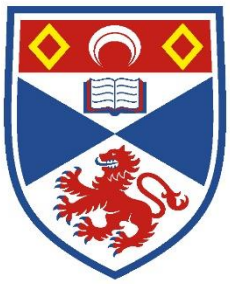
CS5011 Artificial Intelligence

Assignment 2: Search – Rescue Simulation



University of
St Andrews

Student ID: 170009479

Contents

Appendix1

A list of the parts implemented:

1. Part1:
   (1) Implement depth-first search (DFS), and breadth-first search (BFS).
   (2) Construct and the path from the starting point to Bob, to the goal.
   (3) Print the current node.
   (4) Print the list of states expanded.
   (5) Print the frontier at each step in the search.
   (6) Print the current solution details

2. Part2:
   (1) Implement best-first search (BestFS) and A* search.
   (2) Use three individual different heuristics to implement best-first search and A* search.
   (3) Combine any two of them with both best-first search and A* search.
   (4) Construct and the path from the starting point to Bob, to the goal.
   (5) Print the current node.
   (6) Print the list of states expanded.
   (7) Print the frontier at each step in the search.
   (8) Print the current solution details.

Appendix2

Full name	Abbreviation
depth-first search	DFS
breadth-first search	BFS
best-first search	BestFS

Search Rescue Simulations

1. Introduction

Today, there are many various types of computer games. Amongst, Tower Defence games, as a type of classic game, are still very popular. In this practical, I develop a simple Tower Defence game based on the provided UML class diagram (Game, Tower, Enemy, Catapult, Slingshot, Rat and Elephant). This report illustrates the process of developing this game that is consist of the design of basic class, the implementation of user interface enhancement and the result of this system.

2. Literature review

2.1. Introduction of search

Search is an important part of Artificial Intelligence (AI) and help to solve many AI problem. “Search is a universal problem-solving mechanism in AI. In many problems, sequence of steps required to solve is not known in advance but must be determined by systematic trial-and-error exploration of alternatives [1].” AI search algorithms mainly can be categorized by three classes: “single-agent path-finding problems, two-players games, and constraint-satisfaction problems [1].” Generally, searching means to find something. ‘’In computer science, searching techniques are strategies that look for solutions to a problem in a search space. The solutions or ‘goal states’ could sometimes be an object, a goal, a sub-goal or a path to the searched item [2].’’ Take finding car key for example, “the search goal is the car key and the search space is confined to the owner’s home. The car key can be located anywhere in the owner’s house [2].”

2.2. Introduction of 4 main searching algorithms

2.2.1. Breadth-First Search

“Breadth-first search (BFS) is an algorithm for traversing or searching tree or graph data structures. It starts at the tree root (or some arbitrary node of a graph, sometimes referred to as a search key') and explores the neighbor nodes first, before moving to the next level neighbours [3].” “It starts from the root node, explores the neighboring nodes first and moves towards the next level neighbors. It generates one tree at a time until the solution is found. It can be implemented using FIFO queue data structure. This method provides shortest path to the solution.If branching factor (average number of child nodes for a given node) = b and depth = d, then number of nodes at level d = bd.The total no of nodes created in worst case is b + b2 + b3 + ⋯ + bd.Disadvantage − Since each level of nodes is saved for creating next one, it consumes a lot of memory space. Space requirement to store nodes is exponential.Its complexity depends on the number of nodes. It can check duplicate node [4]s.”

2.2.2. Depth-First Search

“It is implemented in recursion with LIFO stack data structure. It creates the same set of nodes as Breadth-First method, only in the different order.As the nodes on the single path are stored in each iteration from root to leaf node, the space requirement to store nodes is linear. With branching factor b and depth as m, the storage space is bm [5].” The disadvantage of this algorithm may not terminate and continue to search infinitely on one branch of search tree.

To solve this problem，the method of setting a limit depth can be chose [5]. “If the ideal cut-off is d, and if chosen cut-off is lesser than d, then this algorithm may fail. If chosen cut-off is more than d, then execution time increases. Its complexity depends on the number of paths. It cannot check duplicate nodes [5].”

