Proceedings of the Federated Conference on
Computer Science and Information Systems pp. 643–650

ISBN 978-83-60810-22-4

Modeling Agent Behavior Through Online Evolutionary and Reinforcement Learning

Robert Junges and Franziska Klügl
Modeling and Simulation Research Center
Örebro University, Sweden
Email: {robert.junges,franziska.klugl}@oru.se

***Abstract*—The process of creation and validation of an agent-based simulation model requires the modeler to undergo a number of prototyping, testing, analyzing and re-designing rounds. The aim is to specify and calibrate the proper low-level agent behavior that truly produces the intended macro-level phenomena. We assume that this development can be supported by agent learning techniques, specially by generating inspiration about behaviors as starting points for the modeler. In this contribution we address this learning-driven modeling task and compare two methods that are producing decision trees: reinforcement learning with a post-processing step for generalization and Genetic Programming.**

I. Motivation

AGENT-BASED simulation as an innovative paradigm is particularly apt for the modeling and analysis of complex systems. Based on (mostly) local, low-level interactions, the agents together produce some higher level phenomenon. This bottom-up approach (see [1] coining the notion of social science from the bottom-up) supports the understanding of why and when a phenomenon emerges. It goes beyond only describing macro-level behavior or pattern and requires a highly expertise-based development process for the model. It is basically exploratory as – specially with emergent phenomena – the explicit link between the agent and the system level is missing. Thus, the success of a modeling process is highly depending on the experience of the modeler about what low level behavior might generate the desired macro-level phenomenon.

However, if we want to make agent-based simulation accessible to more people – specially to people without experience in modeling and simulating complex systems –, new ways of systematically developing agent-based simulation models have to be tackled. Our idea is to use adaptive agent architectures for enabling the modeler to develop the model on a higher abstraction level, assuming that this approach will make modeling easier. That means the modeler focuses on the characterization of the phenomenon he/she is interested in, and based on given functionality of effectors and sensors, the behavior model of the agents is developed in a self-adaptive way. Finally, we hope to establish a learning-driven analysis and design approach using self-adaptive agents.

Our main objective for this contribution is to explore the suitability of different learning techniques for a particular modeling problem. Thus our idea is not to evolve or learn a perfect behavior control, but a behavior model for which the source code can be understood by a human modeler. Decision trees form an obvious behavior representation candidate for this task: they are an intuitive representation for decision-making processes and there are a number of learning techniques that can operate on them. However, supervised decision tree learners such as C4.5[2] or other classification techniques[3] have requirements for a sufficient number of appropriate cases – in our case good situation-action assignments – that cannot be applied directly.

In the following, we first give a short survey on related work concerning modeling for simulated agents and learning technology. Then we will shortly describe the two techniques that we want to analyze here: a combination of Q-Learning with a Decision Tree learning for generalization, and Genetic Programming. After a short introduction of our simple test scenario, we will provide a set of experiments and results. The paper ends with a conclusion and next steps.

II. Self-Adaptive Agents Modeling

We propose a learning-driven analysis and design approach for using self-adaptive agents in the behavior modeling task. This approach is based on the following core idea: an appropriate conceptual model of the overall system can be developed by setting up a simulation model of the environment allowing to evaluate agent performance and integrating agents that may learn their decision making behavior.

The design strategy starts with the definition of an environmental model together with a function that evaluates agent performance in this environment. After that, the modeler determines what an individual agent shall be able to perceive and to manipulate. In the next step, the designer selects an appropriate combination of agent learning procedures, used by the agents to determine a behavior program that generates the intended overall outcome in the given environment. At the end of this process, ideally a decision tree representing the agent behavior is available in a way that it fits the environmental model and the reward given and thus produces the aggregate behavior intended.

The basic assumption is that the learned decision tree then is sufficiently elaborated that it can serve as a starting point for further steps in a development methodology, such as technical design or implementation. Thus, we transfer the initial agent behavior design from the human modeler to a simulation system. This strategy could be also described as a variant

of an environment-driven strategy for developing multiagent simulation models[4].

Specially in complex systems, a higher number of degrees of freedom have to be handled. This could make a manual modeling process cumbersome, particularly when knowledge about the requirements for the overall system or experience for bridging the micro-macro gap are missing. We assume that using agents that learn at least parts or initial versions of their behavior is a good idea for supporting the modeler in finding an appropriate low level behavior model.

III. Modeling and Learning

Adaptive agents and multiagent learning have been one of the major focuses within Distributed Artificial Intelligence since its very beginning [5]. The following paragraphs shall give a few general pointers and then a short glance on directly related work on agent learning for behavior design, not for optimizing. It is important to keep in mind that the objective of our work is not addressing mere learning performance but suitability for the usage in a modeling and analysis support context.

Reinforcement learning [6] and evolutionary computing [7] are recurrent examples for categories of learning techniques applied in multiagent scenarios.

