Using Learning Automata in Coordination Among Heterogeneous Agents in a Complex Multi-Agent Domain

Mohammadreza Khojasteh *khojasteh@iaushiraz.ac.ir*
Department of Computer Engineering,
Shiraz Branch, Islamic Azad University,
P.O.Box: 71345-1645, Shiraz, Iran

Aida Kazimi *aida.kazimi@yahoo.ca*
Department of Computer Engineering,
Shiraz Branch, Islamic Azad University,
P.O.Box: 71345-1645, Shiraz, Iran

Abstract

This paper describes our use of Learning Automata as a reinforcement learning method in coordination among three heterogeneous teams of agents acting in RoboCup Rescue Simulation environment. We provide a brief introduction to Learning Automata and Cellular Learning Automata, the reinforcement machine learning methods that we have used in lots of parts of our agents' development. Then we will describe the major challenges each team of agents should be concerned about in such a complex domain and for each challenge, we propose our approaches to develop cooperative teams. Finally, some results of using Learning Automata in coordinating these heterogeneous teams of agents that cooperate to mitigate the disastrous damages in a simulated city are evaluated.

Keywords: Distributed Artificial Intelligence, Learning Automata, Coordination, Heterogeneous Agents, RoboCup Rescue Simulation.

1. INTRODUCTION

The idea of using Learning Automata as a model for cooperation among homogeneous members of a team of agents acting in a complex multi-agent domain was first investigated in [1, 2] and evaluated in [3]. In those researches we had used RoboCup [4] Soccer Simulation as a test-bed for our multi-agent simulations. One might also find many researches in the literature using RoboCup Soccer Robot (as the one in [5]) since RoboCup has gained popularity as an excellent platform to foster intelligent (physical and simulated) robotics research. In this paper, we have used Learning Automata in coordination among (three) heterogeneous (teams of) agents in another branch of RoboCup known as Rescue Simulation [4].

In this domain, agents cooperate to mitigate the disaster in a simulated urban environment after an earthquake. Comparing to what we had done in [1-3], we have extended our use of Learning Automata (and one of its derivations; Cellular Learning Automata) in implementing much more parts of our simulated teams in RoboCup Rescue Simulation domain.

To accomplish the task of minimizing disastrous damages in RoboCup Rescue Simulation, agents must have effective cooperative behaviors despite incomplete information. In fact, the main goal in this domain is minimizing the damage by helping trapped agents, extinguishing fiery collapsed buildings, and rescuing damaged civilians. Figure 1 illustrates a part of a simulated city after the disaster, depicted by a standard monitor used in RoboCup competitions [4].

To reach to this goal, agents must be flexible. It means that they should be able to work in different (even center-less) situations efficiently and make rational decisions coordinated with each other. Since each agent has some limitations (for example limitation of vision,

communication, water quantity and etc.) due to its type, decision-making in such a domain is very complex. We have tried to implement flexible agents in spite of these problems.

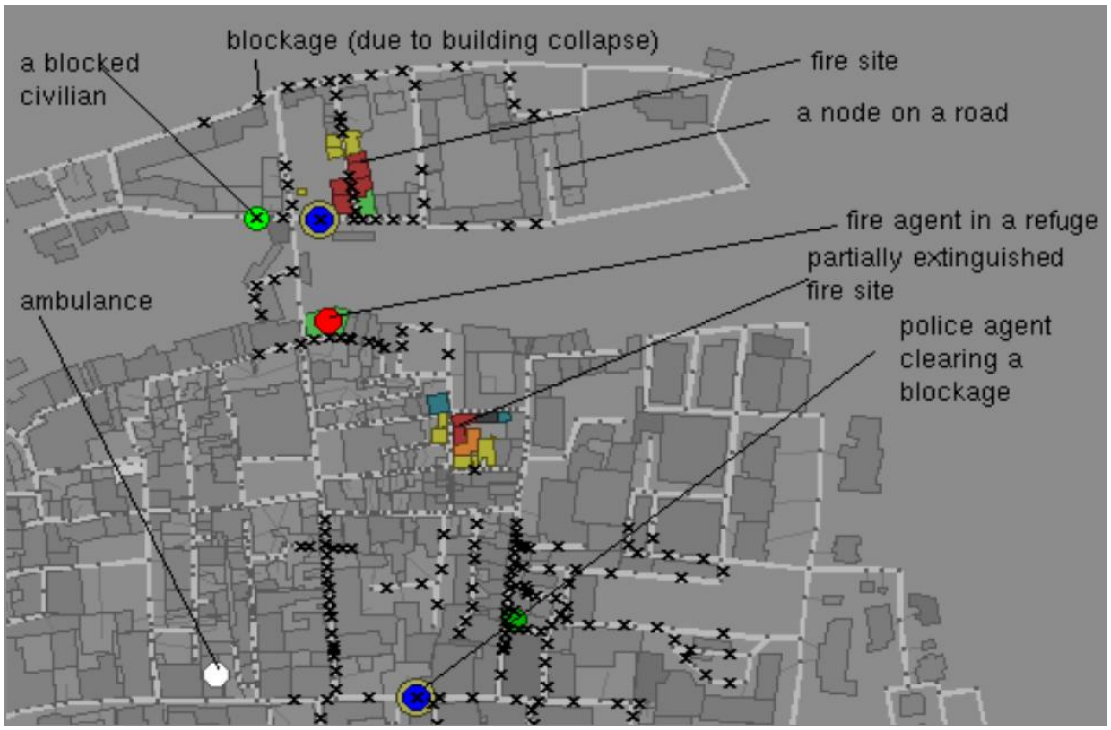

FIGURE 1: A part of a simulated city depicted by the RoboCup Rescue Simulation monitor.

In this paper, we have just concentrated on our learning approaches to rescue agents' major challenges. There obviously exist several other challenges for these sorts of agents to act effectively in this domain that are not mentioned in the paper (like agents' world model, their architecture, and the way they communicate, to name a few).

In the first parts of the paper, we give a brief introduction to Learning Automata and Cellular Learning Automata. These two approaches are the learning methods used in lots of parts of our agents' development.

Next, the challenges each type of rescue agents is concerned about are presented and then our solutions which are mostly based on Learning Automata are described. In order to have a benchmark for evaluating our approach, we have compared our (more distributed) strategy results with the results of a (more fixed) strategy used in [6] which uses pre-computed grid-like sectors and assigns agents (like fire brigades and ambulance teams, each accompanied with some police forces) to the sectors which have the highest risks (regarding fire spread and the like) during the simulation.

2. LEARNING AUTOMATA

As a model of reinforcement learning, Learning Automata act in a stochastic environment and are able to update their action probabilities considering the inputs from their environment, so optimizing their functionality as a result [7].

Among various fixed structure Learning Automata and variable structure Learning Automata investigated in [1-3], we have mostly used L_{rp} in our simulations of this research. Therefore, we give a brief description of L_{rp} as a variable structure Learning Automata.

Variable structure automata are represented by a sextuple $\langle \underline{\alpha}, \underline{\beta}, \underline{\Phi}, \underline{P}, G, T \rangle$. In this sextuple, α is a set of outputs, $\underline{\beta}$ is a set of inputs, $\underline{\Phi}$ is a set of internal states, $\underline{P}$ denotes the state probability vector governing the action chosen in each state at each stage k, G is the output mapping, and T is the learning algorithm. The learning algorithm is a recurrence relation and is used to modify the state probability vector $\underline{P}$ (i.e. the probabilities to choose the actions in each state).

It is evident that the crucial factor affecting the performance of the variable structure Learning Automata is the learning algorithm for updating the action probabilities. Various learning algorithms have been reported in the literature [7]. An Example of the variable structure type is L_{rp} automata that we summarize its behavior in the following paragraphs.