2.2.3. Best-first Search

“Best-first search is a search algorithm which explores a graph by expanding the most promising node chosen according to a specified rule.” [6]

Judea Pearl described best-first search as estimating the promise of node n by a "heuristic evaluation function f(n) which, in general, may depend on the description of n, the description of the goal, the information gathered by the search up to that point, and most important, on any extra knowledge about the problem domain." [7]

2.2.4. A* search

A* is a computer algorithm is broadly used to explore path and traverse graph [8], “the process of plotting an efficiently directed path between multiple points, called nodes [8].” It is widely believed that A* is more accurate to calculate estimated cost. Informally speaking, A* Search algorithms has “brains”, which means it is a much smarter algorithm than other conventional algorithms. Moreover this algorithm is widely used to explored the relative much shorter path very efficiently in many games and web-based maps [8].

3. Design

Since fundamental design elements including frontier and explored node list among BFS,DFS, BestFS and A* are same, the significant things that are needed to be considered are as following:

3.1. The choice of data types as frontiers for different algorithms

One of the most important keys of designing should be the choice of data type as a frontier for a specific algorithm. This because the mechanism of taking node out of frontiers based on different algorithms are different. There are four algorithms in this practical including BFS, DFS, BestFs and A*.

It is rational to use Queue as a frontier for BFS and Stack can be used as the frontier of DFS (the reason is given later in design part); For BestFs and A*, the common ground is that the node who has the minimum estimated cost is taken out preferentially although there is some differences in computing methods between them. Thus, Priority Queue in java can be efficiently in this situation.

In fact, other data type, for example can also be used in these four algorithm. However, to make codes concise, I chose use Queue, Stack and Priority Queue to represent them respectively.

3.2. The choice of order of checking if the child node can be added into frontier

When want to add an existent child node to a frontier (if the parent node on the edge of map, the amount of child nodes is smaller than 4, which means if the value of horizontal axis of the assumptive child node or the value of vertical axis of the assumptive child node is smaller than 0, the assumptive child node is not a real or existent child node), in this practical there are other three things need to be checked. Firstly, check if the position of the child node is the position of one of obstacles. In addition, check if the child node has been in the frontier or explored node list. Personally, the order of last two checking things are worth to being evaluated.

In my checking sequence, I choose to check if the child Node is in the frontier firstly. The reason is that it is a more efficient checking way in this assignment. The number of nodes in frontier is smaller than the number of nodes in explored node list.

Although at beginning, the frontier size is slightly bigger than the size of explored node list, the number of node in explored node list become increasingly bigger and after first several steps the number of nodes keeps in low because of grids making the high repetition rate between child nodes and frontier or the high repletion rate between child nodes and explored node list. Hence, it takes much more time to check if the child Node is in the explored node list than to check if the child node is in the frontier when traversing. It is more cost-effective to weed out ineligible nodes at a lower cost through preferentially checking if the child Node is in the explored array list. It can improve efficiency of this programme. And also I improve the efficiency though using break to jump out of loop once there's any repetition being detected.

Besides, the way of checking as above are right. However, when use A*, if there is the same node as the current child node in frontier, the estimated cost of these two nodes need to be compare. If the estimated cost of the current child node is smaller than the cost of that same node in frontier, the current child node should be added in the frontier and that same node should be removed from the frontier.

3.3. The choice of setting parent node to child node

To get the final path, I use parent node. The method is to find the parent of node (start from goal) layer by layer until find the initial node. Every time, when I add a child node of current node into frontier, I also set the current node as the parent node of this child node. When the goal is detected, the path can be created by going back to find parent node.

3.4. The choices of the time of adding node into explored node list

When the current node is fetched from frontier, there are two choices of adding node into explored node in different time. It can be added into explored node list after checking its child nodes. If it is, instead of checking if there is same node as child node in frontier of explored node list, if the child node is the node of the current node's parent node also should be checked. If this new check step has not been done, the search can trap in infinite loop because the current node and its parent node are removed from frontier alternately.

3.5. Part1:

3.5.1. Describe the state space, initial state and goal, successor functions, and actions in this search problem, path cost, and details of their respective implementation.

State space: State space is used to describe all states of the problem. Here the problem is to find a proper path to arrive Bob and the goal. And state space also reflects the relationship between different states.

Initial state: In this problem, initial state is the state that initial node has not moved. It is the state representing the start of searching. Take map1 for example, initial state can be expressed by the coordinates of all elements in this map.

Goal: Goal is the result you want to get. Here goal means the initial node starting from initial position, finds path to the position of Bob and finds path from the position of Bob to the position of goal. Briefly, goal is the destination of searching.

