Shortest path using $\mathcal{A}^*$ Algorithm

Phaneendhar Reddy Vanam
Computer Science
Indiana State University
Terre Haute, IN, USA

December 13, 2011

Abstract

The main aim of this project is to find the shortest path using $\mathcal{A}^*$ **Algorithm.**

Contents

1 Introduction

The $\mathcal{A}^*$ Algorithm is a best-first search algorithm that finds the least cost path from an initial configuration to a final configuration.The most essential part of the $\mathcal{A}^*$ Algorithm is a good heuristic estimate function. This can improve the efficiency and performance of the algorithm.It is an extension of Dijkstra.s algorithm.$\mathcal{A}^*$ algorithm uses the function $f(n) = g(n) + h(n)$.

* $f(n) = g(n) + h(n)$.
* $g(n)$ is the path-cost function, which is the cost from the starting node to the current node.
* $h(n)$ is the heuristic estimate of the distance to the goal.

$\mathcal{A}^*$ Algorithm guides an optimal path to a goal if the heuristic function $h(n)$ is admissible.In this projct I am going to explain the algorithm and how this algorithm is going to implemented.

1.1 Applications

The Real time applications of $\mathcal{A}^*$ Algorithm are:

- $\mathcal{A}^*$ mainly used in Computer Gaming, Robotics and Google maps.
- The $\mathcal{A}^*$ for hueristic search is applied to construct a Neural Network structure(NS).

2 Statement of the problem

$\mathcal{A}^*$ depends on the heuristic.$\mathcal{A}^*$ is faster and give good results if we have a good heuristic.In this project I am going to path finding problems, that is, planning routes from a start node to some goal nodes in a graph.Such problems arise in many fields of technology, for example, production planning, message routing in large networks, resource allocation and vehicle navigation systems.I concentrate mostly on planning a minimum cost path using the $\mathcal{A}^*$ algorithm.

In some cases, $\mathcal{A}^*$ is an optimal method in a large class of algorithms.This means, roughly speaking that $\mathcal{A}^*$ explores a smaller region of the search space than the other algorithms in the given class.

A heuristic controls the search of $\mathcal{A}^*$ so that unnecessary branches of the tree of nodes that $\mathcal{A}^*$ visits are pruned.The new method also finds an optimal path to any node it visits for the first time so that every node will be visited only once.The later is an important property considering the efficiency of the search.

In some cases, the $\mathcal{A}^*$ is an optimal resource allocation method, which means that the number of the nodes the path finding algorithms together visit is minimized.

$\mathcal{A}^*$ gives the same results as Dijkstra, but faster when we use a good heuristic.$\mathcal{A}^*$ Algorithm has some conditions for to work correctly such as the estimated distance between current node and the final node should be lower than the real distance.$\mathcal{A}^*$ is guaranteed to give the shortest path when the heuristic is admissible.

3 History of $\mathcal{A}^*$ Algorithm

1. In 1964 Nils Nilsson invented a heuristic based approach to increase the speed of Dijkstra's algorithm.This algorithm was called A1.
2. In 1967 Bertram Raphael made dramatic improvements upon this algorithm, but failed to show optimality.He called this algorithm A2.
3. Then in 1968 Peter E.Hart introduced an argument that proved A2 was optimal when using a consistent heuristic with only minor changes.His proof of the algorithm also included a section that showed that the new A2 algorithm was the best algorithm possible given the conditions.
4. He thus named the new algorithm in Kleene star syntax to be the algorithm that starts with A and includes all possible version numbers or $\mathcal{A}^*$.

4 Steps involved in $\mathcal{A}^*$ algorithm

1. Let's characterize a class of admissible heuristic search strategies, using the evaluation function:$f(n) = g(n) + h(n)$.
2. $\mathcal{A}^*$ can be implemented more efficiently.roughly speaking, no node needs to be processed more than once.
3. As $\mathcal{A}^*$ traverses the graph, it follows a path of the lowest known cost, keeping a sorted priority queue of alternate path segments along the way.
4. If, at any point, a segment of the path being traversed has a higher cost than another encountered path segment, it abandons the higher-cost path segment and traverses the lower-cost path segment.
5. Starting with the initial node, it maintains a priority queue of nodes to be traversed, known as the open set.
6. The lower f(x) for a given node x, the higher its priority.