A reinforcement learning approach for automatically programming a behavior-based robot is described in [8]. Using Q-Learning algorithm, new behaviors are learned by trial and error, based on a performance feedback function as reinforcement. In [9], also using reinforcement learning, agents share their experiences and most frequently simulated behaviors are adopted as a group behavior strategy. The authors conclude that both learning techniques are able to learn the individual behaviors, sometimes outperforming a hand coded program, and behavior-based architectures speed up reinforcement learning. However, these approaches are for learning "controllers". The actual behavior program is secondary as long as the given tasks are fulfilled.

Evolutionary Computing (EC) has also been applied for behavior generation in multiagent systems. In [10], Genetic Programming (GP) is used to evolve agent behavior in a Predator and Pray scenario. Agents derive their decision-making process from a decision tree model, that is built throughout the execution. Agent performance is the focus.

Additionally, [11] and [12] discuss the performance of evolutionary generated behavior in multiagent systems. The former presents optimization problems solved by single agent and multiagent approaches. The latter presents a robotic soccer scenario and the concept of layered learning, as a form of problem decomposition.

In [13], Denzinger and Kordt propose a technique for generating cooperative agent behavior using evolutionary online learning. An experiment is developed for this multiagent scenario applied to a pursuit game, where agents are guided by situation-action pairs, or SApairs. The authors compare the proposed online approach with an offline approach for the same problem. In addition to their own SApairs, the agents have in their memory SApairs that model the other agents' behaviors. These pairs can be added by observing other agents or by communicating from one agent to another. The results show that incorporating this online learning phase improves the agents' performance in more complex variants of the scenario and when randomness is introduced, when compared to the offline approach. In the proposed online learning strategy the agents are not required to learn a complete strategy, but only how to perform well in the next steps, after the learning.

Some differences can be pointed out between the work of this paper and the work presented in the last paragraph: in their case, although the results of the learning phase are used in an online way, the learning itself is offline, it does not take place in the "real" scenario where all agent are adapting at the same time; they use Genetic Algorithms (GA), where the individuals are composed of a number of SApairs and genetic operators act on SApairs – not changing its content, but exchanging them among individuals. This way, the GA only operates on creating new programs (individuals with their rules) and not on creating new rules, as it happens in our GP approach, presented in section IV-B; they focus on learning parts of the problem as the execution evolves, and not to learn a complete model of the agent for the problem, as in our case; they assume the agents have knowledge about all the scenario, which can be unrealistic in certain scenarios. Since their focus is on cooperative behavior, a model of other agents is crucial for the success of the learning.

To extend the work presented in [13] and improve the performance of the GA: in [14], the authors include a mechanism to collect data about the usage of the SApairs – such as number of times used and how it changed the fitness – and use this information for applying the genetic operators during the evolutionary learn phase; in [15], the authors present several case studies encoding application specific features into the fitness function. The conclusion is that there is not much difference in terms of performance by refining knowledge already available in the fitness function, however adding new application knowledge improves the performance significantly; in [16], the authors address the problem of modeling other agents' behavior. The authors point that a model of other agent generated out of few observations often results in inaccurate predictions, while a model comprised of many observations decreases the efficiency of the modeling process. To address the first issue it is proposed to use a method with stereotypes, where the agent, based on current observations can classify the other agent in one of these stereotypes – which in their turn are composed of a set of SApairs – and therefore have a model of the other agent behavior. The second problem is addressed by building trees that branch at each level according to a different feature. The idea is to create a tree-like compact representation, reducing the model to only the relevant observations. The conclusions indicate that when a correct matching of stereotypes is made, there is a significantly increase of performance, and the compactness of representation through trees is a promising approach, but deserves more analysis, specially to minimize the risk of building a model that ignores important observations.

Although there is a wealth of publications dealing with the performance of particular learning techniques, there are not many works focusing on the resulting behavioral model aiming at understandability by a human interpreter. In our learning-driven design approach we transfer the initial agent behavior design from the human modeler to a simulation system. We assume that using agents that learn at least parts or initial versions of their behavior is a good idea for supporting the modeler in finding an appropriate agent-level behavior model.

Nevertheless, the basic question on a way to such a learning-driven analysis methodology is about the availability of appropriate learning techniques, for this form of application, for a particular domain, or maybe just for a particular system. In a previous work we evaluated the applicability of reinforcement learning techniques for this purpose [17]. One of the main problems is the interpretability of the resulting behavior program of the agents. To overcome this problem, in [18] we proposed to use a decision tree learner for post-processing the situation-action pairs with the highest fitness values. However, we reported issues with convergence in RL affecting the generated decision tree, when evaluating its size and quality. In the present work we start our investigation with Genetic Programming. The aim is to overcome the search-space exploration problem – present in reinforcement learning techniques – and include compactness and generalization in the behavior representation by directly working with a decision tree model.

IV. Approaching Agent-Learning

In this section we describe the learning techniques chosen for evaluation in this contribution. Their implementation, as well as the experiments conducted – presented in section V – used the multiagent simulation tool SeSAm (www.simsesam.org).

