Final report
”Establishing bonds for the advancement of the Rescue League”

Iván David Riaño Salamanca[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 evaluate the effect of several 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 decisions as posed by the RoboCup Rescue Simulation League [1] is nicely described by Jennings [2]. 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 heterogeneous multi-agent problem; the agents do not have the same capabilities and have to cooperate to be successful.

Although the goal of the RoboCup Federation sponsored short-visit is not 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.

For the RoboCup competition, decisions have to be made for fire, ambulance and police agents. Yet, the actions of the fire agents have a direct impact on the score, while the actions of the ambulance and police agents have a less direct effect on the score. To clearly demonstrate the effect of state-of-the-art machine techniques, both the studies in Amsterdam and Bogota concentrated on the decisions for fire agents [3,4].

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

1.1 Strategic and Tactical decisions

The fire agents are responsible for extinguishing the fires that are spread over the city. A decision model which takes all relevant information into account is demonstrated to be intractable [5]. The approach of Amsterdam [4] is to distinguish a macro-level behavior responsible for the strategic, high-level decisions and a micro-level behavior dealing with the local particularities of the environment. For the strategic decision on macro level one takes partial observability into account, while the tactical decision on the micro level could be implemented with a simpler Markov decision process. The focus of the thesis was on the strategic decisions, which show excellent results. The parameters for the tactical decisions were learned by both Q-learning and SARSA [6] which resulted in fast convergence to good performances.

The approach of Bogota [3] is to optimize tactical decisions with state-of-the-art machine techniques as reinforcement learning and evolutionary computing. Here both techniques outperformed a handcrafted policy.

2 Differential evolution

The decisions of the fire agents are based on an action selection function which couples a priority to each action. In practice, this is a utility function $E_\theta(b)$ which calculates a score for all nearby buildings b_i, which can be sorted to find the next building to be attended b_b, and then $E_\theta(b)$ is used to calculate the action to make in the building (extinguish, explore or in case of refuge refill the tank). The utility function $E_\theta(b)$ is parameterized with a vector θ_k, which contains 36 scalars which should be learned. This utility function is estimated with a fuzzy inference system, that uses Gaussian membership functions defined by standard deviations σ_{ij} means μ_{ij} and a T-norm product balanced with the weights y_i, as indicated in Eq. 1, where m is the number of input parameters of the fuzzy system and n is the number of Gaussian membership functions per input.

First is found the most important building to be attended b_b for that $E_\theta(b)$ is calculated with $x_{ji} = p_j(b_i)$, $\sigma_{ij} = \theta_k; (k = 0 - 11)$, $\mu_{ij} = \theta_k; (k = 12 - 23)$, $y_i = \theta_k; (k = 24 - 25)$, and $m = 6$.

$$E_\theta(b) = \sum_{i=0}^{i<=n-1} y_i \prod_{j=0}^{j<=m-1} MAX(exp(-\frac{1}{2}.\frac{(x_{ji} - \mu_{ij})^2}{\sigma_{ij}^2})) \tag{1}$$

As a second step, for this building b_b the best action $E_\theta(b_b)$ is calculated, for that $x_{ji} = p_j(b_b)$, $\sigma_{ij} = \theta_k; (k = 26 - 29)$, $\mu_{ij} = \theta_k; (k = 30 - 33)$, $y_i = \theta_k; (k = 34 - 35)$, and $m = 2$. In both cases two Gaussian membership functions per input ($n = 2$) were chosen, which allows to represent a multi-modal distribution. This decision was based on initial experiments with the first 25% of the training set, which showed that one membership function produced a slow learning process, while more than three membership functions increased significantly the processing time. The meaning of the input parameters $p_j(b_i)$

are given in Section 2.3. The influence of each of the input parameters can be positive or negative. The parameters of the membership functions θ_k could be tuned with a differential evolution algorithm [7].

