Poseidon Team Description Paper
Robocup 2015, Hefei, China

Melina Farshbaf Nadi[1], Mahshid Farzan[2], Maryam Gohargani[1], Pooria Kaviani[3], Shiva Sadat Mahdavian[1], Rozhina Pourmoghaddam[2]

[1] Farzanegan High School, Robotics Research Group, No. 56, Shahid Sarparast St., Taleghani Ave., Tehran, Iran
[2] Tehran University, Enghelab Ave., Tehran, Iran
[3] Sharif University of Technology, Department of Computer Science, Azadi Ave., Tehran, Iran

{melinafn15, mahshid.farzan, mgohargani, pooria.kaviani, mahdavianshiva, rozhina.pourmoghaddam}@gmail.com

Abstract. This manuscript describes the algorithms and contribution of Poseidon 2015 to acquire the optimum and most efficient solutions for rescue agent simulation problems. New ideas in World-Modeling structure and the agents' decision were implemented. Each section explains new algorithms of this year's implantation in detail. This version of the Poseidon team is based on the sample code of the server with extensive changes in structure of agents and World-Model. In Poseidon 2015, computational geometry and artificial intelligence algorithms have been used for solving the problems of rescue agent simulation.[1]

Introduction

The purpose of the Rescue Simulation League is to decrease life and financial losses caused by natural disasters such as earthquakes, floods, etc. In order to achieve this goal, a large urban disaster is simulated and indicates the agents' actions in this situation. This simulation matches real world limits and problems as accurately as possible. Rescue simulation agents include ambulance team Agents, police force agents and fire brigade agents. The main task of the police force is to open the closed roads. The main duty of the ambulance team is to save lives and extinguishing the fire is the main duty of the fire brigade agents. In addition, all the agents are responsible to facilitate the other teams' tasks. The Poseidon team is trying to find solutions and improve available algorithms to solve agents' problems.[1]

2. World – Modeling and Communications

2.1 Agent Position

The center chooses tasks for each agent. It can decide more accurately, by knowing every agents' position. Also, having all agents' positions can be useful in searching, as we'll know which buildings have been seen and which remain unvisited.

We transfer this data (the agents' positions), using the radar. When agent receive this

data, they'll compute which buildings have been visited using vision areas and vision points. (The vision areas of a building are the areas that contain a point from which the target can be seen.) This will be inaccurate if we only send the area that the agent has occupied (i.e., if there is a long road, the agent may be standing on its center not being able to see buildings near the corners of the road.) To make this computation more precise, we must send the point that the agent is standing on. This will cause too much information to go through the radar, and the risk of not hearing it will be higher. So we decided to use another method called "Cellularization and Positioning". In this method, every area is divided into smaller pieces called "cells". By doing this we send two integers to determine each agent's position. One telling the area that the agent is standing on, and the other being the cell that the agent has occupied.

2.2 Cellularization and Positioning

We faced some problems last year. The positions of our agents which they communicated through the radar were not accurate enough and the tasks that were assigned in the center couldn't work properly because we couldn't estimate all the buildings that were seen by them. To solve these problems we decided to use "The Boustrophedon Cell Decomposition".

The "Boustrophedon Cell Decomposition (BCD)" is a method used in artificial intelligence and robotics for configuration space solutions.[2] Like other cellular decomposition methods, this method transforms the configuration space into cell regions that can be used for path planning. But we used this method for reasons other than path planning, so we modified this method and used it to increase agents' positions accuracy.

Cellularization algorithm is not suitable because of the mass of information we exchange each cycle. so we used "Quadtree" algorithm to solve this problem.[3]

The quadtree is a data structure appropriate for storing information to be retrieved on composite keys. Each vertex has four extensions, and it is most often used for the purpose of dividing new extensions into four regions repeatedly. Searching is guaranteed to be fast in optimized trees. We divide the map into four cells and after that we divide the cell that contains our agent to four other cells and we will continue this until it has enough precision. This will lower the radar's traffic by estimating buildings which have been seen by them without the agents communicating what they've seen through the radar. [Fig 1]