Successor function: successor function is the function of making Initial node move from the current location to the next location after implementing action. The purpose of this function is to find the result (the next state) of implementing action from the current state.

Actions: in this problem, actions can be regarded as go different directions, namely go North, West, East and Cost.

Path cost: In this problem, path cost means the distance of starting from initial position of “I” to go to the position of “B” and from the position of “B” to the position of “G” based on created path after executing the programme.

3.5.2. Describe the implementation of the search strategy clearly, and the differences between DFS and BFS in your implementation.

(1) The implementation of the search strategy:

（1） start from initial state and regard it as current node

（2） add the current node into frontier

（3） remove the current node out of frontier

（4） add the current node into explored node queue

（5） implement successor function to find the children nodes of the current node.

（6） check if children nodes have not been in frontier and explored node list. If there is no same node, children nodes can be added in to frontier. (when judge if there is same node in frontier, in this problem, just check by using of position of node).

（7） Loop starts from (3) to (6), the loop finishes until finding goal by taking every node out of the frontier in a specific sequence or frontier becoming empty. When frontier becomes empty and no goal is founded, it means that the goal cannot be got to.

(2) the differences between DFS and BFS：

（1） Basically, for BFS, it explores the nodes at same level and explores deeper level until finishing the current lever. It extracts the node which is the first node has been sent into the frontier. Namely, it obeys “first in first out” principle. For DFS, it explores nodes deeper and deeper and change other branch until the current branch finishes. It extracts the node which is the last node has been sent into the frontier. It means that it obeys “first in last out” principle. Therefore, the biggest differences between them are using different types frontiers to represent different mechanisms of them to fetch nodes from a frontier.

3.5.3. Describe your choice(s) of order to add new states to the list.

In my programme , because of only “I” moving, I use the position of the current “I” to represent state. Thus, the order to add new states to the list is actually the order to expand children nodes to frontier because when every new node creates(can be seen as new “I” with new position), it carries its states. Besides the order is decided by moving directions.

There are four directions are allowed including North, East, South and West. The sequence of directions normally decides the sequence of checking if the child node can be added into the frontier, adding new node into frontier, extracting new node from frontier and adding new node into explored node. Therefore, the rational sequence directions can influence how much time is spent to explore goal;

In my programme, I choose the sequence from South to East to North to West. Comparing with the sequence from West to North to East to South, it probably spends lesser time to find goal when the initial node is in the Northwest of the grids and the goal is in the Southeast of the grids by using BFS algorithm. The time of spending on explored goal is also influenced by the positions of obstacles.

3.5.4. Describe how you tested your implementation.

(1) When running the programme, corresponding values include the position of the current node (i.e. state), the actions have been tried, the size of frontier and the size of explored node list are shown on console;

(2) Normally, at the beginning, I check first several loops step by step based on values shown on console and compare the logic of the programme and the logic of the algorithm to judge if the programme logic is right.

(3) By using these values, I can check the rationality of search state

(4) For example. According to moving directions(actions) to judge if the order of adding nodes to frontier is right and if the order of taking out nodes from frontier is right; according to the final path and the feature of the current algorithm to judge the searching procedure is right

(5) Evaluate BFS and DFS for this problem by running them on the data provided. Which algorithm is best in terms of the number of search states visited? Which algorithm produces the best (shortest) routes on the basis of path cost?

Table 1 Compare the number of search state visited

	BFS	DFS	Have solution(Y/N)
Map1	99	148	Y
Map2	147	60	Y
Map3	126	54	Y
Map4	80	80	N
Map5	115	88	N
Map6	79	92	Y

Table 2 compare path cost

	BFS	DFS	Have solution(Y/N)
Map1	19	57	Y
Map2	35	43	Y
Map3	22	48	Y
Map4	/	/	N
Map5	/	/	N
Map6	30	31	Y

3.6. Part 2:

3.6.1. Describe the heuristics chosen and the respective implementation.

In my program, I calculate tree heuristics, and the user can choose any one of them or any two of them to calculate heuristics. The respective implementation is based on formula.

