Mid-Term report "Establishing bonds for the advancement of the Rescue League"

Iván Riaño[1] and Arnoud Visser[2]

[1] Universidad Distrital Francisco José de Caldas, Colombia
[2] Universiteit van Amsterdam, Science Park 904, Amsterdam, The Netherlands

Abstract. The RoboCup Rescue Simulation League aims to benchmark the intelligence of software agents and robots on their capabilities to make autonomously the right decisions in a disaster response scenario. The UvA Rescue and Distribot team have come together to work cooperatively to support this aim in two ways. One aspect is to advance the popularity of this League by presenting the background of the scientific challenges outside the direct RoboCup community. Another aspect is to build the code base which allows to include state-of-the-art machine learning techniques, such as reinforcement learning, evolutionary computing and Markov decision processes. [3]

1 Introduction

The challenge for multi-agent decision as posed by the RoboCup Rescue Simulation League is nicely described by Jennings [1]. The number of tasks to be performed outnumbers the resources of the emergency responders. Further, the time needed to finish the task is not constant, but differs based on the circumstances. In addition, the number of tasks to be performed is not constant; tasks can become obsolete and new tasks can appear which means that the decision making has to be performed online. It is also a real multi-agent problem, the agent do not have the same capabilities and have to cooperate to be successful. Yet, the article is also quite critical:

> "Unfortunately, the problems posed by the simulator and solutions to these have rarely, if at all, been defined, formalized and solved."

Although the RoboCup Federation sponsored short-visit does not have the ambition to define, formalize and solve all problems posed by the RoboCup Rescue simulator, we like to demonstrate how several state-of-the-art machine learning techniques, such as reinforcement learning, evolutionary computing and Markov decision processes, could be used to make optimal decisions for emergency responders.

[3] This is a progress report of a short-time visit sponsored by the RoboCup Federation

1.1 Explore, Extinguish and Refill

The fire brigades are responsible for extinguishing the fires that are spread in the disaster space. Selecting the fire zones is a prediction problem; the fire brigades must estimate the fire propagation, taking into account the properties of the building on fire, and the surrounding buildings, properties such as the construction method and the specifications of how much it is burning.

Every cycle the agent take decision and act, they can move, explore the disaster space, sense information of surroundings and receive information broadcasted by their teammates. The resultant behavior is the key to success when trying to reduce the human and material losses that may be caused by a disaster.

If one looks at the possible situations that may arise during a disaster, one finds that some events must be addressed before others. For instance, if a burning building has people inside, this place ought to be attended before a burning empty building. Depending on the situation a different priority value must be assigned, this value is decisive in determining the behavior of the agents. Just looking the possible set cases is clear that determining priority values is not an easy work.

As answer to the problem presented in the previous paragraphs of this document will be worked out in the following sections. First you will find the Controlled Actions and his expected impact on the agents behavior, next is presented the modified functional structure of the fire brigade agent, the next section includes two approaches for this multi-agent optimization problem; a Differential Evolution Algorithm and a Reinforcement Learning method. It would be interesting to see the difference in performance of those two methods, yet this comparison will be made in the final report.

1.2 Control actions

The priority assignment problem defines the mayor agent states, and based on this priority the fire brigade can chose one building and decide if will explore from outside, extinguish, explore from inside or refill his thank. The refill action is only possible if a refuge was chosen. So combining actions with buildings results in a big set of possible behaviors, which makes for instance the learning process difficult.

1.3 Agents Structure

In order to the control actions could govern the agent, the normal structure of the fire brigade was modified and divided in 5 basic sequential blocks; Sense, Identify, Decide, Action and Communicate, as illustrated in Fig. 1.

The Sense block is responsible of receiving the new environment disaster space information and pre-process it to allow his use in following stages.

