RoboCup 2018 – TDP Rescue Agent Simulation RoboAKUT (Turkey)

H. Levent Akın, Okan Aşık, Gökçe Uludoğan, and H. Kübra Eryılmaz

Boğaziçi University, Turkey
[akin,okan.asik,gokce.uludogan,hatice.eryilmaz]@boun.edu.tr
http://robot.cmpe.boun.edu.tr/rescue/

Abstract. RoboAKUT is a multi-agent rescue simulation team that competes in the RoboCup Rescue Agent Simulation League(RSL). It has been competing in the RoboCup competitions since 2002 and has won the *First Place* in the agent competition in RoboCup 2010 and in RoboCup IranOpen 2017. RoboAKUT code base was rewritten to use Agent Development Framework(ADF) in 2017. The behavior of the agent is divided into two tasks; target selectors and action modules. The target selectors choose the target for the agent and action module calculates the simulator action to achieve the target. To be able to carry out these tasks, these modules uses path planning, clustering, and exploration algorithms. We take the greedy target selection approach and also added reward based exploration approach. We use the Hungarian algorithm for the optimal allocation of agents to the clusters. We show that the total distance taken by all agents to reach the cluster centers is reduced.

1 Introduction

RoboAKUT has been competing in the RoboCup Rescue Agent Simulation League since 2002. We won the first place in RoboCup 2010 and in RoboCup IranOpen 2017, and the third place in RoboCup 2017. We build our agent strategies using greedy algorithms. As shown by Aşık and Akın [2], greedy algorithms have quite robust performance in task allocation in search and rescue task. Agent Development Framework (ADF) requires the development of two types of modules; target detectors, and action generation. Target detector modules choose a target for the agent to act on. For example, building detector module chooses a building on fire to extinguish. The action generation module calculates the required simulation actions. For example, if the building selected by the building detector module is away from the agent, action generation module calculates a path to the target and when gets near extinguishes the building.

We create clusters for every agent types using the k-means clustering algorithm such that the same agent types cover the whole map. When choosing greedy targets, target detectors firstly choose the targets from their own clusters, if there are any targets in the cluster. If there is no target in the cluster, the target detectors select their targets from whole map.

After creating clusters starting from random cluster centers, we assign one agent to one cluster. Previously, we assigned agents greedily one by one. We get one agent find the closest cluster center to the agent and assign the cluster to that agent. However, this

is not as effective as expected. Therefore, we use Hungarian assignment by calculating the shortest path distance of agents to the cluster centers. This assignment results in the optimal assignment of agents to the clusters.

We use Dijkstra's shortest path algorithm to calculate a path between two locations. However, we also keep track of blocked and opened roads regularly so that the agents do not move on the path that is blocked. We use a priority queue to improve the computational complexity of Dijkstra's algorithm.

As an exploration method, we use a reward based algorithm. The agent keeps track of the locations he/she has moved. When there is no target to act on, the agent chooses an exploration target based on the reward of the targets. The reward of the target is calculated as the count of unexplored locations to reach the target. Therefore, the agent chooses the target that leads more exploration.

2 Modules

2.1 Clustering

The clustering algorithm is used to create a hierarchical task allocation for the agents. By assigning every agent to a particular area (roads and buildings), we aim to achieve a balanced distribution of the agents on the map. We have building clustering algorithm for *Fire Brigade* and *Ambulance Team* agents and road clustering algorithm for *Police Force* agents.

Building Clustering The buildings are clustered by k-means clustering algorithm [1]. We set the number of clusters same as the number of agents that are same type, such as *Fire Brigade* agents. Firstly, the algorithm sets random cluster centers. Then, it iterates over the buildings and adds them to the closest cluster based on the air distance to the cluster centers. Based on the members of the cluster, a new cluster center is calculated. Finally, this process continues for a given number of iterations. An example visualization of building clustering can be seen in Figure 1.

Road Clustering We would like to cluster the roads as a connected graph. Since the *Police Force* agents need to clear roads, they need to move on the connected roads. Therefore, if the road cluster is not a connected graph, the agent needs to move on the other agents' clusters. Also, we use task allocation scheme using leaf of the graph as explained in Section 3.1.

