Path Planning for Robotic Boats in a Rescue System

S. M. Masudur Rahman , Al-Arif , A. H. M. Iftekhar Ferdous, Mohammad Sohrab Hasan Nizami
Department of Electrical and Electronic Engineering (EEE)
Islamic University of Technology (IUT)
Board Bazar, Gazipur – 1704, Dhaka, Bangladesh

ABSTRACT
Water is life. There is no denying. But water is also a source of many disasters and dangers. There are a lot of rivers, canals and waterways in South Asian countries. In some cases water transports are the only mode of transportation for movement and trade. Many natural and man-made causes like flood, cyclone, tsunamis, jacking, looting; sometimes, people get stuck in a water surrounded environment. So it an unavoidable issue to rescue the people when they fall in such situation. Two kinds of automated rescue mission can be possible in those cases, one is air-borne and another one is water-borne. Water vehicles shows better efficacy instead of air vehicle for developing countries in terms of economy and complexities. Therefore here in this paper, a basic principle and methods towards an automated water-borne rescue system is outlined. The architectures of distributed system along with multi-logics are presented. The proposed system is actually comprised of Artificial Intelligence (AI) and Mobile Robotics. The proposed system is then investigated by all available path finding algorithms, to find a most suitable which can conduct rescue operation for different map systems with better efficiency and better economy.

General Terms
Artificial Intelligence (AI), Rescue Systems, Algorithms.

Keywords
Robotic rescue system, Path planning algorithms, graph-search algorithm, breadth-first algorithm, A* (astar) algorithm; dijsktra algorithm.

1. INTRODUCTION
South Asian countries (Bangladesh, Pakistan, India etc) are full of rivers. The major means of communication of those countries are rivers and canals. Therefore, big/small boat is an important mode for transportation. Sometimes, for man-made invention (hijacking a ship/loot of wealth etc.) and mechanical failures fall the passengers in a danger situation. Also many natural disasters like flood, cyclones and tsunami force people to get stuck in water surrounded situation. Hence, it is an unavoidable issue to rescue the people from danger. Two types of approaches are generally applied for this type of rescue campaign: one is air vehicles: another is water (automatic boat). Aircraft is too expensive to bear a better service for the developing countries. Therefore, water vehicles shows better efficacy in the view of economic tolerance.

With the advent of wireless technology and technique from applied engineering has changed the way to save the endangered people. To deal with an un-named rescue system concerns with many items and technical disciplines. For dealing with so many items within a limited time, a whole system activity is distributed over times, and for speeding-up of such activities leads the challenges (integration etc.) in applied engineering. The activity involves what is the relative co-ordination and communication of a team member with the others, and how the data of generalized vehicles are created and dealt with. This paper presents towards a real prototype of a rescue system. We used artificial intelligence (AI) and the concept of mobile robots to represents the proposed approach. AI is the most effective issue in our paper. Mobile robots are the principle working medium here as they perform all the necessary physical works. So the main theme is that we tried to implement the whole system using Artificial Intelligence (AI) through mobile robot. Many of the available path planning algorithms are used to check the systems response to different rescue modes. Finally, a perfect algorithm is chosen for each type of rescue mode.

2. BACKGROUND
Artificial intelligence (AI) can be defined as "the study and design of intelligent agents", where an intelligent agent is a system that perceives its environment and takes actions which maximizes its chances of success. Intelligent agents which may be a device or vehicle or robot, must be able of set destinations and achieve them. They need a way to understand the situation from available information and be able to make choices that maximizes the utility of the available choices.

Robotics is very closely related with AI. Intelligence is a must thing for a robot to be able to handle certain tasks like navigation and motion planning. Different artificial intelligence (AI) techniques can be used to provide the robots with this kind of intelligence and flexibility. Those techniques belong to three areas of artificial intelligence:

i) Learning

ii) Reasoning and

iii) Problem solving

Among many diverse learning algorithms, inductive learning is most widely used in robotics, in which the robots learn from pre-selected examples. In our case we have used some similar maps and rescue modes as pre-selected examples. Then among all path planning algorithms the most efficient one for each mode of operation was chosen to be used in real case scenario.

In past many researcher have shown tremendous amount determination in the similar fields, which were very help for us. A sensor based intelligent autonomous control method is proposed for tele-operated robotic system. A number of sensors are used in the system to obtain the environmental information and that input is then used into different level autonomous controller to fulfill the primary target [1-4]. A sensor based network system for a rescue robot working under a similar disaster situation is also presented in a work [5]. Here a network system is proposed and an algorithm for a rescue robot to obtain its position under collapsed area is also

considered which in our case will be a water surrounded area. A lot of other works are going on about different algorithms that we are going to compare here. Dijkstra algorithm is a very commonly used algorithm. Researchers are working around the world to improve this algorithm more and more [6], [7], [8]. A-Star (A*) Algorithm is also very useful for search and rescue operations [9]. Works are going on around the world to make an efficient hardware engine for this algorithm [10] and to use this algorithm in hazardous environment [11]. A very important work [12] compared dijkstra and A* algorithm in a grid based map which will be our primary tool to find the most suitable algorithm for our case.