A differential evolution algorithm works with vectors of continues numbers (in this case a vector with 36 values θ_k). Each generation consist of a list of possible instantiations of these vectors. From this list each time a target $\overrightarrow{t}$ and a base vector $\overrightarrow{b}$ are chosen. The base vector $\overrightarrow{b}$ is modified (if a random value is bigger than the crossing probability) by the respective value of a difference vector $\overrightarrow{d}$, which is the difference between two other random chosen vectors, into a mutated vector $\overrightarrow{m}$. The difference vector $\overrightarrow{d}$ defines the step size of the search, and should get smaller when the algorithm converges. A new trail vector $\overrightarrow{n}$ is generated, which is for each of the elements of the vector is the element of the target vector $\overrightarrow{t}$ or the element of the mutant vector $\overrightarrow{m}$. This process is called crossover. The score of the mutated vector $\overrightarrow{m}$ is compared with the score of the base vector $\overrightarrow{b}$. When the score is better, the new trail vector replaces the $\overrightarrow{n}$ target vector $\overrightarrow{t}$ in the next generation. The variant of a differential evolution algorithm chosen here (*DE/best*) takes as base vector $\overrightarrow{b}$ always the vector with the best fitness score and define the difference vector $\overrightarrow{d}$ as the difference between two random vectors multiplied by a mutation rate F, plus the vector with the best fitness score.

Differential Evolution is ideal to work with continues variables, such as the values θ_k in the function $E_\theta(b)$. Further the algorithm is able to cope with multi-modal distributions and can automatically adapt the step-size while going from global search to fine-grained, local search.

The mutation rate F is a constant which scales the difference-vector, for each element $m_j = b_j + F * d_j$, where d_j is the difference between the elements of two random chosen vectors from the previous generation ($d_j = r_{1j} - r_{2j}$). The crossing probability CR determines the probability if an element of the target vector t_j or an element of the mutant vector m_j is chosen. The chosen learning parameters are shown in Table 1 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 paramaters.

2.1 Mutation

The mutation vector $\overrightarrow{m}$ is generated from three individuals of the current generation, one of them is the best individual of the previous generation $\overrightarrow{b}$ and

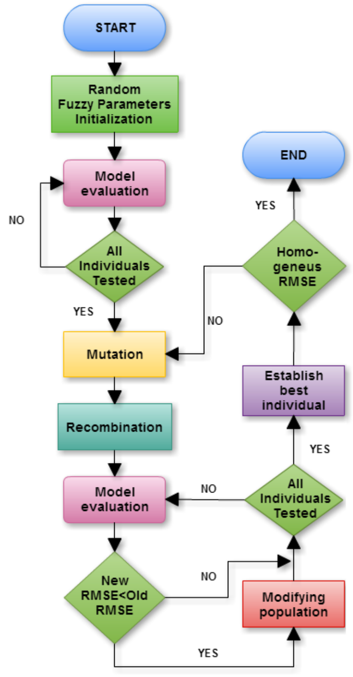


Fig. 1. Flowchart of the implemented differential evolution algorithm.

the other two are randomly selected ($\overrightarrow{r_1}$ and $\overrightarrow{r_2}$). Care has been taken that all selected individuals are different in order to propagate the characteristics of all vectors [8].

2.2 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.

2.3 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 (p_0).
Temperature: Reported temperature of the building ($p_1(b_i)$).
Distance: Euclidean distance between the agent and the building ($p_2(b_i)$).
Buildings Area: Ground area of the buildings($p_3(b_i)$).

Fieriness: State of the building that represent the fire state ($p_4(b_i)$)
Neighbors Fire energy: Fire energy of the adjacent buildings ($p_5(b_i)$).

The latter four parameters depend of the moment when the observation of the burning building was done. The latter parameter to find the biggest fire depends on a relation of the characteristics of each nearby building.

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 [9].

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 2.3, which have a value in the range $[0, 1]$. The score function used in the action selection algorithm is the same as the function used in the Differential Evolution algorithm:

$$E_\theta(b_i, p_j) = \sum_{j=0}^{j<6} \theta_j p_j(b_i) \tag{2}$$

Note that the first input parameter $p_0(b_i)$ 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).

Several experiments have been performed to find a good balance between *exploration* and an *exploitation*. As described in [3], a ratio of 1:10 gave the best result. The reinforcement learning converged typically after 1500 simulations.

4 Results

The agents are trained on the small test world which is provided by the simulator (as illustrated in Fig. 4).

