OPTIMAL PATH PLANNING USING AN IMPROVED A* ALGORITHM FOR HOMELAND SECURITY APPLICATIONS

Microsystems and Machine Vision Laboratory
Materials and Engineering Research Institute
Faculty of Arts, Computing, Engineering and Sciences
Sheffield Hallam University

ABSTRACT
Path finding is an important sub task in mobile robotics and Homeland Security applications and has been subjected to extensive research. This paper analyses a variety of search algorithms used for path finding and planning. Using the Breve simulation environment, a general search algorithm and then the A* algorithm have been implemented. An improvement to the A* algorithm is introduced and presented. In this paper it has been proved through experimental results that the performance of the A* algorithm improves considerably after adding an additional heuristic. Dynamic path planning has also been implemented in this paper by allowing the vehicle to check for changes in the environment at every simulation time step and recalculate paths if there is a change in the environment. In the past ad-hoc sensor networks have been used in homeland security. In this paper ad-hoc sensor networks have been modeled using the patch class in the Breve simulation tool and path finding techniques have been used in this environment. Time and space complexity analysis of the various algorithms implemented in this paper have been presented.

KEY WORDS
Robotics, artificial intelligence, path planning, homeland security, search algorithms, sensor networks.

1. Introduction

The problem of dynamic collision free path planning is vital to mobile robots and robots used in homeland security applications. The area of homeland robotics has assumed a great importance in the present age and robots are now being used extensively to rescue survivors from dangerous environments and when dealing with hazardous substances. The robots are used for tasks that rescuers or canines are unable to perform, for example to either investigate in spaces that are too small for humans to pass through, or in areas consumed by fire with limited breathable air in an attempt to reach a survivable void.

The output of a search algorithm is either a failure or a solution. The efficiency of a search algorithm can be evaluated in four ways [1]

- *Completeness:* Is the algorithm guaranteed to find a solution when it exists?
- *Optimality:* Does the strategy find the optimal solution?
- *Time Complexity:* The time taken to find the solution
- *Space Complexity:* The memory needed to perform the search

1.1 Background

Search algorithms have been used extensively to solve various problems like the Queens problem, the dining philosophers problem etc. In this paper, we restrict ourselves to the use of search algorithms for path finding problems. Path finding techniques can be grouped into local methods and global methods [2]. One well known local planning method is the potential field method. In this method the robot follows the gradient of a force field. The field is generated by attractive potentials from a start position towards a target and by repulsive potentials that point away from obstacles. The potential field method has a low computational requirement. In contrast, global methods need complete information about the world and hence would require relatively large computational power.

The probabilistic roadmap which is a skeletonization method offers more possible routes and thus deals better with wide open spaces. The graph is created by randomly generating a large number of configurations, and discarding those that do not fall into free space. We then join any nodes by an arc if it is easy to reach one node from the other. Theoretically this method is incomplete, because a bad choice of random points may leave us without any paths from the start to the target [3].

1.2 Simulation

The Breve simulation environment was used to implement path finding techniques [5]. Breve is a free, open-source software package which makes it easy to build 3D simulations of decentralized systems and artificial life organisms. Time complexity analysis of the different search algorithms to find optimal paths was performed using a timer function in the Steve language, which is part of Breve. Time complexity analysis of the various search algorithms showed the efficiency of the different algorithms. Space complexity analysis of the search algorithms was also performed. Space complexity analysis showed not only the efficiency of the different algorithms but also shows why a particular algorithm fails. Dynamic path finding and dynamic placement of sensors (light sensitive objects in the simulation) were also used in the simulation.

1.3 Outline of the paper

This paper is organised as follows. The search algorithms and heuristics implemented in this paper are discussed briefly in section 2. Section 3 gives a short description of the aspects of Homeland security implemented in this

paper. Section 4 discusses the details of the implementation done using Breve. A few snapshots of the simulation performed in this paper are also shown in section 4. Time and space complexities of the algorithms are shown in the form of tables and graphs containing the experimental data obtained through the simulations implemented in Breve in section 5. Section 6 gives the conclusions and possible future work.

2 Search Algorithms Used in Path Planning

In this study we have compared the general search algorithm to the A* algorithm and a modified A* algorithm that makes use of preset heuristics.

2.1 General Search Algorithm

In the case of a general search algorithm, the agent does not use any heuristic. The only knowledge the agent possesses is the start point and the end point. An exhaustive search of all possible routes is performed and all routes are given the same importance. The only inbuilt intelligence is to ignore circular paths. Circular paths are defined as paths that start and end at the same node.