Apart from those works, many researchers have also shown tremendous improvement in the field of mobile robots, path planning algorithms and rescue robotics. A 3D active sensing mechanism is used in [13] to operate search and rescue mission in urban areas. They used a sensory system that provides high resolution 2-D and 3-D information of a cluttered scene that can be used by a robot operator for real-time viewing as well as to develop a 3-D map of the disaster scene.

In [14] a new and innovative type of incremental multi-scale search algorithm is demonstrated for path planning in a dynamic way with low worst-case complexities. This incremental multi-scale algorithm leads to an improvement both in terms of robustness and computational complexity–in the worst case–when compared to the classical algorithms.

Recently in [15] they propose another fast path algorithm for finding the best shortest paths in the road network. There they tried to minimize costs between the origin and destination nodes just like our case. The proposed algorithm was compared with the dijkstra algorithm in order to find the best and shortest paths using a sample of Tehran city road network. In our case we will discuss all available algorithms including dijkstra to do the same in water affected area for rescue operation.

3. THE PROPOSED RESCUE SYSTEM

Three rescue boats (RB) are initially considered to rescue the victim from the Endangered boat (EB) to a safe place. Whenever EB finds a problem, it sends a signal to the Base station (BS). BS receives EB's location and it knows where actually the RB's are, because RB's are connected to each other and to the BS via computer network.

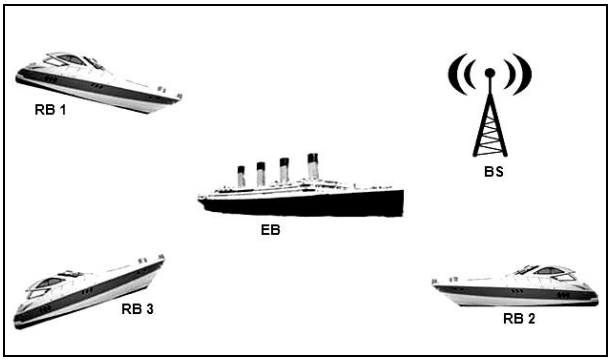

Fig 1: Proposed System

Whenever BS receives a signal from any EB, it finds out which RB is the nearest to that EB and sends signal to that RB to perform rescue operation and hence that RB reach to that EB and rescue the victims. The main reasons of finding the nearest one are:

i) To reduce the cost

ii) To minimize the rescue period and

iii) To keep other RB free so that they can perform another operation if needed.

4. PATH FINDING

Path finding problem is the fundamental problem for mobile robots i.e. the RB. The graph search algorithms are the most known solutions for this problem. The most known algorithms for the shortest path problem are:

i) The graph search algorithm

ii) Breadth-first algorithm

iii) Dijkstra algorithm and

iv) A* (A Star) algorithm

So far most of the related studies are focused on one of these algorithms and for urban based map. In this paper, these algorithms are summarized and simulated. Depending on our case various advantages and disadvantages are defined. Some assumptions are taken into account before the simulation. The map we have used is divided into same size square cells. The ability of traversing is accepted as 900 and 4-adjacent traversable neighbor's is considered for simplicity. 4- and 8-adjacency definition can be shown in Fig. 2 and Fig. 3 respectively.

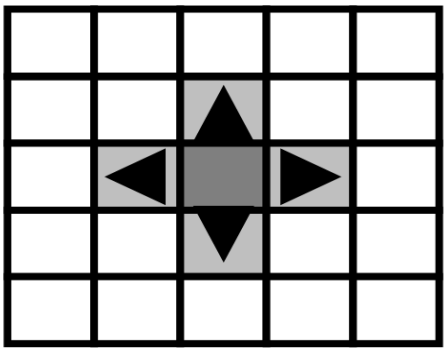

Fig 2: 4-adjacency

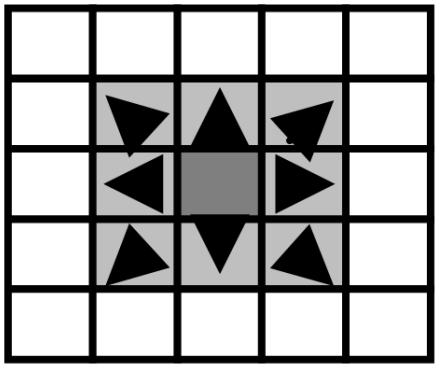

Fig 3: 8-adjacency

4.1 The Graph Search Algorithm

The graph search algorithms are very old and primitive. These algorithms are basically based on node-edge notation but this notation lacks when a modern system like GPS gets an image frame, converts it to a map matrix and uses this map matrix as the grid based map. In these situations using matrix notation gives the advantage of simplicity and comprehension. But in our particular case this type of algorithm is of lesser use.

