RoboAkut 2004 Rescue Team Description

Barıs Eker and H. Levent Akın

Boğaziçi University, Department of Computer Engineering, 34342 Bebek, Istanbul, TURKEY
{akin}@boun.edu.tr

Abstract. RoboAkut is a simulated multi-agent rescue team that combines reinforcement learning with cooperation among agents so that agents can learn mutually from each other. There is a hierarchy among agents, and agent coordination is centralized. Each agent is capable of deciding for itself when there is no support from other agents. The agents make use of the sensory information they obtain to learn the state of the environment, and consequently to decide on the actions to perform. Each agent learns from the results of the actions it does. Two kinds of q-learning is used, table based and neural network based. RoboAkut is one of several projects carried out in the Artificial Intelligence Lab of the Computer Engineering Department of Bogazici University. RoboAkut has participated in the RoboCup Rescue Simulation League 2002 and 2003.

1 Introduction

Intelligent agents are entities that can perceive their environment and act upon their percepts [1]. They are implemented and used in diverse areas such as meeting planning and air traffic scheduling. The knowledge base and the inference mechanism of agents are among the important factors determining their effectiveness. Another issue is the learning capabilities of agents. In order to make progress or to adapt to the changing conditions of the environment, an agent must be equipped with a learning mechanism.

Although an agent may be capable of doing a certain task with high efficiency, there are tasks that cannot be accomplished by a single agent. Multiple agents of the same kind may be needed in order to speed up the accomplishing of a task such as cleaning a room. In addition, the task may be one that requires expertise in many different areas, such as mechanical device designing. In this case, agents that are experts in different areas are built, possibly by different researchers, and they together accomplish the task. Moreover, the nature of the task itself may make it necessary to use multiple agents, such as robotic soccer.

Disaster rescue is an area that is suitable for multi-agent design. Designing intelligent agents that have rescuing, cooperating and learning capabilities can help reduce the problems associated with disaster rescue. The agents can be built in large quantities to be able to reach the whole disaster area. With their compact size and sensors for visual and audible information, they can reach the areas where human rescuers cannot reach. In addition, the agents can be designed so that they can cooperate with human rescuers and those agents that have been implemented by other researchers. In a disaster environment, agents need to cooperate and act fast in order to save victims, making it a challenging problem for multi-agent research.

The RoboCup Rescue simulation league of the RoboCup organization [2] provides a simulated disaster environment in which the researchers can test their implementations of rescue teams. The disaster simulation takes place on modeled maps of cities. The cities are either ones where there has previously been a disaster, or virtual ones designed specially for the simulation. Taking the modeled map data as input, several simulators simulate different aspects of a disaster. The environment simulators are the collapse, fire, traffic, and road blockage simulators. The behaviors of civilians are simulated in addition. These simulators work in a coordinated fashion. For example, when buildings collapse and roads are blocked, traffic jams occur in that area and civilians cannot move through blocked roads.

RoboAkut is one of several projects carried out in the Artificial Intelligence Lab of the Computer Engineering Department of Bogazici University. RoboAkut has participated in the RoboCup Rescue Simulation League 2002 and 2003. In this work, we have proposed and implemented a multi-agent model for disaster rescue teams in which the agents cooperate with each other and learn from experience.

2 The General Structure

In order to benefit from the usage of multiple agents in an environment, there must be mechanisms for negotiation and cooperation between agents. Moreover, a cooperative learning mechanism should be present so that achievements can be made both in terms of cooperation and in terms of the action decisions.

In rescue operations, the agents must be capable of making the right decision among many alternatives, and the decision must be timely.

The implemented multi-agent rescue team incorporates learning, cooperation, hierarchy among agents and autonomy. The agents of the team can be classified into two groups in terms of their social roles: platoon agents and dispatcher agents. Dispatcher agents are responsible for keeping their world information up to date, and assigning jobs to platoon agents. Platoon agents are responsible for bringing reorder to the disaster site. The agents can also be classified into three groups in terms of their jobs: Police, firefighter, and ambulance. Ambulances aim to rescue injured civilians and bring them to refuges, firefighters extinguish fires, and police officers clear blocked roads. The simulation proceeds in cycles of 'sense' and 'act' sequences. The agents have the following main components: World model, the learning component, the I/O component, the route finder and the performance module.

