Route Planning Software

Hao Wu, Yiming Yao

University of Minnesota - Twin Cities

May 11, 2020

Abstract

We design a software to find optimal (shortest) path on the University of Minnesota map by using the A* algorithm. Using GUI (Graphical User Interface) to visualize the path on the map and print the list, which contains the transient nodes. It is definitely efficient way for students to make route planning in the campus. The approach supports that query the shortest path and manages the time in the campus.

1 Introduction

Maps are kind of diagrammatic form that represents physical features such as buildings and roads, and maps play an essential role in the development of civilization. Most of people are familiar with using maps to find routes and locate their position on the maps. However, it is very challenge to find a shortest path in the real world. Therefore, using computers and algorithms are efficient ways to solve route planning problems. Using graph theories to take the place of real buildings and available roads with nodes and edges, then a transportation network has been established by connecting them. Meanwhile, computer software can generate a shortest path in such transportation network. The path shows the optimal solution in the corresponding transportation network, what is more, the tool we designed should be tested against real world, complex situation in order to detect any bugs existing in it.[2]

2 Description

Map is a graphic way to represent spatial concepts. A type which people most familiar with is transportation map. Map is a medium of interaction between human and environment. That is because vision of human cannot satisfied human when they make plan according to environment. An accurate map could help human to learn where they are, and make correct decision. For the record, Napoleon Bonaparte did that we owe the first systematic use of maps in the conduct of war. [1]Accurate information on map helped French Emperor to coordinate geographically expansive campaigns and discrete armies in the field. They indicate positions of enemy armies and do prediction by labelling on map. By calculating distance and highlighted important topographical features on map, predication could be much more accurate. As “a means of visualizing and managing the future,” the Napoleonic map was “the central part of an information-transformation system”[4]

Technology helped map develop more quickly. Map helps human learn a more accurate world, and also makes life of human more convenient. For example, when students join a new university, or they want to know about campus, students could use visual map software installed in Smartphones to get rough idea about it. Map cannot show all detail about each building in campus but it could provide students the most accurate shapes of buildings in vertical view, distance between them and relations of positions. All these information could help students to learn their campus better. However, it is difficult for some of students to find a route between two locations in a complicated campus which is with a big area, like the University of Minnesota. Current technology is able to provide students with a shortage and accurate path showed on visual map installed on phone. Besides that, its application makes work more efficient when people want to get some locations where they never been.

Such technology always is called route planning. Route planning is for computing most efficient route which contains several nodes by making distance travelled or time minimum. Time could be calculated by distance and speed limiting. Therefore, the way to make distance of route as minimum as possible is the most interesting and pivotal part in route planning. As concept of route planning stated, there will be several nodes on route. The key of task is what standard for choosing next node where user can access from start location. Or what expectation to node which is on the shortest route. Absolutely, all of these need computing by algorithm. Fortunately, there are some algorithms which are efficient for computer to solve route planning.

3 Background & Benchmark Other Algorithms

The PNA stands for personal navigation assistants, also known as Personal Navigation Device or Portable Navigation Device (PND) is a portable electronic product which combines a positioning capability and navigation functions. The features of the 3rd generation of the PNA system include determining locations of users and indicating path planning.[8] To determine the location of users, the software will define N to represent the streets system, N contains a set of curves, which can be called arc as well, then set the position P in the transportation network. The algorithm can find nodes, which are close to P, and find a series of arcs that are incident to the nodes. Using a minimum norm projection to find the closest arc, and calculate the minimum distance between p and the line segment. [8]Since there are many nodes on the transportation network, the problem can be simplified as making decisions to nodes surrounding current nodes. [10]These kinds of problems called combinatorial optimization problems, which build a solution step by step. At a step, a single element will be added in the current solution which is partly under construction.[5]

The "Least-cost", which represents the shortest distance of the route, is also always being treated as an optimal solution in the route planning problem. Greedy algorithm for minimization always chooses one of the least costs. In the greedy algorithm, among feasible elements, the algorithm would select an element with the least cost and put it in solution under construction.[5] Applying the greedy algorithm in route planning, start with object S, naive greedy will choose an object which is closed to S. However, we do not know if the object is an element of the optimal solution. What is more, naive greedy heuristic pay all attention to current cost and ignore target object T. Therefore, advanced greedy algorithm called oriented heuristic greedy heuristic can solve the shortage. Let us represent current location is L. When it selects next object O, the oriented greedy heuristic will consider the value of the distance between current location O and target T. Heuristic will choose object O when (dist(L, O) +dist(S, O) + dist(O, T)) is minimum among all possible choices. [10]Instead of focus the cost of a small part in the path, oriented greedy heuristic is better to reduce total cost, which has a much higher possibility to find a route that the user feels satisfied. However, greedy has its limitation. Like real-world is complicated, if we add more constraints to solutions like avoiding some specific object, the greedy approach sometimes cannot guarantee the optimal solution because the solution might not satisfy all constraints.[9]

