Using A* to find shortest paths in a map of Minneapolis

Yuanzhe Liu
yuanz002
5038957

May 2019

Abstract

A* search algorithm is a greedy search algorithm that balances the current cost and an estimation of future cost. This project implements an A* search algorithm on a map of Minneapolis to find the shortest path between two location. The A* search algorithm shows great efficiency and accuracy compare to a uniform cost search algorithm. The heuristic function used by A* search is the direct distance between a node and goal, which is admissible and consistent. This work confirms the usage of A* search algorithm on pathfinding on maps.

1 Introduction

A* search algorithm is the most widely known form of best-first search[8]. Navigation is one field that a search algorithm most commonly utilized in everyday life. In this project, I am going to implement an A* search algorithm to find shortest paths through roads between two locations in a map of Minneapolis. This project will consisting two parts: an A* search algorithm as the experimental group and a uniform cost search(UCS) algorithm as a reference group. The performance will be measured by the accuracy of the final path and the time cost. Although in this project, I am only implementing a search program on a map of Minneapolis, the final project should have general applicability.

I find this topic is interesting because finding a shortest path is a foundation for navigation. This is one project that I can apply my knowledge to solve real-life problems[2].

2 Approach Review

Uniform cost search algorithm are proven to be able to find optimal solution[3], so the A* search algorithm in this project will be compared with a uniform cost search algorithm. Uniform cost search algorithm will always expand the path with the lowest path cost until it reaches the goal. Although uniform cost search guarantees to find the shortest path, it is very slow. Let C* is the cost of the optimal solution, ϵ is the least cost of each action and b is the branching factor. In worst-case, the time and space complexity is[8]:

$$O(b^{floor(1+C*/\epsilon)})$$

There are some other uniformed search algorithm, as introduced by "Artificial Intelligence A Modern Approach"(Figure 1)[8]. Here we see that Uniform-Cost search is the only search algorithm that always gives optimal solutions.

There are other algorithms are optimal if step costs are all identical[7], but for a map search we cannot make such assumption since the step costs, which are the lengths of roads, are not the same[1].

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Complete?	Yes[a]	Yes[a,b]	No	No	Yes[a]	Yes[a,d]
Time	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(b^m)$	$O(b^\ell)$	$O(b^d)$	$O(b^{d/2})$
Space	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(bm)$	$O(b\ell)$	$O(bd)$	$O(b^{d/2})$
Optimal?	Yes[c]	Yes	No	No	Yes[c]	Yes[c,d]

Figure 3.21 Evaluation of tree-search strategies. b is the branching factor; d is the depth of the shallowest solution; m is the maximum depth of the search tree; l is the depth limit. Superscript caveats are as follows: [a] complete if b is finite; [b] complete if step costs $\geq \epsilon$ for positive ϵ; [c] optimal if step costs are all identical; [d] if both directions use breadth-first search.

Figure 1: Evaluation of tree-search strategies

This project's main method is A* search algorithm. It evaluates nodes by adding the cost to reach the node g(n), and the cost to get from the node to the goal h(n):

$$f(n) = g(n) + h(n)$$

A* guarantees to find optimal solution if the heuristic function is admissible and consistent[2][8]. An admissible heuristic function is one that will never overestimate the cost from the current node to the goal. This way we can make sure the lowest cost path is in the optimal solution we will find. Consistency requires that one path's actual cost shall not be smaller than a path's heuristic cost, if the latter one's real cost is greater than the first one. This may seem not so obvious, but since a map search is a graph search, this may happen if we falsely pick heuristic, where[4]

$$h(n) > c(n, a, n') + h(n')$$

According to triangle inequality, this will not happen if we pick the direct distance of two points as heuristic[5]. Previous works on complexity of A* search shows that with relatively accurate heuristic, A* have time complexity of $O((b^\epsilon)^d$[6]. The time complexity grows as the heuristic fails to estimate the real cost well. An A* with a heuristic function are all zero is the same as uniform cost search. Thus our method should expect improvement on efficiency while keep the correctness of finding optimal paths.

3 Proposed Method

This project uses Python 3.7 as its platform, utilizing numpy and matplotlib modules. .