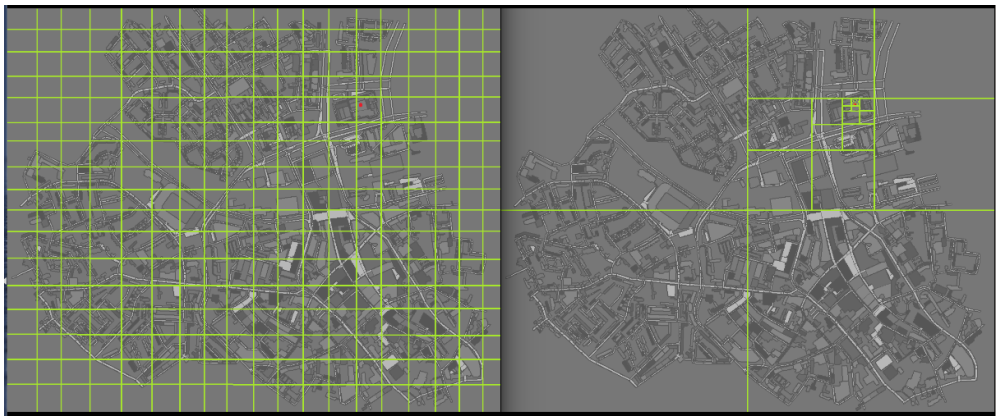

Fig 1. Comparison of Cellularization and Quadtree Cellularization - Eindhoven

2.3 Floyd

The Floyd–Warshall algorithm compares all possible paths through the graph between each pair of vertexes, until it finds the shortest path between that pair of vertexes.[4]

We simulated the map as a graph by assuming each area as a vertex and used this algorithm for finding the shortest path between each two areas of the map. [Fig 2]

Fig 2. Chosen Path with Min-Dist by Floyd-Warshall Algorithm

2.4 Optimizing Radar

The radar plays a significant role in a team's code. The center informs other agents of their target, the cleared ways are communicated through the radar, and last but not least, if a new fire point is seen by one of the agents, this information is transferred with the help of the radar.

While tracing a bug, we noticed the high radar traffic and decided to consider some ways to reduce it. We increased the precision of near areas of an area by appointing them based on corners of each area instead of the center. It affected the radar traffic drastically, as

we explained how it depends on the near areas in previous years.[5] [Chart 1]

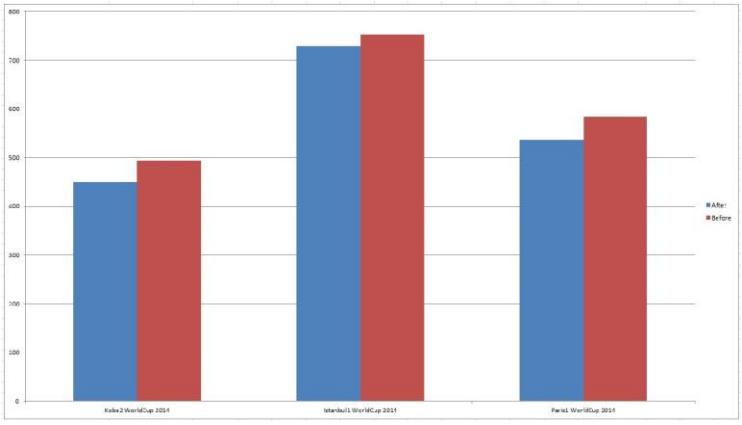

Chart 1. The Comparison of Our Radar Traffic Before and After Optimizing

3. Ambulance Team Agent

The ambulance team agent is one of the most important agents. Their coherence plays an important role in maximizing the number of civilians they save.

We realized that ambulance team agents are highly dependant on on the other agents' tasks, and if they function productively, ambulance teams are well ordered and structured. The radar is of importance in ambulance agents' duty as well, as information goes through it. The ambulance team agents' will operate drastically if the radar is more optimized.

4. Police Force Agent

The principle responsibility of the police force agent is cleaning the roads in a way that the other agents could perform their duties effectively and efficiently. We decided to use a new way of clearing which is faster and more optimized.

4.1 Sim Clear

We saw that some police force agents are not fast enough. (i.e., If a police force agent could reach its target by either clearing-moving-clearing, or moving-clearing, it would choose the first option, instead of the shorter one.) In order to prevent bad choices, we decided to use simulating. Police force agents have two possible commands in each cycle: moving, and clearing. Like BFS algorithm, the nodes of the graph are move, or clear command and this graph will grow continually, until it reached the target. [Fig 3]

After the simulated move command, we save each agent's new position, and after a simulated clear command, we save the new shape of the blockades that have been cleared. We choose the series of commands that reached the target before others.