A. Q-Learning plus C4.5: RL+

Reinforcement Learning is a well-known machine learning technique. It works by developing an action-value function that gives the expected utility of taking a specific action in a specific state. We selected Q-Learning [19] for our investigation. In this technique, the agents keep track of the experienced situation-action pairs by managing the so-called Q-Table, that consists of situation descriptions, the actions taken and the corresponding expected prediction, called Q-Value. Nevertheless, the use of the Q-Learning algorithm is constrained to a finite number of possible states and actions. As a reinforcement learning algorithm, it is also based on modeling the overall problem as Markov Decision Processes (MDP). Thus, it needs sufficient information about the current state of the agent for being able to assign discriminating reward. The Q-Learning algorithm could be implemented by means of the standard high-level behavior language in SeSAm.

However, Q-Learning only gives us rules, mapping the agents perceptions to possible actions and their expected utility, and we need to generate a decision tree representation of the implicit behavior of these rules. Since the number of generated rules can be large, we suggest to use a post-processing step for improving the analysis of the rule set on a detailed level. In this contribution we focus on using the C4.5 algorithm [2] to generate decision trees, using the rules generated by Q-Learning as the input. For a better description of the requirements and implementation of Q-Learning and C4.5 please refer to [18].

The decision tree, returned at the end of the simulation process for a given agent, is generated by selecting the best rules from the Q-Table (the ones with higher Q-Value) and applying them in the C4.5 algorithm. To do that, we map the individual components of the situation description as the features of the algorithm, and the action corresponding to that situation as the correct classification.

B. Genetic Programming

Genetic Programming (GP) is an Evolutionary Computation (EC) technique that aims at solving problems by evolving a population of computer programs. New populations of, hopefully better, programs are created in each generation using the previous generation as the input for transformation operators [20]. Evolution is processed on the basis of the Darwinian principle of natural selection (survival of the fittest) and variations of natural processes, such as sexual recombination (crossover), mutation and duplication.

In our approach, we integrate a standard implementation of the genetic programming functionalities into SeSAm, as described in [21]. Agent behavior programs are represented as decision trees, coded as strings. This way of representing behavior requires the mapping of the agents' perceptions and actions, to the functions (nodes) and terminals (leaves) sets of decision tree. To cope with this, the simulations needs components for: using a decision tree encoded in the string format to determine the actions of an agent; managing the population of behavior programs and evolving the strings.

The GP component provides the set of primitives necessary to handle the decision trees. The central primitive is the **Evaluation** primitive. It receives a decision tree string and the description of a situation from the simulation, and outputs a terminal, that the agent may map to a sequence of action calls. Other primitives implement the creation of new individuals and the genetic operators for crossover and mutation.

We define a unique pool of strings, that are used and evaluated collectively by all agents in the simulation. The finally returned decision tree is the tree with the highest fitness overall generations.

V. Scenario and Performance

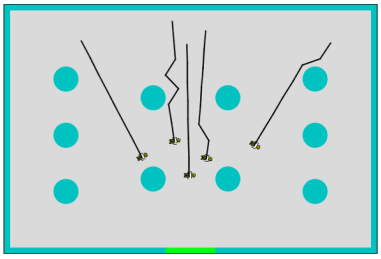
For this contribution we selected a pedestrian evacuation scenario. Although it is a simple problem, it provides a way of evaluating the requirements of the learning techniques and point out the challenges.

A. Scenario Description

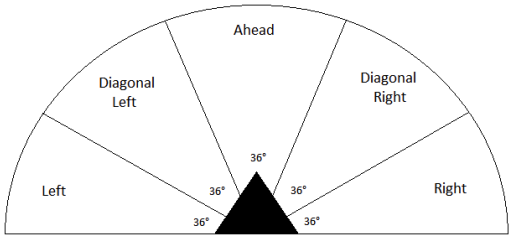
The scenario, depicted in Figure 1(a) consists of a room (20 x 30m) surrounded by walls with one exit and 10 column-type

obstacles (with a diameter of 2m). In this room a number of pedestrians are placed randomly at the upper half part and shall leave as fast as possible without hurting during collisions. We assume that each pedestrian agent is represented by a circle with 50cm diameter and moves with a speed of 1.5m/sec. One time-step corresponds to 0.5sec. Space is continuous, yet actions allow only discrete directions. We tested this scenario using 2 and 4 agents.

Agents can perceive obstacles and the exit when within a field range of 2m. These perceptions are divided in five areas, as depicted in Figure 1(b). Additionally, the agents hold a binary perception telling them whether the exit is to their left or right side.



(a) Pedestrian evacuation scenario



(b) Agent perception sectors

Fig. 1. Scenario and perception sectors

In the reinforcement learning case, all possible perceptions are converted into a string that represents the situation, in the situation-action pairs, or rules, developed by Q-Learning. For the genetic programming case, the perceptions are used as functions, representing the nodes of the decision tree, and are used in combination with the possible actions of the agents, that correspond to the terminals or leaves of the decision tree.