The action selection rules were optimized with the Differential Evolution (DE) algorithm described in section 2, which establish the priority between the nearest buildings for actions as move and extinguish. The progress in learning can be seen Fig. 2, where both the score of the best individual, the worst individual and the average score of every generation is steadily increasing.

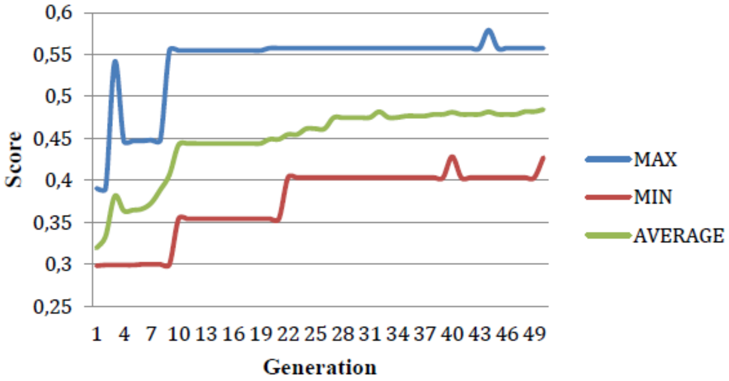


Fig. 2. Learning process for Differential Evolution

Based on the learning process previously presented were selected average individuals of each of the generations 1, 5, 20, 30 and 50 were selected for a deeper analysis. In Fig. 3 the score changes of the aforementioned individuals during a 70 cycles simulation are presented. In Fig. 4 can be seen screenshots of the simulation realized with the aforementioned individuals.

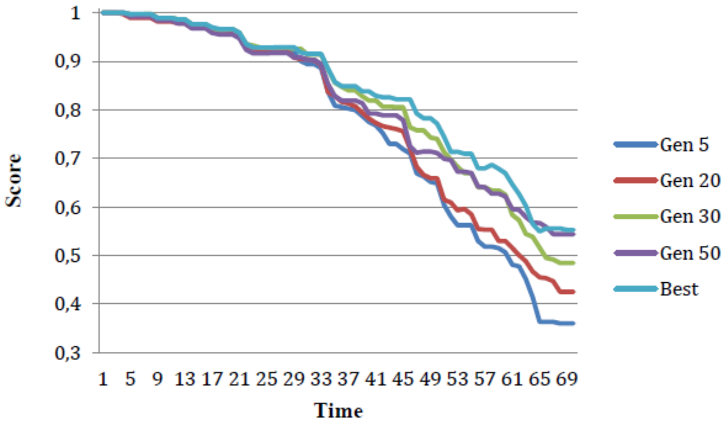


Fig. 3. Score Vs Time of average individuals of each generation

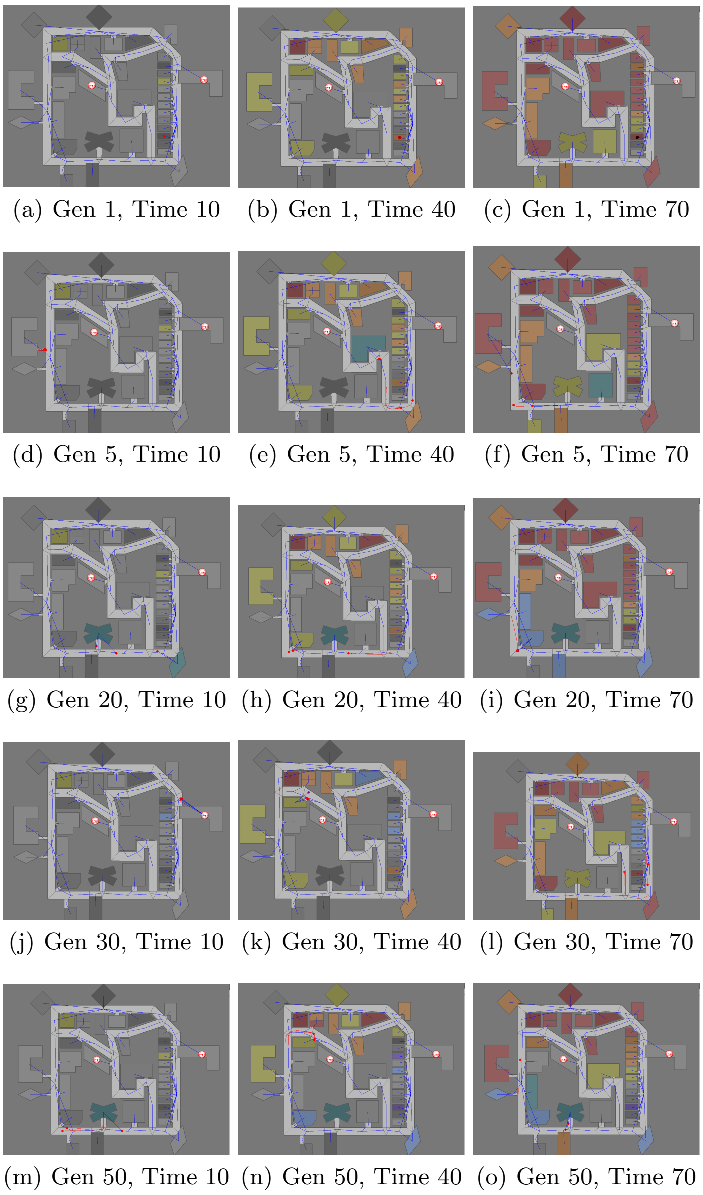
As can be seen in Fig. 3, initially most of the buildings are burnt down. Every 10 generations the performance improves, until at the one-third of the buildings are saved.

