AnthillRL: Multi-agent Reinforcement Learning

Anibal Sólon Heinsfeld, Felipe Meneguzzi

[1]School of Computer Science (FACIN)
Pontifical Catholic University of Rio Grande do Sul (PUCRS)
Porto Alegre – RS – Brazil

anibal.heinsfeld@acad.pucrs.br, felipe.meneguzzi@pucrs.br

Abstract. *Ants are essentially social insects, and their organizational dependency reflects directly on their survival. Due to their social nature, ant society provides a rich model for analysing properties of multiagent systems such as collaboration and effectiveness of collective action. Modelling natural behaviour of ants can help understand their actions, as well as studying better forms of cooperation and competition for other multiagent models. In this paper, we model an ant society within a stochastic environment in which ant behaviour is generated using reinforcement learning to generate paths towards their goal. We test this model using a variable number of agents and learning parameters and report on the results.*

1. Introduction

Ants societies are described as a superorganism because of their social organization and cooperation [Wilson and Hölldobler 2005]. Besides the mutual care of their brood, there is a division of labour based in the ant's reproduction capacity and behavioural groups. According to this morphological and behavioural differences, the ants are classified in caste such as Queens, Workers and Soldiers. To achieve this status as a superorganism, the communication of ants is a differential point and occurs via secreted chemicals, sounds and touch [Wilson and Hölldobler 2005]. Pheromones are chemical substances used for purpose of communication and they are secreted from the body of an individual to affect the decision making of other individuals that detect it. Different types of pheromones are produced, such as to attract mates, to signal danger to the colony or to give directions about a location.

Given that ants are a fundamentally a social species, its modeling in a multi-agent platform is an interesting way to verify the collective behaviour of this decentralized and self-organized systems and to apply its results on optimization heuristics. Many algorithms were created inspired on social behaviour, as of insects: ants [Shtovba 2005] and honey bees [Sonmez 2011], and even from other phylum, as bats [Yang and He 2013] and gray wolves [Mirjalili et al. 2014].

In this paper, we have two main objectives. First, we want to model an anthill in order to better understand behaviours of ants collaborating as a group. Using JaCaMo architecture, each ant is modelled as an agent in order to act collaboratively to achieve the global goal: find the shortest path to environment resources. Second, we aim to test Reinforcement Learning methods with multi-agents system in order to evaluate how agents distributivity affects performance in our scenario. Using Markov Decision Processes to

model environment states and actions that agents can take, we aim to achieve randomness of real-world environment, and with Reinforcement Learning, we want to simulate pheromones that reinforce the shortest path to a resource.

This paper is organized as follows. Section 2 reviews the background on MDP and Reinforcement Learning. In Section 6 we address related work using Multi-agents and Reinforcement Learning. Section 3 and Section 4 describe our method to achieve the proposed objectives and the tests methodology. In the Section 5 we present our tests results and discussion. Finally, in Section 7, we conclude with final considerations and point the direction for future work.

2. Background

2.1. Markov Decision Process (MDPs)

Markov decision processes provide a mathematical framework for modeling decision making in situations where outcomes are partly random and partly under the control of a decision maker, thus, it is a stochastic process. Using this control process, the decision maker is in a known state and can choose an action available for the current state. The action has a probability to occur and lead to the subsequent state, so the process brings about a random state to the decision maker. The probability to reach the subsequent states depends only on the current state and the action taken, satisfying the Markov Property. The previous taken actions do not interfere in the probability distributions of the current possible actions. For example, given an exploratory robot agent that must recognize its territory and control its battery level, the actions available to this robot are: search (as its mission), recharge its battery and wait for the recharging process. In a very simple agent model, the agent just has two states: high level of battery and low level of battery. In Figure 1, the stochastic automaton of an an exploratory robot agent demonstrates how the taken actions lead to states with a certain probability. The automaton has probabilities for an action to happens and a reward. At the *high* state, the agent can take the *wait* action, keeping the agent in the same state with a probability of 100% (denoted as 1) with reward R^{wait}. Otherwise, the action search can lead to both described states with a probability of α (or its complementary value 1-α) with reward R^{search}. It is important to note that, when in *low* state, the search action can lead to a negative reward (a penalty). This penalty for an agent is important in order to make it avoid this action.