The action set consists of: $A = \{$*MoveLeft, MoveDiagonalLeft, MoveStraight, MoveDiagonalRight, MoveRight, Noop, StepBack*$\}$. We assume that the agents are per default oriented towards the exit.

For any of these actions, the agent turns by the given direction (e.g., $+36$ degrees for *MoveDiagonalRight*), makes an atomic step and orients itself towards the exit again.

B. Learning Evaluation

We evaluate the learning using experiments composed by a series of trials, with 100 iteration steps each, representing the number of steps that the agents have to evacuate the room before a new trial begins. With reinforcement learning, these trials are sequenced as explore and exploit trials. During explore, agents execute random actions and build their Q-Table with the situations experienced and the actions executed. During exploit the agents select only the best action for each situation, again based on the Q-Table. We used a total of 3000 explore-exploit trial pairs. At the end of the simulation, the Q-Values are used to select the best rules from the Q-Table that will form the training set used to build the decision tree with the C4.5 algorithm.

For the genetic programming case, each trial represents the use of one of the decision trees in the population. When all decision trees in the population are tested, a new generation is created. We experimented with 3000 generations.

The fitness value is assigned as the result of the agents acting in the environment, for both learning techniques. In the reinforcement learning case, this value is given on an action level, feeding the evaluation of the rule, and as a consequence developing the Q-Table. In the genetic programming case, the fitness of the decision tree is given at the end of the trial, as the sum of the fitness collected from all actions performed in that trial.

The rewards are given to the agent a for executing an action at time-step t:

$$reward(a,t) = reward_{exit}(a,t) + reward_{dist}(a,t) + penalty_{coll}(a,t)$$

Where:

- $reward_{exit}(a,t) = 1000$, if agent a has reached the exit in time t, and 0 otherwise;
- $reward_{dist}(a,t) = \beta \times [d_{t-1}(exit,a) - d_t(exit,a)]$ where $\beta = 10$ and $d_t(exit,a)$ represents the distance between the exit and agent a at time t;
- $penalty_{coll}(a,t)$ was set to 100 if a collision free actual movement had been made, to 0 if no movement happened, and to -1000 if a collision occurred.

Together, the different components of the feedback function stress goal-directed collision-free movements.

All agents perceive the environment in parallel, so their perception is based on the same overall situation at the time-step t.

C. Learning Configuration

Q-Learning was set with a linear learning rate function (from 1 to 0.2) and a discount factor of 0, which means the agents consider only the immediate best action. We also used a balanced exploration feature for selecting random action in the explore trials, in a way that all possible actions are tested equally and therefore we have a better confidence on the Q-Value. In the post-processing case, the decision tree must be tested for correspondence to the original situation-action pairs that were used to produce it. This is basically a matter of convergence which is not trivial in our scenario [18]. At the end of the simulation, the best rules – the ones with higher Q-Values – are selected as input for generating the decision tree in C4.5. Additionally, we exclude the rules that have not been tested sufficiently, according to an experience threshold.

For the Genetic Programming case, we defined the population with a size of 30 individuals. That means each generation consisted of 30 trials of 100 steps each. One trial for each individual. The initial population was generated using the *ramped half and half* method. Each individual was represented by a decision tree with a maximum depth of 5 levels. We set the probability for reproduction (copy the individual in the next generation) to 0.5, the probability of crossover to 0.4 and the probability of mutation to 0.1. Tournament selection was used to select the best fitted from 5 random individuals in the population for any of the before mentioned genetic operations.

D. A Glance on Results

The first result to be analyzed from the simulations concerns the performance of the learning technique: the number of collisions throughout the simulations. Table I shows the average number of collision: over all the 3000 explore-exploit trials for reinforcement learning (considering only data collected from exploit trials); over all the 3000 generations for the genetic programming approach (considering only the best individual in each generation). We average the number of collisions recorded by all the agents in the simulation. Reinforcement learning requires more time to develop a good set of rules to accomplish the task, while the high level representation in genetic programming is able to cope well with this simple scenario.

TABLE I
AVERAGE COLLISIONS

	RL	GP
2 agents	0.62 ±1.04	0.002 ±0.05
4 agents	1.28 ±1.18	0.18 ±0.30

Additionally, it is important to see how the agents behave in the scenario. To do that, we consider example trajectories of an exemplary agent from the 2 agents scenario, at a) 500, b)1000, c)2000 and d) 3000 explore-exploit trials in the reinforcement learning case, and generations in the genetic programming case. In the genetic programming case we consider the best decision tree, according to the fitness evaluation, in those specific generations. This is depicted in Figures 2 and 3. One can see how the decision trees perform and how they are **converging** to a good solution.