7. At each step of the $\mathcal{A}^*$ algorithm, the node with the lowest f(x) value is removed from the queue, the f and h values of its neighbors are updated accordingly, and these neighbors are added to the queue.

8. The $\mathcal{A}^*$ algorithm continues until a goal node has a lower f value than any node in the queue.

9. The f value of the goal is then the length of the shortest path, since h at the goal is zero in an admissible heuristic.If the actual shortest path is desired, the algorithm may also update each neighbor with its immediate predecessor in the best path found.

10. A closed set of nodes already traversed may be used to make the search more efficient.This process continues until the goal is reached.

Values of Heuristic i.e straight line distance to Bucharest

This is the example for A* search, I am going to consider the map of Romania and going to give step by step expalnation.

These are the hueristic values i.e straight line distance to Bucharest.

City	Hueristic value	City	Heuristic value
Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Eforie	161	Pitesti	100
Fagaras	176	Rimnicu Vilcea	193
Dobreta	242	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374

5 Diagrams representing stages in $\mathcal{A}^*$ search [1]

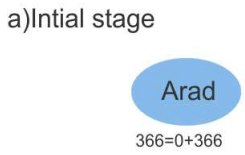

Figure 1: Example

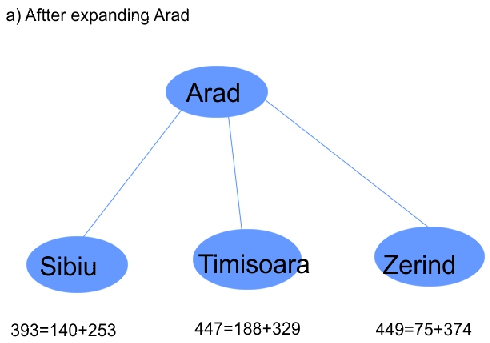

Figure 2: Example2

6 Algorithm

1. Put the start node **bs** in to **OPEN**
2. IF **OPEN** is empty THEN exit with failure.
3. Remove from **OPEN** and place in **CLOSED** a node **n** for which **f** is mimimum.
4. IF **n** is a goal node THEN exit sucessfully with the solution obtained by tracking back the pointers from **n** to **s**.
5. ElSE expand **n**, generating all its succesors,and attach them pointers back to **n**.