4.2 The Breadth-First Algorithm

Unlike the graph search algorithm the breadth-first algorithm works with the method branching from the starting cell to the neighbor cells (just traversable cells), (un-traversable cells and cells out of boundaries are discarded) until the destination cell is found [16, 17]. This kind of algorithm can be very useful in water affected area scenario.

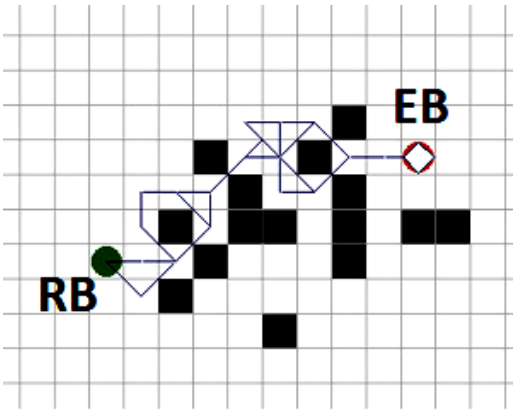

Fig 4: Breadth-first algorithm

To realize simulation of this algorithm we must define some arrays. Here we added all traversable neighbor cells to an array named NEIGHBOURS. So that means NEIGHBOURS is the array of neighbor cells which must be investigated in order to find the destination cell. All NEIGHBOURS elements are checked if one is the destination cell or not. Then NEIGHBOURS arrays include new neighbor cells, which actually are the neighbors of the old listed cells and this procedure goes on until the destination cell is finally added to the NEIGHBOURS. The cost of the starting cell is zero. The cost of each neighbor cell is +defined constant cost of the cell which added it to the NEIGHBOURS. Here the constant cost is taken as one. The costs of the cells are stored in COST matrix with the same dimensions of the map. Then after adding the new neighbor cells, old checked cells are pulled out of NEIGHBOURS. This prevents checking the checked cells again. When the destination cell is added to the NEIGHBOURS, to find the shortest path you just follow from the destination cell to the starting cell step by step by the decreasing cost of the cells from the cost-matrix if the NEIGHBOURS is empty anytime, this means there is no possible paths.

4.2.1 Summary of breadth-first algorithm

The total procedure of the breadth-first algorithm can be summarized as follows:

i) Define the starting and destination cells

ii) Load the map matrix

iii) Add the starting cell to NEIGHBOURS

iv) Add the neighbor cells to NEIGHBOURS

v) If NEIGHBOURS is empty, no possible path

vi) If destination cell is added to NEIGHBOURS, define the PATH using map matrix. Else compute the cost of neighbor cells

vii) Pull out the checked cells from NEIGHBOURS

viii) Go to step iv.

4.2.2 Advantages of breadth-first algorithm

The breadth-first algorithm is simple to implement. It doesn't require too much matrix operations and also doesn't need to use the location of the destination cell (an advantage if the location of the destination isn't defined). This algorithm is quite good for our water surrounded case scenario.

4.2.3 Disadvantages of breadth-first algorithm

Although the breadth-first is good enough to be used in our proposed system but still it have two major drawbacks:

i) One has to search all the available traversable cells until the destination cell is found. So in large maps or in real case scenario it needs very large computational space.

ii) It is impossible to define cells with different costs.

4.3 The Dijkstra Algorithm

This algorithm is almost similar to the breadth-first algorithm but it overcomes a major disadvantage of previous algorithm. It can do the computation of different cost cells. That means it can not only find the shortest path but also the lowest cost path. In this algorithm, again the array of the all neighbor cell NEIGHBOURS exists. Here at starting the neighbors of the starting cell are added to the NEIGHBOURS. Then the costs of the neighbors are calculated. These costs are the costs of moving from starting cell to neighbor cells. Neighbor cells are checked according to their calculated costs. When a cell with the lowest cost is found the neighbors of this cell is added to NEIGHBOURS. That means the lowest cost becomes the comparison criterion. For these new cells the cost from starting cell to these cells are calculated and again the neighbors of the lowest cost cell is added to the NEIGHBOURS. This procedure goes on until the destination cell is added to the NEIGHBOURS. When the destination cell is added to the NEIGHBOURS, following the parents of the cells from the destination cell to the starting cell gives the shortest and the lowest cost path. If the NEIGHBOURS is empty anytime, it means that there are no possible paths.