In order to achieve better results, agent must find a policy which specifies which action to take in each state, so as to maximize the sum of values from a specific reward function. The problem is to specify a function to find a policy with maximum expected reward, called the optimal policy, based on a reward function. In this work, with regard to find the optimal policy, or the shortest path to a resource, we use Reinforcement Learning methods.

2.2. Reinforcement Learning

Reinforcement learning is defined as learning what to do in order to maximize some notion of cumulative reward [Sutton and Barto 1998]. In the context of MDP modelling, the intelligent agent does not know what actions it must take in its environment. Instead it must explore the environment by exploring actions and states and measuring the resulting reward. The chosen action has two consequences: first, it affects the reward of the decision and second, it changes the subsequent possible actions and its rewards.

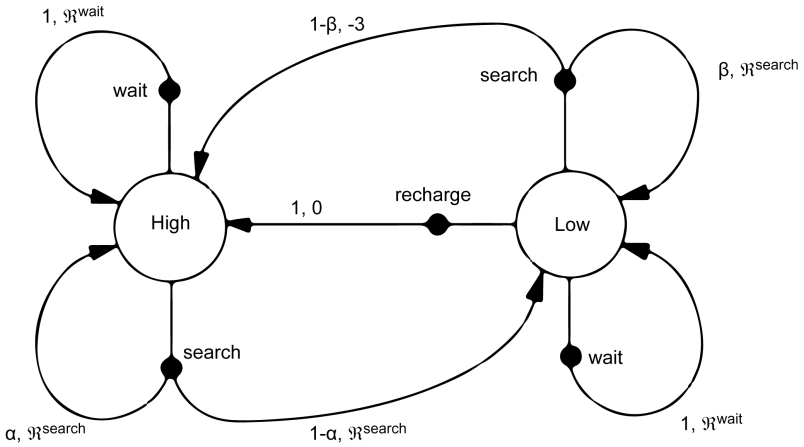

Figure 1. The Markov model of the robot states and actions

The feature of Reinforcement Learning of trial-and-error search makes it a distinguishable area of Machine Learning: it refers to learning from direct interaction with an environment. The learning agent receive just a reinforcement signal from the environment instead of a precise error signal measuring the discrepancy between the realized and a known expected performance. In Reinforcement Learning, the agent must explore the environment to learn. But also, aiming to converge the learned rewards towards an optimal solution, the agent must exploit its current knowledge about the environment, probably leading it to the search space closer to the solution of the problem. The agent should, for example, execute the same actions that in the past produce a good reward, assuming that reproducing the same behaviour will lead to the same reward. Otherwise, through the exploration, the agent can find a better reward than this known reward. So, it is important to define a proper weight between this two strategies, aiming the maximization of the cumulative reward.

There are several algorithms for Reinforcement Learning [Kaelbling et al. 1996]. The algorithm used in this work is the Q-learning. Q-learning is a form of model-free reinforcement learning [Sutton and Barto 1998], in which the agent does not need an internal model of environment to work with it. It works by learning an action-value function that provide the expected utility of taking a given action in a given state and its core is a simple value iteration update.

The remarkable characteristic of Q-learning is that it can determine the immediate rewards and delayed rewards, depending on its parametrisation. The algorithm creates a policy, that is a rule the agent follows in selecting actions, given the state it is in. When such an action-value function is learned, the optimal policy can be constructed by simply selecting the action with the highest value in each state. Q-learning is an on-line reinforcement technique, which means that the policy of the agent is improved on the fly.

The update rule for Q of the state-action combination calculates an estimate value given a state and a chosen action by the agent with the formulae:

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha \times (R(s_t, a_t) + \gamma \times \Delta Q(s_t, a_t 0)) \quad (1)$$

$$\Delta Q(s_t, a_t) = maxQ(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \quad (2)$$

The learning rate (represented by α) defines how much the newly acquired Q-value influences the new Q-value. The learning rate can take values between zero and one. A rate close to zero makes the agent ignores the new information, so it does not learn anything. Otherwise, when the rate tends to one, the agent only considers the new Q-value and ignores the old one.

