2019算法设计与分析课程报告

题目： **A Study on A\* Algorithm Applying Reversed Direction Method for High Accuracy of the Shortest Path Searching**

姓名：孟济模

学号：L170300101

# 论文介绍

【介绍所阅读论文】

## 论文背景

In this paper, Ryu Yeong-Geun and Park Yongjin, those two researchers looking for more efficient, quicker algorithm to find a way to go destination known as shortest path problem.

## 问题定义

The way to find shortest path is to find minimum weight from start node to target node. And the main concept of A\* algorithm is huristic function h(x). So in this paper, they try to change a way to search reversing star node to target node that huristic function can be effected.

## 算法介绍

the edges correspond to road segments, each weighted by the length of the segment. The shortest path can be defined for graphs whether undirected, directed, or mixed.

The most important algorithms for solving this problem are:

<Dijkstra's algorithm> solves the single-source shortest path problem with non-negative edge weight.

<Bellman–Ford algorithm> solves the single-source problem if edge weights may be negative.

<A\* search algorithm> solves for single pair shortest path using heuristics to try to speed up the search.

<Floyd–Warshall algorithm> solves all pairs shortest paths.

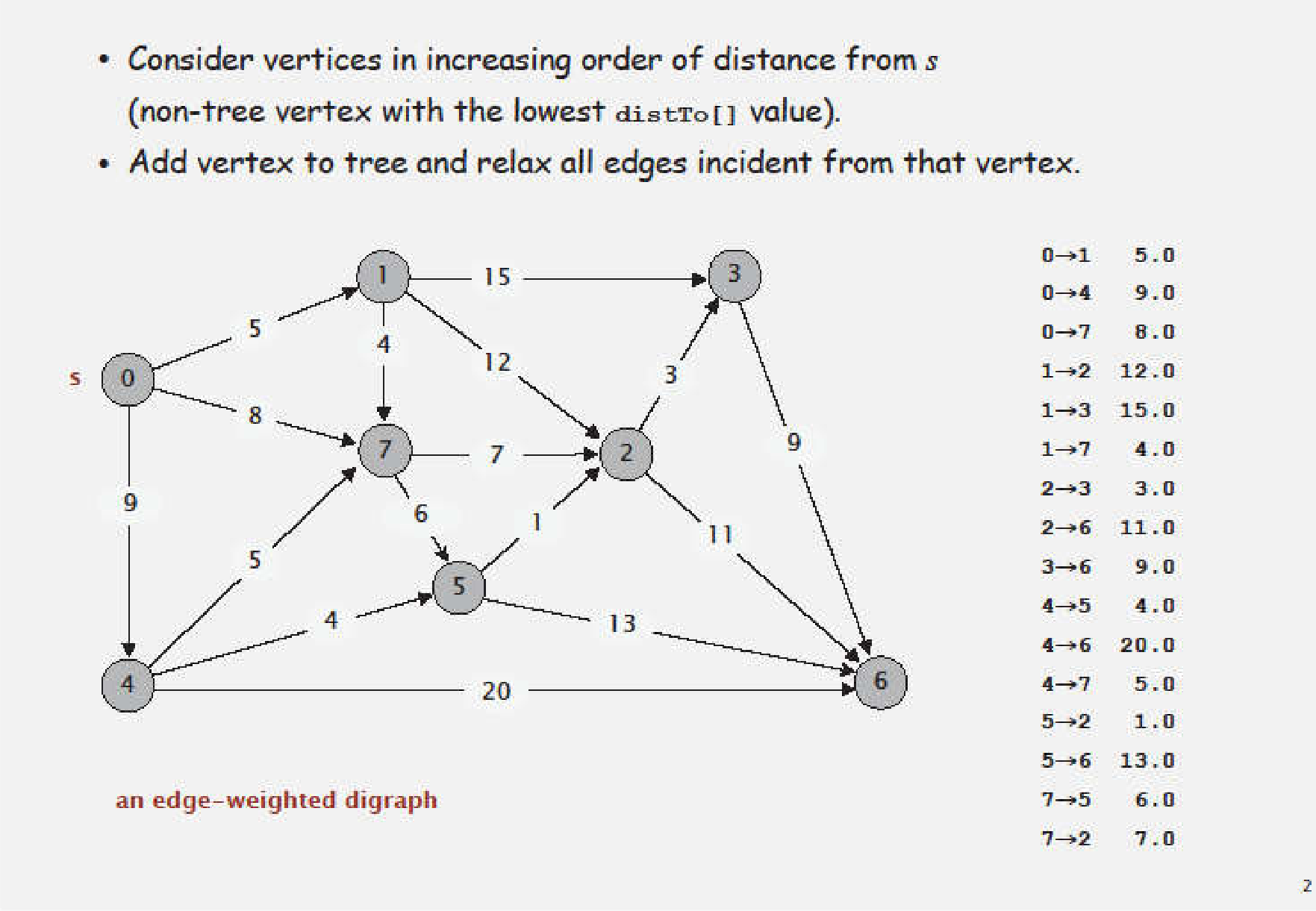
<Johnson's algorithm> solves all pairs shortest paths, and may be faster than Floyd–Warshall on sparse graphs.