4.3.1 Summary of dijkstra algorithm

The total procedure of the breadth-first algorithm can be summarized as follows:

i) Define the starting and destination cells

ii) Load map matrix

iii) Add the starting cell to NEIGHBOURS

iv) Add the neighbor cells to NEIGHBOURS compute the costs, record their parent cell to PARENTS

v) If NEIGHBOURS is empty, no possible path

vi) If destination cell is added to NEIGHBOURS define the PATH using PARENTS matrix. Else go on

vii) If neighbor cell is added NEIGHBOURS before find its new cost and compare to its old cost. If it is lower, update the cost and PARENTS matrix

viii) Pull out the checked cells from NEIGHBOURS

ix) Go to step iv.

4.3.2 Advantages of dijkstra algorithm

The lowest cost criterion and the ability of computing different cost cells makes this algorithm very efficient in large and different cost terrain maps. These two properties are very crucial for our proposed system. As the proposed system has to navigate through water-borne disaster affected areas, the ability to compute different cost cell makes it a natural choice.

4.3.3 Disadvantages of dijkstra algorithm

Although his lowest cost criterion obtains the shortest path but it has two problems:

i) NEIGHBOURS array has to be sorted according to the costs and the new neighbor cells have to be located in the right place in the NEIGHBOURS. And in order to locate the new cells and in order to find the shortest and the lowest cost path the parents of the neighbor cells have to be stored in PARENTS array.

ii) And yet again if the cost of the neighbor cell is lower than its parent cell the neighbor becomes the parent and the costs have to be re-computed.

4.4 A* (A Star) Algorithm

This is the most common and efficient used algorithm in shortest path finding problems.

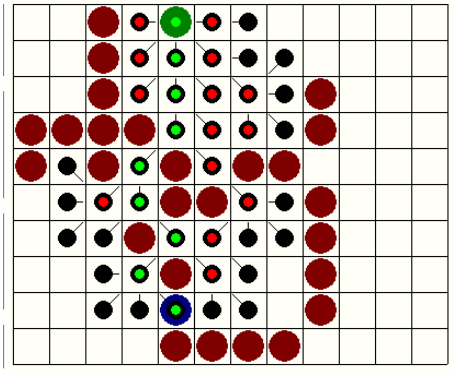

Fig 5: A* Algorithm

For this new algorithm we need to define two list arrays:

i) NEIGHBOURS

ii) CHECKED CELLS

NEIGHBOURS array does the same work and where CHECKED CELLS array holds the cells that have already been checked. Again as previous algorithms first the neighbors of the starting cell are added to the NEIGHBOURS. And like dijkstra these cells are checked according to their costs. But this time two cost functions exist.

i) S = cost of moving from the starting cell to the current cell

ii) D = cost of moving from the current cell to the destination cell

Cost at any point n, $C(n)=S(n) + D(n)$.

The cost function S can be calculated but the cost function D can just be estimated. That's why this cost function is called heuristic cost function. There are several methods for this estimation. As for our case i.e. 4-adjacent traversable cells Manhattan method is the most used method.

$D(currentcell)=|currentX-destinationX|+|currentY-destinationY|$

This method directs the search to the destination cell. The total cost function $C = S + D$ is the comparison criterion for the cells. NEIGHBOURS has to be sorted and in addition as the comparison criterion the C cost array has to be sorted. The parents of the neighbor cells are stored in PARENTS array. Again in this algorithm if the cell exists in NEIGHBOURS its new cost must be compared to the old cost. If it is lower the cell becomes the parent and S and C costs must be re-calculated. The checked cells are placed in the CHECKED CELLS. Again after the destination cell is added to NEIGHBOURS, following the parent cells gives the shortest path. Just like the previous cases if the NEIGHBOURS is empty at anytime, it means that there is no possible path [12].

4.4.1 Summary of A algorithm*

The algorithm can be summarized as follows:

i) Define the starting and destination cell

ii) Load the map matrix

iii) Add the starting cell to NEIGHBOURS

iv) Add the staring cell to CHECKED CELLS

v) Add the neighbor cells to NEIGHBOURS: - If traversable; - If not in NEIGHBOURS before; - If not in CHECKED CELLS; With the order compute S, D and C cost function values. Record the parent to PARENTS matrix. Locate the C cost function value in the right place- If in NEIGHBOURS before; Compute the S cost function value. If it is better than the old value, chance the parent with this parent in PARENTS matrix. Update S and C cost functions

vi) If NEIGHBOURS is empty, no possible path

vii) If the destination cell is added to NEIGHBOURS define the PATH using PARENTS matrix

viii) Find the lowest cost neighbor cell. Add it to CHECKED CELLS and continue the search on this cell

ix) Pull out the checked cells from NEIGHBOURS. Go to step v.

4.4.2 Advantages of A algorithm*

This algorithm is the most efficient algorithm because it uses both the shortest path information from starting cell and the shortest path information to the destination cell.

4.4.3 Disadvantages of A algorithm*

This algorithm cannot be used if the location of the destination cell is not unknown.