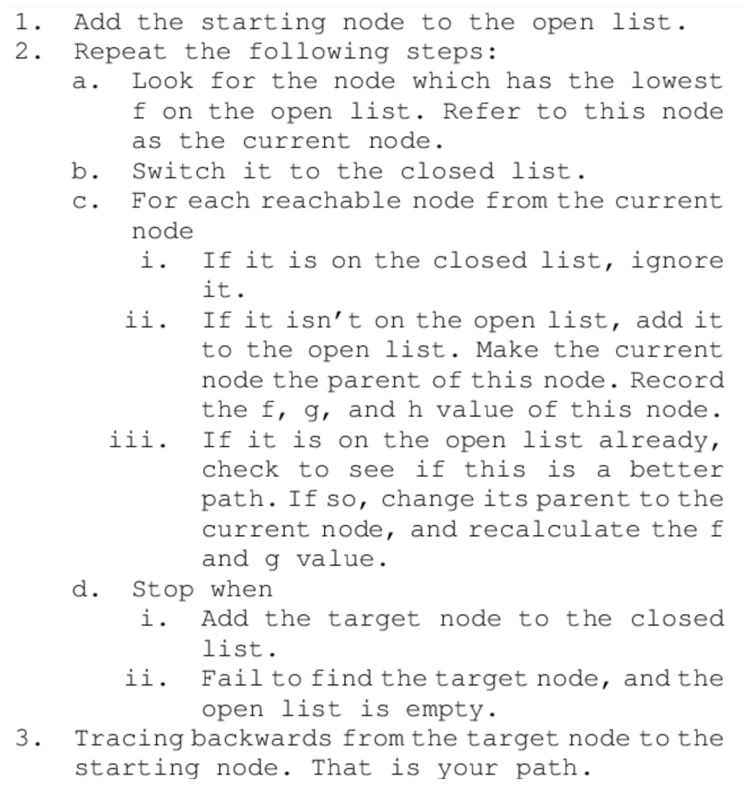
Dijkstra's algorithm aims to find the shortest paths between nodes in a graph. According to the research paper "Customizable Route Planning" written by Daniel Delling, Andrew V. Goldberg, Thomas Pajor, and Renato F. Werneck, they mention that Dijkstra's algorithm still exist weakness and it can be modified. They brought up a new system that allows real-time queries, which has fast customization and keeps very little data for each metric. Compared to the previous technique, their approach implements PCD (Precomputed Cluster Distances) in their basic algorithm and starting with bidirectional Dijkstra's algorithm searches restricted to the source and the goal. Using ACT preprocessing to pick X vertices as landmarks, and stores distances between these landmarks and all vertices in the graph to guide the search towards the goal.[3] All in all, their algorithm speedups the original Dijkstra's algorithm to a new peak.

The differential evolution algorithm (DE) is a kind of approach to find the optimal path. According to the academic journal "An Improved Differential Evolution Algorithm for TSP Problem", the authors provide an enhanced algorithm to solve this problem. Their algorithm improves the convergence speed and optimal quality by regulating the integer sequence to the mutation process instead of using the original crossover operator. The procedures to build this algorithm are creating an initial position by using greedy algorithms, modifying the DE algorithm by regulating integer sequence, implementing Liuhai's crossover algorithm to generate the best solution. [6]The algorithm has a faster convergence rate and can get an optimal solution or approximate optimal solution.

Moreover, the heuristics search can be used for route planning. According to the journal

”Evacuation Route Planning: Scalable Heuristics”, the authors Sangho Kim, Betsy George, Shashi Shekhar presents innovative heuristics scalable to solve route planning problems with very large system. They brought up the concept called ”Intelligence road”, which aims to find the optimal route rather than optimal ”schedule”.[7] The Intelligence road heuristic(ILR) applied to the CCRP algorithm, they modified CCRP by adding some features to implement load reduction formula. They mention that compared with CCRP, ILR heuristic makes more efficient than CCRP because it can skip many iterations that do not produce new routes. This is why the ILR heuristic algorithm can involve more complex network systems.