The pseudo code used in this algorithm is as follows

```
function GENERAL-SEARCH(problem, stratergy)
        returns a solution, or failure
initialise the search tree using the initial
state of problem
  loop do
    1. if there are no candidates for expansion
       then return failure
    2. choose a leaf node for expansion
       according to a fixed strategy
    3. if the node contains a goal state then
       return the corresponding solution
       3.1. else expand the node and add the
            resulting nodes to the search tree
  end
```

2.2 A* Search algorithm

The A* algorithm is a graph search algorithm and is one of the most widely used search techniques. Unlike the general search algorithm above, the A* algorithm uses heuristic rank estimates to make the search more efficient. The nodes are evaluated by combining *g(n)*, the cost to reach the goal, and *h(n)*, the cost to get from the node to the goal:

$$f(n) = g(n) + h(n) \quad (1)$$

Since *g(n)* gives the path cost from the start node to node *n*, and *h(n)* is the estimated cost of the cheapest path from *n* to the goal, we have *f(n)* = estimated cost of the cheapest solution through *n* .

Hence if we are trying to find the cheapest solution, a reasonable route to try first is the node with the lowest value of *g(n)* + *h(n)*. Provided that the heuristic function *h(n)* satisfies certain conditions, A* search is both complete and optimal[1].

2.3 Heuristics

In the A* algorithm when there are two or more paths ending in the same node it makes sense to consider the best possible path and ignore the rest. This is the heuristic that the A* search algorithm makes use of. So whenever it finds that there are more than two paths with the same end node, it computes the *f*-cost of the various paths. Among these paths, the path with the lowest cost is selected and the rest are simply ignored.

The A* search algorithm also sorts the paths placing the lowest cost paths at the end since in this way the nodes at the end of the list are expanded first.

In this paper an additional heuristic has been added to the A* algorithm. This heuristic is described as follows:

The Manhattan distance between the start and target location is computed. It can be assumed that a valid path will not exceed this distance. Hence all paths exceeding this maximum distance can be concluded to be invalid and hence ignored. Domains which are influenced by the presence of obstacles leading to paths greater than this maximum distance have not been considered in this paper.

The Manhattan distance and not the Euclidean distance is considered since the simulated non-holonomic robot is capable of moving only forwards, backwards, left or right and is incapable of moving diagonally.

3 Homeland Security

3.1 Sensor Networks

The application of path finding techniques for homeland applications used in this paper was inspired by work done Li and Rus [4]. Here a versatile information system by using distributed sensor networks, *i.e.* hundreds of small sensors, equipped with limited memory and multiple sensing capabilities which autonomously organize and reorganize themselves as ad hoc networks in response to task requirements and to triggers from the environment. Distributed adaptive sensor networks are reactive computing systems, well suited for tasks in extreme environments, especially when the environmental model and the task specifications are uncertain and the system has to adapt to them. A collection of active sensor networks can follow the movement of a source to be tracked, for example, a moving vehicle. It can guide the movement of an object on the ground, for example, a surveillance robot. Or it can focus attention over a specific area, for example, a fire in order to localize its source and track its spread. A sensor network consists of a collection of sensors distributed over some area that form an ad hoc network. Each sensor is equipped with some limited memory and processing capabilities, multiple sensing modalities, and communication capabilities. These sensors are capable of detecting special events called “danger” (*e.g.* temperature, biochemical agents, *etc.*) that are above a particular threshold. The sensors

that have triggered the special events are considered to be obstacles.

3.2 Sensor Network Implemented in Breve

The sensor network mentioned in the previous section was modeled in Breve using patches. The patch class in the Steve language was utilised for this purpose. In this paper, the patches were equally distributed over the entire region. But efficiency could have been improved by placing the patches in a certain fixed pattern to minimize the number of patches and thus maximizing the safety of a vehicle navigating through the area infested with "danger" (obstacles). In [6] the sensor deployment problem in the context of uncertainty in sensor locations for airdropping situations was considered. Sensor deployment in such scenarios is inherently non-deterministic and there is a certain degree of randomness associated with the location of a sensor in the sensor field. In a sensor network the sensors would sense the special events electronically. This is simulated in Breve by placing light obstacles over the patches. The patches are capable of sensing obstacles placed on them. By getting the location of the light object and by finding the patch present at that location we are able to determine the patches which have obstacles (or "danger") and patches which are safe.

4 Implementation

4.1 Cell Decomposition

According to the principle of Cell Decomposition the entire world is decomposed into equal sized patches. The Patch and PatchGrid classes in Steve were used for this purpose. These patches provide a sense of location. That is, given a location it is possible to obtain the patch object residing in that location.