Let α (with index i) be the action chosen at stage $k-1$ as a sample realization from distribution $\underline{P}(k-1)$. So, the automata should update all of its action probabilities (for action i and all other actions j) depending on the environment's response received at stage k. In linear reward penalty algorithm (L_{rp}) scheme, the recurrence formulas for updating $\underline{P}$ are defined as:

$$p_i(k+1) = p_i(k) + a(1-\beta(k))(1-p_i(k)) - b\beta(k)p_i(k) \quad (1)$$

$$p_j(k+1) = p_j(k) - a(1-\beta(k))p_j(k) + b\beta(k)\left[\frac{1}{r-1} - p_j(k)\right] \quad (2)$$

In these formulas, r is the number of actions and we have assumed $\beta(k) = 0$ when we have received a reward from the environment and $\beta(k) = 1$ when we have received a penalty from the environment. Also, the parameters a, and b represent reward and penalty parameters and determine the amount of increase/decrease in action probabilities respectively. For more information on Learning Automata, the reader may refer to [7].

As a general scenario in a complex domain, we should first generalize the vast number of environmental states (for each agent) into some finite states. Then we embed one L_{rp} Learning Automata with some possible actions in each state. Each of these Automata in turn, is going to learn the best distribution of action probabilities in order to do the best actions in each of the mentioned generalized states. They will do this by tuning the probabilities of their actions in each state as explained above.

3. CELLULAR LEARNING AUTOMATA

Cellular Learning Automata [8, 9] is a mathematical model for systems that are made of simple components. The behavior of each of these components is selected and modified based upon its own and/or its neighbors' behaviors and also its previous experiences. The simple components that build this model can show complex behaviors through interacting with each other.

Each Cellular Learning Automata is made of a Cellular Automata [10] in which each cell is equipped with one or more Learning Automata that identify the state of the cell. As is the case for Cellular Automata, there exists a local rule in the environment which tells whether the selected action by automata in a cell should be rewarded or punished. The act of giving reward or penalty results in updating the structure of Cellular Learning Automata in order to achieve a particular goal.

A Cellular Learning Automata is formally a quintuple $\{\Delta, A, \Omega, R, L\}$ where $\Delta = \{\lambda_1, \lambda_2, ..., \lambda_n\}$ is the set of cells in the Cellular Learning Automata which are positioned in a grid-like network, $A = \{a_1, a_2, ..., a_k\}$ is the set of legal actions in each cell (with $A^t(\lambda_i)$ showing the action done in cell λ_i at time t), R is the rule governing the system, $\Omega = \{\Omega_1, \Omega_2, ..., \Omega_m\}$ is the set of neighborhoods of each cell, and L is a Learning Automata that each cell is equipped with.

Considering above definitions, $\Omega(\lambda_i)$ is the set of neighborhoods of the cell λ_i which has the following two characteristics:
1. $\lambda_i \notin \Omega(\lambda_i)$; $\forall\ \lambda_i \in \Delta$
2. $\lambda_i \in \Omega(\lambda_j)$ if and only if $\lambda_j \in \Omega(\lambda_i)$; $\forall\ \lambda_i, \lambda_j \in \Delta$

Suppose $W(\lambda_i) = \Omega(\lambda_i) \cup \{\lambda_i\}$. The rule that governs the system can be defined as the following function:

$$A^{t+1}(\lambda_i) = R\{A^t(x) \mid x \in W(\lambda_i)\}$$

Two main neighborhood types in Cellular Automata are Von Neumann and Moore [10].

The function of Cellular Learning Automata can be described as follows: In the beginning, each Learning Automata in Cellular Learning Automata chooses an action out of the set of its legal action set. This selection can be based on the previous observations of the cell or can be random (especially in the first steps of learning from zero). After that, the chosen action by the cell is rewarded or punished. This would be done based on the chosen actions by the cell's neighborhoods and the rule that governs the Cellular Learning Automata. Now, depending on whether the selected action is rewarded or punished, the internal structure of the automata updates.

This updating for all the automata in Cellular Learning Automata can be synchronous or asynchronous; if updating the cells occurs synchronous, we call the Cellular Learning Automata synchronous and if it occurs asynchronous, we call it asynchronous. After updating, every automata in Cellular Learning Automata, chooses an action (out of its action list) again and performs it. The result of this action leads to receiving another reward or penalty by the mentioned automata.

This process of selecting an action and giving reward or penalty to it repeats until the system reaches to a steady state (or continues to reach some predefined factors). Updating the structure of each existing automata in Cellular Learning Automata is done by a learning algorithm.
Likewise Cellular Automata [10], the rules in Cellular Learning Automata can be of three types: general, totalistic, and outer totalistic. In general rules, the value of a cell in the next stage is dependent on the values of the neighbors of that cell. In totalistic rules, the value of a cell is only dependent on the sum of the values of the neighbors of that cell. And at last, in outer totalistic rules, the value of each cell is dependent on the sum of either the neighbors' values or the value of the cell itself.

The main features of the Cellular Learning Automata used in this paper are that updating the Learning Automata in cells is done synchronously, Learning Automata in cells are identical (L_{rp}), and the rule used in each cell is outer totalistic. For more information on Cellular Learning Automata and their applications, the reader may refer to [8, 9].

4. FIRE STATION AND FIRE BRIGADES

Fire brigades have an important role in preventing fire spread and in rescuing civilians' lives in RoboCup Rescue Simulation domain. We might categorize fire brigades' goals into two major categories:
1. Finding out fires and preventing their spread by extinguishing them.
2. Searching for the damaged civilians.

As far as there are fire spots in the environment, fire brigades' duty is to extinguish them. To extinguish fires, fire brigades should form groups to act more efficiently. But there are some problems in this regard that we first declare and then, propose our solutions to them.

The first problem is to understand how many fire brigades are needed to be assigned to a specific task (i.e. a specific fiery zone). Also, since we don't know how many tasks (again fiery zones) we will be facing in a typical simulation, the second problem is how we should form groups of fire

brigades to assign to these tasks, so that the resulting formation achieves the best result for the team.

In our approach, the fire station tries to assign an efficient group to each task. The fire station does so based on a learning method that is discussed in the following paragraphs. After these assignments, each group goes on its work and each fire brigade decides itself which fiery buildings in its allocated area must be extinguished first to prevent fire spread. This decision is based on buildings' dangers. By danger we mean the potential damage a building might have if it catches fire. It is also going to be introduced formally later in the next paragraphs.

Our fire brigades compare buildings' dangers in their allocated fiery zone and choose the buildings that have higher danger values. Obviously, if there isn't any center (i.e. fire station) in the simulation environment (the case in center-less simulations), one of our agents will act as a commander to do so. In such conditions, we have used other approaches to read and send messages by our agents due to their communication limitations.

Before answering the first problem proposed above, we first present some definitions:

- Fiery Building: a building with a 1, 2, or 3 fieriness degree.
- Building Neighbors: the immediate neighbors of a building.
- Fiery Zone: a set of fiery buildings and their neighbors.

As a solution to the problems proposed (and using the terms we defined above), the station needs to have enough information about each fiery zone. This information should contain the priority of each fiery zone and the number of the fiery zones we face in the simulation.

In order to define the "priority of each fiery zone" we studied various parameters that exist in RoboCup Rescue Simulation environment and selected some of them that seem to be more important in this regard. These parameters are:

- Area of the fire (FA): It is fiery zone's area.
- Fiery zone's fieriness (F): It is simply the average of fieriness for those buildings which form a fiery zone.
- Fiery zone's danger (D): It is the average of dangers corresponding to those buildings that form a fiery zone.
- Fiery zone's spread area (SA): It is defined as the area that a fiery zone might spread into.