On the Identify block the previous received information is used to update the agent model of world and for instance the path planning algorithm (was

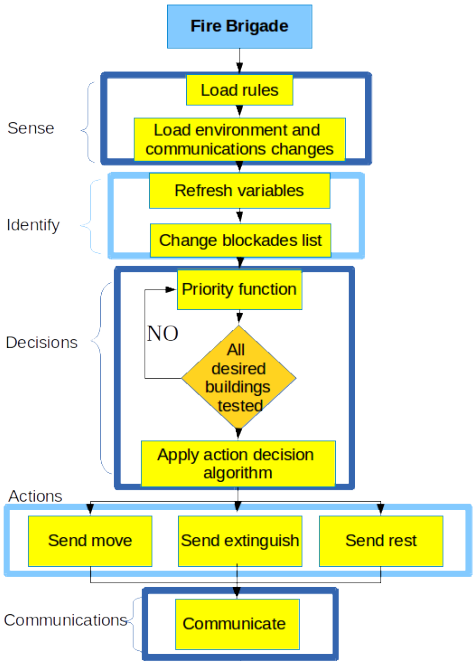

Fig. 1. Agents Structure

implemented a simple greedy algorithm to reduce the computational load) and blocked roads map.

At the Decide block was implemented the priority assignment algorithm and the actions decision algorithm; together define the next action commands to the kernel.

Inside the Action Block the priority list and the action is chosen and translated in action messages for the Kernel.

Finally the new environment information and the task on going is share on the Communication block. Note that the communication's structure follows an all-to-all broadcast scheme.

2 Differential evolution

The decisions are modeled with a Fuzzy Basis Function Expansion model [2]. The fuzzy membership function is a Mixture of Gaussian whose means, variances and weights were tuned using a differential evolution algorithm, as illustrated in Fig. 2. The differential evolution algorithm was chosen due to its advantages in performance and operation in optimization problems [3] compared to classical

evolution algorithms [4,5]. The main characteristic of a Differential Evolution Algorithm is that the new generation is generated thanks to the parameters of the best individual and the difference in chromosome of two random individuals.

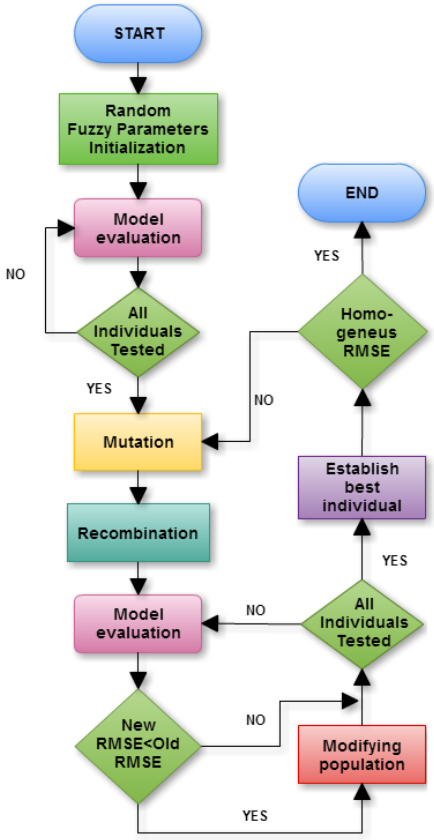

Fig. 2. Flowchart of the implemented differential evolution algorithm.

The Mutation number is the relation that determine the probability that have any individual to start the mutation process in which his chromosome could change, the Crossing number is a relation that determine the probability that have any part of the chromosome of an individual to change thanks to the characteristic of the best individual of the current generation and two others random individuals. These and other main characteristics are shown in Table 1 and Table 2 and described in the following sections.

population size	10
Generations	50
Mutation rate	0,4
Crossing probability	0,4
Fitness function	Score

Table 1. Differential evolution algorithm characteristics.

Fuzzy Functions	Gaussian
Inputs	7
Individuals	10
Outputs	7

Table 2. Fuzzy system characteristics.