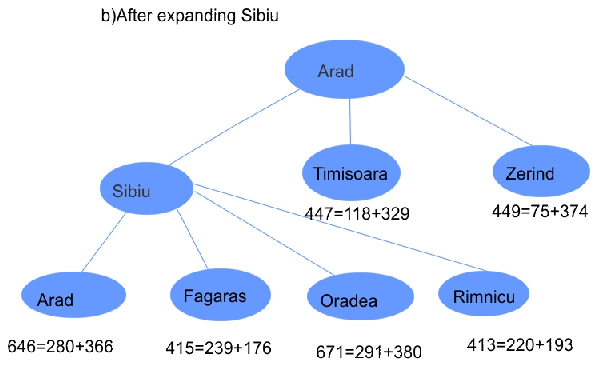

Figure 3: Example3

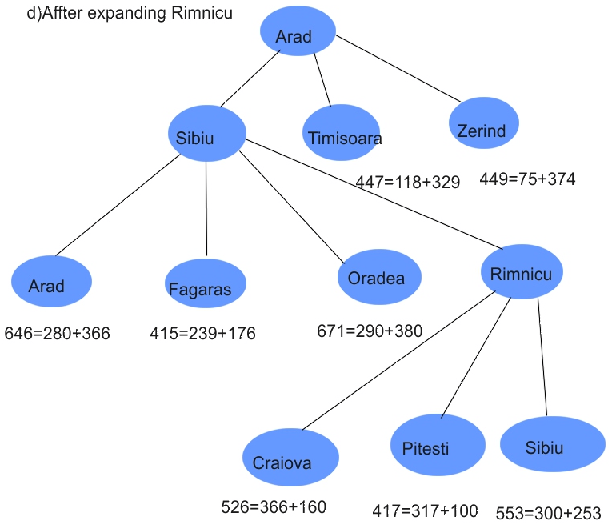

Figure 4: Example4

6. FOR every succesor **n'** of **n** DO IF **n** is not already in **OPEN** or **CLOSED** THEN estimate

$$h(n'),$$

and calculate

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

where

$$g(n') = g(n) + c(n, n')$$

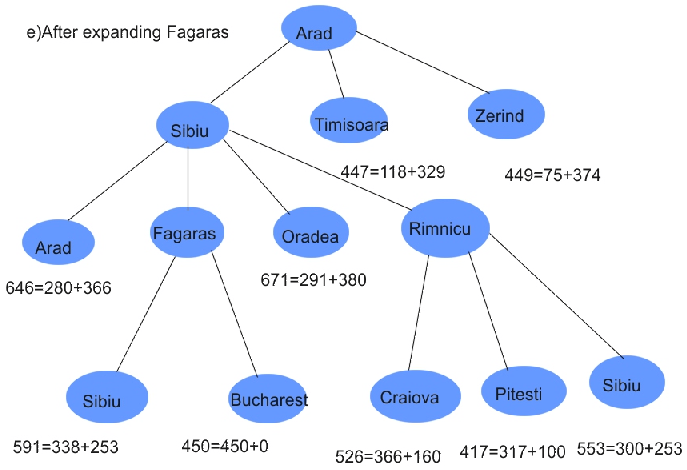

Figure 5: Example5

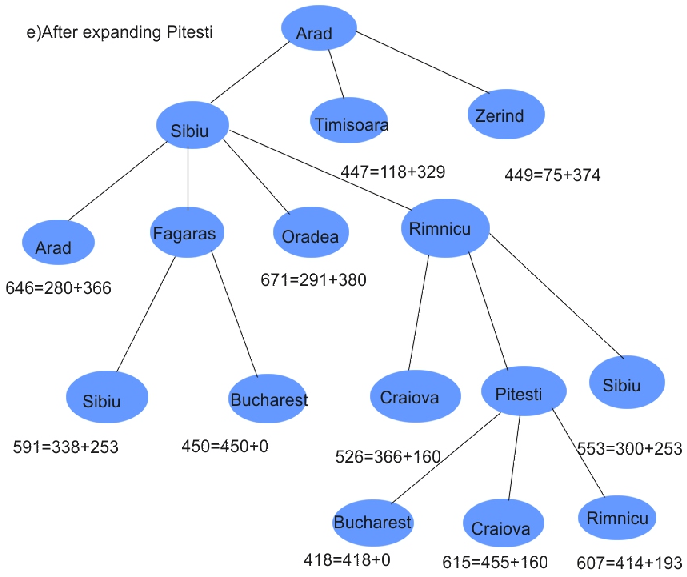

Figure 6: Example6

and put **n'** in to **OPEN**

7. IF **n** is already in **OPEN** or **CLOSED** THEN direct its pointer along the path yielding the lowest

$$g(n)$$

END FOR

8. GO TO step 2

7 Diagrams representing shortest path in Map of Romania [1]

$\mathcal{A}^*$ search in map of Romania

- This figure represents the intial map of Romania.The values representing in red colour are heuristic values(i.e h(n)).
- The values representing in silver colour are path cost values(i.e g(n)).
- The values representing in blue colour are f(n) values i.e

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

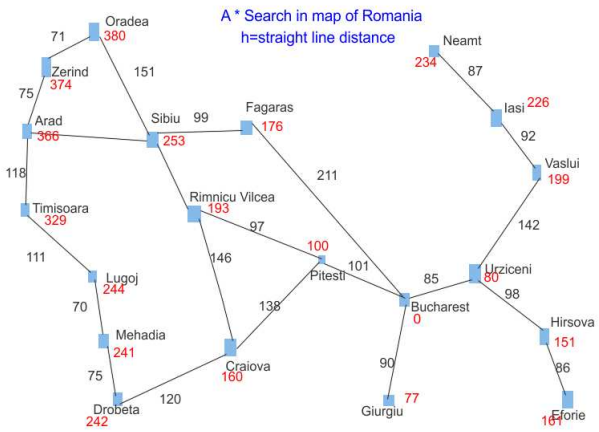

Figure 7: Intial map of Romania

7.1 After expanding Arad

- We have three nodes i.e Zerind, Sibiu and Timisoara.
- As we know f(n)=g(n)+h(n).
- f(Sibiu)=f(n)=140+253=393(g(n)=140 and h(n)=253).
- f(Zerind)=f(n)=75+374=449(g(n)=75 and h(n)=374).
- f(Timisoara)=f(n)=118+329=447(g(n)=118 and h(n)=329).
- From these nodes we have to choose the least f(n) value, so f(Sibiu) is least among these nodes.(Refer example 2.)