By using the above mentioned parameters we tried to find a relationship between them to determine the priority of a typical fiery zone. Let |b| be the number of buildings in the fiery zone, and f_z be the set of buildings that are members of that fiery zone. We define P as the priority of fiery zone with the following formula:

$$P = \frac{\sum_{b \in f_z} F}{|b|} \cdot \frac{\sum_{b \in f_z} D}{|b|} \cdot (FA + SA) \quad (3)$$

To find a building's danger, we have used [6] to calculate a quantity as the building's danger. In order to do so, we have to find each building's neighbor vulnerability (V) first. A building's vulnerability depends on its area (A) and the material (M) that forms that building [6]:

$$V = \frac{1 - \sqrt{\frac{1}{1 + \log(1 + A)}}}{M} \quad (4)$$

We should also define and use another term to find a building's infectivity. Actually, the fire can spread from a burning building to other buildings within a certain radius (r). The closer these buildings are, the more likely it is they will catch fire. For all these neighbors we can already

estimate their vulnerability. Being close to vulnerable buildings affects the risk a building poses. We add all the vulnerabilities of the surrounding buildings, weighted by their distance related chances to get a new factor called Infectivity [6]:

$$I = 1 - \frac{dist^2}{r^2} \quad (5)$$

$$S = \sum_{n \in neighbors} \frac{1}{1 + \log(1 + I_n)} \cdot V_n \quad (6)$$

We also define the fire risk of a building as a static value that is computed at the start of the simulation for each building. We simply add vulnerability and infectivity to find the fire risk (R) of each building [6]:

$$R = V + S \quad (7)$$

This fire risk already contains a hint about how important this building is to its neighbors. The risk value shows how dangerous it would be if none of that building's neighbors were burning. As a result, we can calculate the actual danger (D) this burning building poses during the simulation (assuming $phase_n$ is the current state of neighbor n) as [6]:

$$D = R \cdot \frac{\sum_{n \in neighbors} \begin{cases} 0 & phase_n \notin \{not\ burnning\} \\ R_n & phase_n \in \{not\ burnning\} \end{cases}}{|neighbors|} \quad (8)$$

The last parameter used in our approach is fire spread. We want to know how a fiery zone can spread fire. We have used Cellular Learning Automata [8, 9] to gain this knowledge.

In our Cellular Learning Automata method, $\Delta = \{B_1, B_2, \ldots, B_n\}$, where each B_i represents one of the environment's buildings. $A = \{A_1, A_2\}$, where A_1 predicts that B_i will be burnt in the next cycle and A_2 predicts B_i won't be burnt in the next cycle. $\Omega = \{N_1, N_2, \ldots, N_n\}$, where each N_j shows one of the neighbors of the cell B_i (it's obvious that we have a different Ω for each B_i). Also, we have defined R as an outer totalistic rule and L_{rp} Learning Automata is used as our learning.

We need to have some distinct states in our Cellular Learning Automata. Therefore, we have defined a relation between the mentioned parameters (which seem to be important in order to define states):

$$V_{B_i} \cdot \sum_{n \in neighbors} \left(\frac{1}{dist} (D + F) \right) \quad (9)$$

In above formula, dist means the distance between B_i and each N_j. It is obvious that distance affects each N_j's fieriness and danger. In above relation, F means N_j's fieriness (it is considered to be important because fieriness increases temperature), and D means N_j's danger (it is important because it specifies how much a building would spread fire). At last, V of B_i (vulnerability of B_i) is considered important in the above relation because it specifies how much a building is vulnerable due to its physical properties (its area and its material).

We divided the range between the experimental minimum and maximum values of the above relation into 10 equal intervals. Each interval represents us a state. We are therefore put in a unique state, at any instant of the simulation, by calculating the value of the above relation.

We equipped each state with an L_{rp} Learning Automata with 2 actions. By running a lot of different simulations and after convergence, the Learning Automata learned the probability of a

building's being burnt depending on the above mentioned parameters in that building and also in its neighbors.

As a scenario we used for learning, the fire station identifies the state of each building by calculating the above formula. Then, it looks up the corresponding Learning Automata in that state and chooses one of the two following actions:
1. This building will be burnt in the next cycle based on its current state.
2. This building will not be burnt in the next cycle based on its current state.

After going to the next cycle, that state's Learning Automata can give itself reward (if it had predicted correctly) or penalty (if it had predicted incorrectly).

Up to this point we have declared the "priority of each fiery zone". Unfortunately, it does not provide us with enough information to solve the problem of how to do the best action to control fire spread in such a dynamic environment. It's because the number and the position of fiery zones change in each simulation and also we need several cycles in order to detect all fiery zones.

Actually there are a plenty number of states (considering different number of fiery zones that might exist in the city, their positions, their corresponding dangers, the way they spread, etc.). There are also a plenty number of possible actions regarding the different formations the fire brigades might have. This makes it hard to use a traditional machine learning method from the convergence point of view.

Therefore, we have defined just 10 actions instead of considering all possible actions agents might do. Each of these actions was tested in different cities with various fiery zones in order to find the near optimal action in almost every state:

1. Assigning all agents from the highest priority (zones) to the lowest priority (zones) in a preemptive manner (priorities are calculated as explained above). By preemptive, we mean agents will leave their zone if a fiery zone with a higher priority is detected at any instant.

2. Assigning all agents from the highest priority (zones) to the lowest priority (zones) in a non-preemptive manner. By non-preemptive, we mean once agents are assigned to one fiery zone, they won't leave there until they have finished the job at that assigned fiery zone (extinguishing the fire completely).

3. Assigning all agents from the lowest priority (zones) to the highest priority (zones) in a preemptive manner.

4. Assigning all agents from the lowest priority (zones) to the highest priority (zones) in a non-preemptive manner.

5. Dividing agents to groups proportional to every detected fiery zone's minimum necessity (discussed later) and assign them in a preemptive manner.

6. Dividing agents to groups proportional to every detected fiery zone's minimum necessity and assign them in a non-preemptive manner.

7. Assigning agents to cover fiery zones' minimum necessities as far as possible, from the highest fiery zone's minimum necessity to lowest fiery zone's minimum necessity in a preemptive manner.

8. Assigning agents to cover fiery zones' minimum necessities as far as possible, from the highest fiery zone's minimum necessity to lowest fiery zone's minimum necessity in a non-

preemptive manner.

9. Assigning agents to cover fiery zones' minimum necessities as far as possible, from the lowest fiery zone's minimum necessity to highest fiery zone's minimum necessity in a preemptive manner.

10. Assigning agents to cover fiery zones' minimum necessities as far as possible, from the lowest fiery zone's minimum necessity to highest fiery zone's minimum necessity in a non-preemptive manner.

Actions 1 to 4 assign all agents to only one fiery zone at a time. Actions 5 and 6 assign agents in groups so that every detected fiery zone would have at least one fire brigade being allocated to it. Actions 7 to 10 also assign agents in groups but there might be some detected fiery zones without any fire brigades being allocated to them.

To be able to find each fiery zone's minimum necessity, we have used another L_{rp}. In this L_{rp}, we generalized the states by calculating our priority formula for each fiery zone again. We divided the range between the experimental minimum and the experimental maximum values of the priority function into 10 equal intervals. Each interval represents us a state. We have equipped each state with an L_{rp}. But some problems arise with the selection of actions for this L_{rp}; is it better to assign some number of fire brigades or to assign some amount of water as actions?

Each of the two choices does have some problems. Therefore, we let the actions be assigning the minimum number of fire brigades with average water (i.e. half of the maximum amount of water they might have in their tanks).

As a scenario for learning, we administrated offline simulations with different fiery zones (but just one fiery zone in each simulation). The fire station identifies the state of this fiery zone by calculating the priority formula. Then, it looks up the corresponding Learning Automata in that state and assigns some (in the beginning of the simulation, it could be one or some predetermined number of) fire brigades to that fiery zone. If they could extinguish and control the fire by the end of simulation time, the station gives itself reward. But if not, it will give itself penalty, and so the probability that one more fire brigade will be assigned to the same fiery zone's state (by the station), is increased.

Doing so and after a lot of simulations, our L_{rp} would be able to assign the minimum number of fire brigades (with average water) in each state that a typical fiery zone is in.

5. AMBULANCE CENTER AND AMBULANCE TEAMS

Ambulance teams play an important role to rescue civilians and agents that are buried under collapsed buildings. In order to reach this goal, they should first search the city and then do the best possible actions. So they require a lot of information to accomplish their tasks. Some of the items forming this information include the position of the collapsed agents and civilians, some parameters showing whether they are dead or alive, and the knowledge about fire spreads. Some parts of this information are kept in the ambulance center which sends the necessary information (for example reported injured civilians) to ambulance teams. To achieve a better team result, we have divided the entire simulation time of our ambulance teams into two major phases.