2.1 Chromosome

Chromosome contains the characteristics of each individual, it is divided in two parts one for each evaluation (priority, and action) and each part into three sections: means, deviations and centers. Means and deviations are equal length to the product between the number of rules and the number of inputs. The center section is equal length to the number of rules. The parameters of the chromosome are represented with real numbers which are randomly initialized between 0 and 1, this in order to cover the whole spectrum of solutions.

2.2 Mutation

The mutation vector is generated from three individuals of the current generation, one of them is the best individual of the previous generation and the other two are randomly selected, all selected individuals are different in order to propagate the characteristics of all vectors [3].

2.3 Recombination

The recombination is performed individual by individual and is only possible if the fitness function of the mutated individual is better than the current individual.

2.4 Input Parameters

The input parameters are normalized representations of the environment and the agent characteristics, were selected taking into account the importance of the environment information that poses and previous experiments:

Tank Level: Quantity of water on board.
Buildings Area: Ground area of the buildings.

Distance: Euclidean distance between the agent and the building.
Temperature: Reported temperature of the building
Fieriness: State of the building that represent the fire state
Neighbors Fire energy: Fire energy of the adjacent buildings

The latter four parameters depend of the moment when the observation of the burning building was done. The latter two parameters to find the biggest fire in the map depend on a relation of the characteristics of each building involved.

3 Reinforcement learning

Reinforcement learning makes it possible to learn the optimal policy by a simple "trail-and-error" [6]. Yet, to apply such leaning process to the real world a method is necessary which reduces the number of trails required to learn over the way. A good example of such method is TEXPLORE [7].

Inspired by the TEXPLORE method, the Distribot team has divided its action selection algorithm in two phases: an *exploration* and an *exploitation* phase. The choice between *exploration* and *exploitation* is made randomly. During the *exploration* a search is started for the optimal parameters θ of the action selection function $E_\theta(b_i, p)$. The action selection function $E_\theta(b)$ calculates a score for all nearby buildings b_i, which can be sorted to find the next building to be extinguished. The parameters θ_j are in the range $[-1, +1]$ and one of this parameters is incremented or decremented with a value of 0.2, which is remembered as a step in the parameter space $\Delta\theta_k$. The score function is also of the input parameters p_j defined in section 2.4, which have a value in the range $[0, 1]$. The score function used in the action selection algorithm is defined as follows:

$$E_\theta(b_i, p_j) = \sum_{j=1}^{j<=6} \theta_j p_j(b_i) \tag{1}$$

Note that the first input parameter $p_1(b_i)$ is only a function of the agent, not the building b_i which could be selected, so this gives only an offset to the action selection function $E_\theta(b_i, p)$. The other input parameters $p_j(b_i)$ are strongly depended on the building b_i. The action selection function $E_\theta(b_i, p)$ is applied and an extinguish action is applied on building b_i. The simulator score after the action is interpreted as the reward and is stored for the combination $(\theta_j, \Delta\theta_k)$. The trace of steps of $\Delta\theta_k$ is also stored, which allows to trace back to all previous parameter combinations θ_j and the same reward is stored there (if the new reward is bigger than the old reward).

In the *exploitation* phase not a random change in the parameters $\Delta\theta_k$ is chosen, but the $\Delta\theta_k$ with the highest reward. In this way the optimal set of parameters for the action selection function $E_\theta(b_i, p)$ can be learned. The parameters θ_j will always be slightly modified, but it could be possible that a stable state can be found by modifying one parameter every time with a value of $+0.2$ followed by modification with a value of -0.2.