<Viterbi algorithm> solves the shortest stochastic path problem with an additional probabilistic weight on each node.

2) Dijkstra algorithm

Dijkstra's algorithm is an algorithm for finding the shortest paths between nodes in a graph, which may represent, for example, road networks. It was conceived by computer scientist Edsger W. Dijkstra

Dijkstra's original algorithm does not use a min-priority queue and runs in time { O(|V|^{2})} O(|V|^{2}) (where {|V|} |V| is the number of nodes). The implementation based on a min-priority queue implemented by a Fibonacci heap and running in { O(|E|+|V|\log |V|)} O(|E|+|V|\log |V|) (where { |E|} |E| is the number of edges) is due to Fredman & Tarjan 1984. This is asymptotically the fastest known single-source shortest-path algorithm for arbitrary directed graphs with unbounded non-negative weights.



Way to execute Dijkstra’s algorithm

Let the node at which we are starting be called the initial node. Let the distance of node Y be the distance from the initial node to Y. Dijkstra's algorithm will assign some initial distance values and will try to improve them step by step.

1. Mark all nodes unvisited. Create a set of all the unvisited nodes called the unvisited set.

2. Assign to every node a tentative distance value: set it to zero for our initial node and to infinity for all other nodes. Set the initial node as current.

3. For the current node, consider all of its unvisited neighbours and calculate their tentative distances through the current node. Compare the newly calculated tentative distance to the current assigned value and assign the smaller one. For example, if the current node A is marked with a distance of 6, and the edge connecting it with a neighbour B has length 2, then the distance to B through A will be 6 + 2 = 8. If B was previously marked with a distance greater than 8 then change it to 8. Otherwise, keep the current value.

4. When we are done considering all of the unvisited neighbours of the current node, mark the current node as visited and remove it from the unvisited set. A visited node will never be checked again.

5. If the destination node has been marked visited (when planning a route between two specific nodes) or if the smallest tentative distance among the nodes in the unvisited set is infinity (when planning a complete traversal; occurs when there is no connection between the initial node and remaining unvisited nodes), then stop. The algorithm has finished.

Otherwise, select the unvisited node that is marked with the smallest tentative distance, set it as the new "current node", and go back to step 3.

3) A\* algorithm

A\* is a computer algorithm that is widely used in pathfinding and graph traversal, which is the process of finding a path between multiple points, called "nodes". It enjoys widespread use due to its performance and accuracy. However, in practical travel-routing systems, it is generally outperformed by algorithms which can pre-process the graph to attain better performance, although other work has found A\* to be superior to other approaches.

A\* algorithm is formulated in terms of weighted graphs. starting from a specific starting node of a graph, aims to find a path to the given goal node having the smallest cost. It does this by maintaining a tree of paths originating at the start node and extending those paths one edge at a time until its termination criterion is satisfied.

At each iteration of its main loop, A\* needs to determine which of its paths to extend. It does so based on the cost of the path and an estimate of the cost required to extend the path all the way to the goal. Specifically, A\* selects the path that minimizes { f(n)=g(n)+h(n)} f(n)=g(n)+h(n)

Where n is the next node on the path, g(n) is the cost of the path from the start node to n, and h(n) is a heuristic function that estimates the cost of the cheapest path from n to the goal. A\* terminates when the path it chooses to extend is a path from start to goal or if there are no paths eligible to be extended. The heuristic function is problem-specific. If the heuristic function is admissible, meaning that it never overestimates the actual cost to get to the goal, A\* is guaranteed to return a least-cost path from start to goal.