The patches were created using
patches = (new PatchGrid init-at location (0,0.75,0) with-patch-size (5, 0.1, 5) with-x-count X_SIZE with-y-count Y_SIZE with-z-count 6 with-patch-class "LifePatch").

By changing the *x* and *z* values it was possible to create patches of different dimension *e.g.*, 4x4, 6x6 *etc.* A 32x32 patch would fill the entire work space created in this simulation.

4.2 Light Objects Modelled as Obstacles

The light objects which are part of the sample Braitenberg class in Breve were used as obstacles. Light objects are mobile objects and can be moved around during the simulation, which provides us with dynamic obstacles *i.e.* obstacles that would change their position with time. In this work it was possible to infer a safe path avoiding the obstacles dynamically by checking for changes in the position of the obstacles at every iteration time step. If it were not for computational limitations, this type of replanning from scratch would be the ideal, since it guarantees optimal plan generation and execution given all known information at the time it is acquired. The use of the D* algorithm (Dynamic A*) could lessen the computational work by producing an initial plan based on known and assumed information, and then incrementally update the plan as new information is discovered. The implementation of the D* algorithm is left as future work.

4.3 Visual Representation of the implementation

Figure 1 and Figure 2 show two snapshots of the simulation performed in this paper. The vehicle position is on the start patch. The patches with protrusions contain obstacles. The path found is shown by light colored patches.

Figure 1 below shows the optimal path found in a world containing obstacles placed in a triangular pattern.

In Figure 2 the optimal path is found in a world containing obstacles placed in a rectangular manner.

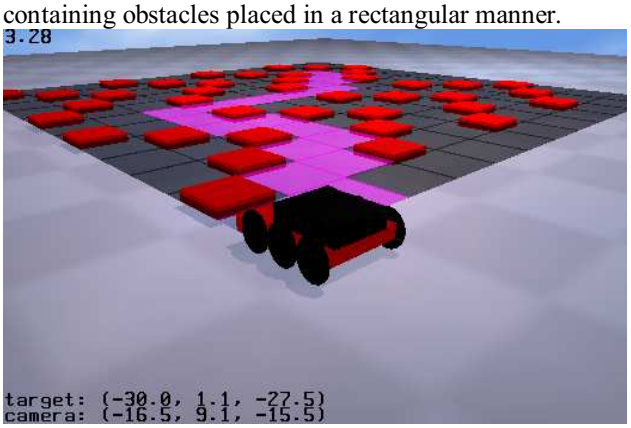

Figure 1: Path found in a world containing obstacles placed in a triangular fashion

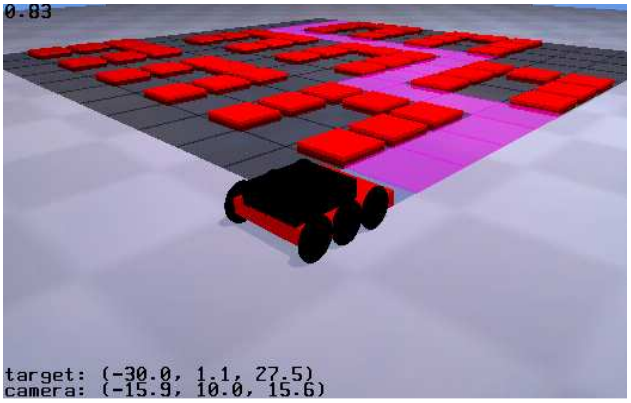

Figure 2: Path found in a world containing obstacles placed in a rectangular fashion

5 Time and Space Complexities

Time and space complexities of the various algorithms implemented in this paper are discussed in this section. The following definitions are important to understand the space complexity analysis.

Maximum paths: The maximum number of possible paths that the algorithm considers during any iteration as it progresses to find the final safe path.
Maximum nodes: The number of nodes in the path which contains the maximum number of nodes before a valid safe path is found

5.1 Time and Space Complexity of the General Search Algorithm

Table 1 shows the time taken to compute the optimal paths for different Manhattan distance values using the general search algorithm.

Manhattan Distance	Time taken to compute path in seconds
40	0
45	1
50	11
55	116

Table 1: Time Complexity using the general search algorithm.

The general search algorithm fails when the distance between start patch and target patch exceeds a Manhattan distance of 55.

Table 2 shows the maximum paths and the corresponding maximum nodes for different Manhattan nodes using the general search algorithm.

Manhattan Distance	Maximum paths	Maximum nodes
30	110	7
40	447	9
45	2282	10
50	5614	11
55	13722	12

Table 2: Space Complexity using the general search algorithm.

The space complexity table shown above proves that the general search algorithm failed due to the large number of maximum paths, and hence memory requirements, that could be computed in a single iteration.