5. MULTI-DESTINATION CELLS

All of above algorithms are described for cases like one starting cell - one destination cell. But in practical cases of a rescue mission one starting cell-multiple destination cells without the importance of destination cells order are most likely. A single RB may have to perform rescue of more than one EB. There is no order between the EB's. In such cases all

possible paths have to be calculated. First n! Paths between the points (n = number of destination points, |AB1|, |AB2 |, |AB3| , |B1B2| , |B1B3| , |B2B3|) have to calculated and then the shortest path from starting point to multi-destination points (all points have to be visited once) have to calculated.

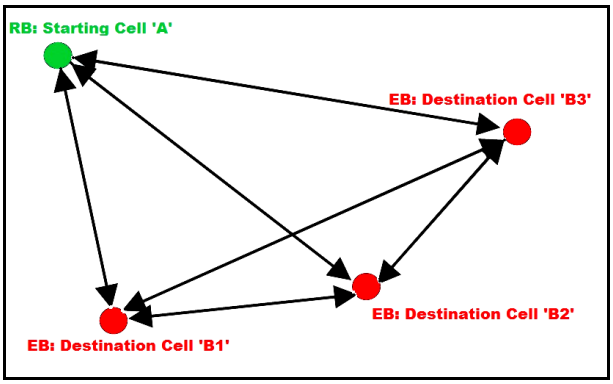

Fig 6: One starting - multi-destination points (S: starting point , G1-G2-G3 : destination points)

This is a most known problem Traveling Salesman Problem (TSP) in graph theory [18, 19]. As like a traveler salesman has to visit a number of towns once and has to turn back to the starting town. The number of entire paths that has to be calculated is (n-1)!/2 (n: number of destination points+1). This number is 3 for 3 destination points but for 10 destination points this number becomes 18,14,400. Total computation time for this number is not acceptable and some techniques are used to decrease the number of computation. This subject is not destination of our paper and some useful resources can be found in [18, 19].

6. Result and Simulation

All the simulation works are mainly divided into two groups.

i) One starting cell - One destination cell

ii) One starting cell - multiple destination cells

At first the algorithms are compared for one starting-one destination cells. As mentioned before, the breadth-first algorithm is lack of computing different cost cells. So two types of maps are used for this group. One that only has same cost cells (Breadth-first, Dijkstra and A*) and another one for different cost cells (Dijkstra , A*).

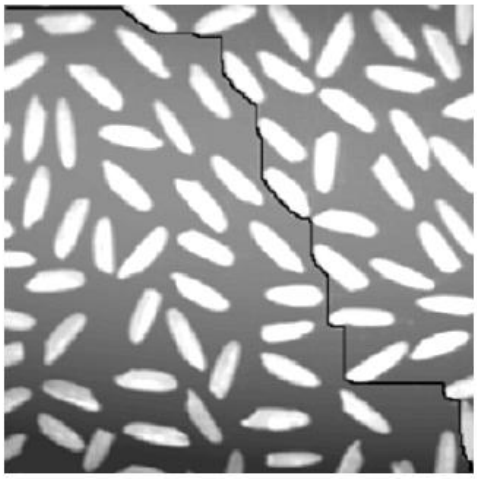

Fig 7: Same cost cell-map and path found by breadth-first algorithm

Fig 8: Same cost cell-map and path found by dijkstra algorithm

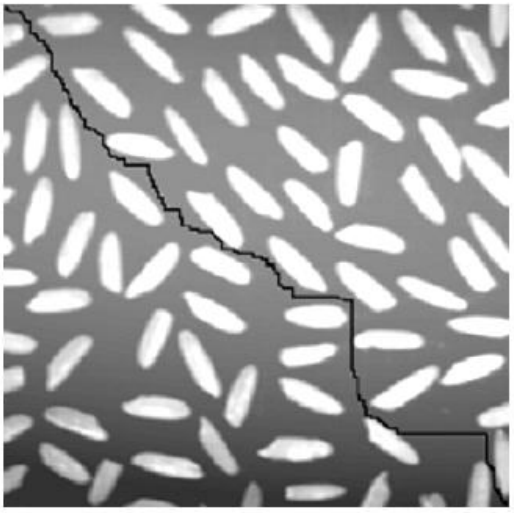

Fig 9: Same cost cell-map and path found by A* algorithm

Table-1 lists the comparison of algorithms for CPU time, the sum of the cells, the cells visited, and the path cells. It can be seen that although the Breadth-first algorithm visits more cells, its CPU time is better than Dijkstra and A*.

The cause of this efficiency is the simplicity. Dijkstra and A* algorithm need a lot of matrix operations and in a map with same cost cells, the costs of the cells must be updated very frequently.