4 A* Algorithm and Code Analysis



```
1.  Add the starting node to the open list.
2.  Repeat the following steps:
    a.  Look for the node which has the lowest
        f on the open list. Refer to this node
        as the current node.
    b.  Switch it to the closed list.
    c.  For each reachable node from the current
        node
          i.  If it is on the closed list, ignore
              it.
         ii.  If it isn’t on the open list, add it
              to the open list. Make the current
              node the parent of this node. Record
              the f, g, and h value of this node.
        iii.  If it is on the open list already,
              check to see if this is a better
              path. If so, change its parent to the
              current node, and recalculate the f
              and g value.
    d.  Stop when
          i.  Add the target node to the closed
              list.
         ii.  Fail to find the target node, and the
              open list is empty.
3.  Tracing backwards from the target node to the
    starting node. That is your path.
```

Figure 1: A * Pseudocode

The project is to solve the problem that find a shortest route between two location among all nodes on the map. In simplified situation will be discussed in project, A* algorithm, or called A* search is more easy to handle and it is efficient. It evaluates node with total cost, which denoted f (n). What decide f are costs which are denoted g (n), the cost to reach the node, and h (n), the cost to get from the node to the goal. In mathematics, equation is: f (n) = h (n) + g (n). During selection of nodes, f(n) is supposed to be estimated cost of the cheapest solution through node n. Different from uniform-cost-search which only consider value of g, A* search is both of complete and optimal.

Figure shown above contains pseudocode and main idea which is way to solve route planning with A* algorithm. From start node which is also current node, all of its neighbor node will be put in open list for choosing. Absolutely, algorithm will choose the node with lowest f value, set it to new current node. And it will be put in close list. Then denote last node as parent node, which is for make connect between nodes on solution. Repeat these steps until target node appear in close list. If open list was empty, which means there is no more choice for solution. Algorithm will fail to find a solution. Algorithm will record g and h value of all node in open list and go back to check if there is any node in open list will be better than current node. When algorithm gets the path to target node, solution is found. Algorithm will backtrack to start node to label all nodes on route by using parent nodes we set during searching.

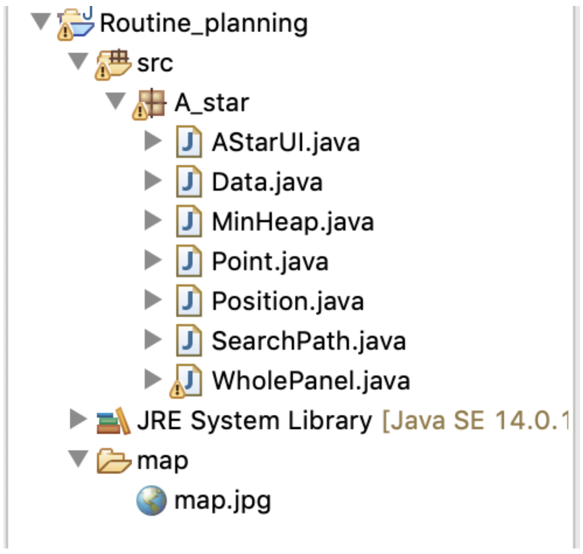

```
Routine_planning
  src
    A_star
      AStarUI.java
      Data.java
      MinHeap.java
      Point.java
      Position.java
      SearchPath.java
      WholePanel.java
  JRE System Library [Java SE 14.0.1
  map
    map.jpg
```

Figure 2: Source Codes

The figure 2 shows the main source codes in the file. In the AStarUI.java file, we import JFrame from the java library to pack the other java files. Data.java, Point.java, position.java, Search Path are Object-Oriented Programming files. The main structures and algorithms wrote in the WholePanel.java file. MinHeap.java is a helper function, which stores all possible nodes. Map.jpg is at the university map on the google maps.

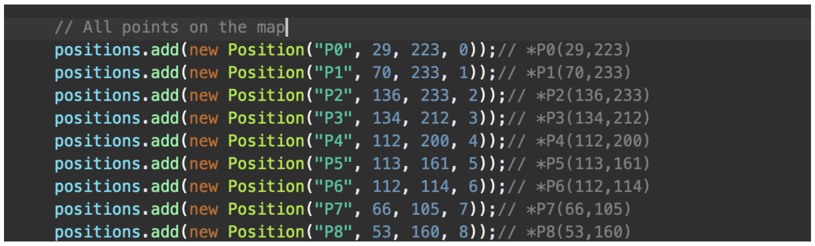
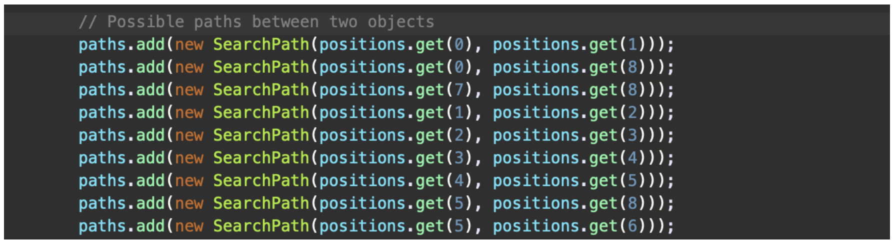

```
    // All points on the map
    positions.add(new Position("P0", 29, 223, 0));// *P0(29,223)
    positions.add(new Position("P1", 70, 233, 1));// *P1(70,233)
    positions.add(new Position("P2", 136, 233, 2));// *P2(136,233)
    positions.add(new Position("P3", 134, 212, 3));// *P3(134,212)
    positions.add(new Position("P4", 112, 200, 4));// *P4(112,200)
    positions.add(new Position("P5", 113, 161, 5));// *P5(113,161)
    positions.add(new Position("P6", 112, 114, 6));// *P6(112,114)
    positions.add(new Position("P7", 66, 105, 7));// *P7(66,105)
    positions.add(new Position("P8", 53, 160, 8));// *P8(53,160)
```