In first phase (the beginning cycles of the simulation), most of the roads are blocked. Because of blocked roads and moving limitations in this phase, we decided that all ambulance teams work together and rescue the same target. They will detect the end of this phase (i.e. the start of the second phase) through a police force message which indicates that roads are mostly open.

In second phase, because ambulance teams don't have a lot of blocked roads ahead or they don't have moving limitation problems (as it was the case in the first phase), we tried to test different approaches to improve the task the ambulance teams have (i.e. to rescue injured agents and civilians). At last, because of the large number of states and the vast number of possible actions in such a complex environment (as it was the case in fire brigades' section), we found out that it is better to select the best approach out of eight specific methods using offline learning. After running a lot of simulations with various situations, it seems that the best learned method out of the following eight methods promise to give a near optimal result:

1. Rescuing all together, from highest priority to lowest priority:
- Non-Preemptive: In this method, all ambulance teams work together and aim a certain target to rescue. This target is advised by the ambulance center using the maximum priority known by the time (i.e. the moment of decision). We will talk about the priorities in the coming paragraphs. We call this method non-preemptive because ambulance teams will stay on the target until its rescue will be done.
- Preemptive: This method is the same as the previous one until the time the ambulance center hasn't detected any other target (i.e. civilian) with a higher priority. At that time, all ambulance teams will leave the current target and will go to rescue the new target.

2. Rescuing in two groups, from highest priority to lowest priority:
- Non-Preemptive: In this method, the ambulance center divides the ambulance teams into two balanced groups and assigns them to the two targets with highest priorities. We call this method non-preemptive because each group of the ambulance teams will stay on the allocated target until its rescue will be done.
- Preemptive: This method is the same as the previous one until the time the ambulance center hasn't detected any other target (i.e. civilian) with a higher priority. At that time, it will assign the group with the lowest target's priority to the new target. The group's ambulance teams will leave the current target (even if they aren't finished with its rescue) and will go to rescue the new target.

3. Rescuing in three groups, from highest priority to lowest priority:
- Non-Preemptive: In this method ambulance center divides ambulance teams into three balanced group and assigns them to the three targets with the highest priorities. We call this method non-preemptive because each group of the ambulance teams will stay on the allocated target until its rescue will be done.
- Preemptive: This method is the same as the previous one until the time the ambulance center hasn't detected any other target (i.e. civilian) with a higher priority. At that time, it will assign the group with the lowest target's priority to the new target. That group's ambulance teams will leave the current target (even if they aren't finished with its rescue) and will go to rescue the new target.

4. Rescuing in some groups based on the number of required ambulance teams for each civilian, from highest priority to lowest priority:
- Non-Preemptive: In this method, the ambulance center computes the number of required ambulance teams for the 4 highest priorities, and then divides ambulance teams into some non-predetermined (but less than 5, of course) unbalanced groups and assigns them to the targets that have the highest priorities. We call this method non-preemptive because each group of the ambulance teams will stay on the allocated target until its rescue will be done. In order to calculate number of required ambulance teams for rescuing a target, the ambulance center uses this formula:

$$\text{Number of required ambulance teams} = \left\lceil \frac{\text{buriedness}}{\text{Deadtime} - (\text{TCR} + \text{ATC})} \right\rceil \quad (\mathbf{10})$$

In this formula, TCR is ambulance teams' traveling costs (defined in next paragraphs) to reach the refuge from that target, and ATC is the average of idle ambulance teams' traveling costs to that target.

- Preemptive: This method is the same as the non-preemptive one until the time the ambulance center hasn't detected any other target (i.e. civilian) with a higher priority. At that time, it will assign the number of ambulance teams this new target requires to it by preempting the members of the group that its target priority is the lowest. They will leave the current target (even if they aren't finished with its rescue yet) and will go to rescue the new target.

It shall be noted that in all cases above, the ambulance teams inform the ambulance center when they have arrived to the target. Also, the ambulance center estimates the time they will finish their job based either on the information it has received by now or on the information it receives from the ambulance teams during doing their jobs. This estimation is useful for the ambulance center to predict the time it should recalculate its priority list to assign the ambulance teams to their next jobs.

We again used an L_{rp} Learning Automata (with eight actions corresponding to above mentioned eight methods) and administered several simulations to test each of these methods in various situations. We exploited the results gained in these simulations to help us choose the method that gives the best average result. The chosen method was used in our simulations, especially in our fire brigades' learning from zero's method.

To define and calculate the priorities, ambulance teams should consider some parameters to help them choose the most important civilians to rescue. These parameters include their dead time, their buried-nesses (a parameter given by the server), the fire spread, and the traveling cost it takes for ambulance teams to reach the civilians. We have given a brief explanation of these factors below:

1. Dead Time: It is the remaining time before the death of an injured civilian. We again tried to estimate this by use of a simple learning method.
2. Fire Spread Time: It is the way the fire spreads (related to a civilian being exposed to the danger of catching fire). We find the value of this parameter using the Cellular Learning Automata which was explained in the fire brigades' section.
3. Traveling Cost: It is the number of cycles that will be spent by our ambulance teams to reach the civilian.

To assign priorities to civilians, we sort them first by their dead times (ascending), then by their fire spread times (ascending), and at last by their distances (descending).

In center-less situations our ambulance teams use the same methods as fire brigades. A commander substitutes the ambulance center and does the above task assignments. The commander goes to the police office or fire station and tries to get the injured civilians' information.

6. POLICE OFFICE AND POLICE FORCES

Police forces' main job is to clear the blocked roads in the city. Although their work doesn't affect the score of a team directly, it can have a great effect on other agents' duties. That's because the performance of other agents does heavily depend on the number of open roads in the city. Our police forces try to learn how to do their job better from zero and in doing so, they are counting on what our fire brigades and ambulance teams have learned by now. It means that, through offline simulations mentioned in previous sections, fire brigades and ambulance teams must have already learned how to do their best actions. In this phase, we have let the whole rescue team

(fire brigades, ambulance teams, and police forces) cooperate together. Doing so, we let police forces learn their best actions by giving appropriate services to other agents.

We have again used Learning Automata to reach this goal. Our police forces use two various methods; one in normal (center-based) situations and the other in center-less situations. We will describe the method which is used in normal situations first and then our center-less method will be described.

In normal situations our police forces work autonomously without receiving any commands from the police office. Our police forces do their jobs using a technique based on attraction and repulsion vectors similar to what we had done in [11].

In this approach we assume some virtual attraction vectors pull each police force towards each of the following items (to attract it to important positions in the city) at any instant:

- Position of each fiery building known by now which there is (are) still blocked road(s) reported on the way to it.
- Position of each refuge which there is (are) still blocked road(s) reported on the way to it.
- Position of each trapped agent toward which there is (are) still blocked road(s) reported on the way to it.
- Position of each reported blocked roads.
- Positions mentioned in each fire brigade's and ambulance team's important (and still unanswered) request messages.

The beginning point of an attraction vector is the police force's position and its end point is an item's position. Therefore, the direction of each attraction vector is towards the above items' positions and the magnitude of that is conversely proportional to the distance between the police force's position and the item's position. This makes closer items more important to police forces.

We also assume some virtual repulsion vectors push each police force away from other police forces (to distribute them in the city) at any instant. The beginning point of any repulsion vector for a police force is another police force's position and its end point is its own position. Therefore, the direction of each repulsion vector is towards the police forces' positions and the magnitude of that is conversely proportional to the distance between the two police forces' positions. We try to prevent traffic jams by distributing our police forces in the city using this repulsion vectors.

As an implemented scenario, each idle police force calculates the resultant vector by the vector addition of these attraction and repulsion vectors for itself. Then the police force finds the nearest target to the end point of this resultant vector and finds a path towards it. The police force tells its intention to the police office and starts to move on its path. If the police office finds out more than one police force moving to the same target (it might happen because each police force does the above procedure autonomously), it will prevent all except the nearest one from moving to the target by sending messages.

Whenever a police force finishes its job by clearing its target (i.e. clearing the blocks in the roads leading to the target) it will again calculate its resultant vector for its next target. Police forces send new information to the police office at any instant and they count on information they receive from the police office in this approach.