Typical implementations of A\* use a priority queue to perform the repeated selection of minimum cost nodes to expand. This priority queue is known as the open set or fringe. At each step of the algorithm, the node with the lowest f(x) value is removed from the queue, the f and g values of its neighbors are updated accordingly, and these neighbors are added to the queue. The algorithm continues until a goal node has a lower f value than any node in the queue. The f value of the goal is then the cost of the shortest path, since h at the goal is zero in an admissible heuristic.

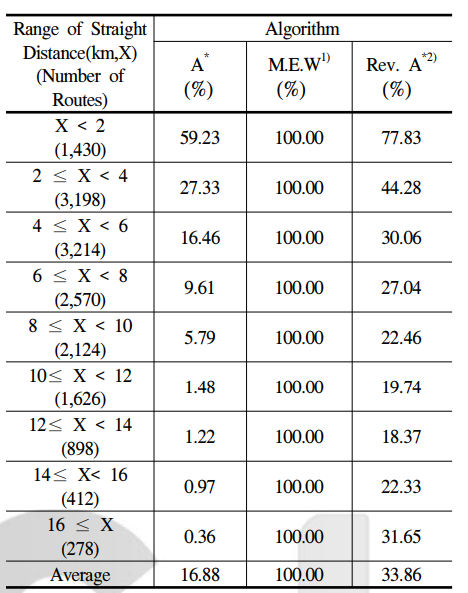
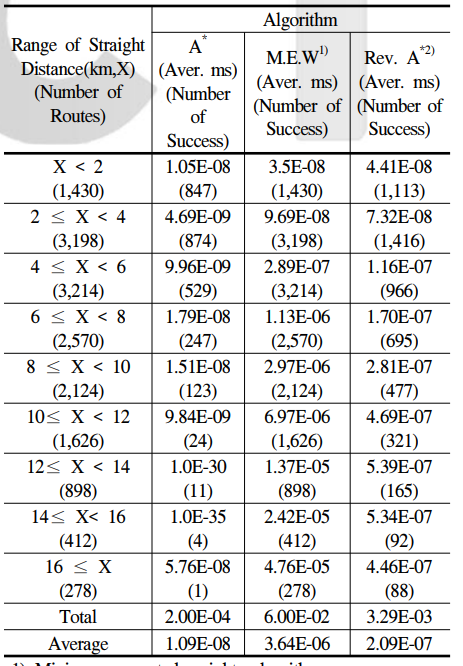
The algorithm described so far gives us only the length of the shortest path. To find the actual sequence of steps, the algorithm can be easily revised so that each node on the path keeps track of its predecessor. After this algorithm is run, the ending node will point to its predecessor, and so on, until some node's predecessor is the start node.

As an example, when searching for the shortest route on a map, h(x) might represent the straight-line distance to the goal, since that is physically the smallest possible distance between any two points.

If the heuristic h satisfies the additional condition h(x) ≤ d(x, y) + h(y) for every edge (x, y) of the graph, then h is called monotone, or consistent. In such a case, A\* can be implemented more efficiently and A\* is equivalent to running Dijkstra's algorithm with the reduced cost d'(x, y) = d(x, y) + h(y) − h(x).

## 理论和实验结论

The research verified using accuracy, lead time, and seeking time three component.

success rate of minimum path searching Running time of minimum path searching

According to success rate table can figure out, success rate of A\* algorithm is the further starting node to destination node, lower probability to success.

The average of running time of minimum path searching turns out A\* = 1.09E-08, minimum Euclidean weight = 3.64E-06, reverse A\* = 2.09E-07 (ms). In this research, Reverse A\* algorithm is spending more time than A\* algorithm but has higher accuracy so that is deserve to use as part of higher efficiency.

# 领域综述

## Research of A\* algorithm

1) Duplex explore of A\* algorithm

Duplex explore means let starting node and destination node execute A\* algorithm in the same time. The success condition of this process is both processes need to meet in same node in the middle of progress. But it turns out not that efficient because it is not better than simplex explore about finding optimized condition that when to stop explore.

2) Restrict of searching area

Restrict of searching area is to purpose overcome shortage of Dijkstra’s algorithm. In 2006,Fu suggest that when seeking process of A\* algorithm the evaluation value exceed estimated limiting value, not to continue process. g(n)+h(n)=E(o,d) -–g(n) : dose of starting node to n node, h(n) : estimated dose of n node to destination node--, E(o,d) =K\*h(n) means limiting value of minimum dose of starting node to destination node. This idea has a problem that coordination factor K is hard to estimate and adjust.