The evolution of the fitness value is also considered. The Figure 4 shows the fitness distribution – represented by the Q-Value – over all the rules for one exemplary agent in the simulation, in the RL+ case. A low number of rules with high Q-Value reflect the situations where the agent perceives the exit, and by reaching that, the reward gets increased. There is also a number of rules with a Q-Value of 0, meaning the rules have not been tested during the simulation, mainly because the agents did not experienced those situations.

The Figure 5 show the evolution of fitness for the decision trees population in GP, considering the best individuals in each generation. After a number of generations the value stabilizes, however a big variance can be verified, meaning that there is a need for improvement on the GP settings, mainly the genetic operators probabilities.

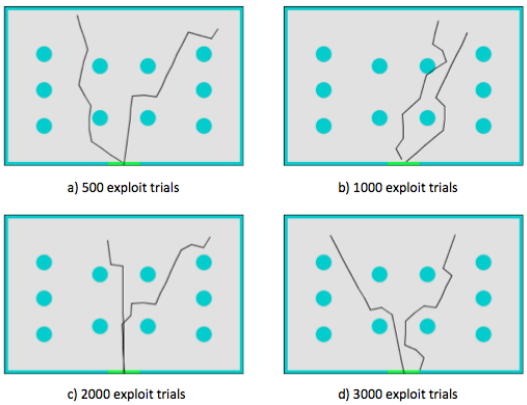


Fig. 2. Agent example trajectories for RL+ with 2 agents, over exploit trials

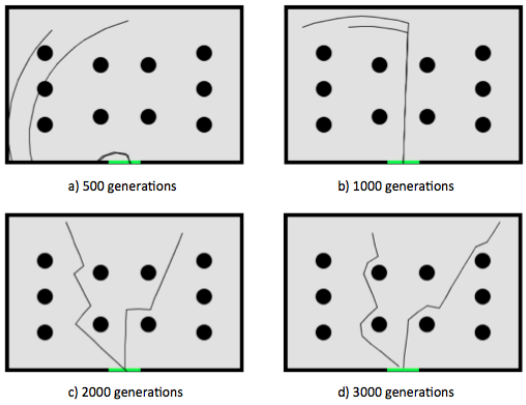
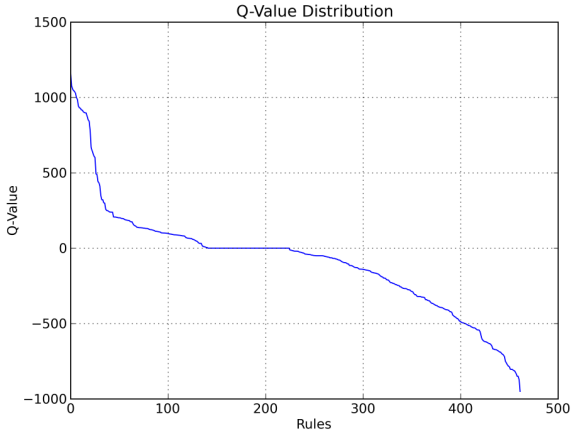


Fig. 3. Agent example trajectories for GP with 2 agents, over generations

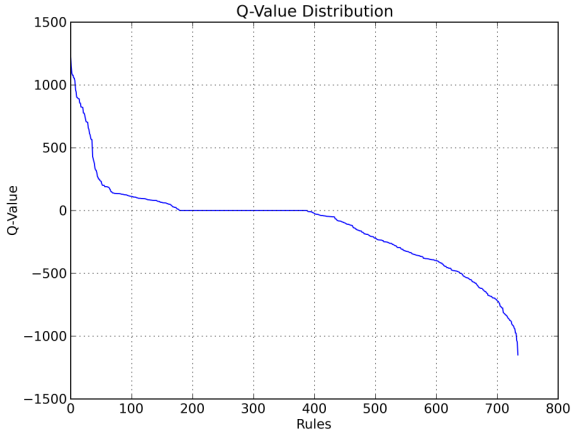
E. Evaluation of the Decision Tree

As our objective is not to develop a black box behavior controller for this simple scenario, but to generate advice for the modeler about potential behavior models for the individual agents, it is central to have a look onto the decision trees themselves. That means, we have to analyze the resulting decision trees not just according to their performance in the given scenario, but also their value as a source of inspiration for the modeler.