This approach seems to have two great advantages. The first one is that police forces don't go to clear a barricade simultaneously because they try to be far from each other using repulsion vectors. The second one (considering the fact that police forces learn how to do their job after fire brigades and our ambulance teams have learned how to do their best actions) is that the most important barricades seem to be cleared in appropriate time using attraction vectors.

To complete the procedure, we should consider some scalar coefficients for the magnitudes of each vector type to make some items more significant. That's because it seems some items like (clearing the roads toward) trapped agents might be more important than others. In our experience, these coefficients are composed of two parts; an experimentally known part (the priorities that each vector type should essentially have and can be set manually), and an unknown part that should be learned by police forces.

For the known part, we use our domain experiments like the importance of trapped agents comparing to others, the amount of danger parameter for each fiery building discussed in the fire brigades section, and etc. By doing so, we give priorities to each vector type. For the unknown part, we can't have any predetermined assumption and this is again the place Learning Automata come into the scene. In fact police forces should learn these coefficients (based on how fire brigades and ambulance teams work together) in order to improve the whole rescue simulation's score.

Let's call these coefficients p_1, p_2, p_3, p_4, and p_5, respectively for each of the vector types mentioned above. At the beginning of our learning method, we initialized all these coefficients to the value 0.5. Our police forces tried to tune these values using a learning method we are going to describe. After running many simulations in different cities and with many situations in the learning phase offline, we were able to fix the values of coefficients learned and use them during the test phase online. It should be mentioned that our agents are also slightly allowed to explore these values online.

In the learning phase we define an L_{rp} Learning Automata with ten actions. The first five actions add 0.05 to each of the above 5 coefficients and the second five actions subtract 0.05 from each of the above 5 coefficients, respectively.

L_{rp}'s action probabilities for these actions are each set 0.1 initially. At the beginning of each simulation during the learning phase, police forces choose one of these actions based on their corresponding probabilities, using a uniform random number generation and they perform the specified action (changing the value of a coefficient with a factor of +/- 0.05). At the end of each simulation, police forces give themselves reward if the simulation's final score has improved comparing to the previous simulation' final score. In this case the changes to the coefficients are preserved. Police forces give themselves penalty if the simulation's final score has decreased comparing to the previous simulation' final score. In this case the changes to the coefficients are undone.

In center-less situations, we should provide a different approach. In the previous (center-based) approach each police force needs to be aware of other police forces' positions in order to do an action. In that approach, the police office transfers information of the police force positions amongst them and each police force can receive the information it needs in a short time.

In the center-less situations police forces can't send their positions to each other in each cycle because of the communication limitations they have. Therefore, it seems that the previous approach does not work efficiently in center-less situations. Because of this problem we have designed and implemented another approach for center-less situations. In this approach a police force will act as a commander and assigns jobs to other police forces.

The entire city is divided into four major parts and each of these major parts is subdivided into 20m*20m cells as shown in figure 2. We have chosen 20m*20m cells because agents are able to see everything in the radius of 10 meters from themselves, according to server rules. Therefore, almost the whole square should be visible for an agent located in it.

For each police force, we assume a virtual attraction vector pulls it towards the position of each of

the following items:

- Fiery buildings
- Refuges
- Trapped agents
- Positions mentioned in fire brigades' and ambulance teams' important requests

The magnitude of each vector is conversely proportional to the distance between the above mentioned items and the police force, multiplied by a constant coefficient. Coefficients are different based on the type of each item. In this center-less approach, these coefficients were chosen just by using our previous simulation experiments.

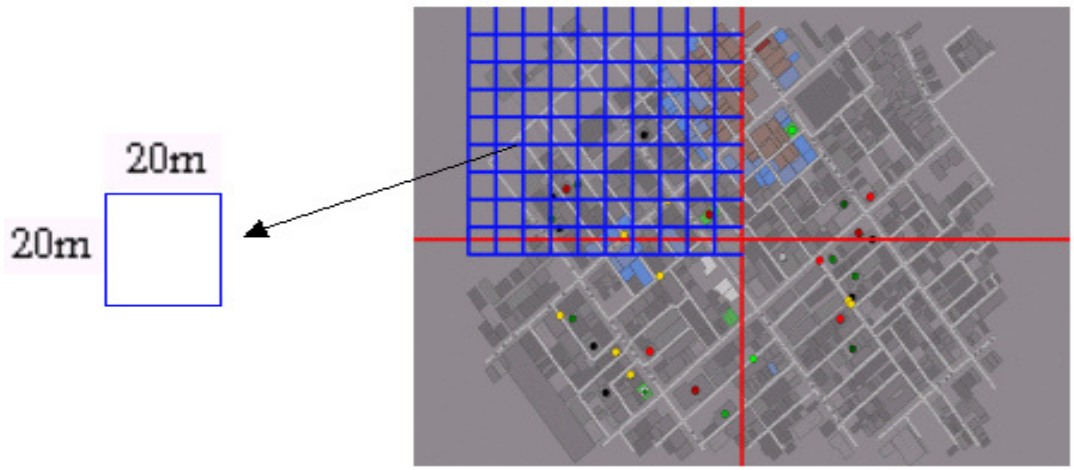

FIGURE 2: A map of Kobe city virtually divided into 20m*20m cells

Because of process time restrictions (2 seconds as the rule says), we have considered no attraction vectors towards blocked roads. Instead, we have considered attraction vectors originated from the police forces towards the centers of each cell which contains blocked roads in addition to the above mentioned vectors. The magnitude of this vector is conversely proportional to the distance between the police force's position and the cell. Also the coefficient for this magnitude is directly proportional to the number of roads inside the cell. It's obvious that after assigning a police force to a cell, we will ignore this cell in forming vectors. Also, if a path toward a refuge (or a fiery building) was opened, its corresponding vector will be ignored thereafter.

The commander calculates the resultant vector for each police force. Naturally the resultant vector points to one of the four major parts of the city. We call this part as the active part of that police force. Then the commander assigns a police force to one of the cells of this active part via an L_{rp} Learning Automata (explained briefly in the next paragraph). Then, the police force begins to clear the roads in the allocated cell.

The commander should learn which cell is more important to clear and it uses Learning Automata to do so. The rewards and penalties for the commander's selected actions are given regarding the decision of this allocation. As an example, if there is a trapped agent in one of the cells of the active part but the police force was assigned to a cell which doesn't bear any special items (like trapped agents or fiery buildings), this commander's decision will get a penalty.

Since the commander knows the police forces' work areas, it is able to calculate the resultant vector of each police force easily. The commander considers the center of the cell in which the police force works as the police force's position. Then it calculates the resultant vector using above procedure and assigns the police force to another cell. This way, the commander

determines each police force's next target.

After the process of determining police force's next target is done by the commander, the new job is sent for the police force. The commander also gives such information to every police force in each cycle. On the other hand, police forces save the commands which were sent by the commander in each cycle. Each police force extracts the latest commands after finishing the job in its allocated cell and begins to accomplish it. Each police force informs its position (the cell it is located in) to the commander during clearing the blockades.

7. EVALUATION

To evaluate the learning method we used for our teams in such a complex domain, we would first like to discuss the convergence of our agents' action probabilities briefly. To do so, we take our ambulance teams as an example. As mentioned in section 5, we have used an L_{rp} Learning Automata with eight actions (each corresponding to a method through which ambulance teams should act). We administered several offline simulations to test each of these methods in various situations and to let our Learning Automata converge to their best decisions of actions.

Since our selected domain is very changeable and dynamic, there is always the possibility of convergence to a different action from the one selected in a previous run in consecutive simulation runs. If we administrate sufficient simulation runs to let our Learning Automata have enough time to test each of their actions in various situations, we can expect it to give us a near optimal action (to be done) in each and every state. Our experiments revealed that this need of running enough simulations might take considerable time for the Learning Automata to converge.

In our experiments, we noticed that in most cases, the Learning Automata used were able to report us the best selected action and so, were able to converge to one of their actions at the end. But it seems inevitable that in some cases, although they have had enough opportunities to act during offline training simulations, our Learning Automata give us probability distributions as the final outcome of their learning.