Manhattan distance: $|x_A - x_B| + |y_A - y_B|$

Euclidian distance: $\sqrt{(x_A - x_B)^2 + (y_A - y_B)^2}$

Chebyshev distance: $max\,(|x_A - x_B|, |y_A - y_B|)$

Combining Heuristics: h(n) = max{h1(n), h2(n)}

3.6.2. Describe the search algorithms in your implementation.

Both of BestFS and A* need to estimate cost as a criteria of deciding which node should be fetched from frontier preferentially. However they have different formula to calculate estimated cost (f(n)is estimated cost of path from the state of n to goal, h(n) is heuristic and g(n) is the cost of the path from the start to n):

BestFS: f (n) = h(n);

A*: f (n) = g(n) + h(n)

According to this two formulas, The search algorithm of Best first search and A* are that choose the node with minimum f(n) from frontier, extract the node from frontier and add this node into explored node list.

3.6.3. Describe how you tested your implementation.

The answer is same as part2

3.6.4. Evaluate BestFS and A* for this problem by running them on the data provided. How do they compare to DFS and BFS in terms of number of states visited and path cost? Which of the algorithms and which of your heuristics is best in terms of the number of search states visited? Which algorithm produces the best (shortest) routes on the basis of path cost?

Note that, here heuristics is calculated in Manhattan distance.

Table 3 Table 3 Compare the number of search state visited

	BestFS	A*	Have solution(Y/N)
Map1	19	53	Y
Map2	45	93	Y
Map3	23	55	Y
Map4	80	80	N
Map5	85	94	N
Map6	34	62	Y

Table 4 compare path cost

	BestFS	A*	Have solution(Y/N)
Map1	19	19	Y
Map2	35	35	Y
Map3	22	21	Y
Map4	/		N
Map5	/		N
Map6	30	30	Y

4. Examples and Testing

There are four examples for four different algorithms. Every algorithm uses map1. Note that: there is no path in map4 because Bob is encompassed by obstacles. The results based on different algorithms are shown as following:

4.1. BFS:

4.1.1. Find Bob(from “I” to “B”):

The path for map from I to B is (positions are represented by two coordinates in Euclidean Space (x and y)):

(0,0) → (0,1) → (0,2) → (0,3) → (0,4) → (0,5) → (0,6) → (0,7) → (1,7) → (2,7) → (3,7) → (4,7) → (5,7) → (6,7) → (7,7) → (8,7)

the number of elements in path is: 16

Output explored node:

(0,0) → (0,1) → (1,0) → (0,2) → (1,1) → (2,0) → (0,3) → (1,2) → (2,1) → (3,0) → (0,4) → (1,3) → (2,2) → (3,1) → (4,0) → (0,5) → (1,4) → (2,3) → (3,2) → (4,1) → (5,0) → (0,6) → (1,5) → (2,4) → (3,3) → (4,2) → (5,1) → (6,0) → (0,7) → (1,6) → (2,5) → (4,3) → (5,2) → (6,1) → (7,0) → (0,8) → (1,7) → (3,5) → (4,4) → (6,2) → (7,1) → (8,0) → (0,9) → (1,8) → (2,7) → (4,5) → (5,4) → (6,3) → (7,2) → (8,1) → (9,0) → (1,9) → (2,8) → (3,7) → (4,6) → (5,5) → (6,4) → (7,3) → (8,2) → (9,1) → (2,9) → (3,8) → (4,7) → (6,5) → (7,4) → (9,2) → (3,9) → (4,8) → (5,7) → (6,6) → (7,5) → (8,4) → (9,3) → (4,9) → (5,8) → (6,7) → (7,6) → (8,5) → (9,4) → (5,9) → (6,8) → (7,7) → (8,6) → (9,5) → (6,9) → (7,8) → (8,7)

the number of elements in explored Node is: 87

4.1.2. Find the goal (from "B" to "G"):

The path for map is from B to G (positions are represented by two coordinates in Euclidean Space (x and y)):

(8,7) → (8,8) → (8,9) → (9,9)

the number of elements in path is: 4

Output explored node:

(8,7) → (8,8) → (9,7) → (8,6) → (7,7) → (8,9) → (9,8) → (7,8) → (9,6) → (8,5) → (7,6) → (6,7) → (9,9)

the number of elements in explored Node is: 13

4.2. DFS

4.2.1. Find Bob (from "I" to "B"):

The path for map from I to B is (positions are represented by two coordinates in Euclidean Space (x and y)):

