RoboCup 2017
Rescue Simulation League Team Description
GUC_ArtSapience (Egypt)

Patrick Attia, Youmna Salah, Ather Hegazy, Mark Moheb, Ahmed Nader, Mohamed Ahmad Azab, Veronika Wissa, Ahmed Osama, Maggie Moheb, Ahmed Diaa, Sara Nagui, Miada Nour, Mark Morcos, Nada Hammad, Dina Helal, Heba Aamer, Fadwa Sakr and Slim Abdennadher

German University in Cairo, Egypt
[fadwa.elhussini, dina.helal, slim.abdennadher]@guc.edu.eg

Abstract

This paper describes the contribution of the GUC_ArtSapience team in the Rescue Agent Simulation competition in RoboCup 2017. The following sections present the different approaches that are adopted by the team since last year [8]. The changes this year are mainly related to target allocators. Decision making in Police Force Agents and Fire Brigade Agents this year includes reinforcement learning. Moreover, Hungarian algorithm is applied for centers in Ambulance Team Agents to allocate tasks. Finally, we added slight change to our clustering module, where we used KMeans || as a speed up for KMeans ++ algorithm.

1 Introduction

The Agent Development Framework (ADF) provides an interesting test bench for many algorithms and techniques in the field of Artificial Intelligence and Multi-Agent Systems. It is built on top of the Rescue Simulator by providing new structured modules such as *Clustering*, *PathPlanning* and *TargetAllocator* modules.

In the ADF, agents are extended from the RMAS agents in a more structured way. Moreover, agents have two different handlers that should be implemented; the tactic aspect of the agent, and the center one.

This paper explains the modifications done this year to improve the agents' performance. The changes mainly adopts machine learning approaches for the target allocation tasks; especially reinforcement learning and supervised learning. The rest of the paper is divided as follows, section 2 describes the changes to our clustering, path planning and target allocator modules. In section 3, we describe our preliminary results and conclusions.

2 Modules

2.1 Clustering

Last year, we used k-means++. K-means++ is one of the most known modifications to the initialization of k-means algorithm, where only the first center is chosen at random

and the rest of the centers are chosen according to a probability, which prefers data points, that are further away from the previously chosen centers. It was a great enhancement over the regular k-means, where random points could be selected very close to each other and would take a lot of time until the data reached a stable state.

A disadvantage of the k-means++ is that it needs n iterations to set each of the k centers; where n is the number of data points. Hence, if the number of clusters is huge, the algorithm will take long computation time accordingly. As an enhancement for k-means++, we implemented k-means‖ [3], which is a parallel version of k-means++.

To set the base for the algorithm, here are the formulas and notations used by the algorithm: X is the data set of points that need to be clustered, C is the set of centers and a centroid

$$Y = \frac{1}{|Y|} \sum_{y \in Y} y$$

The cost of Y with respect to C is

$\phi_{\mathrm{X}}(C) = \sum_{x \in X} {d_{\mathrm{C}}}^2(x, C)$; where $d(x, y)$ is the minimum distance between x and y.

In any k-means algorithm the main goal is to find a set C that minimizes $\phi_{\mathrm{Y}}(C)$. K-means‖ is very similar to k-means++ as we start also with one randomly chosen center point. But instead of choosing a single data point in each iteration of k-means++, we sample l points in each iteration, l being a chosen oversampling factor, for this to be done after the random point is chosen we compute

$$\psi = \phi_{\mathrm{X}}(C)$$

and then start sampling each point x with a probability

$$ld^2(x, C)$$

and update $\phi_{\mathrm{X}}(C)$ accordingly. This process is repeated for O(log ϕ) iterations. We are left with kx points which we then re-cluster until we reach the k initial centers that are needed for the regular k-means. Figure 1 is an example for k-means‖ initialization.

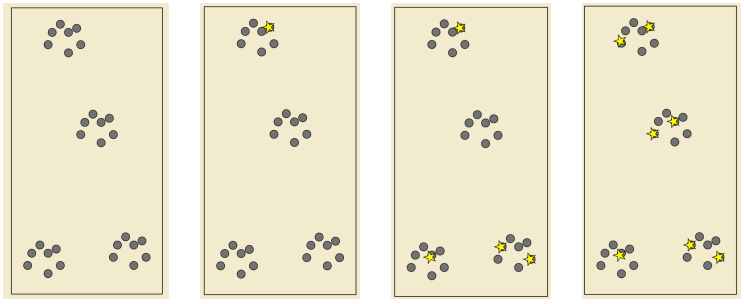

Figure 1: K-means‖ initialization example

This introduces an overhead in the initialization of the algorithm, but it reduces the probability of a bad initialization leading to a bad clustering result and helps the k-means clustering to reach a stable state faster.