2.1 The World Model

Each agent in the team keeps its own world model. The structural information on the map is provided initially to all agents. However, the agents know the road blockages, building collapses, and injuries only within their visible range, which is predefined as 10 meters.

Other than the provided information, each agent also keeps an abstraction of the world. The world model abstraction is done by determining the status of civilians, buildings, and roads and combining these statuses to obtain a single index. This index represents the environment state as seen by the agent.

The index indicating the status of buildings is calculated using Eq. (1)

$$B_i = a * B_1 / B_t \tag{1}$$

where B_i is the building severity index, B_1 is the area of burned buildings and B_t is the total area of buildings on the map. The index of the status of the civilians is calculated as follows.

$$C_i = 1 - (C_h + ((\sum HP_i)/(C_t * MAX\_HP)))/(C_t + 1) \tag{2}$$

where C_i represents the civilian severity index. Each civilian is assigned a health point, HP. In Eq. (2), the expression $C_h + (\sum HP_i)/(C_t * MAX\_HP)$ represents sum of the number of healthy civilians and the ratio of the total HP of all civilians to the condition where all civilians are healthy. This expression is used by the simulator in the evaluation of the performance of the ambulance agents. In the state calculation, the ratio of this expression to the condition where all agents are healthy is taken and subtracted from one. The index for the roads is calculated with Eq. (3).

$$R_i = a * R_1 / (R_0 + R_1) \tag{3}$$

where R_i, is the road severity index, R_0 is the number of clear roads, R_1 the number of blocked roads, and the rate to normalize the severity index value.

The indices above are in the range of [0,1], where zero describes an ideal state in which everything is under control, and one describes a disastrous state. In the worst-case conditions, not all of the buildings burn and not all of the roads are blocked. In order to make use of the index as much as possible, a normalization rate is introduced to the road and building severity indices.

The road, building and civilian severity indices are consumed by the learning component to calculate an index for the state the agent believes to be in. The index for the world state is formed by combining $< B_i, C_i, R_i >$ triples into one number. This calculation yields a whole number that describes the current state of the environment as seen by the agent. We can increase the detail with which we can describe the state of the environment by increasing the level of detail of each component of the state.

2.2 The Performance Module

The performance module of the agents executes in a loop and controls the flow of actions. The loop starts with receiving messages from the kernel, and making actions upon the received information. Two types of messages can be received by each agent "sense" message or "hear" message. Each "sense" message sent by the kernel to the agents indicates the start of a new cycle. The agents keep track of the elapsed time and the available number of messages that can be heard or said in a cycle by keeping track of the received "sense" messages.

Platoon Agents. The platoon agents start by connecting to the kernel, getting their identification numbers and sensory information. In each cycle the world information, abstract world information and the internal state representation is updated. If the agent does not have a target, decides whether it will choose a target suggested by the dispatcher or choose the target by itself. If it does have a target, it checks whether there is a more prior target in the environment or suggested by the dispathcer. When the agent decides on the target for the current cycle, it acts to save the target. When there is no target, the agent goes to random targets to explore the environment.

Dispatcher Agents. In each cycle the dispatchers assign their platoons to targets. Upon receiving feedback from the platoons, they update their Q-values and learn about the performance of their assignment methods. Also, the dispatchers act as a communication center, buffering and forwarding each message to the appropriate receivers.

2.3 The Routing Module

The structural information provided to an agent via 'sense' messages includes the positions and neighbors of each building and road in the environment. Using this information, A^* search is carried out for finding a path from the source to destination. There is a limit on the depth of the search tree for safety.

An agent may need more than one simulation cycle to move to its target if the target is far or one of the roads on the path is blocked. In such cases, since the environment is dynamic, the route is re-calculated.

2.4 The I/O Module

I/O component is responsible for communicating with the coordinator of the simulators via UDP. Each agent sends commands to the coordinator when it wants to do a specific task such as to move, to rescue an injured person, or to spray water to a burning building. Communication among agents also passes through the simulator coordinator. The agents communicate with each other in order to report an injured civilian or a blocked road, platoon agents send position updates to their dispatchers and request job from them and the dispatchers assign targets to their platoons. The messages spoken by the agents can be heard by all other agents within 30 meters. Wireless communication can also be used. The receivers of a wireless message when the speaker is a platoon agent and a dispatcher agent are shown in Figure 1.