Each crossroad coordinate is represented as a tuple in the project: (x,y). Since starting and ending location of some segments of road should be the the same, we can consider these ends as nodes of a graph. The heuristic function will be the direct distance from any node to the destination. If the destination happens to be on a crossroad i, then $h(i) = 0$. Since the actual distance is always smaller than or equal to the direct distance (by the inequality of triangle[8]), this heuristic function is admissible. The cost function is simply the sum of length of all segments of road in the path. A* shall not have superiority in the time and space complexity theoretically, since it relay heavily on the effectiveness of the heuristic function. But in practice it shall perform much better.

3.1 Constrains

We make the following assumptions for simplicity:

- The map data shall be given as segments of roads, with the starting number as the allowed travel direction(i.e.one-way road or two-way road), followed by the coordinates of the starting and ending location(two crossroads)
- For start point, the project finds the closest crossroad to the input coordinate.
- For goal, the project uses a crossroad provides the best f(n). The destination could be any arbitrary location, but we can still use A* with that location. A* algorithm terminates when it cannot find any other node that gives a smaller f() value. Consider Figure 2: the destination is in a block surrounded with crossroad A,B,C and D. With a starting location at lower-right corner, A* will find crossroad D. Then A* can no longer find any other crossroad provides a smaller f(), since

  $$g(D) < g(x), x \in \{A, B, C\}$$

  And by the inequality of triangle[8]

  $$f(D) < f(x), x \in \{A, B, C\}$$

  This ensures our new goal does not on the opposite side of the best path.

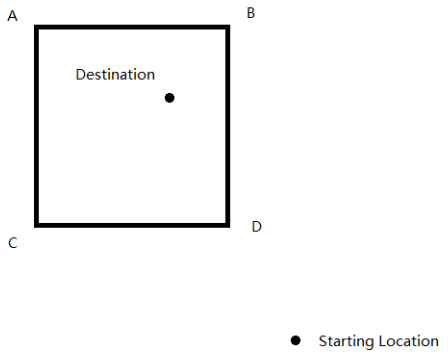

Figure 2: Validity of finding the closest node

3.2 Data Structure

The project has the following key data structure:

- **Map**
  A map holds all segments of road with direction. It is a dictionary with keys as starting coordinate and values as an array of ending coordinate and the length of road. I calculate the length of a segment of road when creating the map to improve the performance when searching. For instance, suppose point K has coordinate of (Kx,ky), a key-value pair in the map dictionary has the representation of :

  $$(Ax, Ay) : [[Bx, By, lB], ..., [Cx, Cy, lC]]$$

  This means the road segments starting from point A are: AB,..., AC, with ending point B,...,C respectively and road segment length of lB,...,lC respectively.

- **Openlist and pathlist**
  The openlist maintains all possible nodes to be searched. It is an array of coordinate and the f value to that point. Pathlist is an array stores the path to the nodes in openlist, respectively. An element in pathlist is an array of coordinates, representing the path from the first coordinate to the last coordinate.
  For instance:
  An element in openlist: $[x, y, f]$
  Its path in pathlist (an element of the pathlist)accordingly:

$[(x1, x2), ..., (xn, yn)]$

- **Searchedlist**
  Searchedlist holds all expanded nodes. This make sure one node will not be visited twice. It is an array of coordinates.

3.3 Pseudo Code

In this A* search algorithm, the heuristic function is the distance between the point and the goal(as in line 8: f(e) = g(e) + distance(e,GOAL)). The uniform cost search algorithm can be mutated from the A* search algorithm, for there is no heuristic function, which is the same as keep the heuristic value a constant 0.
The pseudo code of A* search algorithm is shown as Algorithm 1.

Algorithm 1 A* search

```
1:  initialization from START
2:  final_path ← Null
3:  final_distance ← 0
4:  while True do
5:      next ← Null
6:      min_f ← inf
7:      for e ∈ openlist do
8:          f ← e.g + h(e)
9:          if f < min_f then
10:             next ← e
11:             min_f ← f
12:     delete next from openlist
13:     path ← path(next)
14:     delete path from pathlist
15:     if GOAL = next then
16:         final_path ← path + GOAL
17:         final_distance ← next.g(e))
18:         break
19:     for e ∈ From(e) do
20:         if e ∉ openlist then
21:             openlist ← openlist + [e, g(e)]
22:             pathlist ← pathlist + (path + next)
    return (final_path, final_distance)
```

4 Experiment

I test the A* search algorithm and uniform cost search algorithm on a same set of start and goal under the proposed constraints. The raw map data is shown as Figure 3. An found path(from point(15000,5000) to (25000,15000)) is drawn on the map using red line, i.e. in Figure 4.