Clearly, the size and compactness of a decision tree is a relevant descriptor for how good a modeler can analyze its contents. In the RL+ case the size and compactness of the tree is correlated to the number and diversity of situation-action pairs that are used for its generation. This is influenced by the experience threshold, stating how often a situation action pair has to be tested. Actually, this filtered all rules with observations of the exit and just left over situation that the agents more often perceive resulting in trees that are not appropriate for all situations. The internal nodes of the trees refer to single perceptions, thus the compactness in scenarios with only a few agents is reduced as the most frequent



(a) RL 2 agents



(b) RL 4 agents

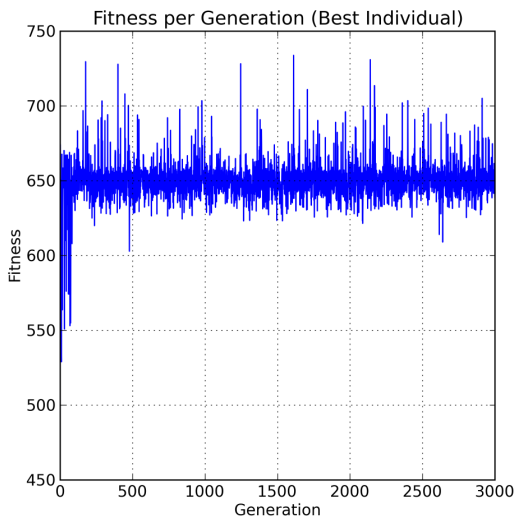
Fig. 4. Q-Value evolution

perceptions only contain at most one other obstacle. This is even the case in the scenarios with 4 agents.

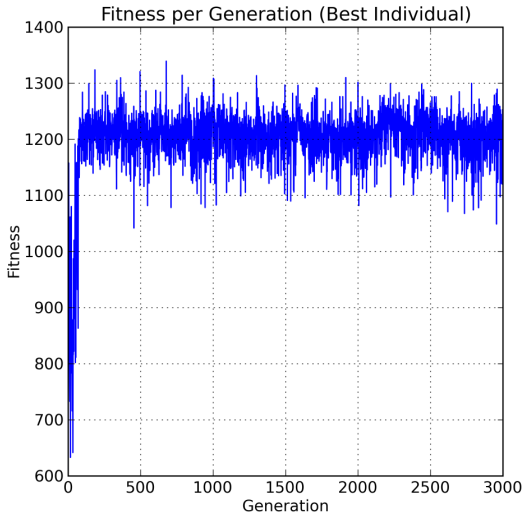
For being readable, the trees learned by the GP approach have to be pruned as well. In all trees that we analyzed, there are many branches that will never be used as they represent conditions that have been excluded before. Thus, the resulting effective size of the tree is much smaller than initially observed. Again, not all situations that an agent can perceive are handled appropriately.

Figures 6, 7 and 8 allow a direct comparison between a hand-made decision tree and examples generated from RL+ and GP.

However, size and compactness are only weak indicators of how well a learned behavior model can serve as a basis for modeling. Only the modeler can finally state whether the learned decision tree contains something “interesting” for the particular modeling problem. In Figure 6 a decision tree is depicted that was created by a human modeler who supposed that this is a good behavior model for the scenario.



(a) GP 2 agents



(b) GP 4 agents

Fig. 5. Fitness over generations for best individuals

It does not just avoid collisions but even considers the coarse direction toward the exit when deciding about the avoidance direction. However, potential problems with other simulated pedestrian, moving to positions where the agent following this tree decided to go in the next step are not regarded. An analysis of the learned trees immediately shows that they are not of the same complexity than the manual tree, but can indeed point to alternative, better solutions. Specially the RL+ tree shows that it is not sufficient to just avoid the obstacle, but the turning behavior must be larger for avoiding immediate collisions coming from movement to colliding sectors. That means an analysis of the learned behavior actually has the potential to help improving the manually developed tree.

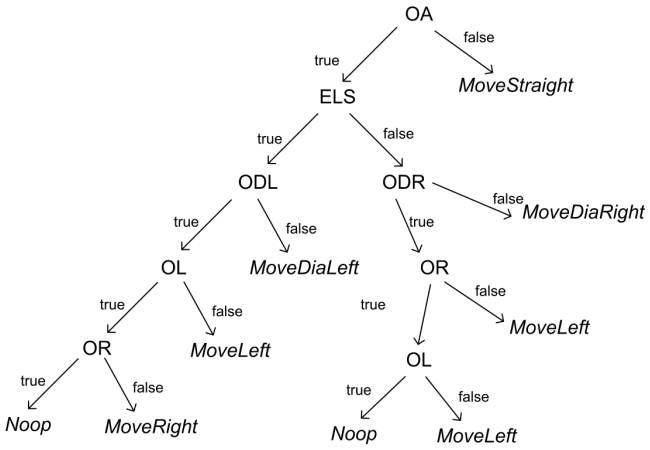
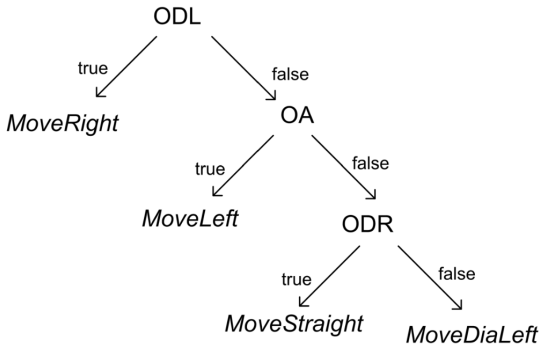
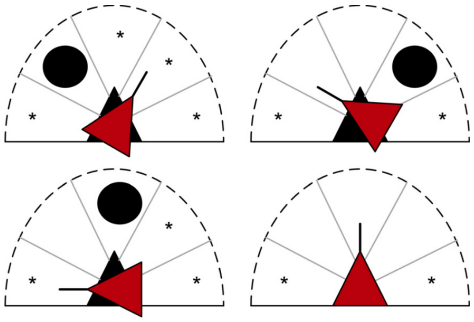


