Page 662