The discount factor (represented by γ) makes rewards earned earlier more valuable than those received later. It assumes values between zero and one.

Analysing Figure 2, it is possible to visually understand how information influence the algorithm. The assumed action (blue) will have its Q value changed, gathering the greater Q value of the possible actions (green) that the selected action will lead to.

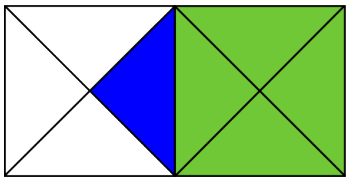

Figure 2. Taking an action (green) will trigger the Q(State, Action) calculus, using the neighbour's (blue) Q-value.

3. Proposed Approach

The proposed approach revolves around the search of an optimal action-selection policy for any given Markov Decision Process. Using a Multi-Agent System model, the goal is the convergence to the shorten path to a given destination. The environment surrounds the anthill world and, once the ant (the agent) is outside of the anthill, it must search the proposed grid for a source of sugar (modelled as a donut). The rewards of Q-Learning state-action pairs is modelled as pheromones. This system uses Multi-Agent approach in order to optimise the Reinforcement Learning algorithm by computing rewards concurrently and distributing their values across agents in the form of pheromones. Thus, it models correctly the ants behaviour by its perceiving, acting and sharing information through pheromones. The agents follow the Belief-Desire-Intention model, separating the activity of selecting a plan provided by its plan library from the execution of currently active plans.

We model the Multi-Agent System using JaCaMo [Boissier et al. 2013]. JaCaMo's joins three Multi-Agent technologies aiming the integration of agents, environments and organizations. These technologies are, respectively: Jason [Bordini et al. 2007], a agent-oriented programming language and platform that extends AgentSpeak; Cartago [Ricci et al. 2011], a platform that provides an general-purpose shared environment to agent interactions; and Moise [Hübner et al. 2010], an organisational model that provides structural, functional and deontic specifications. This integra-

tion aids on rewards synchronization between agents, since Cartago shares equally the environment information to Jason-programmed agents.

The movimentation of the agents across the environment was developed in the AgentSpeak-like language, Jason, with a set of conditionals which gives the direction of what plans must be executed for the current state. The decision process modeled over the Markov Decision Process for the learning and reasoning about the next actions and states is developed in Java, the language and environment that wraps the Jason, Cartago and Moise.

The map described in the Figure 3 was used for the tests. The modeled pheromones are the trail and danger pheromones, in order to stimulate ants follow or avoid a certain path, respectively. Each triangle, that represents the possible actions in a location of the grid, has its own measure of pheromone. So the ant can reason about which direction is the most valuable.

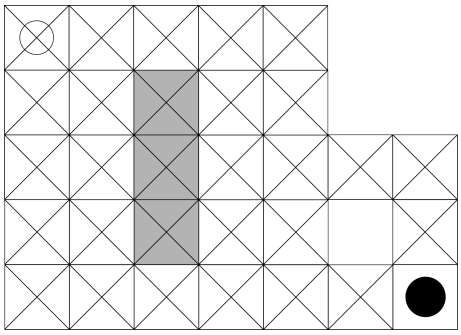

Figure 3. The map used for the tests. The white circle is the initial state, the black circle is the goal and the gray states are dangerous areas.

At this simulation, an ant can choose the actions UP, DOWN, LEFT, RIGHT when it is possible. In the borders of the scenario, where there is no surrounding state, it is impossible to assume an action aiming regions outside our scenario.

The agents use an ϵ-Greedy policy. For a given ϵ, between 0 and 1, the agent assume a greedy position choosing the most valuable utilities of the actions or choose a random action, based in a random distribuition. In the random mode, the action is described as exploratory, and when the next action is choosen based on its valuation, this actions is called exploitatory. In this system, ϵ assumes an initial value and, according to the current episode, the ϵ tends to an exploitatory value. It ensures that the ants take benefits of the learned knowledge. The ϵ value of the episode is described with the following formula:

$$\epsilon_{Episode} = \epsilon_0 + ((1 - \epsilon_0) * Episode/FinalEpisode) \tag{3}$$