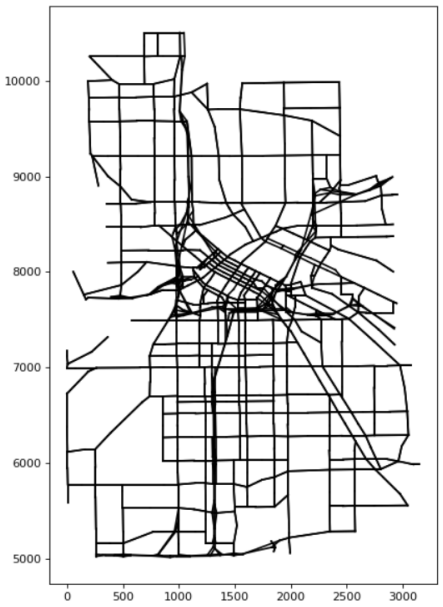

Figure 3: Map of Minneapolis

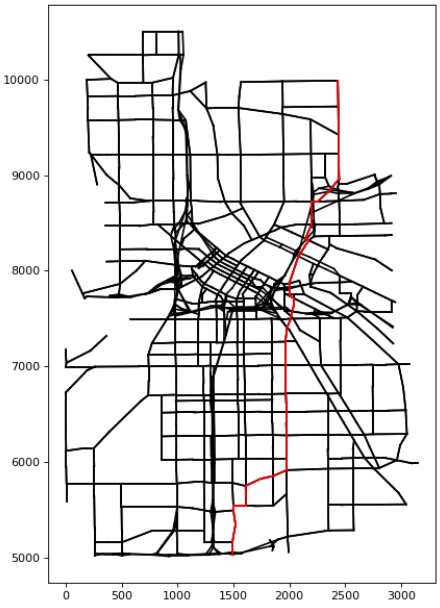

Figure 4: A found path in the map

The terminal prints the total distance found by both algorithm. It also prints the total nodes they expanded respectively. Figure 5 is the terminal output for Figure 4. This way we can compare the efficiency of the algorithms. Lastly the project will also print if the found path are the same. Since uniform cost search guarantees to find the best path, we can check if A* has found the best path.

```
A* search expanded  337  nodes
Uniform cost search expanded  862  nodes
Distance by A*:  5526.963585362886
Distance by ucs:  5526.963585362886
Both searches found the same path
```

Figure 5: Sample output from terminal

The best way to test efficiency of this implementation of A* search algorithm is by comparing the nodes expanded. But it is obvious that longer paths will require longer exploration. 10 sample test results are recorded in Table 1.

Test	Start	Goal	A*	UCS
1	(2690,4844)	(2869,9192)	231	755
2	(234,6675)	(3761,9965)	300	827
3	(3759,6338)	(1627,8125)	90	439
4	(2256,5990)	(209,7795)	493	796
5	(2409,6165)	(2599,5633)	9	59
6	(27,5366)	(3305,9752)	309	861
7	(2051,8273)	(3046,7077)	48	496
8	(1238,7658)	(3081,6007)	145	845
9	(62,7604)	(3207,7003)	238	752
10	(3913,5772)	(1139,6940)	84	309

Table 1: Nodes expanded by both algorithms for 10 searches

5 Analysis

If we print out all paths that algorithms explored, one typical example is shown in Figure 6. We see that uniform cost search expand from the start location towards all directions, while A* search algorithm has a general direction towards its goal.