Figure 3: Example for adding coordinate points on the map

The strategies of designing this software are: Marking several points nearby "Northrop" in the university campus map and give them coordinate points. The Figure 3 shows the example that adding the coordinate points on the campus map.



```
// Possible paths between two objects
paths.add(new SearchPath(positions.get(0), positions.get(1)));
paths.add(new SearchPath(positions.get(0), positions.get(8)));
paths.add(new SearchPath(positions.get(7), positions.get(8)));
paths.add(new SearchPath(positions.get(1), positions.get(2)));
paths.add(new SearchPath(positions.get(2), positions.get(3)));
paths.add(new SearchPath(positions.get(3), positions.get(4)));
paths.add(new SearchPath(positions.get(4), positions.get(5)));
paths.add(new SearchPath(positions.get(5), positions.get(8)));
paths.add(new SearchPath(positions.get(5), positions.get(6)));
```

Figure 4: Possible Paths

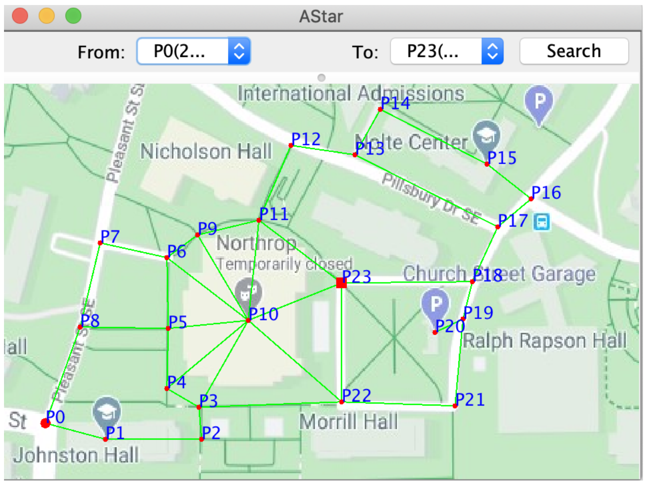
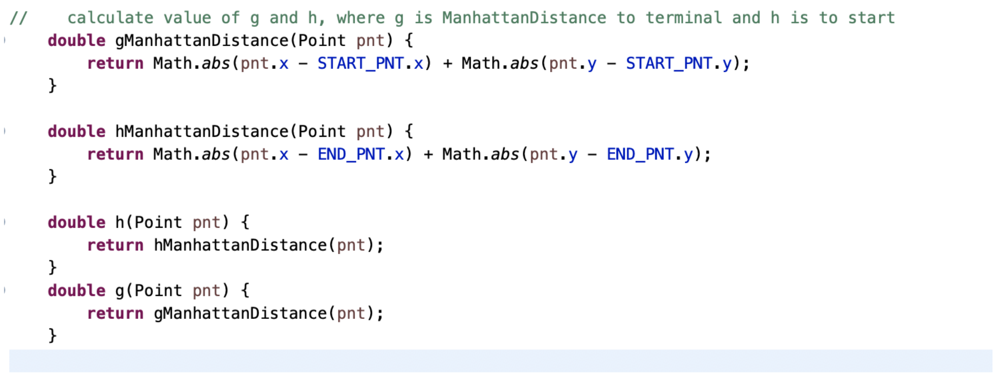