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. 4(a-c).

The average individual of the generation 5 exhibit a preference for large buildings, as can be seen in the Fig. 4(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. 4(e-f). Is important to note here that the average agents have learned that use the extinguish command increase the final score.

The Fig. 4(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. 4(h-i) can be seen that the preference for large buildings that was observed in Fig. 4(d-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. 4(j). The Fig. 4(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



(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

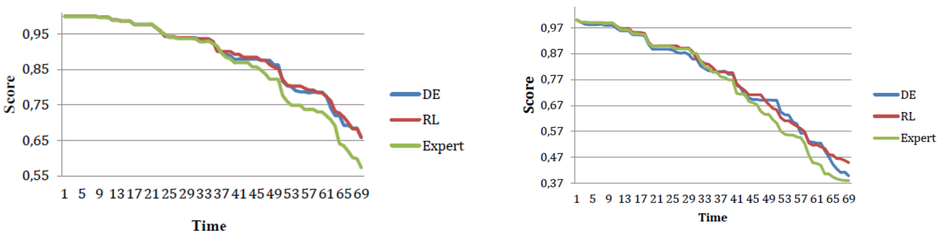
Fig. 4. Simulation of selected individuals learning process

was under control in the earlier generation (see Fig. 4(i)) is now on fire (see Fig. 4(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. 4(m), where instead a small fire was detected the fires brigades continue the disaster space exploration. in the Fig. 4(n-o) can be seen that the individual is choosing the bigger buildings that are close to the actual position.

4.1 Comparison of methods

The Differential Evolution (DE) and Reinforcement Learning (RL) algorithm described in the previous sections are compared to a handcrafted policy for a number of scenarios. In Fig. 5 the three algorithms are compared for scenario T1 and T3 (with respectively 1 and 9 fire agents). Both the DE and RL algorithms outperform the handcrafted policy, although they are not able to get the fires fully under control (which would have resulted in horizontal line).



(a) Scenario T1

(b) Scenario T3

Fig. 5. Comparison of Differential Evolution (DE), Reinforcement Learning (RL) against a handcrafted policy (Expert)

Yet, the differences between the learning algorithms and the handcrafted policy are not substantial. This can partly be attributed to the fact that it is difficult to design an experiment which contains such a challenge that smarter algorithms can outperform less advanced algorithms. At competitions one often see maps which are too hard (all teams fail to get the situation under control, the only question how long they could postpone this fate) or too easy (all teams get the situation under control, the only question is how fast). A nice example of designing a challenge which is just right is described in the utility function experiment of the master thesis [4]. At the Kobe-hard map (Fig. 4.19) four different utility functions show differences in performance in the order of 10%. Yet, it was hard work to design such a challenge. Unfortunately, this challenge is computationally too large for a comparison of DE and RL algorithms.