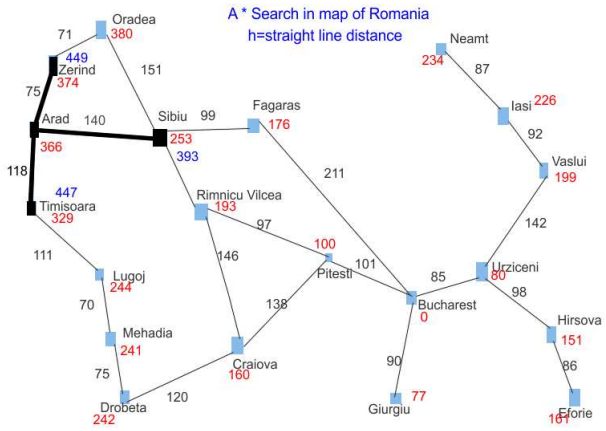

Figure 8: After expanding Arad

7.2 After expanding Sibiu

- After expanding Sibiu we have four nodes i.e Arad, Oradea, Fagaras and Rimnicu Vilcea.
- As we know f(n)=g(n)+h(n).
- f(Arad)=f(n)=280+366=646(g(n)=280 and h(n)=366).
- f(Oradea)=f(n)=291+380=671(g(n)=291 and h(n)=380).
- f(Fagaras)=f(n)=239+176=415(g(n)=239 and h(n)=176).
- f(Rimnicu)=f(n)=220+193=413(g(n)=220 and h(n)=193).
- From these nodes we have to choose the least f(n) value, so f(Rimnicu) is least among these nodes, f(Rimnicu)=413.(Refer example 3).

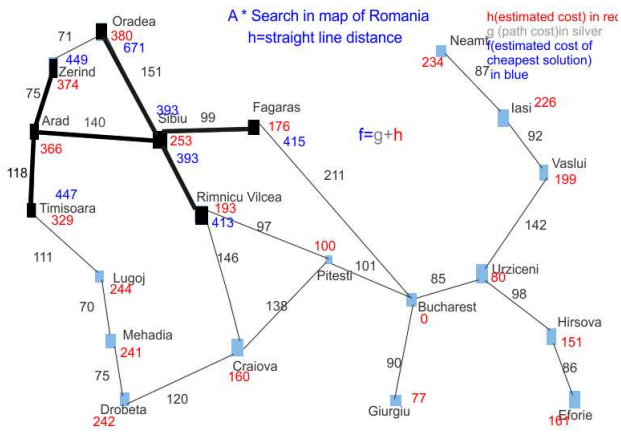

Figure 9: After expanding Sibiu

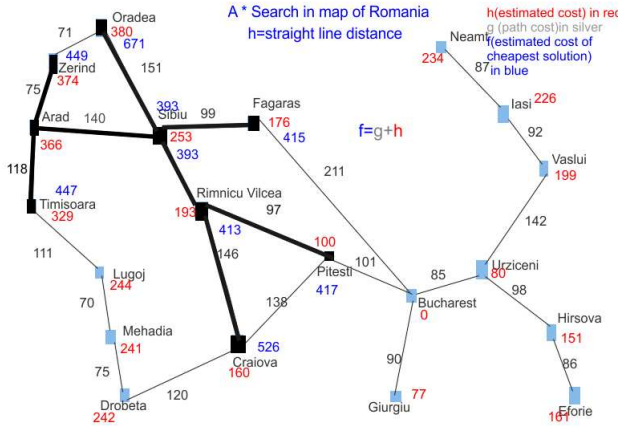

Figure 10: After expanding Rimnicu Vilcea

7.3 After expanding Rimnicu

- After expanding Rimnicu node we have three nodes i.e Craiova, Pitesti and Sibiu.
- As we know f(n)=g(n)+h(n).
- f(Craiova)=f(n)=366+160=526(g(n)=366 and h(n)=160).
- f(Pitesti)=f(n)=317+100=417(g(n)=317 and h(n)=100).
- f(Sibiu)=f(n)=300+253=553(g(n)=300 and h(n)=253).