In RoboCup Rescue Simulation, the lapse of time is defined in terms of simulation cycles. Although the number of cycles is configurable, an ordinary simulation run consists of 300 simulation cycles. The actual time lapse per cycle can be altered too, but generally equals one second, bringing the total simulation run to 5 minutes [6].

Our results show that in one of our simulation series (results of which depicted in figure 3), the probability values (first each initialized to 0.125 for each of the eight actions) have converged almost fast. After something about 140 minutes (28 consecutive simulation runs), the probability of doing action 2 has reached to 0.995 that seems to be enough to claim for convergence.

In another series of simulation (results of which depicted in figure 4), after something about 550 minutes from the beginning of the simulation (110 consecutive simulation runs), none of the probability values reached to a value good enough to claim for convergence. The actions 3, 4, and 8 had been selected as the near optimal action, each in a time slice during these 550 minutes. Although action numbered 8 had the most probability value at the time we stopped the simulation, it didn't seem to lead to any convergence.

As illustrated, in some time slices of the time passed, the probability of each of the above-mentioned actions had increased, but declined very soon. In such cases, we have gained probability distributions (for doing actions) as the outcome of learning in our simulations.

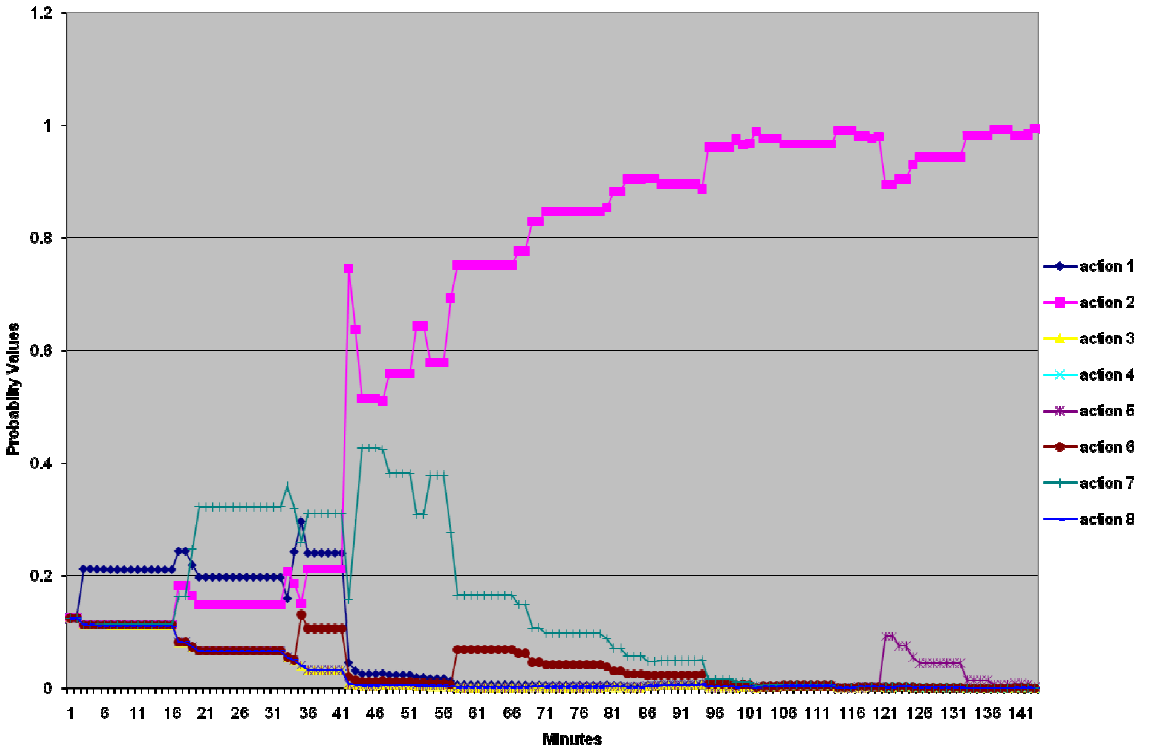

FIGURE 3: First simulation series' changes in probability values of each of the eight actions of L_{rp} Learning Automata for ambulance teams over time elapse

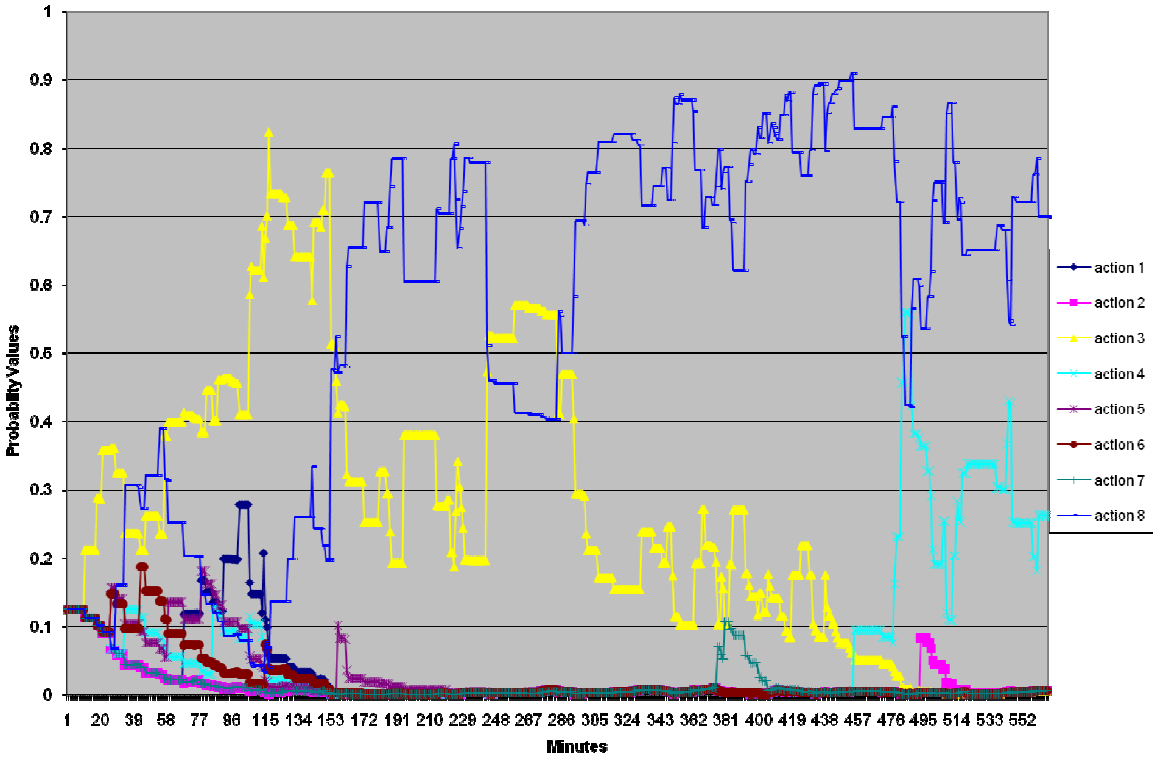

FIGURE 4: Second simulation series' changes in probability values of each of the eight actions of L_{rp} Learning Automata for ambulance teams over time elapse

And at last, in the third series of simulation (results of which depicted in figure 5), the probability value of action 2 has reached to 0.98 after something about 230 minutes (46 consecutive simulation runs) that seems to be enough to claim for convergence.