This approach can be interpreted as a standard Markov Decision Process (MDP) which is composed by a set of states S, a set of actions A, a reward function $R(s, a)$, and a transition function $P(s, a, s')$. The state S corresponds with the parameters θ_j of action selection function $E_\theta(b_i, p)$. The set of actions A corresponds with the modification of those parameters $\Delta\theta_k$. The reward function $R(s, a)$ is a lookup table of the combination $(\theta_j, \Delta\theta_k)$. The transition function $P(s, a, s')$ is a simple jump-table, because all modifications $\Delta\theta_k$ are performed in steps of 0.2. Note that by selecting as action set A the modifications $\Delta\theta_k$, the set has a fixed number of 12 actions. When instead the action set A' would have been chosen which would consist of extinguish the fire in one of the nearby buildings b_i, the MDP would scale with the dynamic number b_i (which could become quite big for large maps).

4 Results

The agents are trained on the small test world which is provided by the simulator (as illustrated in Fig. 6).

The fuzzy rules were tuned following the parameters of the Differential Evolution (DE) Table 1. These rules allow actions like move, extinguish and rest, and establish the priority buildings. The tuning process known as learning this is represented in the Fig. 3 and Fig. 4 where can be seen the score of the best individual, the worst individual and the average score of every generation.

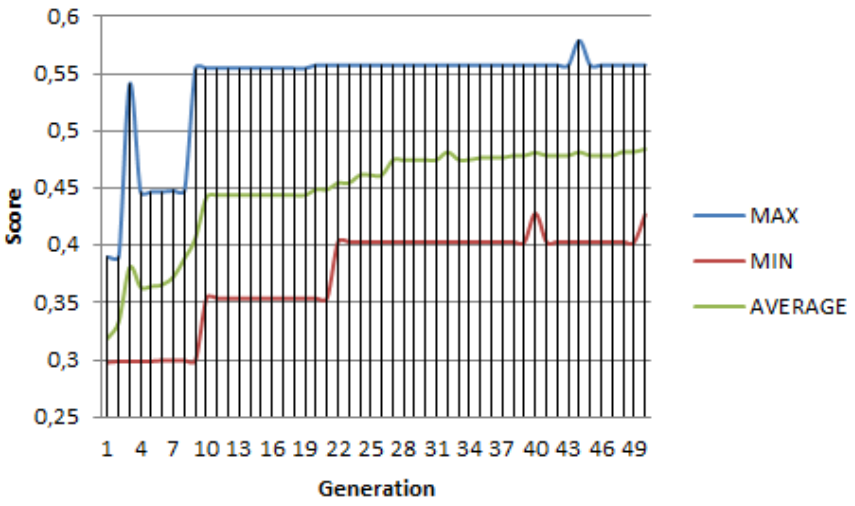

Fig. 3. Learning process

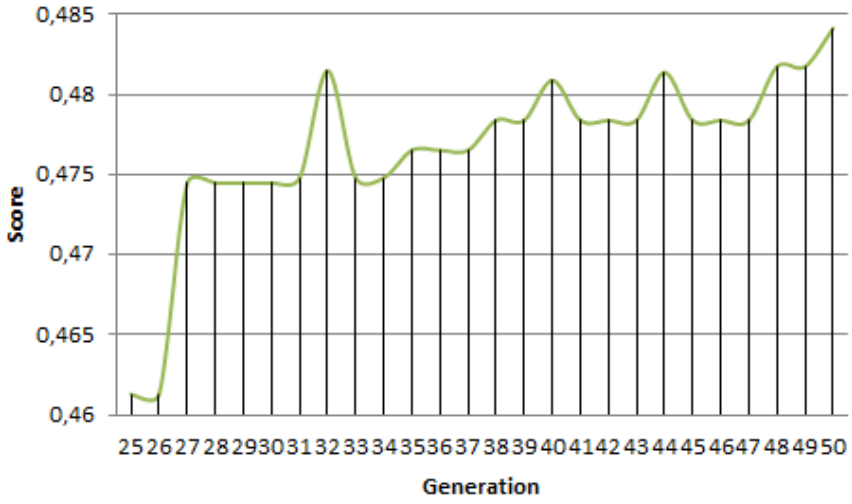

Fig. 4. Details average learning process

Based on the learning process previously presented were selected the average individuals of the generations 1, 5, 20, 30 and 50, and the best individual of the last generation to realize a deeper analysis. In Fig. 5 the score changes of the aforementioned individuals during a 70 cycles simulation are presented. In Fig. 6 can be seen screenshots of the simulation realized with the aforementioned individuals, were selected the simulation cycles 10, 40 and 70 to present more details of the found solutions.