Figure 5: Output Actions

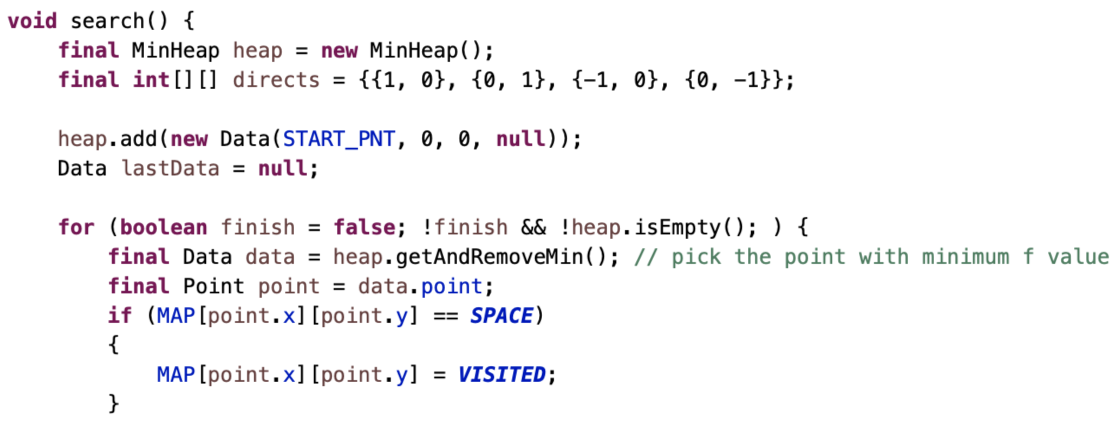
Then, searching all the possible paths in the each position, and add them in the paths (Figure 4). And the Figure 5 shows the map after marking the points and their possible paths.



```java
//    calculate value of g and h, where g is ManhattanDistance to terminal and h is to start
    double gManhattanDistance(Point pnt) {
        return Math.abs(pnt.x - START_PNT.x) + Math.abs(pnt.y - START_PNT.y);
    }

    double hManhattanDistance(Point pnt) {
        return Math.abs(pnt.x - END_PNT.x) + Math.abs(pnt.y - END_PNT.y);
    }

    double h(Point pnt) {
        return hManhattanDistance(pnt);
    }
    double g(Point pnt) {
        return gManhattanDistance(pnt);
    }
```

Figure 6: Manhattan Distance

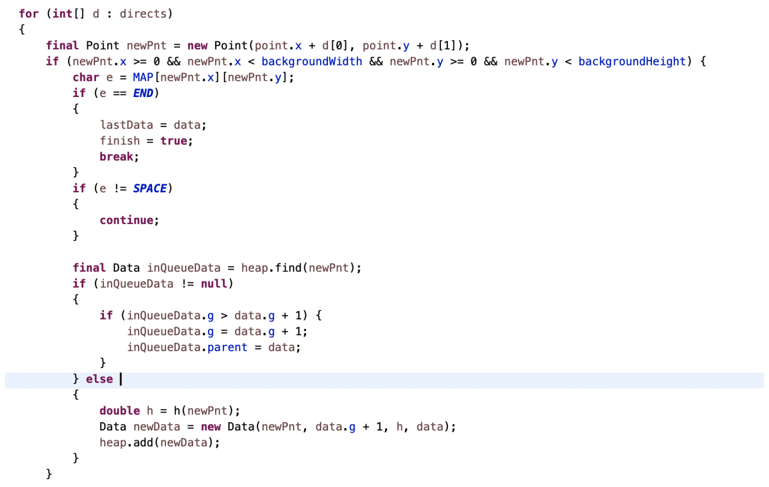
In the figure 6, we use the Manhattan Distance to get the g and h value, according to the A* algorithm, we were looking for the path, which has minimum f value (f = g + h). These two "Manhattan Distance" functions are to calculate the length between current position to the starting position or ending position.



```java
void search() {
    final MinHeap heap = new MinHeap();
    final int[][] directs = {{1, 0}, {0, 1}, {-1, 0}, {0, -1}};

    heap.add(new Data(START_PNT, 0, 0, null));
    Data lastData = null;

    for (boolean finish = false; !finish && !heap.isEmpty(); ) {
        final Data data = heap.getAndRemoveMin(); // pick the point with minimum f value
        final Point point = data.point;
        if (MAP[point.x][point.y] == SPACE)
        {
            MAP[point.x][point.y] = VISITED;
        }
```