Table 1. Comparison of algorithms for same cost cell-map

Algorithm	CPU Time (s)	Sum of the cells visited	Sum of the path cells
Breadth-First	1.078	43002	506
Dijkstra	2.625	43004	506
A*	2.297	25134	506

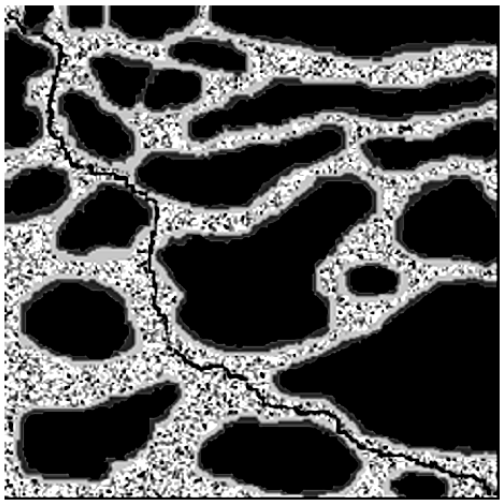

Fig 10: Different cost cell-map and found path by dijkstra

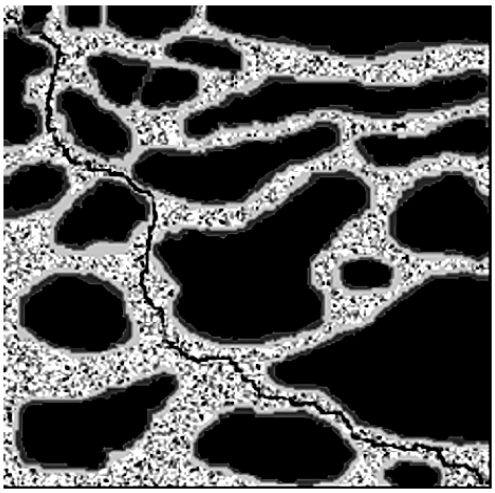

Fig 11: Different cost cell-map and found path by A*

Fig. 10 and Fig. 11 show the maps with different cost cells and the paths for the algorithms. We kept the dimensions similar to the same cost map. The starting cell is at (3, 3) and destination cell is at (255, 255). This map is manually created and different shades of grey define different costs. Finally the black curve shows the path found.

Table-2 lists the comparison of algorithms for CPU time, sum of the cells, the cells visited, the path cells and cost sum of the path cells. It can be seen from the table that A* algorithm doesn't give the shortest and the lowest cost path. The quality of A* algorithm depends on the quality of the heuristic cost function D. If D is close to the true cost of the remaining path, A* algorithm guarantees finding the shortest and lowest cost path. In other condition A* gives no guarantee but it is still efficient.

Table 2. Comparison of algorithms for different cost-cell map

Algorithm	CPU Time (s)	Sum of the cells visited	Sum of the path cells	Cost sum of the path cells
Dijkstra	2.097	35280	537	6970
A*	1.718	26990	545	7000

From the tabular data shown in table 2 we can see that the cost sum of the path cells found by A* is 0.4% higher than Dijkstra's but it is 21.9% faster and it needs 30.7% less memory according to the sum of the cells visited.

In the second group algorithms are compared for one starting-multi destination cells. Here the map with different cost cells is used. Three destination cells are defined on the map. The coordinates of starting cell and destination cells are given below.

A: (3,3) *B1: (120,5) B2: (190,140) B3: (70,185)*

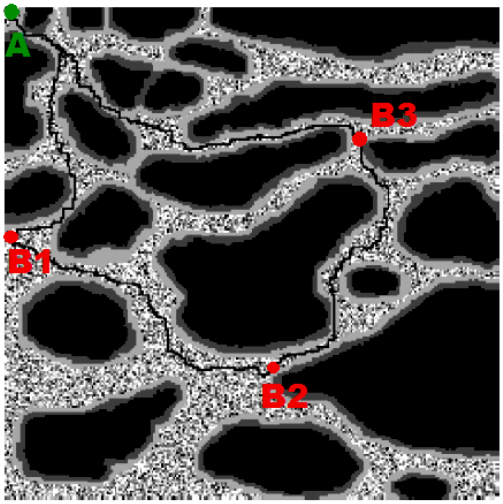

Fig 12: Maps and path for multi destination with dijkstra

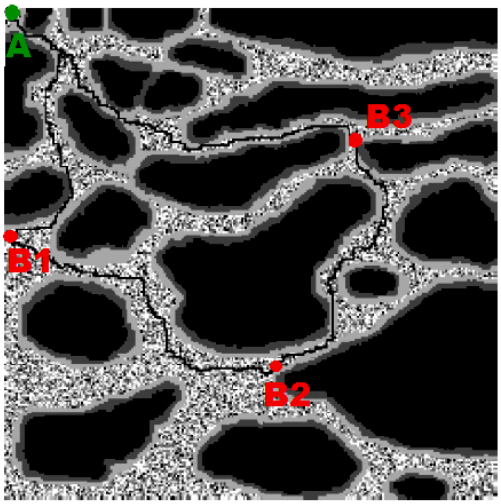

Fig 13: Maps and path for multi destination with A*

Fig. 12 and Fig. 13 show the maps and the found path for dijkstra and A* algorithms. Table 3 lists the comparison of algorithms for different start-destination points , CPU times , sum of the path cells cost sum of the path cells and selected path ,total CPU time , sum of the path cells and the cost sum of the path cells.