The algorithm below describes how the action-selection through ϵ-Greedy works:

$\epsilon \leftarrow \epsilon_0 + ((1 - \epsilon_0) * Episode/FinalEpisode)$

$R \leftarrow Random[0, 1]$

if $\epsilon \geq$R **then**

Action ← Most-Valuable Action

else

Action ← Random Action

end if

Putting it all together, with the ϵ-Greedy policy and Q-learning in our MDP-modeled environment, the algorithm executed by the agents for each episode is:

CurrentState ← State_0

while CurrentState is not the Goal State **do**

Action ← Action chosen by ϵ-Greedy policy

Compute Q(CurrentState, Action)

CurrentState ← State

end while

4. Experimental Methodology

To measure the performance of the whole Multi-Agent System, the chosen metric is the number of episodes before convergence. An episode in this work is described as the traversing of the agent from initial position to the goal. So, an episode ending when some terminal state is reached. Convergence occurs when in the next 5 episodes the values remain unchanged. For a single agent, the episode of convergence is the episode that the agent reaches. The considered episode of convergence for multiple agents is the superior episode that one of the agents reaches. This metric supplies the lack of determinism on multi-agents systems because each agent can reach different episodes of convergence and the learning process is only finished when all agents reach the convergence.

In order to evaluate different scenarios for the proposed set of algorithms and architecture, the following parameters are changed across the tests:

- Number of agents: A single agent or multiple agents;
- Maximum number of episodes: The stop criteria for the learning algorithm. This metric influences the ϵ for each episode.
- Initial ϵ: The value of the ϵ variable at the ϵ-Greedy policy;

The test was executed five times for each parametrization, because of stochastic of the system. The multiple results are summarized through a mean operation. The α, the learning parameter of Q-Learning formula, is set to 0.8 in order to learn new information, but also consider the old one. The γ, the discount factor, is set to 1.0. This value was chosen to achieve additive rewards towards long-term rewards for a faster convergence.

5. Experimental Analysis

The experiments follow the variation of parametrization: the number of agents is varied in one, three and five agents; the maximum number of episodes that the run can reach is

varied in 30 and 100; and the initial ϵ of Q-learning algorithm is varied in 0.4, 0.6 and 0.8. The gathered results is shown in Table 5 and Table 5.

Number of Agents	Max Episode	Start Epsilon	Episode of Convergence
1	30	0.8	23
3	30	0.8	14
5	30	0.8	9
1	30	0.6	24
3	30	0.6	14
5	30	0.6	8
1	30	0.4	26
3	30	0.4	16
5	30	0.4	9

Table 1. Results of parameter permutation: Max Episode equals to 30

Number of Agents	Max Episode	Start Epsilon	Episode of Convergence
1	100	0.8	50
3	100	0.8	38
5	100	0.8	26
1	100	0.6	53
3	100	0.6	35
5	100	0.6	29
1	100	0.4	50
3	100	0.4	39
5	100	0.4	31

Table 2. Results of parameter permutation: Max Episode equals to 100

The number of agents influences convergence directly, because the knowledge is distributed among the agents and they execute multiple valuation of the actions at the same time. When there is just one agent, it takes more episodes to reach the converged policy.

It is clear that the decrease of the maximum number of episodes converges faster because of the influence of this parametrization in the ϵ value at each episode. Since the numerical difference between the current episode and the maximum episode is lesser, the tendency of the ϵ to one (describing 100% of chance to use the Greedy policy, in the exploitatory mode).

As showed in the results, the value of the initial Epsilon do not affect directly the convergence. This indifference probably occurs because of the high influence of the variation of ϵ during the execution.

The convergence of the map is showed at the Figure 4. The green marks indicate the possible actions of the position. It is possible to see that the most-valuable actions (marked as blue) leads to the food.