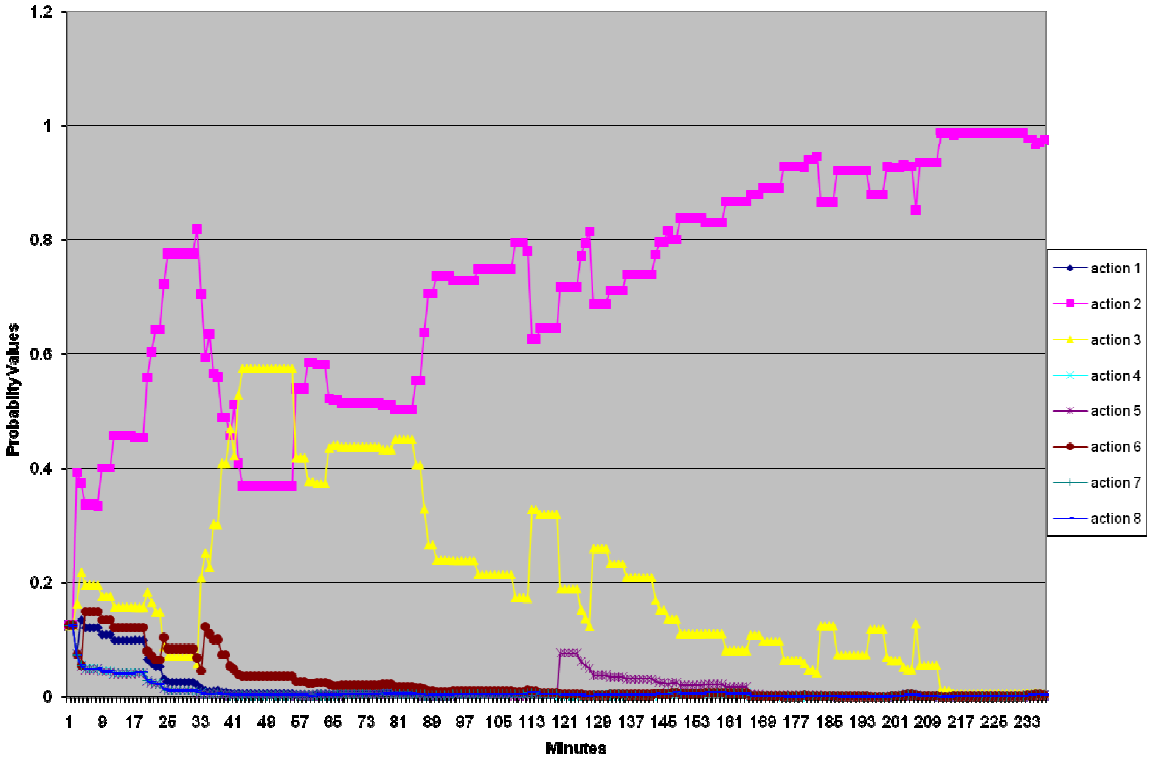

FIGURE 5: Third simulation series' changes in probability values of each of the eight actions of L_{rp} Learning Automata for ambulance teams over time elapse

As the above results show, because the environment is very changeable and dynamic, we can't always expect the probability values to converge to a unique (or predefined) behavior to be selected as the optimal behavior of the Learning Automata and we can't even expect for convergence. The results show the need for and the necessity of having enough simulation runs in order to have the best average decision of action in each and every state of the environment.

As we experimented in this research's simulations, our previous simulations in RoboCup Soccer2D Simulation domain had also revealed that Learning Automata converge slowly comparing to many typical machine learning methods (like Q-Learning) [1-3]. Although we have investigated some modification methods that slightly increase their speed of convergence in [1], the fact is that Learning Automata is very time demanding to converge. That would be because of their natural need to have enough (and occasionally plenty of) time in order to make them able to test each of their actions in various situations.

On the other hand, we would like to compare the results of our research's simulations with those of other known researches using this test-bed which have used other learning strategies. A lot of teams taking part in international RoboCup Rescue Simulation [4] competitions have published their strategies.

In order to have a benchmark for evaluation, we have compared some results of our strategy with those of the strategy used in UvA team [6] which utilizes pre-computed grid-like sectors in the simulated city. UvA has been one of the teams taking part in RoboCup Rescue Simulation competitions which in its essential part, assigns agents (like fire brigades and ambulance teams)

to the sectors which have the highest risks (due to fire spread and the like) during the simulation.

In our approach each of the three heterogeneous teams uses its own learned behavior to be assigned to any point in the map, while in [6], police forces accompany fire brigades and ambulance teams after their assignment to a sector. In fact, we have used a more distributed approach to make our agents more flexible and efficient in this dynamic and non-deterministic environment.

We also believe that our approach would be more robust comparing to the method used in [6]. That's because it seems there would be no jobs to do for the police forces who accompany those fire brigades and ambulance teams which encounter sudden malfunction (like disconnecting from the server which happens occasionally). While in the approach we have used, police forces are assigned to their jobs based on the information they receive and independent of the other two teams of agents acting in the environment.

As mentioned in section 6, police forces can have a great effect on the betterment of other agents' duties by clearing the blockades and the performance of other agents does heavily depend on the number of open roads in the city. It would be obvious that the more nodes (especially those on the blocked roads) the police forces visit during a simulation run, the higher chances exist for fire brigades and ambulance teams to act more efficiently in that run. The reader might have a second look at the situation depicted in figure 1 to have a better feeling of the objects (such as nodes and roads) in a simulated city.

To have an estimation of the number of buildings, roads, and nodes in typical simulated cities, three of the most used classic maps in RoboCup Rescue Simulation [4] competitions are shown in table 1 along with their key features [6]. In general, the number of civilians in the simulation lies between 70 and 90, whereas the number of agents is 15 to 30 and agents can only see (sense) objects at sufficiently close range [6].

map name	total area (m2)	#buildings	#roads	#nodes
Kobe	417.4*316.5	739	820	765
Foligno	2038.1*1517.4	1084	1480	1369
Virtual City	413.1*417.8	1266	621	532

TABLE 1: Three maps used in RoboCup Rescue Simulation competitions and some of their key features [6]

In order to be able to compare our method with the method used in [6], many different evaluation tests might be run to illustrate the difference of performance between the two. As its significance was briefly mentioned above, one such test might be evaluating the function of police forces in visiting the nodes. Visiting more nodes has a direct effect on discovering and so, clearing more blockades which might exist in corresponding roads and/or corresponding building entrances. More blockage clearing during the simulation cycles by police forces might be very critical for both fire brigades and ambulance teams in doing a better job in the environment. That would be because of their having access to more roads which facilitate their routings to their targets (civilians to rescue and fiery buildings to extinguish).

We have called our agents' strategy "Distributed strategy using LA", and the strategy in [6] (implemented on our base code) "Fixed strategy". While we have implemented similar teams regarding base codes and agents' behavioral capabilities, table 2 shows the results gained during a simulation on Kobe map. The test has been administrated after all the three heterogeneous teams of agents' learning values of both teams have been converged and agents of both teams are able to exploit their learned values.

As the table shows, although using a fixed strategy does better in the first cycles of the simulation (and that might be because of using pre-computed grid-like sectors in the simulated city), the overall function of the Distributed strategy using Learning Automata is better (visiting 331 nodes vs. 303 nodes during a complete simulation run). It can be observed that the number of nodes visited in our approach surpasses that of those visited by the other approach as the cycles pass and as we reach closer to the end of the run.

Cycles / Strategy	0-50	51-100	101-150	151-200	201-250	251-300
Distributed strategy using LA	49	55	67	54	63	43
Fixed strategy	68	74	64	34	37	26

TABLE 2: Comparison of the number of nodes visited by police forces between "distributed strategy using LA" and "fixed strategy" in consecutive cycles of the simulation

Finally, the main objective of a RoboCup Rescue Simulation is to save as many civilians' lives as possible and to minimize the damages from the fires. The degree of success of the clients (i.e. the teams of agents acting in the domain) to reach this objective is calculated by the server as an overall final score.

Therefore, another typical evaluation is to compare the strategies from the final score point of view. Doing this, we would be able to have an overall comparison between the two strategies evaluating the coordination among heterogeneous teams of agents to mitigate the disaster in the city. Figure 6 shows a diagram that compares the final scores of both teams averaged over 10 complete simulation runs.

It shall again be noted that these results are gathered after agents of our teams have passed their offline training simulations and are able to exploit their learned values. Also, due to the inherent complexity which dynamically exists in such domains, we should equip the learning methods used by our agents with some (though little but not zero) online exploration for their learned (action probability) values. As illustrated, our approach has an overall better performance.