Fig. 6. Hand-made decision tree



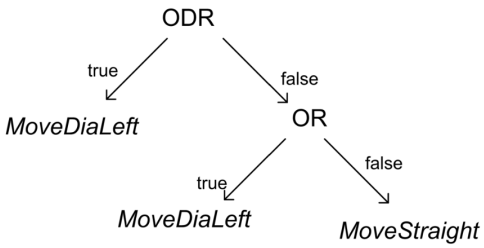
(a) RL+ tree, 4 agents scenario, experience threshold = 150



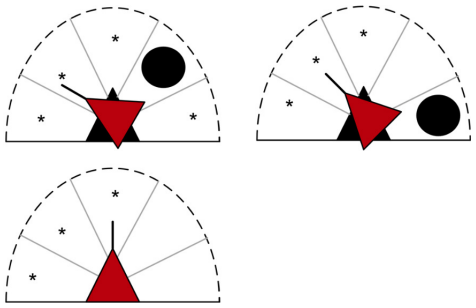
(b) Visualization of the content of the RL+ tree

Fig. 7. RL+ decision tree

Analyzing their contents, both learned trees are far from being optimal and are suspect to assumptions that cannot hold in any case. They both contain movements into sectors that are not tested whether there is an obstacle or not. This shows that learned behavior models may play an important role for detecting bugs in the environmental model, scenario or the functions describing the validity, i.e. fitness of an agent. The example GP tree only considers obstacles on the right side of the agent – a next step must be to test whether there is an inherent bias in the scenario that is responsible for that. The basis for this depicted tree was the best tree found during



(a) GP tree, 4 agents scenario



(b) Visualization of the content of the GP tree

Fig. 8. GP decision tree

3000 generations. Maybe, more generations are needed to evolve a more complete tree. Moreover, we intend to include a check after each genetic recombination operation, preventing the creation of trees with redundant checks in the same branch. This way, evolution can focus on finding relevant relations among the perceptions, considering that redundancies in the original representation are only partially helpful for that.

VI. Conclusion and Future Work

In this contribution we compared decision tree based behavior models learned from the results of a Reinforcement Learning approach or directly evolved using a Genetic Programming approach. We not just considered performance as a controller, but also had a look onto the resulting behavior models - going a step further to our original goal of modeling support by generating suggestions for a modeler when he is getting stuck with developing a multiagent simulation model. Clearly, we are just at the beginning of our endeavor trying to find out the appropriate learning techniques for our goal in general and for simulation problems with particular structures.

The next steps are related to further improvements of the learning algorithms. Whereas for RL+ we already did extensive tests concerning effects of different configurations and alternative setups, this has still to be done for the GP approach. A systematic analysis of the influence of the many different parameter configurations and scenario setups should be conducted, deepening the comparison of the two learning techniques. We want to avoid integrating components into the objective function that intentionally influence the shape of the tree. This would involve meta-level considerations, making the development of the fitness function even more complex and would confuse a modeler

as performance and modeling concerns would not be separated.

An important question in our setup concerns the robustness of the learning approach with respect to small changes in the objective function that contains the characterization of valid behavior. It is clear that developing this function is the most difficult task when using a learning-based modeling approach. Therefore it is essential to know how sensitive the learning algorithm is to slight changes in this objective function.

Additionally, we want to test variations of the learning techniques that focus on the modeling support goal. Having in mind the design strategy, starting from the definition of sensors and actuators, and going to a decision tree behavior representation, the learning techniques differ on how they evolve such a model. RL essentially works on developing a set of rules, evaluated individually, that need to be translated to a tree representation. In this translation process we lose information about which were the original rules and what was their assessment. In case the modeler wants to further develop this model by modifying branches of the tree, it becomes difficult to integrate this knowledge back in the learning process. On the other hand, GP evolves directly a tree representation. There is no need for converting information. The human modeler can alter parts of the program and use it back in the simulation for validation. However, the fitness assessment is done in a program level. There is no information about the influence of a particular branch of the tree on the final value. We intend to modify our GP approach to include individual action fitness evaluation – on a similar level as it's done with RL+ – in order to develop an editable tree program that can be interpreted not only on the global performance level. This would represent an important step towards the development of a systematic way to analyze the learned tree and identify elements that should be integrated into the final model. How to present a tree that the modeler is able to evaluate, which are the problems in the manual behavior model and which elements of the learned tree are responsible that the learned tree does not face these problems. Up to now we were mostly focusing on finding the most appropriate learning techniques, supporting the modeler in using the learned results cannot be neglected.