Figure 7: Search algorithm 1

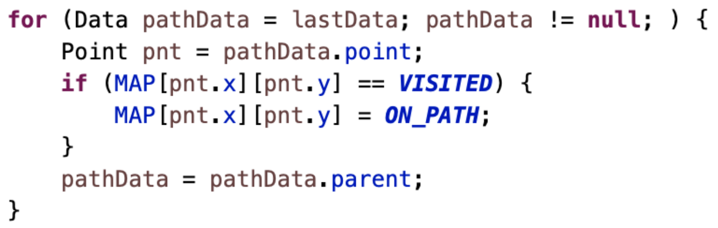
We defined four unit vectors to represent the direction to expand, and add the starting point in the min-heap.(Figure 7) As I mentioned above the min-heap file is a helper function to return the node, which has minimum f value (the purpose of A*). In the for loop, we recorded the node that we visited once to avoid repetition. The iteration terminates when there were no nodes in the min-heap.



```java
for (int[] d : directs)
{
    final Point newPnt = new Point(point.x + d[0], point.y + d[1]);
    if (newPnt.x >= 0 && newPnt.x < backgroundWidth && newPnt.y >= 0 && newPnt.y < backgroundHeight) {
        char e = MAP[newPnt.x][newPnt.y];
        if (e == END)
        {
            lastData = data;
            finish = true;
            break;
        }
        if (e != SPACE)
        {
            continue;
        }

        final Data inQueueData = heap.find(newPnt);
        if (inQueueData != null)
        {
            if (inQueueData.g > data.g + 1) {
                inQueueData.g = data.g + 1;
                inQueueData.parent = data;
            }
        } else 
        {
            double h = h(newPnt);
            Data newData = new Data(newPnt, data.g + 1, h, data);
            heap.add(newData);
        }
    }
```

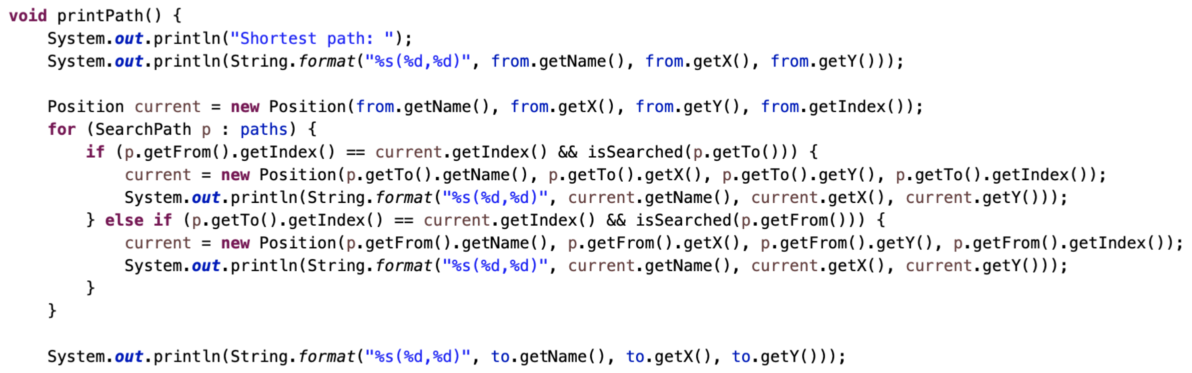
Figure 8: Search algorithm 2

In the figure 8, we iterated four directions in the inner loop, if the nodes is the end then break the loop, and to determine whether expand or not (only the node is SPACE).If the new point has already in the heap then keep updating the g value, otherwise add it to the min-heap.



```java
for (Data pathData = lastData; pathData != null; ) {
    Point pnt = pathData.point;
    if (MAP[pnt.x][pnt.y] == VISITED) {
        MAP[pnt.x][pnt.y] = ON_PATH;
    }
    pathData = pathData.parent;
}
```

Figure 9: Search algorithm 3



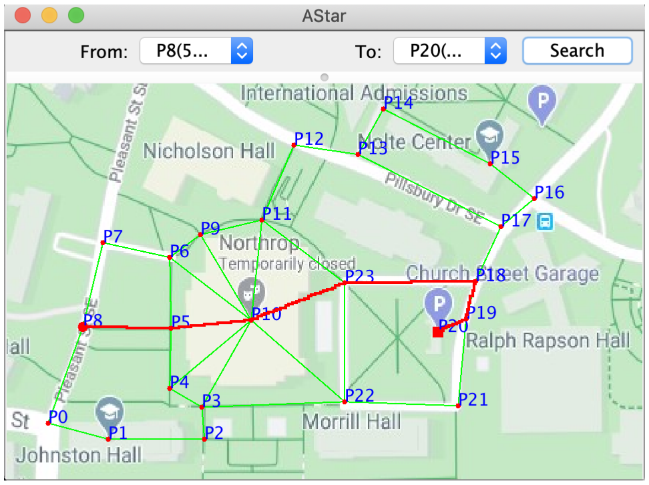
```java
void printPath() {
    System.out.println("Shortest path: ");
    System.out.println(String.format("%s(%d,%d)", from.getName(), from.getX(), from.getY()));

    Position current = new Position(from.getName(), from.getX(), from.getY(), from.getIndex());
    for (SearchPath p : paths) {
        if (p.getFrom().getIndex() == current.getIndex() && isSearched(p.getTo())) {
            current = new Position(p.getTo().getName(), p.getTo().getX(), p.getTo().getY(), p.getTo().getIndex());
            System.out.println(String.format("%s(%d,%d)", current.getName(), current.getX(), current.getY()));
        } else if (p.getTo().getIndex() == current.getIndex() && isSearched(p.getFrom())) {
            current = new Position(p.getFrom().getName(), p.getFrom().getX(), p.getFrom().getY(), p.getFrom().getIndex());
            System.out.println(String.format("%s(%d,%d)", current.getName(), current.getX(), current.getY()));
        }
    }

    System.out.println(String.format("%s(%d,%d)", to.getName(), to.getX(), to.getY()));
```