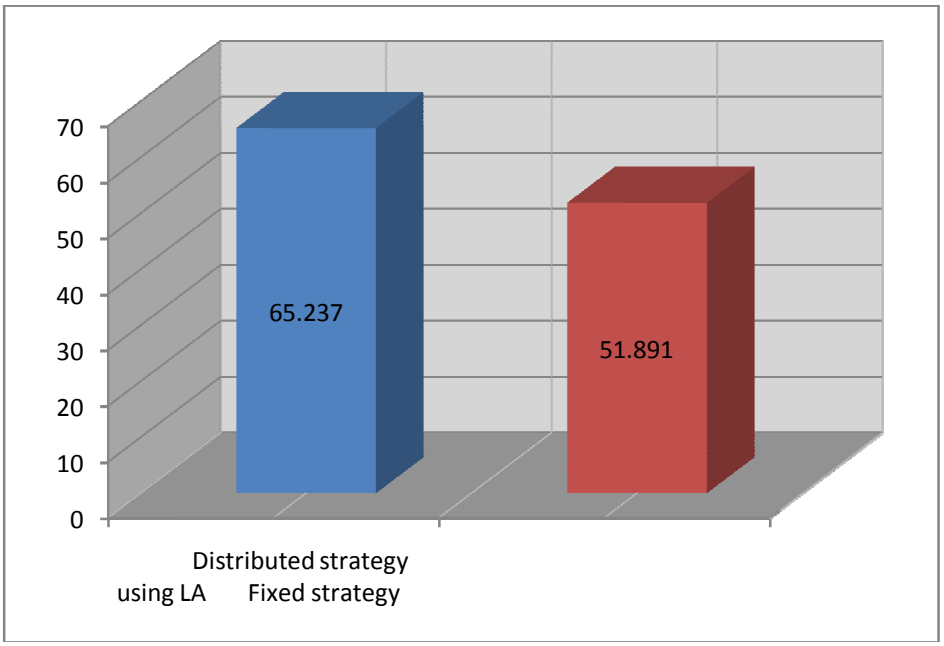

FIGURE 6: Comparison of the average overall final score between the two strategies of coordinating heterogeneous teams of agents, "distributed strategy using LA" vs. "fixed strategy"

8. CONCLUSION

Our goal has been creating a complete working rescue simulation team that can act well in a complex domain using Learning Automata-based machine learning techniques. We had previously used Learning Automata for successful production of a series of actions for homogeneous agents that were members of a team, such that the resulting team can act well in a multi-agent, noisy, real-time, and most important collaborative environment [1-3].

Before our use of Learning Automata, various machine learning methods have been used in developing teams acting in RoboCup simulation domain as a complex multi-agent test-bed. CMUnited [12] introduces Layered Learning and uses decision trees, artificial neural networks, and Q-learning in different layers. In RoboCup Rescue Simulation domain, some teams like ResQ Freiburg [13] and S.O.S. [14] have used A* and Dijkstra's algorithm [15] to plan paths for their agents. Eternity [16] has used artificial neural networks to approximate the relationship between fire situations and their priorities. Roboakut [17] uses reinforcement learning in some parts of its team development in order to predict the long term effects of agents' actions. The Black Sheep team [18] uses D* in agents' path planning.

In this paper we have used Learning Automata in coordination among heterogeneous agents in RobCup Rescue Simulation test-bed. We used Learning Automata for our agents' major challenges to give solutions to complex strategic high level decisions such as when and how to form groups of agents in order to have the best team result.

As a brief scenario in a lot of parts of our team development, the agents determine their current states by generalizing some formulas based on various important parameters in the Robocup Rescue Simulation domain. Then, they perform the action that is advised by the corresponding automata in that state. The agents (the center or commander in center-less situations) then percept their actions' results and give themselves reward or penalty depending on those results.

We have simulated our agents to learn from zero (i.e. without any previous knowledge of the

environment before starting our offline simulations). Also, we have used the agents themselves for the judgment about their actions' results and this let us have what we call "distributed judgment", again a multi-agent approach. We should also point that we have used $(a = b = 0.1)$ for the L_{rp} that we used in our training simulations. The parameters a, and b are reward and penalty parameters respectively as mentioned before.

Our simulations in RoboCup Soccer2D Simulations [1-3] and in RoboCup Rescue Simulations reveal that although Learning Automata converge slowly comparing to many typical machine learning methods, they are able to perform well in order to have a coordinative team of agents in complex multi-agent domains.

The methods introduced in this paper are general methods that can be implemented, applied, and used in other domains or test-beds with minor changes. Our experiments illustrate that Learning Automata adapts itself well with noise and changes in the environment and also with hard situations such as malfunctioning of some of the agents.

9. ACKNOWLEDGEMENTS

This work was supported in part by the office chancellor for research of Islamic Azad University for the research project based on which this article is written.

10. REFERENCES

[1] M. R. Khojasteh. "Cooperation in multi-agent systems using Learning Automata." M.A. thesis, Amirkabir University of Technology, Iran, 2002.

[2] M. R. Khojasteh, M. R. Meybodi. "Using Learning Automata in Cooperation among Agents in a Team." In Proc. the 12th Portuguese Conference on Artificial Intelligence, 2005, pp. 306-312.

[3] M. R. Khojasteh, M. R. Meybodi. "Evaluating Learning Automata as a Model for Cooperation in Complex Multi-Agent Domains," in *RoboCup-2006: Robot Soccer World Cup X*. G. Lakemeyer, E. Sklar, D. Sorenti, and T. Takahashi, Eds. Berlin: Springer-Verlag, 2007, pp. 410-417.

[4] "RoboCup." Internet: www.robocup.org.

[5] Z. Luo, Q. Cao and Y. Zhao. "A Self-adaptive Predictive Policy for Pursuit-evasion Game." Information Science and Engineering, Vol. 24, pp. 1397-1407, 2008.

[6] S. B. M. Post, M. L. Fassaert. "A communication and coordination model for RoboCup Rescue agents," M.A. thesis, Universiteit Van Amsterdam (UvA), Netherlands, 2004.

[7] K.S. Narendra, M.A.L. Thathachar. *Learning Automata: An Introduction*. Prentice Hall, 1989.

[8] M. Taherkhani. "Proposing and Studying Cellular Learning Automata as a Tool for Modeling Systems." M.A. thesis, Amirkabir University of Technology, Iran, 2000.

[9] M. R. Meybodi, M. R. Khojasteh. "Cellular Learning Automata as a model for Commerce Networks." In Proc. 6th annual conference of Computer Society of Iran (CSICC'2001), 2001, pp. 268-281.

[10] S. Wolfrom. *Theory and Application of Cellular Automata*. World Scientific Publishing, Singapore, 1986.

[11] M. R. Khojasteh, M. R. Meybodi. “The Best Corner in State Square technique for generalization of environmental states in a cooperative multi-agent domain.” In Proc. 8th annual conference of Computer Society of Iran (CSICC'2003), 2003, pp. 446-455.

[12] P. Stone. “Layered Learning in Multi_Agent Systems.” Ph.D. thesis, Carnegie Mellon University, USA, 1998.

[13] M. Brenner, A. Kleiner, M. Exner, M. Degen, M. Metzger, T. Nussle, and I. Thon. “Resq Freiburg: Deliberative limitation of damage.” Team Description Paper, 2004.

[14] S.A. Amraii, B. Behsaz, M. Izadi, H. Janzadeh, F. Molazem, A. Rahimi, M.T. Ghinani, and H. Vosoughpour. “S.O.S. 2004: An attempt towards a multi-agent rescue team.” Team Description Paper, 2004.

[15] T.H. Cormen, C.E. Leiserson, and R.L. Rivest. *Introduction to Algorithms*. MIT Press, McGraw-Hill, New York, NY, 1990.

[16] A.A. Bitaghsir, F. Taghiyareh, A. Simjour, A. Mazloumian, and B. Bostan. “Layered learning in robocup rescue simulation.” Team Description Paper, 2004.

[17] B Eker, H.L. Akin. “Roboakut 2004 rescue team.” Team Description Paper, 2004.

[18] C. Skinner, J. Teutenberg, G. Cleveland, M. Barley, H. Guesgen, P. Riddle, and U. Loerch. “The black sheep team.” Team Description Paper, 2004.