Finally, we will peruse further experiments in more complex scenarios. Complexity, at first, can be increased by: having more agents in the simulation; broadening the perception range of the agents, to include more perception variables; adding more elements to the objective/fitness evaluation; and also by including heterogeneous agents, that are required to perform different roles and are subject to different objective/fitness functions.

GP seems to be more appropriate to the space exploration problem, yet additional processing may be required depending on how the complexity increase will impact the size of the decision tree. A possible solution would be to split the search space, providing different program trees for different sub-problems. However, this division depends on the problem domain and additional steps would have to be included in the design strategy.

REFERENCES

[1] J. M. Epstein and R. L. Axtrell, *Growing Artificial Societies: Social Science from the Bottom Up*. MIT Press, 1996.

[2] R. S. Quinlan, *C4.5: programs for machine learning*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1993.

[3] S. B. Kotsiantis, "Supervised machine learning : A review of classification techniques," *Informatica*, vol. 31, pp. 249–268, 2007.

[4] F. Klügl, "Multiagent simulation model design strategies," in *MAS&S Workshop at MALLOW 2009, Turin, Italy, Sept. 2009*, ser. CEUR Workshop Proceedings, vol. 494. CEUR-WS.org, 2009.

[5] G. Weiß, "Adaptation and learning in multi-agent systems: Some remarks and a bibliography," in *IJCAI '95: Proceedings of the Workshop on Adaption and Learning in Multi-Agent Systems*. London, UK: Springer-Verlag, 1996, pp. 1–21.

[6] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[7] A. E. Eiben and J. E. Smith, *Introduction to Evolutionary Computing*, ser. Natural Computing Series. Springer, 2003.

[8] S. Mahadevan and J. Connell, "Automatic programming of behavior-based robots using reinforcement learning," *Artificial Intelligence*, vol. 55, no. 2-3, pp. 311 – 365, 1992.

[9] M. R. Lee and E.-K. Kang, "Learning enabled cooperative agent behavior in an evolutionary and competitive environment," *Neural Computing & Applications*, vol. 15, pp. 124–135, 2006.

[10] T. Francisco and G. M. Jorge dos Reis, "Evolving predator and prey behaviours with co-evolution using genetic programming and decision trees," in *Proceedings of the 2008 GECCO conference companion on Genetic and evolutionary computation*, ser. GECCO '08. New York, NY, USA: ACM, 2008, pp. 1893–1900.

[11] L. Hanna and J. Cagan, "Evolutionary multi-agent systems: An adaptive and dynamic approach to optimization," *Journal of Mechanical Design*, vol. 131, no. 1, p. 011010, 2009.

[12] W. H. Hsu and S. M. Gustafson, "Genetic programming and multi-agent layered learning by reinforcements," in *In Genetic and Evolutionary Computation Conference*. Morgan Kaufmann, 2002, pp. 764–771.

[13] J. Denzinger and M. Kordt, "Evolutionary online learning of cooperative behavior with situation-action pairs," in *MultiAgent Systems, 2000. Proceedings. Fourth International Conference on*, 2000, pp. 103 –110.

[14] J. Denzinger and S. Ennis, "Improving evolutionary learning of cooperative behavior by including accountability of strategy components," in *Multiagent System Technologies*, ser. Lecture Notes in Computer Science, M. Schillo, M. Klusch, J. Müller, and H. Tianfield, Eds. Springer Berlin / Heidelberg, 2003, vol. 2831, pp. 205–216.

[15] J. Denzinger and A. Schur, "On customizing evolutionary learning of agent behavior," in *Advances in Artificial Intelligence*, ser. Lecture Notes in Computer Science, A. Tawfik and S. Goodwin, Eds. Springer Berlin / Heidelberg, 2004, vol. 3060, pp. 146–160.

[16] J. Denzinger and J. Hamdan, "Improving modeling of other agents using tentative stereotypes and compactification of observations," in *Intelligent Agent Technology, 2004. (IAT 2004). Proceedings. IEEE/WIC/ACM International Conference on*, sept. 2004, pp. 106 – 112.

[17] R. Junges and F. Klügl, "Evaluation of techniques for a learning-driven modeling methodology in multiagent simulation," in *MATES*, 2010, pp. 185–196.

[18] ——, "Learning convergence and agent behavior interpretation for designing agent-based simulations," in *Proceedings of the Eighth European Workshop on Multi-Agent Systems EUMAS 2010*, 2010.

[19] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, no. 3, pp. 279–292, 1992.

[20] J. R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection (Complex Adaptive Systems)*, 1st ed. The MIT Press, Dec. 1992.

[21] R. Junges and F. Klügl, "Evolution for modeling - a genetic programming framework for sesam," in *Proceedings of ECoMASS@GECCO 2011. Evolutionary computation and multi-agent systems and simulation (ECoMASS)*, 2011.