2.2 Path Planning

As our last year's implementation, we still use *Dijkstra* searching algorithm for path planning module. It finds the shortest path between two nodes in the graph. *Dijkstra* is a greedy approach where it depends on a heuristic function that affects the order in which the nodes are explored. The heuristic we used last year was the distance between the nodes, given that the roads are clear. This will result that agent won't get a path if all the paths to the target contains blockades. In order to enhance this approach, we modified the heuristic used in the algorithm. The heuristic now is affected by the distance and number of blockades in the roads. In case all the paths to the target are blocked, we will get the path containing the least number of blockades. The motive for this change is that the path containing the least number of blockades, most probably will be cleared faster than other paths.

2.3 Target Allocators

2.3.1 Centers

In the ADF last year, we were not allowed to use centers in the technical challenge competition. This year we compared the use of a greedy algorithm; that we implemented last year in the rescue agent simulation competition, and Hungarian algorithm in task assignment done by the centers. As seen in figure 2, the Hungarian algorithm outperforms the greedy algorithm in three out of five test maps. However, the decline in the performance of the Hungarian algorithm is due to the algorithm out running the time of the step preventing the agents from completing the think phase. We used the Ambulance Team Agents to gauge it against the greedy algorithm since their performance can be quantified using the number of civilians rescued which can be acquired easily.

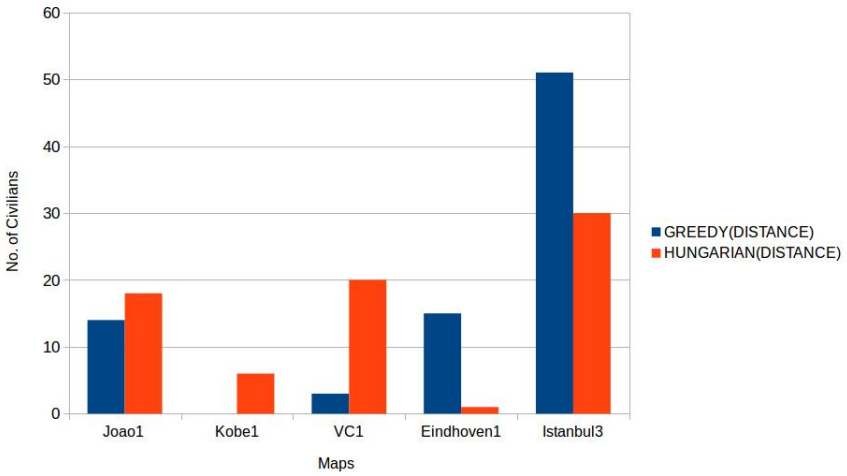

Figure 2: Greedy Algorithm vs. Hungarian Algorithm

Figure 2 compares the number of civilians rescued in five maps between an algorithm

that chooses the closest agent to the civilian and Hungarian algorithm where the cost used is the distance between the ambulance agent and the civilian. We are going to optimize the Hungarian algorithm in our coming implementation and change the cost function such that the cost would be equal to the sum of three parameters: the distance from the ambulance agent to the target, the distance from the target to the refuge, and the estimated time of death of the target calculated by our learning model[12].

2.3.2 Police Force Agents

Police Force Agents play an essential role which has a direct effect on other agents performance. They are mainly concerned with clearing blocked roads so as to facilitate the movement of other agents, Fire Brigades and Ambulance Team Agents, to their respective targets. We have therefore put a great focus this year on optimizing the performance of the Police Force Agents.

There are 2 main challenges concerning the police force performance. The first is to choose the optimal blockade to clear, in a way that assures clear paths for other agents. The second challenge is to optimize the clearing action to avoid having dentate blockades that trap other agents, and to clear the least that allow agents to pass.

We present this year a novel approach to enhance the Police Force Agent's choice of blockades to clear. Seeking the aid of Machine Learning, we are using Reinforcement Learning[1] to train the police agents to remove the blockades efficiently. Efficient actions should lead Fire Brigades to control fire before spreading and rescuing buried civilians by the Ambulance Team Agent. Our approach is inspired by related findings introduced in [2, 1, 4].

Environment: Our learning model consists of states, actions and rewards. A state is represented by the agent's world information and the clearing requests received by other agents.
Actions are either generated from received requests or by the agent himself. Our Defined actions are:

- Clear the closest blockade to me.
- Clear the closest blockade to the center of the cluster.
- Clear the closest blockade to humans either civilians or other agents.
- Clear the smallest blockade, with respect to its area.

The agent chooses an action according to the policy followed. Each Action-State pair has a value which represents the corresponding pair. The policy evaluates this value to output an action. We use Epsilon-Greedy policy which means that the agent chooses, with a probability 1-Epsilon, the action associated with the highest value and chooses randomly otherwise.