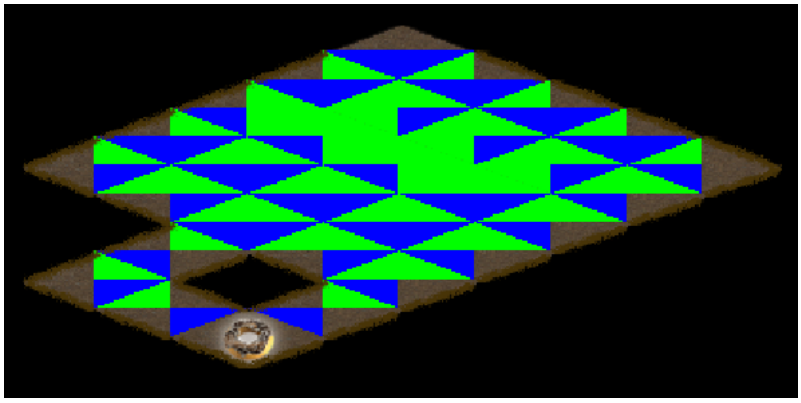

Figure 4. The policy generated by Q-learning.

6. Related Work

Several Multi-agent Systems use Reinforcement Learning with independent or cooperative approaches.

Tan [Tan 1993] uses both in a scenario with two sets of rivals: the hunter and the prey. He use Q-Learning to estimate the reward of each pair of state and action. He demonstrate that reinforcement learning agents can learn cooperative behaviour in a simulated social environment and, as this work, assume three types of interactions: a online communication of perceptions, an episodic communication and the sharing of the learned knowledge.

In the work of Subramanian et al. [Subramanian et al. 1998], the idea of mimicry of ants is used to find better routes in a dynamic computational network. They propose two algorithms that are resilient against random corruption of the current path: the first algorithm is a shortest path algorithm, while in their second approach, they introduce a concept of uniform ant. The probability of the uniform ant choose a route is equal to each other possible routes, so it does not suffer with route oscillation problems, which is a common problem in the first algorithm.

The work of Zhang et al. [Zhang et al. 2012] approaches the application of Reinforcement Learning to optimize the path-finding algorithm for robots. It is based on Markov Decision Process and use Q-Learning to learn the optimal policy through trial-and-error, an intrinsic property of Reinforcement Learning. They also explore the point of non-deterministic rewards and actions, with an interesting formula that consider the number of visits of the agent to that state as an learning rate.

In the work of Gambardella et al.[Gambardella et al. 1995], they propose a distributed algorithm based on Q-learning for combinatorial optimization using the metaphor of ant colonies, to solve the solution of symmetric and asymmetric instances of the traveling salesman problem, a combinatorial optimization problem. The symmetry is defined by the distance between two cities: if the distance to go from city A to city B is equal to go from city B to city A, it is assumed to be symmetric. Otherwise, it is called asymmetric. The remarkable modification made from Q-learning is the use of an heuristic value.

Finally, the survey of Busoniu et al. [Busoniu et al. 2008] brings about the use of Reinforcement Learning in a Multi-Agent System. It state that are two key points to the

design decision of the Multi-Agent Reinforcement Learning: the stability of the agents' learning dynamics and the adaption to the changing behaviour of the other agents. The survey introduces the single-agent and Multi-Agent approach for Reinforcement Learning using the Markov Decision Process and the Stochastic Game, an generalization of Markov Decision Process that in a environment for one or more agents it assumes probabilistic transitions. Surrounding this previous topic, it also cites the Nash Equilibrium, an important concept in Multi-Agent System that is an situation where no agent can benefit by changing its strategy as long as all other agents keep their strategies constant. In the section of benefits and challenges of the survey, it is important to note that the use of multi-agent can bring an speed-up to the algorithm, because of the parallel computation when the agents exploit the decentralized structure of the task. But also sharing the learning information among the agents can help to learn fast and better a converged solution: the authors argue about methods of learning from the knowledge of other agents. They cite the possibility of modeling this interaction as an skilled agents that may serve as teachers for the learner or the learner may watch and imitate the skilled agents to reason about its observation and learn. In the other part of the section, they bring about the importance of a good policy to trade-off between the exploratory and exploitatory strategy.

7. Conclusions and Future Work

This work demonstrates that intelligent agents based on reinforcement learning can cooperate to reach a good result on a multi-agent environment. Also the work involves a variety of subjects, from theoretical to technical, demonstration the application of complex learning on BDI-based multi-agents, as the simulation of a insect that uses a natural reinforcement learning methodology.