As can be seen in Fig. 5, initially two-thirds of the buildings is burnt down. Every 10 generations the performance improves, until at the end the score is nearly two times as high.

On Fig. 6 examples are shown how the disaster evolves for different generations. Each column of 6 shows the situation at the time cycles 10, 40 and 70. Each of rows shows an average individual typically of that generation.

The average individual of generation 1 shows an undesired behavior; all the fire brigades chose the same building and remained inside during all the simulation time as can be seen in the Fig. 6 (a) (b) (c)

The average individual of the generation 5 exhibit a preference for large buildings, as can be seen in the Fig. 6 (d), all agents checked and waited in front of the biggest building of the disaster space, and when fires are perceived, by an ineffective way the agents start extinguish the big buildings of the disaster space Fig. 6 (e)(f). Is important to note here that the average agents have learned that use the extinguish command increase the final Score.

The Fig. 6 (g) shows the disaster space state at time 10 with the average individual of the generation 20. Here can be seen that the agents have learned that throwing water to non-fire buildings is a useful preventive action. In Fig. 6

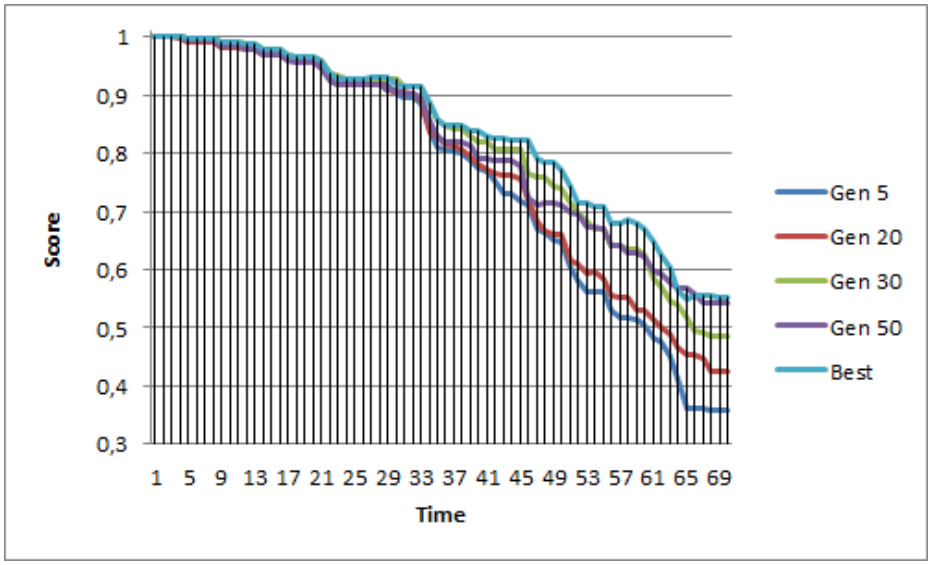

Fig. 5. Score Vs Time of average individuals

(h)(i) can be seen that the preference for large buildings that was observed in Fig. 6 (d)(e)(f) is still determinant.

For first time in the present tested group, the average individual of generation 30 shows a preference for small area buildings, and the undesired behavior of throwing water to refugees as can be seen in the Fig. 6 (j). The Fig. 6 (k) shows the agents extinguish a medium size building that is far of the first efforts to control the fire expansion, this policy is not good; the area at the right which was under control in the earlier generation (see Fig. 6 (i)) is now on fire (see Fig. 6 (l)) and the far extinguished buildings at top catches fire again.