This time A* gives the shortest path. It can be seen that the total CPU times are very close. This result comes from the advantage of computing paths using visited cells. In dijkstra |AB1|, |AB3| and |B1B2| cells are visited in the previous path and there is no need to re-compute the cells. A* is lack of this advantage but it is still more efficient in operations [12].

Table 3. Comparison of Dijkstra and A* algorithms

Algorithm	Start Destination points (farthest)	CPU Time (s)	Sum of the path cells	Cost sum of the path cells
Dijkstra	|SG2|	1.453	359	4770
	|SG1|*	0.078	198	2900
	|SG3|*	0.094	276	4220
	|G1G3|	1.594	265	3380
	|G1G2|**	0.078	210	2490
	|G2G3|	1.875	198	2820
* visited in |SG2|		**visited in |G1G3|		
Selected Path		**Total CPU Time (s)**	**Sum of the path cells**	**Cost sum of the path cells**
|SG1|- |G1G2|-|G2G3|-|G3S|		5.172	882	12430
Algorithm	**Start Destination points (farthest)**	**CPU Time (s)**	**Sum of the path cells**	**Cost sum of the path cells**
A*	|SG2|	1.156	359	4770
	|SG1|	0.672	200	2900
	|SG3|	0.953	272	4220
	|G1G3|	0.891	265	3380
	|G1G2|	0.704	210	2490
	|G2G3|	0.781	198	2820
Selected Path		**Total CPU Time (s)**	**Sum of the path cells**	**Cost sum of the path cells**
|SG1|- |G1G2|-|G2G3|-|G3S|		5.157	880	12430

7. CONCLUSION

All the algorithms presented in this paper have some computational advantages in path planning for multi-destination cells. They seem to find the shortest path between starting and destination cells but in fact these algorithms can find all the shortest paths from the starting cell to all visited cells. So we had to examine Breadth-first, Dijkstra and A* discreetly. Now Breadth-first and Dijkstra don't use the location of the destination point in the computations that's why they can find all the shortest paths for all visited cells. Where as A* can find paths for all visited cells but doesn't guarantee the shortest path because A* uses the location of the destination point in its computation. This benefit gives Breadth-first and Dijkstra one computational advantage. If the destination cell is not in the visited cells, the computation for the shortest path between cells to cell has to be repeated. In these types of computations, starting the computation between the most far destination cells can give an advantage.

In this paper we have discussed the path planning part of a total water-borne rescue system by comparing, simulating four path planning algorithms on grip based map for both one starting - one destination cell and one starting - multi destination cells. From the tabular and graphical results of the experiments and the inferences from the algorithms, we found some important information for path planning for maps with same cost cells, different cost cells and with one starting - one destination and one starting - multi destination cells. For maps with same cost cells, with one starting-one destination cell and multi destination cells, using Breadth-first algorithm is the best if the computational time is the primary desire criteria. But if the size of memory is the major criteria then using A* can be a better alternative. For maps with different cost cells and with one starting - one destination cell A* is best in both computational time and size of memory. But the heuristic cost function D for A* must be chosen in order to find the shortest and lowest cost path. Again for maps with different cost cells and with one starting-multi destination cells A* is best in computational time with no certainty for the shortest path. But it must be understood that Dijkstra, using visited cells advantage especially in enormous multi-destination cells and shortest path guarantee, can be a good choice for these maps.

The algorithms used 4-adjacent traversable cells related to the mobile robot. If a mobile robot with more movement abilities is accepted, using 8- and 16- adjacent traversable cells give better results. Again in our simulation A* uses Manhattan method as the heuristic function. Using other functions can also give better results. In future instead of using manually drawn map, it is planned to use real geographical maps to get more realistic and practical results.

8. ACKNOWLEDGMENTS

We would like express out heartiest thank to Tolga Yüksel, Abdullah Sezgin of Ondokuz Mays University, Kurupelit, Samsun, Turkey for their helpful discussions and encouragements to apply their simulations into some practical life saving automated rescue system.

9. REFERENCES

[1] Chou Wusheng; Wang Tianmiao; You Song; "Sensor-based autonomous control for telerobotic system," Intelligent Control and Automation, 2002. Proceedings of the 4th World Congress on, vol.3, no., pp. 2430- 2434 vol.3, 2002.