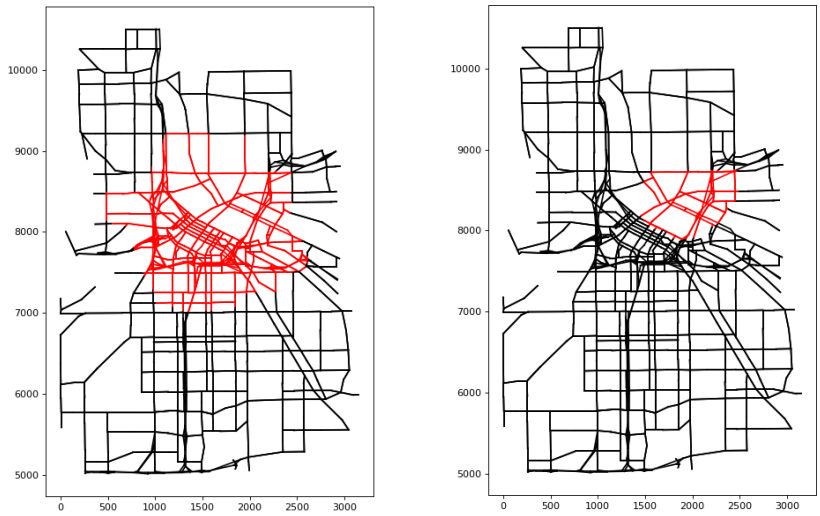

Figure 6: A found path in the map

Uniform cost search generally expands more nodes than A* search and thus takes longer time. Since in map pathfinding, our goal is clear and we have a very simple way to define the heuristic function, A* becomes very effective. As shown in Figure 6, uniform cost search expand its searching like a circle, while A*'s search history is only a circular sector.

To test the improvement on performance, we perform 1000 searches with randomly generated coordination and calculate the ration of nodes expanded by A* search and uniform cost search. The result is shown in Table 2. Meanwhile, both algorithm will always return the same path.

Total tests	Min	Max	Average	SE
1000	100%	2728%	427%	158%

Table 2: Ration of nodes expanded by UCS and A* search

From this table we see that on average uniform cost search expands over 4 times more nodes than A* search. There are cases where both algorithms expand same nodes, such as the goal is only one road segment away from the starting point. Besides being quicker, A* search expands less nodes means it requires less space. Not only nodes to be expanded are stored in openlist, all their optimal path is also stored in pathlist respectively. Although opening nodes takes linear time, storing path history takes exponential storing space. As a result, A* search greatly improves space efficiency compare to uniform

cost search.

6 Conclusions and Future Work

It is proven that A* search algorithm finds optimal solution given admissible and consistent heuristic function. In order to verify the correctness of A* and effectiveness on this particular map, this project is using another proven method – uniform cost search as a comparison. The project successfully implemented A* search algorithm on a Minneapolis map. The implementation secured both efficiency and correctness. Our A* search algorithm performs better than uniform cost search with quicker search time and much better space efficiency.

Future work on this particular project can be done by improving the assumption for start and goal coordinates. Currently the project finds the closest crossroad. This may not be optimal since the closest crossroad could be on the opposite direction of the best path. We could find the projection of the given coordinate on the road, which intersects the line that connecting start and goal. This is a better estimation of real-life situation.

Since the map data we use is just static information, we can update our heuristic function with traffic, road construction, lanes of road and other information to provide the fastest route.

References

[1] Vadim Bulitko. Per-map algorithm selection in real-time heuristic search. In *The Twelfth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 2016.

[2] Rina Dechter and Judea Pearl. Generalized best-first search strategies and the optimality of A*. *Journal of the Association for Computing Machinery*, pages 469–476, 2008.

[3] Ariel Felner. Position paper: Dijkstra's algorithm versus uniform cost search or a case against dijkstra's algorithm. In *The Fourth International Symposium on Combinatorial Search*, 2011.

[4] Pedro F. Felzenszwalb and David McAllester. The generalized A* architecture. *Journal of Artificial Intelligence Research*, pages 153–190, 2007.

[5] David Ferguson, Maxim Likhachev, and Anthony (Tony) Stentz. A guide to heuristic-based path planning. In *Proceedings of the International Work-*

shop on Planning under Uncertainty for Autonomous Systems, International Conference on Automated Planning and Scheduling (ICAPS), June 2005.

[6] Akshay Kumar Guruji, Himansh Agarwal, and D.K.Parsediya. Time-efficient a* algorithm for robot path planning. In *3rd International Conference on Innovations in Automation and Mechatronics Engineering*, pages 144–149, 2016.

[7] Robert C. Holte, Ariel Felner, Guni Sharon, Nathan R. Sturtevant, and Jingwei Chen. A bidirectional search algorithm that is guaranteed to meet in the middle. *Artificial Intelligence*, pages 232–266, 2017.

[8] Stuart Russell and Peter Norvig. *Introduction to Machine Learning (third edition)*. Prentice Hall, 2009.