It is important to note that with a larger number of agents, the learner converges faster, since the reinforcement learning approach applied in this work distributes the knowledge among the agents and the agents can explore quickly the proposed map. The variable ϵ-Greedy action-selection policy gives a faster convergence, since the agents tended to follow the most-valuable path instead of explore even more the map.

For future work, we aim to use other reinforcement learning algorithms, as SARSA, an on-line Q-learning method. Comparison with other algorithms can bring different performance results and possible integrations and optimizations. The simulation of partially observable Markov Decision Process (POMDP), a generalization of Markov Decision Process that assumes probabilistic distributions of the underlying state, would be interesting, since it model the real world more precisely. The design of dynamic characteristics, such as the resource location and quantity, as well the competition between sets of agents to reach the resource, are also good cases to explore with reinforcement learning, since the applied algorithm and the agents must deal with changes in the environment.

References

Boissier, O., Bordini, R. H., Hübner, J. F., Ricci, A., and Santi, A. (2013). Multi-agent oriented programming with jacamo. *Sci. Comput. Program.*, 78(6):747–761.

Bordini, R. H., Hübner, J. F., and Wooldridge, M. (2007). *Programming Multi-Agent Systems in AgentSpeak using Jason*.

Busoniu, L., Babuska, R., and De Schutter, B. (2008). A comprehensive survey of multiagent reinforcement learning. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 38(2):156–172.

Gambardella, L. M., Dorigo, M., and Bruxelles, U. L. D. (1995). Ant-q: A reinforcement learning approach to the traveling salesman problem. pages 252–260. Morgan Kaufmann.

Hübner, J., Boissier, O., Kitio, R., and Ricci, A. (2010). Instrumenting multi-agent organisations with organisational artifacts and agents. *Autonomous Agents and Multi-Agent Systems*, 20(3):369–400.

Kaelbling, L. P., Littman, M. L., and Moore, A. W. (1996). Reinforcement learning: A survey. *J. Artif. Int. Res.*, 4(1):237–285.

Mirjalili, S., Mirjalili, S. M., and Lewis, A. (2014). Grey wolf optimizer. *Adv. Eng. Softw.*, 69:46–61.

Ricci, A., Piunti, M., and Viroli, M. (2011). Environment programming in multi-agent systems: an artifact-based perspective. *Autonomous Agents and Multi-Agent Systems*, 23(2):158–192.

Russell, S. and Norvig, P. (2010). *Artificial Intelligence: A Modern Approach*. Prentice Hall series in artificial intelligence. Prentice Hall.

Shtovba, S. (2005). Ant algorithms: Theory and applications. *Programming and Computer Software*, 31(4):167–178.

Sonmez, M. (2011). Artificial bee colony algorithm for optimization of truss structures. *Applied Soft Computing*, 11(2):2406 – 2418. The Impact of Soft Computing for the Progress of Artificial Intelligence.

Sorici, A., Picard, G., Boissier, O., Santi, A., and Hübner, J. F. (2012). Multi-agent oriented reorganisation within the jacamo infrastructure. In *Proceedings of The Third International Workshop on Iinfraestructures and tools for multiagent systems: ITMAS*, pages 135–148.

Subramanian, D., Druschel, P., and Chen, J. (1998). Ants and reinforcement learning: A case study in routing in dynamic networks. In *In IJCAI (2*, pages 832–838. Morgan Kaufmann.

Sutton, R. S. and Barto, A. G. (1998). *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1st edition.

Tan, M. (1993). *Multi-Agent Reinforcement Learning Independent vs Cooperative Agents*. PhD thesis, GTE Laboratories Incorporated.

Wilson, E. O. and Hölldobler, B. (2005). Eusociality: origin and consequences. *Proceedings of the National Academy of Sciences of the United States of America*, 102(38):13367–13371.

Yang, X.-S. and He, X. (2013). Bat algorithm: Literature review and applications. *Int. J. Bio-Inspired Comput.*, 5(3):141–149.

Zhang, Q., Li, M., Wang, X., and Zhang, Y. (2012). Reinforcement learning in robot path optimization. *Journal of Software*, 7(3):657–662.