- From these three nodes we have to choose the least f(n) value, so f(Fagaras) is least among the nodes(Refer exapmle 4.)

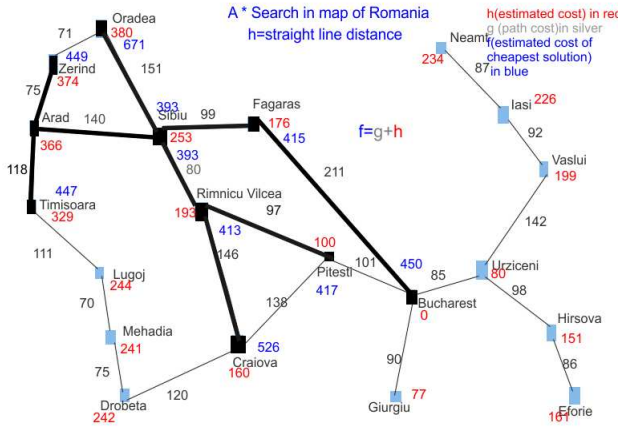

Figure 11: After expanding Fagaras

7.4 After expanding Fagaras

- After expanding Fagaras node we have two nodes i.e Sibiu and Bucharest.
- As we know f(n)=g(n)+h(n)..
- f(Sibiu)=f(n)=338+253=591(g(n)=338 and h(n)=253).
- f(Buchrest)=f(n)=450+0=450(g(n)=450 and h(n)=0).
- From these three nodes we have to choose the least f(n) value, so f(Fagaras) is least among the nodes(Refer exapmle 5.)

7.5 After expanding Pitesti

- After expanding Pitesti node we have three nodes i.e Bucharest, Craiova and Rimnicu.
- As we know f(n)=g(n)+h(n).
- f(Craiova)=f(n)=455+160=615(g(n)=455 and h(n)=160).
- f(Buchrest)=f(n)=418+0=418(g(n)=418 and h(n)=0).
- f(Rimnicu)=f(n)=414+193=607(g(n)=414 and h(n)=193).

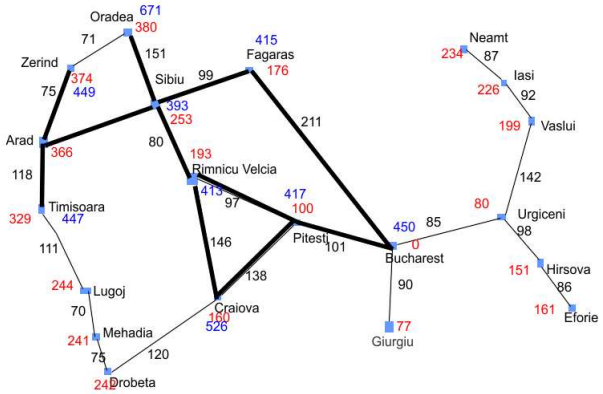

Figure 12: After expanding Pitesti

- From these three nodes we have to choose the least f(n) value, so f(Bucharest) is least among the nodes(Refer exapmle 6.)
- Since Bucharest is the goal node, so h(n)=0 at Bucharest.

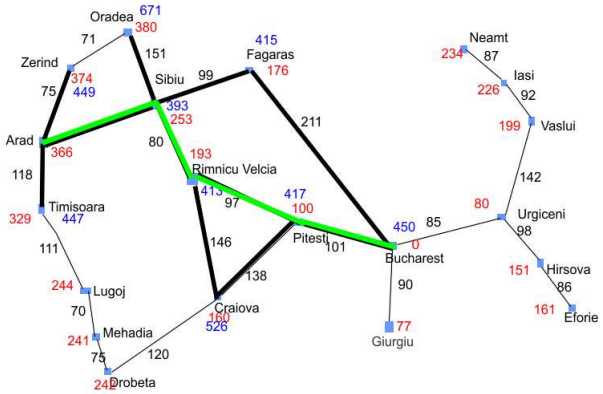

Figure 13: Shortest path from Arad to Bucharest

8 Time complexity

The time complexity of $\mathcal{A}^*$ depends on the heuristic.In the worst case, the number of nodes expanded is exponential in the length of the solution.

$$|h(x) - h^*(x)| = O(log(h)^*(x))$$

where h^* is the optimal heuristic and it is also defined as true cost of getting from n to the goal.For almost all heuristics in practical use, the error is at least proportional to the path cost, and the resulting exponential growth eventually.