5.2 Time and Space Complexity of the A* Algorithm

Table 3 shows the time taken to compute the optimal paths for different Manhattan distances using the A* algorithm.

Manhattan Distance	Time taken to compute path(seconds)	Manhattan Distance	Time taken to compute path(seconds)
80	1	170	27
110	2	175	31
120	3	180	39
125	4	185	45
130	5	190	41
135	7	195	43
140	8	200	74
145	10	205	86
150	13	210	99
155	16	215	148
160	20	220	168
165	23	225	214

Table 3: Time Complexity using the A* algorithm.

A comparison of table 1 and table 3 shows that the A* algorithm is more efficient than the general search algorithm.
Table 4 shows the maximum paths and the corresponding maximum nodes for different Manhattan nodes using the A* algorithm.

Manhattan Distance	Maximum paths	Maximum nodes
30	13	7
35	18	8
40	21	9
45	27	10
50	31	11
55	37	12
60	43	13
65	51	14
85	83	18
125	171	26
150	223	31
200	321	41
220	358	45

Table 4: Space Complexity using the A* algorithm.

Comparing table 2 with table 4, it can be seen that the A* algorithm expands much lesser number of paths for any Manhattan distance.

5.3 The General Search Algorithm with additional heuristics

As previously mentioned, the additional heuristic involves deleting paths that exceed the maximum Manhattan distance. These paths are defined by the Manhattan distance between the start node and the end node. It is assumed that any path which exceeds this maximum Manhattan distance is not an efficient path.

Table 5 shows the time taken to compute optimal paths for different Manhattan distances after adding the additional heuristic to the general search algorithm.

Manhattan Distance	Time taken to compute path(seconds)
40	1
45	1
50	3
55	57
60	Infinite

Table 5: Time Complexity using general search algorithm after adding the additional heuristic.

Comparing the values in table 5 and table 1, we see that adding this heuristic has certainly improved the performance of the general search algorithm by decreasing the time taken to compute paths. Hence if we add this heuristic to the A* algorithm, it should improve the performance of the A* algorithm.

Table 6 shows the Maximum paths and Maximum nodes to compute optimal paths for different Manhattan distance values after adding the additional heuristic to the general search algorithm.

Manhattan Distance	Maximum paths	Maximum nodes
30	80	7
35	201	8
40	452	9
45	1020	10
50	2198	11
55	4776	12
60	10106	13

Table 6: Space Complexity using general search algorithm after adding the additional heuristic.

Comparing the values obtained in table 1 and table 6, it can be seen that the addition of this heuristic has decreased the space complexity of the general search algorithm. The next section shows how the addition of this heuristic improves the A* algorithm.

5.4 The A* algorithm with additional heuristics

Table 7 shows the time taken to compute optimal paths for different Manhattan distances after adding the additional heuristic to the A* algorithm.

Manhattan Distance	Time taken to compute path(seconds)
160	11
180	25
210	70
215	91
220	123
225	140
230	165
235	220
240	238

Table 7: Time Complexity using the A* algorithm after adding the additional heuristic.

Comparing table 3 and table 7 it can be concluded that adding this additional heuristic has improved the performance of the A* algorithm in terms of time taken.

Table 8 shows the Maximum paths and Maximum nodes to compute optimal paths for different Manhattan distances after adding the additional heuristic to the general search algorithm.

Manhattan Distance	Maximum paths	Maximum nodes
35	9	8
40	11	9
45	12	10
50	13	11
55	15	12
60	19	13
65	21	14
70	24	15
75	28	16
80	30	17
85	33	18
125	58	26
150	115	31
200	273	41
220	347	45
225	351	46
230	374	47

Table 8: Space Complexity using the A* algorithm after adding the additional heuristic.

Comparing table 4 and table 8 it can be seen that the additional heuristic has decreased the number paths expanded by the A* algorithm even though there is no improvement in the maximum number of nodes expanded.

The time and space complexity of the A* and modified A* algorithms discussed above are also shown in the graphs of figure 3 and 4 respectively.