3)Improve evaluate function

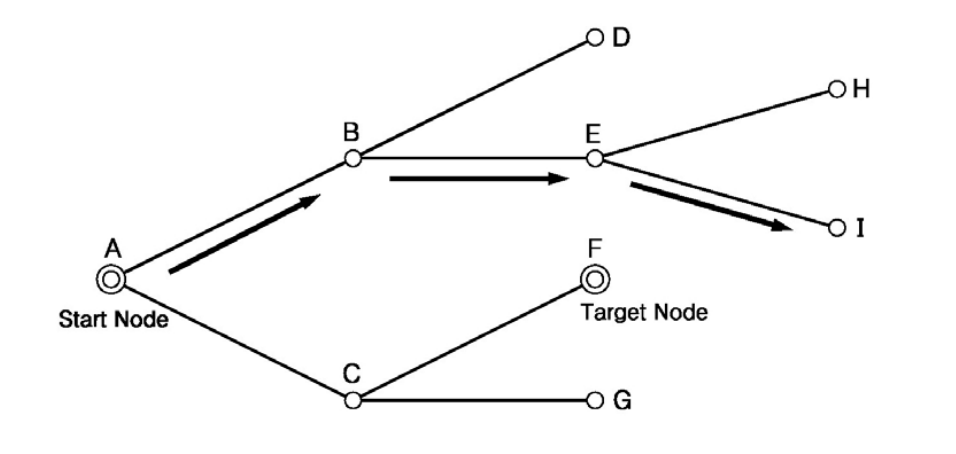
 In normal situation of A\* algorithm, estimated does h(n),distinguish searching node in evaluate function, uses Euclidean distance. In 1996, Fu suggested divide all Euclidean distance to average velocity. Although it has a benefit when according to adaption of average velocity can be higher explore speed, but haven’t improve accuracy.

Ryu Yeong-Geun invented method that h(n) multiple Euclidean distance to calculate minimum estimated does.

## Way to improve accuracy of A\* algorithm

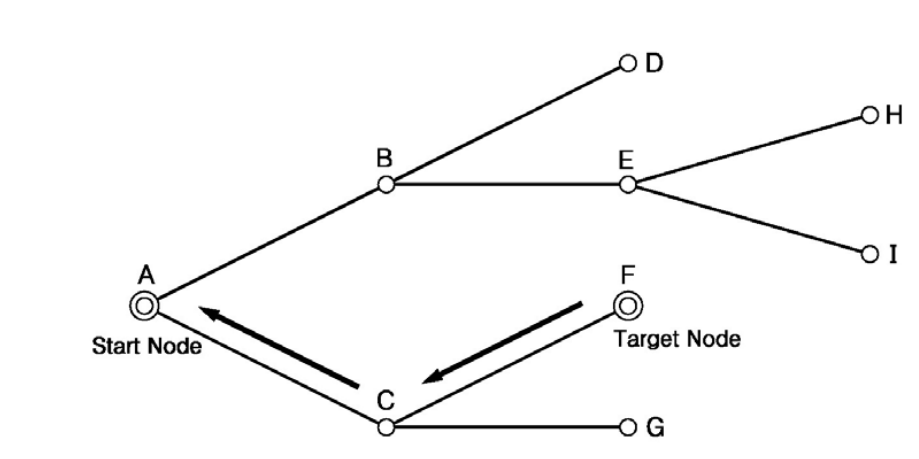
1) development direction

Development direction is trying not to touch explore time(advantage of A\* algorithm) diminish probability of fail shortest path explore. Although Dijkstra’s algorithm is the most stable way to find shortest path, but it need to take too much time. And A\* algorithm has a benefit using DFS, without compare another path, can find shortest path, but it has high risk when cannot find destination node.



example of failed A\* algorithm’s application

Because A\* algorithm doesn’t has function call back, so when a network that is not regular form , it is easy to failure. To avoid this, can figure out link situation of starting node and destination node preferentially. Like let starting node start, find child node, and let this child node as parent node find child node again. Repeat this process until find a destination node. After do this, arranged this link situation, start A\* algorithm from destination node. This process is better than ordinary A\* algorithm in part of that without failure to finding shortest path.



example of reverse direction application of A\* algorithm

# 改进点概述

【描述改论文可以改进的点】