5 Publications

The initial results of this visit were published in a mid-term report [10], which was presented on the poster session of the RoboCup symposium. The final result of this cooperation are two theses; one bachelor thesis from the Universidad Distrital Francisco José de Caldas [3] and a master thesis from the Universiteit van Amsterdam [4].

The bachelor thesis [3] compares three actions selection algorithms. The first algorithm is based on Expert Knowledge, the second one in Reinforcement Learning, and the last one on tuning of priorities using Differential Evolution. Parameters of the aforementioned methods were optimized for the characteristics of the RoboCup Rescue Simulator System. The experimental results shows that in this setting the Differential Evolution and Reinforcement Learning approaches outperform the Expert knowledge approach.

The master thesis [4] has chosen a Decision Theoretic Planning paradigm as underlying framework. However, due to the complexity of the considered problem, using a single model for the agents' behavior could prove unfeasible. A rich enough model taking into account all relevant particularities of the domain would be intractable, while a much simpler model would abstract away too much information. As such, a hierarchical control structure is considered, with a macro-level behavior responsible for the strategic, high level decisions and a micro-level behavior dealing with the local particularities of the environment.

Several method were considered in developing the macro level behavior, including one based on the Bayesian Game Approximation, an algorithm for finding approximate solutions in multi-agent partially observable domains, and another following the DCOP formulation, a popular framework in the scholarly literature studying this domain. The micro level behavior was implemented using a simpler MDP based method, due to the strict computation limits.

Experimental results show a good overall performance of the methods considered, with their differences being better highlighted on harder, custom made configurations of the official contest maps.

6 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 has visited Colombia during the Robotics week. At this event he has given a tutorial and keynote speech titled 'The RoboCup Initiative - embody Artificial Intelligence research with a soccer challenge'.

(a) Primary school presentation

(b) Robot competition presentation

(c) Tutorial in the main lecture hall. ©ICT Ministry Bogotá

Fig. 6. Impressions from the Bogota Robotics week

For this event the organization had arranged four exhibition halls, with presentations from industry (hall 1), university (hall 2), primary & high-schools (hall 3) and robot competitions (hall 4).

In the last hall Colombia's small size team was present, together with the IEEE competition 'Robot at the Park' (high speed robot racing along a black line).

7 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 a reinforcement learning and an evolutionary algorithm, which demonstrated that the efficiency of the hand-crafted policies could be improved by nearly a factor two for a small testing world.

Acknowledgment

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

References

1. Visser, A., Ito, N., Kleiner, A.: RoboCup rescue simulation innovation strategy. In: Proceedings of the 18th RoboCup Symposium. (2014)
2. Ramchurn, S.D., Farinelli, A., Macarthur, K.S., Jennings, N.R.: Decentralized coordination in robocup rescue. The Computer Journal (2010) bxq022
3. Riaño Salamanca, I.: Diseño de un sistema auto organizado de agentes de rescate aplicado al robocup rescue agents simulator system. Bachelor thesis, Universidad distrital Francisco José de Caldas (2014)
4. Trăichioiu, M.: Hierarchical decision theoretic planning for robocup rescue agent simulation. Master's thesis, Universiteit van Amsterdam (2014)
5. Oliehoek, F.A., Visser, A.: A Decision-Theoretic Approach to Collaboration: Principal Description Methods and Efficient Heuristic Approximations. In: Interactive Collaborative Informations Systems. Volume 281 of Studies in Computational Intelligence. Springer-Verlag, Berlin Heidelberg (2010) 87–124
6. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. Adaptive Computation and Machine Learning. MIT press (1998)
7. Price, K., Storn, R., Lampinen, J.: Differential Evolution a Practical Approach to Global Optimization. Natural Computing Series. Springer (2005)
8. Villate, A., Rincon, D.E., Melgarejo, M.: Sintonización de sistemas difusos utilizando evolución diferencial. Laboratorio de Automática, Microelectrónica e Inteligencia Computacional, LAMIC (2011)
9. 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)
10. Riaño, I., Visser, A.: Mid-term report "establishing bonds for the advancement of the rescue league". In: Proceedings of the 18th RoboCup Symposium. (2014)