[1]Reinforcement Learning is a Machine Learning approach, in which an agent is trained to reach the optimal policy by trying to maximize the rewards received from doing a set of actions. [6]

Action selection is also affected by a set of parameters in the environment. Each parameter has a corresponding weight, which gets updated according to the parameter's effectiveness in the action chosen. Examples for these parameters are the number of buried civilians around a blockade and the distance between the blockade and the police agent.

Rewards: Reward is a function that calculates the effect of the chosen action. In our environment, we set all rewards to be negative. The reason for negative rewards is that in natural disaster environments, the situation only gets worse, thus the agent must always get a penalty and should try to reduce it somehow. The value of rewards vary according to the action's effect; some rewards have smaller negative values while others have bigger ones.

Learning: We followed the Lesson-by-Lesson approach during the learning process to overcome the problem of not being able to identify the correct action to take. The Lesson-by-Lesson approach means to train the agent in phases which have gradual increase in difficulty. For example we train our agent first in an environment that has only 1 blockade, then gradually we increase the number of blockades and the difficulty of choice.

An extra modification in the learning process is to follow the learning approach presented in [4] by Google's Deepmind[2] company. The idea is to parallelize the learning process for multi-agent system by having 2 asynchronous threads. A central learner thread that updates the State-Action values and N worker threads where N is the number of agents. These worker threads act as experience collectors where they send their experience to the central thread. The central thread appends the collected experience to a shared memory buffer. This decoupling between the learner and collector threads helped significantly in decreasing the learning time.

2.3.3 Fire Brigade Agents

So far various methods were proposed for decision-making of Fire Brigade Agents. Among these methods is reinforcement learning which is used in [11], [7] and [10] to determine the number of Ambulance Team Agents which should cooperate to rescue a civilian and for the Fire Brigade to learn to allocate tasks and choose the best building to extinguish to maximize the score and how to be allocated in the city respectively. The common parameter between these approaches is that they all used reinforcement learning yet they all lacked homogeneous and heterogeneous communication between agents, that is communication between team members and between teams respectively. Our cooperative social environment describes best the effect of communication and cooperation on learning, we learn not only by trial and error but also through cooperation by constantly sharing information, experience and knowledge. That was indeed the case with agents in virtual simulations as shown in [14], intelligent cooperation between agents enable them to conduct cooperative learning, speeding up the learning process of each individual agent since each agent have more resources and better chances of receiv-

[2]https://deepmind.com/

ing rewards. There are two types of information that agents could share: sensational information and policies,that is information about the current location of the fire and that could be shared between both homogeneous and heterogeneous agents, within and between teams, while information regarding policies such as best action done in a similar situation is shared only between homogeneous agents, within the same team. We are intending to incorporate the reinforcement learning technique to both the centres (if exists) and the Fire Brigades. Since not all map configurations have centers, incorporating a learning model to fire agents will be considered. However, that might result in yet another problem which is having several agents targeting the same fire-location while ignoring other fire-locations, that is fire brigades might cluster instead of spreading over existing fire locations. Such a problem was already solved using the previous approach of clustering used in the RAS [9] and [8]. The agents learning model would be done through supervised learning, as when the center is assigning tasks to the agents, the agents start learning the pattern in their assignments for the current scenario. Thus, if the map does not contain a center, the agent can take a decision similar to what the center would have given it.

Our Model: We plan to extend and implement the work discussed by [1, 14] in an ADF environment. First, we will discuss the proposed model by [1], then we will introduce our approach for applying different communication models discussed by [13, 5, 14] on the proposed model as well as including supervised learning on top of reinforcement learning for agents. In [1], they defined the following 5 actions to be executed by the agents:

- Extinguish the building that has the lowest temperature.
- Extinguish the nearest fiery building to civilians.
- Extinguish the nearest building to the center of the city.
- Extinguish the nearest building to the fire brigade.
- Refill with water.

In our model agents are distributed into clusters where each agent is responsible for extinguishing fire from buildings in its cluster. Consequently, we can exclude the 3^{rd} action. In addition to the rewards mentioned later, we will add a reward of -1/step if the agent runs out of water. The reward function was defined by [1] as follows:

- The agent is rewarded -1/step for each building that is extinguished and damaged of too much water.
- The agent is rewarded -2/step for each fiery building.
- The agent takes a reward of -3/step for each burned out building.
- The agent takes a reward equals to the civilian's health points - 1000 at each time step for each civilian.

Although this model seems quite fulfilling for a fire brigade system, it lacks agents coordination (it is agreed that agents which coordinate/communicate outperform independent agents). Agents doing the same task can learn different things due to interacting with different parts of the world. Thus, exchanging policies between agents led to a faster convergence of these agents. [13, 5] introduced different scenarios for communication between agents. We plan to implement the following "policy exchanging" settings and test which setting would lead to a better performance:

- Averaging the learned policies between agents after a certain number of steps (different values will be tested to know the best number of steps to be used with other parameters).
- For each (s, a) pair where s represents a state and a represents an action, we set $Q(s, a)$ for all the agents to the value of the $Q(s, a)$ of the agent which performed action a in state s the highest number of times after a certain number of steps (this agent will be the most experienced one).