The number of spoken messages that can be received per simulation cycle by an agent is limited. From all spoken messages arriving to an agent, the agent must decide on which ones to hear. If all platoon agents tell the maximum number of messages they can tell in each cycle (currently 4), then total number of messages spoken in a cycle is $4*N$, where N is the total number of agents. Thus, if an agent can hear four messages each turn and if we leave out the messages an agent itself said, then each agent can hear only $1/(N-1)$ of all messages spoken each turn. As a result, an agent may end up in hearing messages not important to it and miss important messages. Therefore, the agents must speak only when it is necessary, and each agent must know which agents to listen to. In order to make sure that each agent receives the most important messages, a hierarchical communication method is used. In this method, all agents use wireless communication and all communication goes through the dispatchers. Thus, all messages destined to a platoon agent are told to it by its dispatcher. This makes it possible for messages not destined to a platoon to be filtered out. A platoon can receive the necessary messages by listening only to its dispatcher. A buffering mechanism is implemented for the dispatchers for storing the messages that could not be sent because of the communication quota. When possible, the messages are sent to their destination. As a result, message losses because of communication quota are highly reduced.

2.5 The Learning Module

Table Based Q-Learning Each agent has a learning module. The platoon agents learn to choose among the targets proposed by the dispatcher agents, targets they see in their vicinity and the targets they are after currently. In addition, they learn which target in the vicinity they will save first. The dispatcher agents assign targets to platoon agents, and receive feedback on the final status of the targets. From the feedback, dispatcher agents learn which method of assigning jobs to agents is suitable in which state. Reinforcement learning [3] is used as the learning method by all agents of the team. In reinforcement learning, an agent receives a positive or a negative reward at the end of a sequence of actions. By repeatedly trying several combinations of actions and storing the rewards, the agents form a policy, which determines which action is best in which state. There are several methods of using the rewards for forming a policy. One such method is the temporal difference of Q-values. A Q-value is the value given to a state-action pair. It determines how successful an agent can be if it does the given action in the given state. In the temporal difference method, at the end of the sequence of actions, the Q-value associated with the initial state-action pair is updated considering two different values. The first value is the external reward obtained, the second value is the Q-values available in the arrived state. If by doing a sequence of actions the agent arrives to a state where there is a high possibility of doing good actions, therefore actions with high Q-values, then the Q-value associated with the initial state-action pair is increased.

The dispatcher agents assign jobs to their platoon agents by following some predefined actions. The actions available to platoon agents are the following.

• Assign each target to the closest agent.

• Assign each job to the closest agent, starting by the most urgent (or hardest) job.

• Assign each job to the closest agent, starting by the least urgent (or lightest) job.

For each action possibility listed above, the assignments can be done so that there are one, two, or three agents assigned to each target. Therefore, there are nine actions available to a dispatcher agent. Two different Q-value update methods were used for the dispatcher agents as shown in Eq. (4) and (5).

$$Q_{ai} = Q_{ai} + lr(Res_a/n) \tag{4}$$

$$Q_{ai} = Q_{ai} + lr(R'[i] + Res_a/100 + (R'[j] - R'[i]))/n \tag{5}$$

where Q_{ai} is the Q-value of the assignment action done in state i, Res_a is the result of the action done, n is the number of agents assigned to a specific target, lr is the learning rate. R'[i] and R'[j] are rewards associated with states i and j respectively. The alternatives of target sources for the platoons are the following.

• Continue saving the current target.

• Revert to a target suggested by the dispatcher

• Revert to a target in the environment

The alternatives of targets for platoon agents when deciding on a target in the environment are the following.

• Save the closest target

• Save the target that is most severe

• Save the target that is most lightly damaged

Each alternative in a given state is associated with a different Q-value. Two different methods are used in updating Q-values of the platoon agents as shown in Eq. (6) and (7).

$$Q_{ai} = Q_{ai} + lr(R[i] + Res_a + (maxQ_aj - Q_{ai})) \tag{6}$$

$$Q_{ai} = Q_{ai} + lr(R'[i] + Res_a/100 + (R'[j] - R'[i])) \tag{7}$$

where Q_{ai} is the Q-value of action a in state i, where state i is the state action a was decided in, $maxQ_a$ is the maximum Q-value for all actions in state j, where j is the state in which the action was completed, Res_a is the result of action a, positive if successful, negative if unsuccessful, lr is learning rate. R[i] and R'[i] represent the reward of being in state i. R[i] has both negative and positive values, where R'[i] has only positive.