8.1 Proving $\mathcal{A}^*$ is optimality

$\mathcal{A}^*$ is the name given to the algorithm where h(node) function is admissible.In the words it is guaranteed to provide underestimate of the true cost to the goal. $\mathcal{A}^*$ is optimal and complete.In the other words, It is guaranteed to find a solution, and the solution is guaranteed to be the best solution. $\mathcal{A}^*$ is complete if the graph it is searching is locally finite.

Proof of $\mathcal{A}^*$ is complete:

Conside two Goals G1 and G2.

The path cost of G1 is f1.

The path cost of G2 is f2, where

$$f2 > f1$$

G1 is the goal with lower cost.Let us assume A* algorithm reached G2 without exploring G1.Let us conside node n, that is an optimal path from root node to G1, the h is admissible heuristics.

$$f1 >= f1(n).$$

the only reason algorithm would not choose to expand n before it reaches G2 would be

$$f(n) >= f(G2).$$

By combining above 2 equations we get

$$f1 >= f(G2)$$

since G2 is goal state

$$h(G2) = 0$$

and thus

$$f(G2) = g(G2)$$

Thus we have

$$f1 >= g(G2)$$

. This, there fore contradicts our orginal assumption that G2 had a higher path cost than G1, Which proves that $\mathcal{A}^*$ can only choose least cost path to a goal.

8.2 Proving $\mathcal{A}^*$ is complete

As said before $\mathcal{A}^*$ expands in incresong f, it must expand to reach a goal state.This is true, of course, unless there are infinetely many nodes with

$$f(n) < f*$$

the only way there could be infinite number of nodes is either there is a node with an infinite branching factor or there is a path with finite path cost but an infinite number of nodes along it.Thus the correct statement is that $\mathcal{A}^*$ is complete on locally finite graphs.

d	IDS	A*(h1)	A*(h2)
2	2.45	1.79	1.79
4	2.87	1.48	1.45
6	2.73	1.34	1.30
8	2.80	1.33	1.24
10	2.79	1.38	1.22
12	2.78	1.42	1.24

Comparing effective branching factor of $\mathcal{A}^*$ algorithm with h1/h1.

8.3 Heuristic accuracy on perfromance

Consider the 8-puzzle problem.let h1=number of tiles that are in wrong position.Generally it has none of the tiles in the goal position, so the start state would be have h1=8, h2=the sum of the distances of tiles from their goal positions.Because tiles cannot move along diagonals, the distance will count is the sum of the horizontal and vertical distances.This is called manhattan distance.

d	IDS	A*(h1)	A*(h2)
2	10	6	6
4	112	13	12
6	680	20	18
8	6384	39	25
10	47127	93	39
12	364404	227	73

Comparing search cost of $\mathcal{A}^*$ algorithm with h1/h2

we can charcterize the quality of a heuristic is the effective branching factor b*.if the total nodes expanded by $\mathcal{A}^*$ for a particular problem is N and the solution depth is d, the b* is the branching factor that a uniform tree of depth d would have in order to contain N nodes.

$$N = 1 + b^* + (b^*)^2 + (b^*)^3 + ...(b*)^d.$$

A well designed heuristic would have a value of

$$b^* = 1$$

To test the heuristic functions h1 and h2, I randomly generated 100 problems each with solution lenth 2,4.....12 and solved them using $\mathcal{A}^*$ search with h1 and h2.This shows how h2 is better than h1, The above statement is always true for any node n.

$$h2(n) > h1(n)$$

$\mathcal{A}^*$ using h2 will expands fewer nodes tha A* using h1.

References

[1] Stuart Russell,Peter Norvig *Artificial intelligence a modern approach, Second Edition*

[2] David L.Poole and Alan K.Mackworth *Foundations of computational agents* Cambridge University Press, 2010

[3] Ben Coppin *Artificial Intelligence Illuminated*

[4] wikipedia A* search algorithm http://en.wikipedia.org/wiki/A*_algorithm

[5] A* algorithm for beginners http://www.policyalmanac.org/games/aStarTutorial.htm