We could use the k-means algorithm, but this would result in a disconnected graph since the distances are measured as air distance. Also, some members of the clusters could be close by air distance but might be very far away by the path distance (distance that is needed to be covered by the agent). Therefore, we need to measure the distance between the cluster center and the road by path distance. However, this requires the shortest path calculation at the assignment of every road. This algorithm performs very slow compared to air distance calculation. Therefore, we propose to use air distance, but after k-means clustering, we use an algorithm to connect the disconnected graphs of the cluster by the shortest path. Finally, the roads on the connecting shortest paths

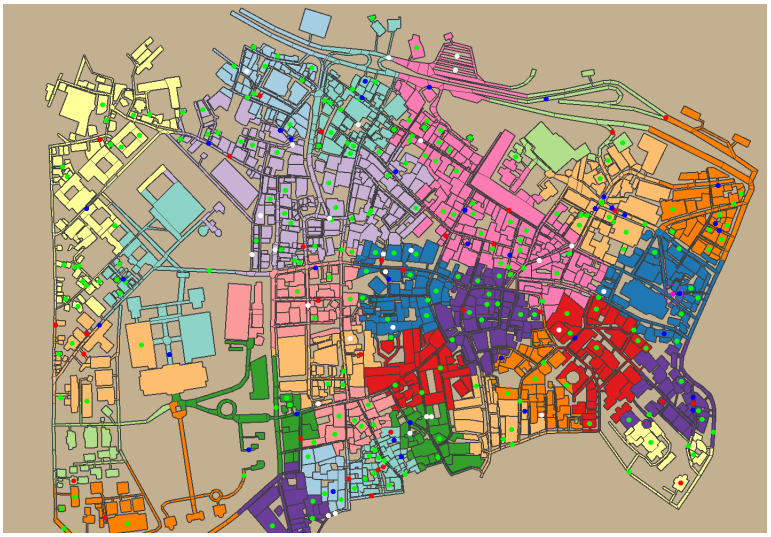

Fig. 1: An example visualization of building clustering. The different colors represent different clusters, but if two clusters with the same color becomes side to side they seem like a single cluster (such as pink colored cluster).

are also added to the clusters. An example visualization of road clustering can be seen in Figure 2.

Hungarian Agent-Cluster Assignment In the previous years, we used greedy assignment of agents to the clusters. The greedy assignment iterates the agent and calculates the shortest cluster center to the agent and assign this cluster to the agent. However, this method is not as effective as expected since the total distance taken by whole agent team is not optimal. Therefore, we use the Hungarian assignment algorithm [4]. We first calculate a cost between every agent and every cluster. A shortest path is calculated from the agent's current position to the cluster centers. We set the cost of the assignment as the travel distance of the agent to the cluster center. We want to find the minimum total cost of tasking all agents to the assigned cluster centers. The Hungarian assignment algorithm calculates the assignment that results in the minimum total cost. In Figure [**?**], we show the percentage reduction in total cost of the Hungarian assignment in place of Greedy assignment. We show that on average the agents will travel 10% to 15% less to reach their cluster centers.

2.2 Path Planning

The path planning algorithm calculates the shortest path between two positions on the map. The map of RSL simulator consists of *roads* and *buildings*. They construct a connected graph where the edge between vertices are defined as the neighbors. The distance between neighbor *roads* and *buildings* are calculated as the distance between the

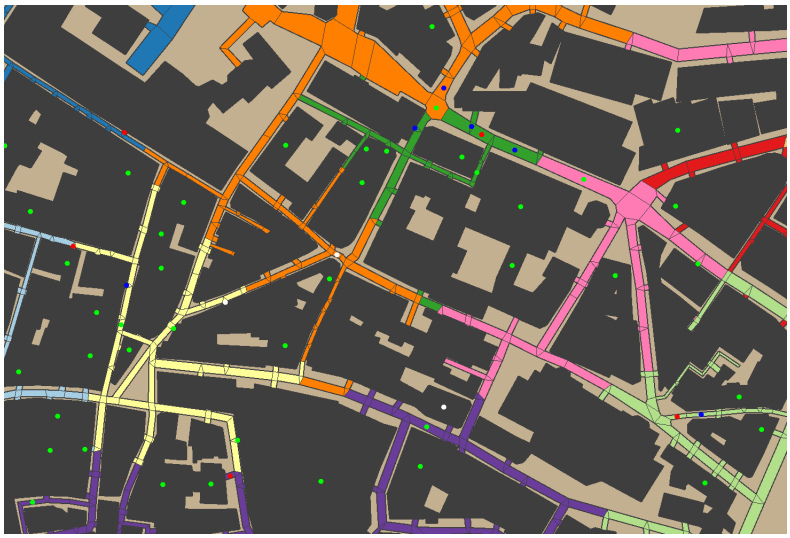

Fig. 2: An example visualization of road clustering. The different colors represent different clusters. These clusters create a connected graph so that some roads may be member of more than one clusters.