It is important to note that states i and j are not successive states, i is the state where the agent decides on an action, j is the state where the action is ended either by success or failure. Since the state of the environment involves the results of the actions of the other agents, the start and end states are not deterministic. This nature of the environment makes Q-learning the most suitable reinforcement learning technique to be used for this task.

The agents can be run and decide on their actions in two different modes: greedy or explorer. If the agent is running on greedy mode, it always tries the action with the best Q-value so far. If the agent is running in explorer mode, it can accept actions with lower Q-values with a predefined probability, which is the exploration rate. The explorer mode is used during training, and greedy mode is used during real action.

Using Neural Networks for Q-learning Since the environment where agents live is continuous, having continuous states will be a better idea. Applying reinforcement learning using neural network may give us this flexibility. In order to do this we propose a model. Our model mainly depends on RoboAkut system. In RoboAkut, each agent calculates three indexes while determining their current state and discretize them. We provide these 3 indexes as input to a neural network without discretizing them. Our neural network has a hidden layer that has 5 hidden nodes and an output layer that has 3 nodes since there are always 3 possible actions. In the greedy mode, the action that corresponds the output node having the highest activation value is chosen. The neural network model can be seen in Figure 1.

In the learning phase, we use the same formulas used by RoboAkut2003. However, instead of updating Q table, we update neural network weights. We start with random weights in our neural network. As an example, here we show how we update weights for "job choice" for platoon agents. The formula used for "job choice" was Eq.(7).

Our neural network is used to calculate Q_{ai} values. We should update weights of our neural network using above formula. In our case,$i$is not a single value but a combination of 3 values which are the values of input nodes. Using current values of weights we first calculate Q_{ai} at the right hand side, then using the formula what new Q_{ai} should be. At that point, we have inputs, current outputs and what outputs should be for neural network. Therefore using backpropagation algorithm, we can update neural network weights. If agents explore enough around the environment, we expect neural network to converge.

In the table-based system, the agents explore independently and at the end of each run, their Q table values are averaged. However, in our case we obtain a neural network for each agent and combining neural networks for same types of agents and obtaining a same size network is not an easy task. We could obtain such a network by using a supervised learning at the end of each run, however conversion of this network would take much time and make the learning time much longer. Instead, we propose making this combining operation while running in greedy mode. We thought that agents would converge to very similar

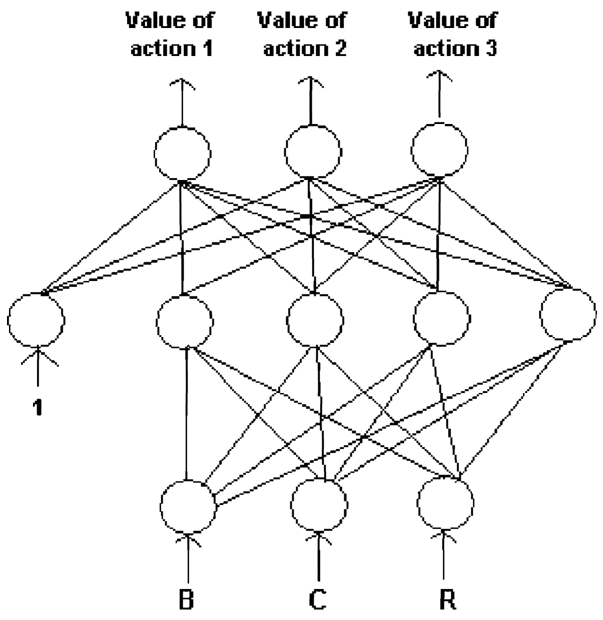

Fig. 1. The neural network architecture for Q-learning.

networks since they run on same environment; only they start from different positions. In our approach the same agent always updates the same network and it is not combined with other networks at the end of runs. When the agents run in greedy mode, they load all networks of same type of agents and calculate outputs for all of these networks and sum them. The action that corresponds to output node having the highest sum is chosen as the best action in the current state.