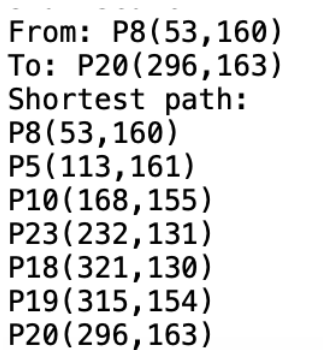
Figure 10: Search algorithm 3

Until expending the node to the end point, Figure 9 shows the loop to find the path by searching their parents. In the figure 10, printPath () function, printing the past nodes from starting point to the ending point, and showing their coordinate points.

5 Outputs

After testing a lot of examples, outputs are satisfied expectation. It could find shortage path between two nodes selected. Some different conditions shown below.

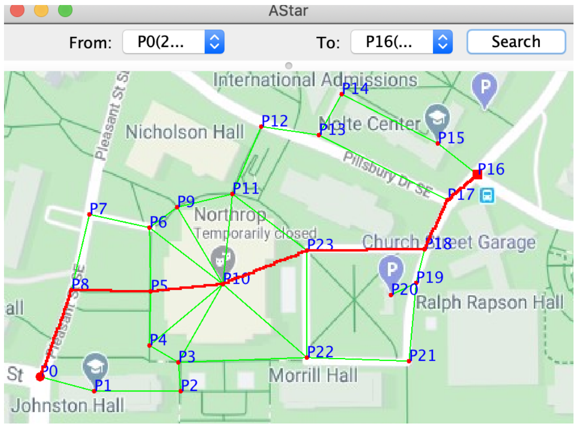




```
From: P8(53,160)
To: P20(296,163)
Shortest path:
P8(53,160)
P5(113,161)
P10(168,155)
P23(232,131)
P18(321,130)
P19(315,154)
P20(296,163)
```

Figure 11: Initial: p8; Goal: p20; Pathway

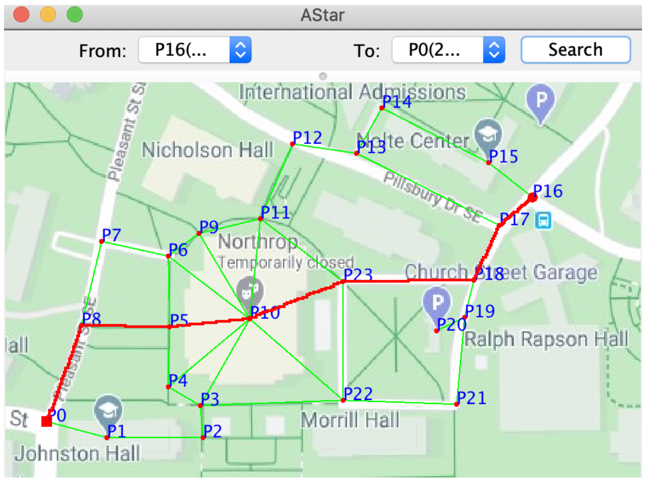
In first condition, random output shows route from P8 to P20. At beginning, P8 has neighbors which are P0, P7 and P5. It selects P5 because it is easy to see f value of P5 is smallest. Then algorithm selects P10 in same way. In selection between P23 and P22, algorithm chooses P23 because it has smaller f value. It is correct. With same horizontal distance to P20 and P8, P23 has smaller vertical distance to start and target nodes. F value is calculated in Manhattan distance which needs both of vertical and horizontal distances. Absolutely, P23 will be selected with smaller f value. And it is best solution even in reality. The route is perfect to expectation.



```
From: P0(29,223)
To: P16(361,76)
Shortest path:
P0(29,223)
P8(53,160)
P5(113,161)
P10(168,155)
P23(232,131)
P18(321,130)
P16(361,76)
```