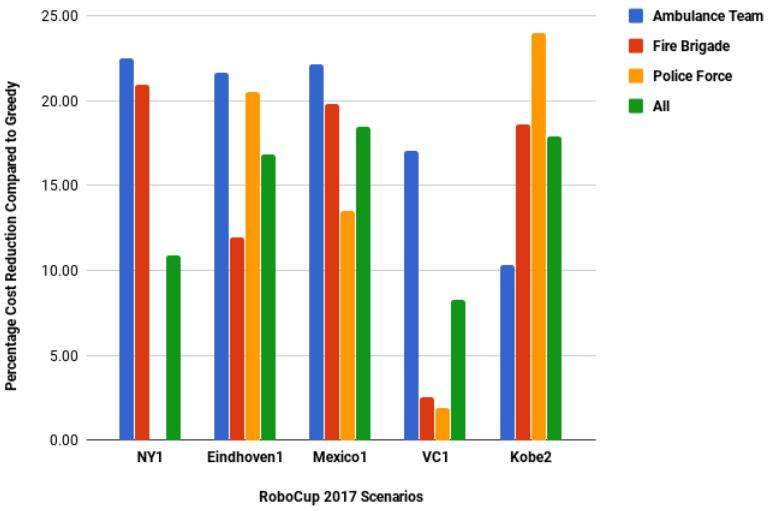

Fig. 3: The percentage reduction of the total travel distance by agents compared to greedy assignment.

centers of these areas. We use Dijkstra's shortest path algorithm [3] to calculate the shortest path between two areas. Dijkstra's algorithm starts from the start vertex and builds a path using the shortest path from possible neighbors. The complexity of the algorithm is $O(|E| + |V|^2)$, where E stands for the number of edges (neighbors) and V stands for the number of vertices (areas). At every iteration the algorithm chooses a vertex having the shortest path starting from the start vertex. By implementing the list which keeps the paths sorted according to their distance as the priority heap, we can reduce the complexity to $O(|E| + |V|log(|V|))$.

We also keep track of the blocked roads and temporarily do not calculate them as a neighbor for T time steps only for *Ambulance Team* and *Fire Brigade* agents. As the agent moves on the map, the perceived roads are updated. This ensures the calculation of a path that is clear of blockades. After T time steps, the blocked roads are reset and assumed to be clear of blockades until the agent discovers that the road is still blocked.

3 Strategies

We do not use center agents to coordinate agents. The platoon agents use decentralized greedy task allocation scheme.

3.1 Police Force

Police Force agent uses the *Road Detector* and the *Clear Action* modules. The detector module selects the target and action module calculates the action to carry out the selected task.

Road Detector The road detector chooses a *clearing* target with a decision tree that is tuned through experiments. The decision tree implements a priority based targets. At every decision level, the agent checks the condition for the behavior of that level. For example, if the agent perceives a human that is stuck in a blockade, clearing this blockade has the highest priority. The behaviors of the *road detector* is as follows starting from the highest priority:

1. **Clearing for a Human:** The agent looks for humans (civilians or mobile agents) in its world model. For every human, the agent checks whether the human is stuck and alive. A human can be stuck in two ways; it is either just inside a blockade (a polygon inside a road polygon), or a blockade is just in front of the building where the human stays in. The agent chooses the closest of the stuck humans if there are more than one stuck human.
2. **Clearing the Cluster:** If there is no agent to rescue in the world model of the agent, the agent will clear the roads of its own cluster. Every agent has an assigned cluster. The clustering algorithm assigns a cluster to every agent based on the distance of the agent to the center of the clusters. The algorithm also ensures that every cluster constructs a connected graph. The agent keeps track of the visited roads. The ultimate aim of this behavior is to maximize the number of cleared roads. Therefore, we formalize the problem as a task allocation where the tasks are the *leaf* nodes of

the cluster graph. We define the *leaf* as the node having only one edge. The utility of a *leaf* node is calculated as the number of the *unvisited* roads that agent needs to take to reach the *leaf* node. The path to a *leaf* node is calculated using a shortest path algorithm. The behavior chooses a *leaf* node having the highest utility value (see Figure 4)

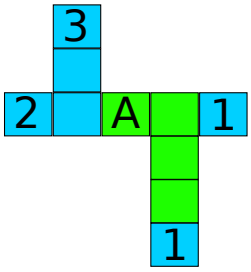

Fig. 4: An example illustration of cluster clearing behavior. The blue and green squares represent the roads in the cluster. The visited ones colored as green. *A* represents the agent position and values at the leaf nodes show the utility values calculated as the sum of unvisited roads on the path.