[2] S.M.M.R Al-Arif, N. Quader, A.M. Shaon and K.K. Islam, "Sensor based autonomous medical nanorobots: A cure to demyelination"; Cyber Journals: Multidisciplinary Journals in Science and Technology, Journal of Selected Areas in Nanotechnology (JSAN), page: 1-7; Vol. 2, No. 11 September Edition, 2011.

[3] Quader, N.; Al-Arif, S.M.M.R.; Shaon, M.A.M.; Islam, K.K.; Ridwan, A.R.; "Control of Autonomous Nanorobots in Neural Network"; 4th International Conference on Biomedical Engineering and Informatics (BMEI 2011), 15-17 Oct. 2011; Shanghai, China; Vol.: 3; pp. 1399-1402.

[4] S. M. Masudur Rahman Al-Arif; "Control System for Autonomous Medical Nanorobots"; International Conference on Biomedical Engineering (ICoBE 2012); 27 - 28 February 2012; Perlis, Malaysia. pp. 161-164.

[5] Miyama, S.; Imai, M.; Anzai, Y.; "Rescue robot under disaster situation: position acquisition with Omni-directional Sensor", IEEE/RSJ International Conference on Intelligent Robots and Systems, 2003. (IROS 2003), 27-31 Oct. 2003, vol.3, pp. 3132 – 3137.

[6] Yin Chao, Wang Hongxia, "Developed Dijkstra shortest path search algorithm and simulation", International Conference on Computer Design and Applications (ICCDA), 2010, 25-27 June 2010, vol.1, pp. 116-119.

[7] Hwan Il Kang, Byunghee Lee, Kabil Kim, "Path Planning Algorithm Using the Particle Swarm Optimization and the Improved Dijkstra Algorithm", Pacific-Asia Workshop on Computational Intelligence and Industrial Application, 2008. PACIIA '08, 19-20 Dec. 2008, vol.2, pp.1002-1004.

[8] Zhang Fuhao, Liu Jiping, "An Algorithm of Shortest Path Based on Dijkstra for Huge Data", 6th International Conference on Fuzzy Systems and Knowledge Discovery, 2009. FSKD '09, 14-16 Aug. 2009, vol.4, pp.244-247.

[9] Xiang Liu, Daoxiong Gong, "A comparative study of A-star algorithms for search and rescue in perfect maze", International Conference on Electric Information and Control Engineering (ICEICE), 2011, 15-17 April 2011, pp. 24-27.

[10] Woo-Jin Seo, Seung-Ho Ok, Jin-Ho Ahn, Sungho Kang, Byungin Moon, "Study on the hazardous blocked synthetic value and the optimization route of hazardous material transportation network based on A-star algorithm", 5th International Joint Conference on INC, IMS and IDC, 2009. NCM '09, 25-27 Aug. 2009, pp. 1499 –1502.

[11] Ma Changxi, Diao Aixia, Chen Zhizhong, Qi Bo, "Study on the hazardous blocked synthetic value and the optimization route of hazardous material transportation network based on A-star algorithm", 7th International Conference on Natural Computation, 26-28 July 2011, vol.4, pp. 2292 – 2294.

[12] Tolga Yüksel, Abdullah Sezgin; "An Implementation Of Path Planning Algorithms For Mobile Robots On A Grid Based Map", Publisher: Citeseer, 2008.

[13] Mobedi, B.; Nejat, G.; , "3-D Active Sensing in Time-Critical Urban Search and Rescue Missions," IEEE/ASME Transactions on Mechatronics, vol.99, pp.1-9.

[14] Yibiao Lu; Xiaoming Huo; Arslan, O.; Tsiotras, P.; , "Incremental Multi-Scale Search Algorithm for Dynamic Path Planning With Low Worst-Case Complexity," IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics, vol.41, no.6, pp.1556-1570, Dec. 2011.

[15] Selamat, A.; Zolfpour-Arokhlo, M.; Hashim, S.Z.; Selamat, M.H.; "A fast path planning algorithm for route guidance system," 2011 IEEE International Conference on Systems, Man, and Cybernetics (SMC), pp.2773-2778, 9-12 Oct. 2011.

[16] K. Manley, "Pathfinding : From A* to LPA", seminar, 21 Apr 2003, Available online: http://csci.mrs.umn.edu/UMMCSciWiki/pub/CSci3903s03/KellysPaper/seminar.pdf

[17] B. Stout, " Smart Moves :Intelligent Pathfinding ", Game Developer, October 1996; Available online: www.gamasutra.com/features/19970801/pathfinding.htm

[18] D. Appplegate, R Bixby, C. Chvatal, W. Cook, " Solving Traveling Salesman Problem ", Available online: www.tsp.gatech.edu

[19] K. Hoffman, "Traveling Salesman Problem", Available online:http://iris.gmu.edu/~khoffman/papers/trav_salesman.html.