(0,0) → (1,0) → (2,0) → (3,0) → (4,0) → (5,0) → (6,0) → (7,0) → (8,0) → (9,0) → (9,1) → (9,2) → (8,2) → (7,2) → (6,2) → (5,2) → (4,2) → (3,2) → (2,2) → (1,2) → (0,2) → (0,3) → (0,4) → (1,4) → (2,4) → (2,5) → (3,5) → (4,5) → (4,4) → (5,4) → (6,4) → (7,4) → (8,4) → (9,4) → (9,5) → (9,6) → (8,6) → (8,7)

the number of elements in path is: 38

Output explored node:

(0,0) → (1,0) → (2,0) → (3,0) → (4,0) → (5,0) → (6,0) → (7,0) → (8,0) → (9,0) → (9,1) → (9,2) → (8,2) → (7,2) → (6,2) → (5,2) → (4,2) → (3,2) → (2,2) → (1,2) → (0,2) → (0,3) → (0,4) → (1,4) → (2,4) → (2,5) → (3,5) → (4,5) → (4,4) → (5,4) → (6,4) → (7,4) → (8,4) → (9,4) → (9,5) → (9,6) → (8,6) → (7,6) → (6,6) → (6,7) → (5,7) → (4,7) → (3,7) → (2,7) → (1,7) → (0,7) → (0,6) → (0,8) → (0,9) → (1,9) → (2,9) → (3,9) → (4,9) → (5,9) → (6,9) → (7,9) → (7,8) → (8,8) → (9,8) → (9,9) → (8,9) → (1,6) → (1,8) → (2,8) → (3,8) → (4,8) → (5,8) → (6,8) → (7,7) → (8,7)

the number of elements in explored Node is: 70

4.2.2. Find Bob (from "B" to "G"):

The path for map is from B to G (positions are represented by two coordinates in Euclidean Space (x and y)):

(8,7) → (7,7) → (6,7) → (5,7) → (4,7) → (3,7) → (2,7) → (1,7) → (0,7) → (0,8) → (0,9) → (1,9) → (2,9) → (3,9) → (4,9) → (5,9) → (6,9) → (7,9) → (8,9) → (9,9)

the number of elements in path is: 20

Output explored node:

(8,7) → (7,7) → (6,7) → (5,7) → (4,7) → (3,7) → (2,7) → (1,7) → (0,7) → (0,6) → (0,5) → (0,4) → (0,3) → (0,2) → (0,1) → (0,0) → (1,0) → (2,0) → (3,0) → (4,0) → (5,0) → (6,0) → (7,0) → (8,0) → (9,0) → (9,1) → (9,2) → (8,2) → (7,2) → (6,2) → (5,2) → (4,2) → (3,2) → (2,2) → (2,3) → (2,4) → (2,5) → (3,5) → (4,5) → (4,4) → (5,4) → (6,4) → (7,4) → (8,4) → (9,4) → (9,5) → (9,6) → (8,5) → (7,5) → (6,5) → (5,5) → (3,3) → (4,3) → (6,3) → (7,3) → (9,3) → (8,1) → (7,1) → (6,1) → (5,1) → (4,1) → (3,1) → (2,1) → (1,1) → (1,2) → (1,3) → (1,4) → (1,5) → (0,8) → (0,9) → (1,9) → (2,9) → (3,9) → (4,9) → (5,9) → (6,9) → (7,9) → (8,9) → (9,9)

the number of elements in explored Node is: 79

4.3. BestFS

4.3.1. Find Bob (from "I" to "B"):

The routine for map from I to B is (positions are represented by two coordinates in Euclidean Space (x and y)):

(0,0) → (0,1) → (0,2) → (0,3) → (0,4) → (0,5) → (0,6) → (0,7) → (1,7) → (2,7) → (3,7) → (4,7) → (5,7) → (6,7) → (7,7) → (8,7)

the number of elements in path is: 16

Output explored node:

(0,0) → (0,1) → (0,2) → (0,3) → (0,4) → (0,5) → (0,6) → (0,7) → (1,7) → (2,7) → (3,7) → (4,7) → (5,7) → (6,7) → (7,7) → (8,7)

the number of elements in explored Node is: 16

4.3.2. Find the goal (from "B" to "G"):

The routine for map is from B to G (positions are represented by two coordinates in Euclidean Space (x and y)):

(8,7) → (8,8) → (8,9) → (9,9)

the number of elements in path is: 4

Output explored node:

(8,7) → (8,8) → (8,9) → (9,9)

the number of elements in explored Node is: 4

4.4. A*

4.4.1. Find the goal (from "I" to "B"):

The routine for map from I to B is (positions are represented by two coordinates in Euclidean Space (x and y)):

(0,0) → (0,1) → (1,1) → (2,1) → (2,2) → (2,3) → (3,3) → (4,3) → (4,4) → (5,4) → (5,5) → (6,5) → (6,6) → (6,7) → (7,7) → (8,7)

the number of elements in path is: 16

Output explored node:

(0,0) → (0,1) → (1,0) → (1,1) → (2,0) → (2,1) → (3,0) → (3,1) → (4,0) → (4,1) → (5,0) → (5,1) → (6,0) → (6,1) → (7,0) → (7,1) → (8,0) → (8,1) → (0,2) → (2,2) → (0,3) → (2,3) → (1,3) → (3,3) → (1,4) → (4,3) → (1,5) → (4,4) → (6,2) → (7,2) → (5,4) → (6,3) → (7,3) → (5,5) → (6,4) → (7,4) → (4,5) → (6,5) → (8,4) → (4,6) → (6,6) → (8,5) → (4,7) → (6,7) → (7,6) → (7,7) → (8,6) → (8,7)

the number of elements in explored Node is: 48

4.4.2. Find the goal (from “B” to “G”):

The routine for map is from B to G (positions are represented by two coordinates in Euclidean Space (x and y)):

(8,7) → (8,8) → (8,9) → (9,9)

the number of elements in path is: 4

Output explored node:

(8,7) → (8,8) → (9,7) → (8,9) → (9,8) → (9,9)

the number of elements in explored Node is: 6

5. Evaluation

(1) In this practical, I have not discussed the solution of if the initial state is the goal, I should add this situation in my programme.

(2) When running the program, the result shows two paths (from “I” to “B” and from “B” to “G”) separately. Therefore, only if two paths can be found, the whole path from I to B to G can be found successfully, namly find the goal sucessfully. If find successfully, the whole path size value should be calculated by add two paths’ size minus 1. If the first path (from “I” to “B”) can be found, but the second path(from “B” to “G”) cannot be found.

6. Running

The running steps is as follows:

- Part1:

  (1) Follow the instructions after running the part1 programme

  (2) There are two questions were asked to the user

  (3) The first instruction is to ask the user to choose an algorithm from BFS and DFS.

  (4) The second instruction is to ask the user to choose a map from 6 maps

- Part2:

  (1) Follow the instructions after running the part2 programme

  (2) There are three questions were asked to the user

  (3) The first instruction is to ask the user to choose an algorithm from BestFS and A*.

  (4) The second instruction is to ask the user to choose a map from 6 maps

  (5) The third instruction is to ask the user to choose amount and types of heuristics that is/are going to be used.

  Note that: To implement programmes properly, Please input valid value, such as pay attention to input capital letters and not to input out of the range.

7. Bibliography

[1] Kanal L, Kumar V, editors. Search in artificial intelligence. Springer Science & Business Media; 2012 Dec 6.

[2] Korf RE. Artificial intelligence search algorithms. Chapman & Hall/CRC; 2010 Jan 1.

[3] Murphy RC, Wheeler KB, Barrett BW, Ang JA. Introducing the graph 500. Cray Users Group (CUG). 2010 May 5.

[4] Yoo A, Chow E, Henderson K, McLendon W, Hendrickson B, Catalyurek U. A scalable distributed parallel breadth-first search algorithm on BlueGene/L. InSupercomputing, 2005. Proceedings of the ACM/IEEE SC 2005 conference 2005 Nov 12 (pp. 25-25). IEEE.

[5] zim T, Neamtiu I. Targeted and depth-first exploration for systematic testing of android apps. InAcm Sigplan Notices 2013 Oct 29 (Vol. 48, No. 10, pp. 641-660). ACM.

[6] Russell S, Norvig P, Intelligence A. A modern approach. Artificial Intelligence. Prentice-Hall, Egnlewood Cliffs. 1995; 25:27.

[7] Dechter R, Pearl J. Generalized best-first search strategies and the optimality of A. Journal of the ACM (JACM). 1985 Jul 1;32(3):505-36.

[8] Lerner J, Wagner D, Zweig K, editors. Algorithmics of large and complex networks: design, analysis, and simulation. Springer; 2009 Jun 29.