Figure 12: Initial: p0; Goal: p16; Pathway

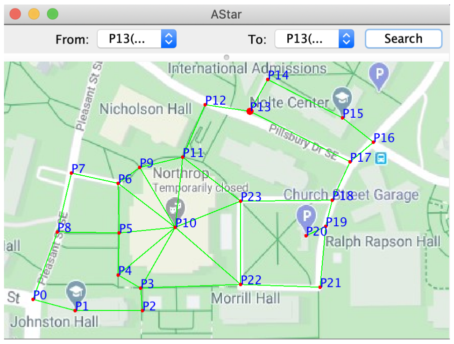
Next we choose to test the most complicated condition, from P0 to P16. Reason why choosing these two nodes is there are more choices during search because both of them located in cater-corner position on map. It is obvious that algorithm find best solution successfully.

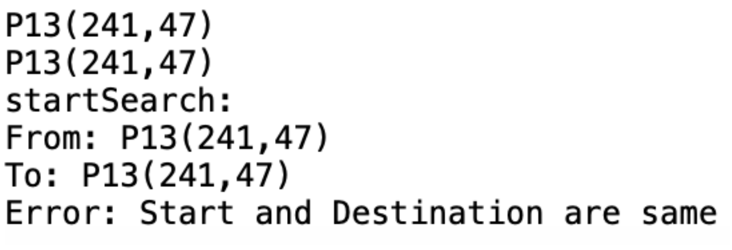


```
From: P16(361,76)
To: P0(29,223)
Shortest path:
P16(361,76)
P17(339,94)
P18(321,130)
P23(232,131)
P0(29,223)
```

Figure 13: Initial: p16; Goal: p0; Pathway

Based on condition 2, condition will test if outputs are same if swap start and target node. As results shown and route on map, it still complete task successfully.





```
P13(241,47)
P13(241,47)
startSearch:
From: P13(241,47)
To: P13(241,47)
Error: Start and Destination are same
```

Figure 14: Initial: p13; Goal: p13; Pathway

If start and target node are same, algorithm does not need to find route between them. It would be treated as an error because user sometimes did mistake when input address. Output shows no route on map and reminder user input error in text output.If start and target node are same, the algorithm does not need to find route between them. It would be treated as an error because user sometimes did mistake when input address. Output shows no route on map and reminder user input error in text output.

6 Conclusion

This paper introduces the solution that finding shortest (optimal) path in the University of Minnesota by using A* algorithm. Also, we designed the GUI for users to visualize the path in the map. We determined how accuracy about the solution by comparing theoretical optimal path, which given by our software and actual path in the maps. Route planning can be treated as a combinatorial optimization problem. To obtain the optimal solution we need to search all possible paths and calculate the cost of each them to reach the goal state. In addition, we Searched background information of solving shortest path problems, and modified and improved the A* algorithm in our software. Beside, no one can deny that the software still had bugs and we would like to improve them further, for example, one of the update we consider to add in the future is that calculate the time in the optimal path. Obviously, it is a very easy way for students make route plannings in the campus even in other maps. We believe that the development of the information technology simulates the productivity of human greatly and make people live more conveniently.

References

[1] Thinking with maps: Geospatial reasoning in war. 2010.

[2] Dorothea Wagner Daniel Delling. Time-dependent route planning. *Encyclopedia of GIS*, 2017.

[3] Thomas Pajor Renato F. Werneck Daniel Delling, Andrew V. Goldberg. Customizable route planning. *Experimental Algorithms Lecture Notes in Computer Science*, 2011.

[4] Anders Engberg-Pedersen. Empire of chance: The napoleonic wars and the disorder of things. 2015.

[5] Celso C. Ribeiro Mauricio G. C. Resende. Solution construction and greedy algorithms. *Optimization by GRASP*, 2016.

[6] Zhong Ming Gu Yu Mei Mi, Xue Huifeng. An improved differential evolution algorithm for tsp problem. *2010 International Conference on Intelligent Computation Technology and Automation*, 2010.

[7] Betsy George Sangho Kim. Evacuation route planning: scalable heuristics. 2007.

[8] Christopher E. White. Some map matching algorithms for personal navigation assistants. *Transportation Research Part C: Emerging Technologies"*, 2000.

[9] Gao Cong Xiaokui Xiao Xin Cao, Lisi Chen. Keyword-aware optimal route search. 2012.

[10] Eliyahu Safra Yaron Kanza, R. Levin. An interactive approach to route search. 2009.