Clear Action The module gets a target position to clear. However, that requires the clearing of the blockades that are on the path to the target. Therefore, as the first step of the algorithm, the shortest path between the agent and the target position is calculated. The simulator clear action is a vector starting from the position of the agent. The module creates clear vector towards the next position on the shortest path. If this clear vector intersects with a blockade, the module needs to clear this blockade. If the clear vector does not intersect, a new clear vector is created from the end point of the previous one. This process continues until the clear vector reaches the end of the path. If the clear vector intersects with a blockade, the module checks whether the agent can clear the intersected blockade. If the agent is able to clear the blockade, a clear simulator action is created and sent by the module. If the agent is not able to clear the blockade, a simulator move action is created and sent by the module.

If the target selector module chooses a position to rescue a human from blockade, the clear action needs to calculate a clear vector towards the position of the human. By design, the target selector can choose only the areas to clear, therefore the clear action needs to calculate the position of the human from selected position.

3.2 Ambulance Team

Ambulance Team agent uses *Human Detector* and *Transport Action* modules.

Human Detector The human needs to be rescued when it is buried or wounded. If a human is buried, the agent needs to rescue the agent from the debris. If a human is wounded, the agent needs to transport the human to the refugee. The human detector

module assigns humans that need to be rescued or transported to the agents. This module makes a central assignment similar to the building detector based on the distance of the agents to the humans which need to be rescued.

Transport Action The transport action calculates a simulator action for the selected human. If the target human and the agent is not on the same position (road or building), the module calculates a shortest path to move to the position of the target. Once the agent and the target is on the same position, the module creates and sends a rescue action if the human is buried and the module creates and sends a load action if the human is not buried and is wounded. After the human is loaded to the agent, it is transported to the shortest refugee and unloaded there.

3.3 Fire Brigade

Fire Brigade agent uses the *Building Detector* and the *Extinguish Action* modules.

Building Detector The building detector algorithm chooses building targets in two ways: *reactive planning* and *general allocation*. In *reactive planning*, the module gets all the buildings in the extinguish range that are on fire and chooses randomly one of them. Firstly, in our experiments, we see that the reactive behaviors are quite effective despite their simplicity. Secondly, since we do not coordinate the firefighting agents, we reduce the resource inefficiency due to error in allocations. Random selection ensures the equal distribution of agents for a given set of targets. The algorithm is concurrently running for every agent and the agents may not know the current position of other agents. If an agent does not get any message or does not perceive other agents, it uses the last position of the other agents. If the agent does not have information about the other agents after the start of the simulation, it uses the initial position of the agent. Although this looks like an ineffective method, since agents generally moves around their starting position, it becomes a good approximation for the position of the agent.

If there is no building in the extinguish range, the module uses *general allocation* and assigns buildings on fire to all the agents based on the distance between the building and the agent. The algorithm iterates over the buildings on fire and gets two closest agents to the building at every iteration. If this is the agent that is running the algorithm, it is assigned to the current building and iteration ends. If the agent is not one of two agents, these agents are removed from agent list, and the algorithm continues with the next building.

Extinguish Action The extinguish action calculates a simulator action for the selected building that is on fire. If the agent needs to refill its tank, the module calculates a shortest path to the closest refugee. If the agent has water, it creates and sends an extinguish simulator action for the target building. If the building is not on the extinguish range, the agent creates and sends a move simulator action by calculating the shortest path to the target building.

4 Conclusions

We implement a decentralized greedy strategy for the rescue task allocation problem. The agents reactively choose their targets such as closest building that is on fire. If they perceive another agent, they implicitly coordinate their target selections based on the priority calculated by their ids. We also keep track of the agents' movement on the map to calculate better search targets and marking blocked paths. We have shown that the approach has promising results by finishing the first title in RoboCup Iran Open 2017, and third title in RoboCup 2017.

References

1. Alpaydin, E.: Introduction to machine learning. MIT press (2014)
2. Aşık, O., Akın, H.L.: Effective multi-robot spatial task allocation using model approximations. In: Behnke, S., Sheh, R., Sarıel, S., Lee, D.D. (eds.) RoboCup 2016: Robot World Cup XX. pp. 243–255. Springer International Publishing, Cham (2017)
3. Dijkstra, E.W.: A note on two problems in connexion with graphs. Numerische mathematik 1(1), 269–271 (1959)
4. Kuhn, H.W.: The hungarian method for the assignment problem. Naval Research Logistics (NRL) 2(1-2), 83–97 (1955)