3 Training and Results

3.1 Testing Table-based Q-Learning

Five different initial conditions are used in the training runs. These initial conditions differ from each other in terms of the number and locations of the buildings that catch fire, the number and positions of the rescue agents and the civilians, and the degree of the blockages and collapses. The initial conditions are indexed from 0 to 4 so that in run i the initial condition (i mod 5) is used. After each run, the rescue agents of the same type share their job choice and source priority choice Q-values.

Two different groups of training were done. In the first group, the Q-value update equations used for the dispatcher and platoon agents are Eq. (4) and (6) respectively. The exploration rate of the agents were 90

Table 1. Results obtained in the test of the first and second set of Q-values for the rescue agents

Test Run No.	Unburned Area(1st set)	Unburned Area(2nd set)
1	321,393.50	474,666.20
2	573,172.20	970,345.00
3	953,362.70	626,542.30
4	566,928.70	807,389.80
5	717,295.00	1,001,274.60
6	621,107.40	1,060,766.40
7	309,622.70	971,498.30
8	719,643.90	1,120,146.90
9	503,064.80	726,179.70
10	675,484.10	867,592.30
Average	596,107.50	862,640.15

In the second training group Eq. (5) and (7) were used. These equations were formed analyzing the results obtained in the first training group. In Eq. (4) and (6), the reward function $R[i]$ and Res_a can both have positive and negative values which yields that the Q-value may be decreased when target was saved and it may be increased when target could not be saved. In order to solve this issue, in Eq. (5) and (7) $R[i]$ is replaced with $R'[i]$ which has only positive values, and Q-value update is done only when target is saved. It was also noted that when the environment is in a bad state, the agents have more targets to save, therefore the Q-values of bad states may get highter than Q-values of good states. Therefore in Eq. (5) and (7) difference of Q-values is not taken when updating a Q-value, instead the difference of the reward function is taken. The state definitions guarantee that agents do not perceive state transitions optimistically.

The rescue team was run in greedy mode using the two sets of Q-values obtained from the two groups of training. Table 1 shows the results obtained.

3.2 Comparison of Table-Based and Neural network Based Q-Learning

We trained our neural-network based system using 5 different maps and different number of agents. In the learning phase, we made 300 runs. In the testing phase, we run table-based sysytem and our neural network based system 5 times for each map and averaged the results. The results obtained can be seen in Table 2.

Results show that our new implementation performs slightly better than table-based implementation in most of our test cases. However, the results are very close to each other. The difference for Map1 and Map5 is larger when compared to others. The difference in Map1 is caused by very good performance of our system for one trial out of five. Others were very similar to RoboAkut.

Table 2. The results obtained for Table-based and Neural Network Based Implementations

	Map1	Map2	Map3	Map4	Map5
Table-based	21.323	60.359	50.705	21.136	48.648
Neural Network Based	33.128	61.162	51.553	21.187	58.076

For Map5, our new system obtained results between 44.808 and 66.389, while table-based system obtained results between 41.840 and 54.491. The results show that our new system performs better in Map5.

4 Conclusions

The test results show that in the case of table-based Q-learning, the second training method increased the performance of the agents by 44.7 percent as compared to the first training method, and that both the platoon and the dispatcher agents can differentiate among the options they have and make suitable decisions. Using neural networks for Q-learning also seems to improve results even further, therefore it is also worth considering as an alternative. Moreover, due to the nature of the method used, the infrastructure of the rescue team does not rely on specific properties of a single map, and the learned behaviors can be used on any map without any kind of preprocessing whatsoever.

References

1. S. Russel and P. Norvig, *Artificial Intelligence: A Modern Approach*, Prentice-Hall, New Jersey, 1995.
2. The RoboCup Rescue Technical Committee, "RoboCup-Rescue Simulator Manual Version 0 Revision 4", http://kiyosu.isc.chubu.ac.jp/robocup/rescue/manual-English-v0r4/manual-v0-r4.pdf, 2001.
3. S. R. Sutton and Barto, A. G., *Reinforcement Learning I: Introduction*, MIT Press, Cambridge, MA, 1998.