The average individual of the generation 50 exhibit that the exploratory behavior is prioritized over the actions to extinguish small fires as can be seen on Fig (m), where instead a small fire was detected the fires brigades continue the disaster space exploration. in the Fig (n)(o) can be seen that the individual is choosing the bigger buildings that are close to the actual position.

The Fig. 6 (p)(q)(r) shows the best individual of the generation 50, even though that present the best final score this individual have the undesirable behavior of throwing water to refuges (as can be seen in Fig. 6 (j)). The Fig. 6(q) shows preferences to buildings on fire with small area and close to the current location, the advantages of this politics can be seen in fig. 6 (r) that shows a bigger controlled fire area.

5 Outreach

During the short-time visit Iván Riaño gave one colloquium presentation and attended colloquia from visiting machine learning researchers:

- Iván David Riaño Salamanca, "A Differential Evolution Algorithm for tuning Rescue Agents", Intelligent Autonomous Agent Colloqium, Universiteit van Amsterdam, June 10, 2014
- Ethem Alpaydin, "Design and Analysis of Machine Learning Experiments", Informatics Institute colloquium, Universiteit van Amsterdam, June 10, 2014.
- Bert Kappen, "A statistical physics perspective of control theory", Intelligent Systems Laboratory Amsterdam Colloqium, Universiteit van Amsterdam, June 24, 2014.

In October 2014 Arnoud Visser will visit Colombia during the Robotics week. At this event he will give a keynote speech and a tutorial.

6 Conclusion

The exchange of researchers from Columbia and The Netherlands has been successful in making the RoboCup Rescue Simulation League more visible in both countries. The visit has been used to exchange ideas about how state-of-the-art machine learning techniques could be used to learn to take the right decisions. The potential of this approach have been demonstrated with an evolutionary algorithm, which demonstrated that the efficiency of the policies could be improved by nearly a factor two for a small testing world.

Acknowledgment

We like to thank the RoboCup Federation for making this exchange visit possible.

References

1. Ramchurn, S.D., Farinelli, A., Macarthur, K.S., Jennings, N.R.: Decentralized coordination in robocup rescue. The Computer Journal (2010) bxq022
2. Yin, T.K., Lee, C.G.: Fuzzy model-reference adaptive control. Systems, Man and Cybernetics, IEEE Transactions on **25** (1995) 1606–1615
3. Villate, A., Rincon, D.E., Melgarejo, M.: Sintonización de sistemas difusos utilizando evolución diferencial. Laboratorio de Automática, Microelectrónica e Inteligencia Computacional, LAMIC (2011)
4. Price, K., Storn, R., Lampinen, J.: Differential Evolution a Practical Approach to Global Optimization. Natural Computing Series. Springer (2005)
5. Moraglio, A., Togelius, J.: Geometric differential evolution. In: Proceedings of the 11th Annual conference on Genetic and evolutionary computation (GECCO '09), ACM (2009) 1705–1712

6. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. Adaptive Computation and Machine Learning. MIT press (1998)
7. Hester, T., Stone, P.: The open-source texplore code release for reinforcement learning on robots. In Behnke, S., Visser, A., Xiong, R., Veloso, M., eds.: RoboCup-2013: Robot Soccer World Cup XVII. Lecture Notes in Artificial Intelligence. Springer Verlag, Berlin (2013)

(a) Gen 1, Time 10 (b) Gen 1, Time 40 (c) Gen 1, Time 70

(d) Gen 5, Time 10 (e) Gen 5, Time 40 (f) Gen 5, Time 70

(g) Gen 20, Time 10 (h) Gen 20, Time 40 (i) Gen 20, Time 70

(j) Gen 30, Time 10 (k) Gen 30, Time 40 (l) Gen 30, Time 70

(m) Gen 50, Time 10 (n) Gen 50, Time 40 (o) Gen 50, Time 70

(p) Best individual, Time 10 (q) Best individual, Time 40 (r) Best individual, Time 70

Fig. 6. Simulation of selected individuals learning process