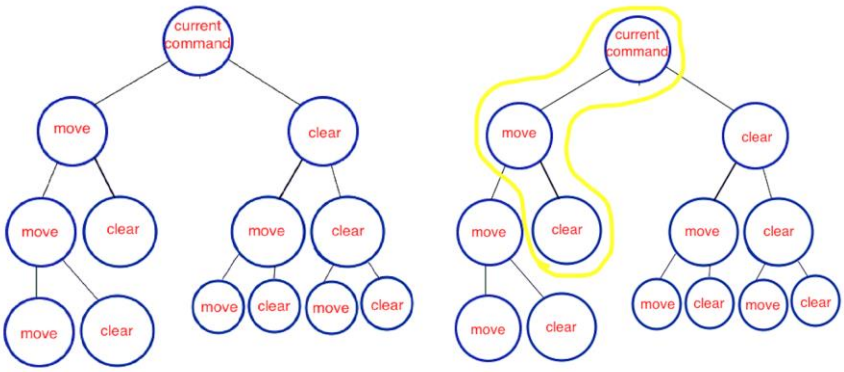

Fig 3. Sim Clear Charts

5. Fire Brigade Agent

Fire brigade agents' main duty is preventing the fire points from spreading.

5.1 Scenario Test Cases

We figured that any part of code that we change or add might affect other aspects of it. A really effective method is to have a way to test every part and make sure that everything is working before moving on to the next part.

One new method to test different parts of a big project is making a test case for each part. For our team, a rescue simulation, scenario test cases are the best option we. We have made different scenario test cases for each and every part (e.g., search, zoning, reachability, etc.) By doing this, not only will we be able to test new parts that we've added, but also we'll be able to check other parts as well and make sure that they've not been affected. [Fig 4]

We are currently using this only for fire brigade, but we have a long term plan to implement this method for other agents as well.

Fig 4. Scenario Test Case of Istanbul – Fire Zoning

6. Search

It has been proved that a good search can affect the operation of a team in a positive way. Search is one of the high priorities of our team because it affects both fire brigades and ambulance team agents' duties. Having a good search algorithm results in more saved civilians and more extinguished buildings.

6.1 Random Search

We admit that a search with random choices, which are more likely to visit the areas that haven't been seen or more time has passed from the time they've been seen, would work for search challenge. Random search chooses a direction from our position. We set weight for each neighbor, which is the sum of that direction's areas' weights in a certain BFS distance limit. Each area's weight is a number based on its BFS distance and the last time it has been seen. [Fig 5]

We used random algorithm in a way that is more probable for a higher weight to be chosen, but there is always a chance for the lowest weight, as well. One privileged aspect of random search is that there is no need for zoning. The center chooses zones for each agent, and informing them of their task takes some of the radar traffic. Also, different agents are assigned to different zones. If an agent gets stuck, dies, or is prevented from its task in any other way, this will interrupt the search process. Another reason we choose this method is that if we randomly choose a place that has been seen before (with a low weight) there is always a chance to find a newly burnt areas because new fire points appear randomly.

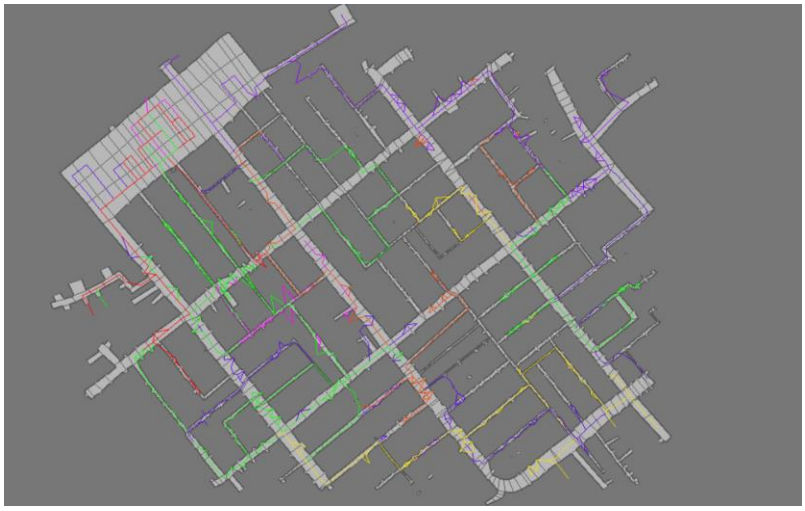

Fig 5. Random Search - Kobe

6.2 Comparing Different Search Methods

After considering different algorithms for a search method, we reached the conclusion that we must test different searched on the same circumstances and choose the

best option.

We assigned different agents to do three different search algorithms, and had the number of the seen buildings printed. (BFS search is specified with red and random search with blue color in the charts.) We also tested sharing information and not doing so, in each map. Our random search seems to work adequately on maps with high data sharing. [Chart 2-3] However, the move command that was sent to server each cycle was path which contained no more than one area, as our random search is at the coding stage. We are still trying to improve the productivity of our random search with new ideas.