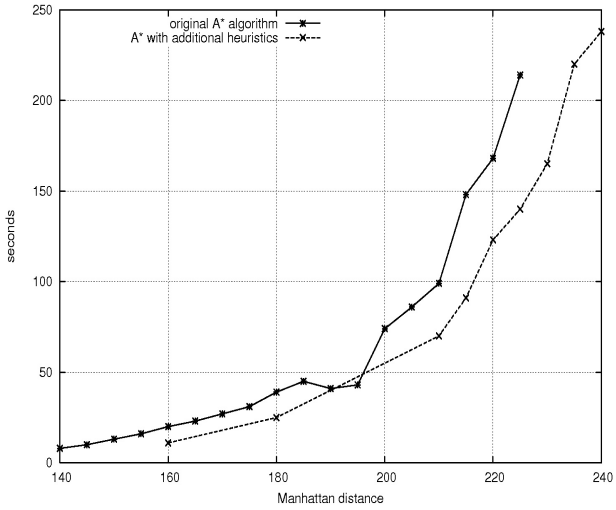

Figure 3: Time complexity analysis of A* versus the modified A* algorithm

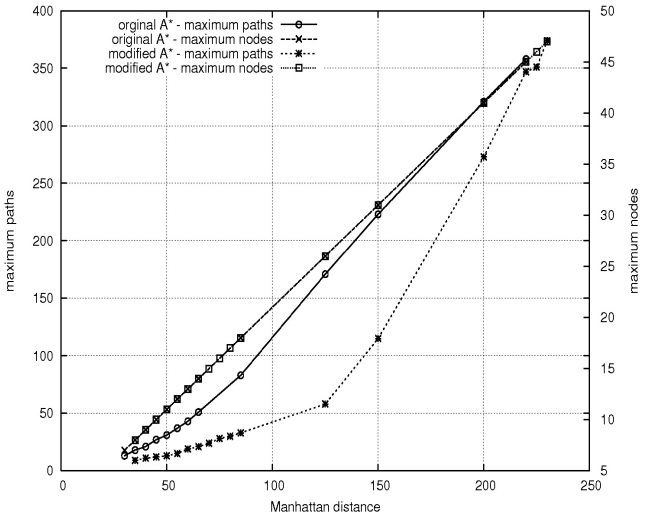

Figure 4: Space complexity analysis of the A* versus the modified A* algorithm

6 Conclusion and future work

It was proved with the help of experimental data that the additional heuristic added to the A* algorithm decreased the time taken to find optimal paths. Space complexity analysis also showed us that the reason for the improvement after adding the additional heuristic was due to the decrease in the number of paths expanded.

A drawback in this algorithm when used in critical homeland applications would be that it computes optimal paths instead of safest paths which may be critical in some homeland security applications. Hence this algorithm is more suitably used in non critical homeland applications and the algorithm needs to be modified for use in critical homeland applications where it is vital to compute safe paths which keep the robot as far away from the obstacles as possible.

This research may be extended in several directions:

- A plain terrain was used in the simulations performed in this paper. Using Breve it is possible to model different types of terrains like hills, valleys and lakes (obstacles). The robots used in the WTC were unable to withstand the rigors of rubble. The effect of using the path finding and navigation techniques on different terrains can be analysed.
- This project deals only with path finding. The algorithms assume that the sensor locations are known. Breve has very efficient collision handling methods. Using this it should be possible to model and implement map building techniques. The robot can be made to explore the environment and mark the safe spots and the danger spots. There is still much research in the area of map building for mobile robots. Simultaneous Localization and Mapping (SLAM) [7] is one of the most used techniques in map building. So another interesting future work would be to first implement map building techniques to build a map of the environment and then use path finding techniques using this information.

7 Acknowledgements

I would like to express my gratitude to

- Dr. Stuart Meikle for contributing his ideas on pathplanning.
- Mr. Jonathan Klien for rescuing me many a times with regards to issues I encountered when programming with Breve.

8 Reference:

[1] Stuart Russell, Peter Norvig, *A modern approach* (Englewood Cliffs, NJ: Pearson Education, 2003).

[2] Sven Behnke, Local Multiresolution Path Planning. *In Proceedings of 7th RoboCup International Symposium*, 2004, 332-343.

[3] L. Kavraki, P. Svestka, J.C. Latombe, and M. Overmars, *Probabilistic road maps for path planning in high-dimensional configuration spaces. IEEE Transactions on Robotics and Automation*, 1996, 566–580.

[4] Qun Li, Daniella Rus. *Navigation Protocols in Sensor Networks, ACM Transactions on Sensor Networks (TOSN)* 2005, 3 – 35.

[5] Klein, J. Breve, *A 3D simulation environment for the simulation of decentralized systems and artificial life, Proceedings of Artificial Life VIII, the 8th International Conference on the Simulation and Synthesis of Living Systems*, 2002.

[6] Y. Zou and K. Chakrabarty, *Uncertainty-Aware Sensor Deployment Algorithms for Surveillance Applications,* 2003, 261-272.

[7] Stefan B.Williams, Hugh Durrant-Whyte, Gamini Dissanayake, *Constrained Initialization of the Simultaneous Localization and Mapping Algorithm, In the International Journal of Robotics Research.*2003, 7–8.