2.3.4 Ambulance Team Agents

Ambulance Team Agents play an essential role in the simulation. Their role involves saving the highest number of civilians and rescuing the highest number of buried agents as well. Our last year approach was to prioritize the targets (other agents, civilians) in their clusters, by ordering the agents and the civilians according to their estimated death time (ETD) giving the human; either civilian or agent, that will die first a higher priority. The Estimated death time equation 1 is calculated using a trained machine learning model [12] from RAS, that uses linear regression and collects data after training it on a lot of maps by providing the model with input (HP, Damage, buridness).

$$ETD = -0.009 * hp - 0.9197 * damage + 0.2056 * buridness + 328.291 \quad (1)$$

This year we give the buried agents higher priority over civilians. So this will allow them to be rescued first in order to be able to continue their respective tasks. Moreover, in case there are neither hurt civilians nor buried agents in the agents' cluster. The agent prioritize the reported targets in other clusters according to the distance and ETD.

3 Preliminary Results and Conclusion

Our changes in the code are still under experimentation. The initial results of the enhancements are promising. The results will be available by the camera ready version.

In conclusion, this paper explained the new strategies being implemented by the team since last year. The approaches used attempt to improve the performance of the agents by adopting machine learning techniques. Moreover, slight enhancements to the clustering and path planning modules were introduced.

References

[1] Abbas Abdolmaleki, Mostafa Movahedi, Sajjad Salehi, Nuno Lau, and Luís Paulo Reis. A reinforcement learning based method for optimizing the process of decision making in fire brigade agents. In *Portuguese Conference on Artificial Intelligence*, pages 340–351. Springer, 2011.

[2] Omid Aghazadeh, Maziar Ahmad Sharbafi, and Abolfazl Toroghi Haghighat. Implementing parametric reinforcement learning in robocup rescue simulation. In *Robot Soccer World Cup*, pages 409–416. Springer, 2007.

[3] Bahman Bahmani, Benjamin Moseley, Andrea Vattani, Ravi Kumar, and Sergei Vassilvitskii. Scalable k-means++. *Proceedings of the VLDB Endowment*, 5(7):622–633, 2012.

[4] Shixiang Gu, Ethan Holly, Timothy Lillicrap, and Sergey Levine. Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. *arXiv preprint arXiv:1610.00633*, 2016.

[5] R Matthew Kretchmar. Reinforcement learning algorithms for homogenous multi-agent systems. In *Workshop on Agent and Swarm Programming*, 2003.

[6] Stephen Marsland. *Machine learning: an algorithmic perspective*. CRC press, 2015.

[7] Ivette C. Martínez, David Ojeda, and Ezequiel A. Zamora. *Ambulance Decision Support Using Evolutionary Reinforcement Learning in Robocup Rescue Simulation League*, pages 556–563. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

[8] Sameh Metias, Mahmoud Walid, Mohamed Farghal, Ahmed Osama, Patrick Attia, Salma Amr, Ebrahim Elgamal, Maggie Moheb, Dina Helal, Menna Bakry, Fadwa Sakr, and Slim Abdennadher. Robocup 2016: Technical challenge team description paper, guc artsapience, egypt. Robocup, 2016.

[9] Sameh Metias, Mahmoud Walid, Mina Sedra, Ahmed Jihad, Salman ElDash, Maggie Moheb, Mohab Ghanim, Fadwa Sakr, Ahmed Abouraya, Dina Helal, and Slim Abdennadher. Rescue simulation league team description guc artsapience. Robocup, 2015.

[10] S. Paquet, N. Bernier, and B. Chaib-draa. From global selective perception to local selective perception. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems, 2004. AAMAS 2004.*, pages 1352–1353, July 2004.

[11] Sébastien Paquet, Nicolas Bernier, and Brahim Chaib-draa. *Comparison of Different Coordination Strategies for the RoboCupRescue Simulation*, pages 987–996. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004.

[12] Fadwa Sakr and Slim Abdennadher. Harnessing supervised learning techniques for the task planning of ambulance rescue agents. ICAART 2016, February 2016.

[13] Ming Tan. Multi-agent reinforcement learning: Independent vs. cooperative agents. In *Proceedings of the tenth international conference on machine learning*, pages 330–337, 1993.

[14] Ping Zhang, Xiujun Ma, Zijian Pan, Xiong Li, and Kunqing Xie. Multi-agent cooperative reinforcement learning in 3d virtual world. In *Proceedings of the First International Conference on Advances in Swarm Intelligence - Volume Part I*, ICSI'10, pages 731–739, Berlin, Heidelberg, 2010. Springer-Verlag.