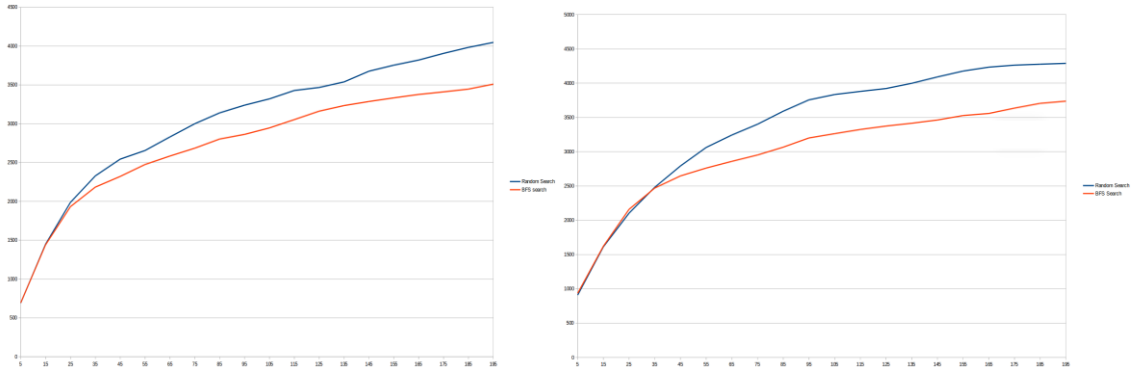

Chart 2. Comparison of Different Searches on Paris with (left) and without (right) Data Sharing

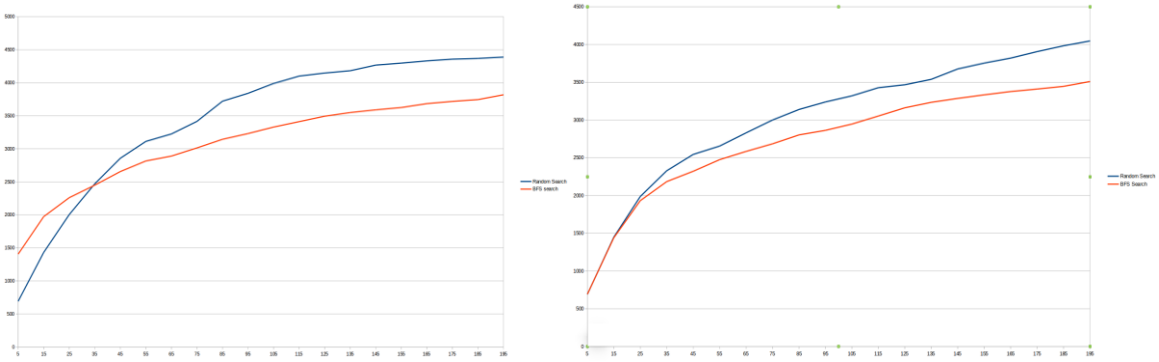

Chart 4. Comparison of Different Searches on Istanbul with (left) and without (right) Data Sharing

6.3 Highway

Having a clear path is helpful to all of the agents' tasks, especially the ambulance team and the fire brigade. One way of doing that is clearing ways which give access to many buildings, first, and then clear other parts of the city. We found a simple algorithm for that. To reach that purpose, we decided to use BFS algorithm to find the Highways. We randomly pick about four or five buildings (not near each other, preferably.) The BFS starts from those buildings and ends with each building in the city. We mark each road that the BFS passes through. The roads that have the highest mark are our Highways and police forces choose them as its high priority to be cleared. [Fig 6]

Despite this algorithm's advantages, there are some cases that make it a little less efficient. We find about four or five Highways for each map, but sometimes it's not the best thing to clear them all because they give access to the same buildings. We ignored these cases because even if clearing these additional paths means wasting some time, it can be advantageous as it means more passable ways and accessible areas. Due to this, we decided that this algorithm can be useful in spite of its cons.

Fig 6. Kobe and Berlin Highways

7. References

1. Abderezaei A., Farzan M., Ghaffari K., Kaviani P., Khodaparast M., Mahdavian Sh., Nasehi F., Pourmoghaddam R.: Poseidon Team Description Paper, 2014
2. Choset, Howie. Coverage of Known Spaces: The Boustrophedon Cellular Decomposition
3. Raphael Finkel and J.L. Bentley (1974). "Quad Trees: A Data Structure for Retrieval on Composite Keys". *Acta Informatica* **4** (1): 1–9. doi: 10.1007/BF00288933.
4. Weisstein, Eric. "Floyd-Warshall Algorithm". *Wolfram MathWorld*.
5. Abadi R., Behazin B., Eslami M., GHasemi Zavie Sadat F., Moradi Dadkhah B., Pourmoghaddam R., Ramezanian M., Rezayat Z., Seyed Majidi N., Shirvani-Mahdavi N., Vartanian A., Kaviani P.: Poseidon Team Description Paper, 2011