Multi-Attribute Decision Making in a Complex Multiagent Environment using Reinforcement Learning with Selective Perception

Sébastien Paquet, Nicolas Bernier, and Brahim Chaib-draa

Laval University, Canada
[spaquet,bernier,chaib]@damas.ift.ulaval.ca

Abstract. Choosing between multiple alternative tasks is a hard problem for agents evolving in an uncertain real-time multiagent environment. An example of such environment is the RoboCupRescue simulation, where at each step an agent has to choose between a number of tasks. To do that, we have used a reinforcement learning technique where an agent learns the expected reward it should obtain if it chooses a particular task. Since all possible tasks can be described by a lot of attributes, we have used a selective perception technique to enable agents to narrow down the description of each task.

1 Introduction

Analyzing many possible tasks and choosing the best one is obviously a hard job for an agent, particularly in a dynamic and uncertain environment. In this case, an agent has to make a choice with only a partial view of the situation and it has to make it quickly, because the environment is constantly changing. To do that, the solution proposed here is to use a reinforcement learning algorithm in which the agent learns the expected reward it should get for every possible task it could be faced with.

In our test environment, the number of possible states is very large, since a task is described by many attributes which can take many possible values. To find the right level of precision for the state description, we have adapted the selective perception technique, developed by McCallum [1], to enable the agent to learn by itself which is the level of precision it needs in all possible situations.

Our test environment is the RoboCupRescue simulation in which the goal is to build a simulator of rescue teams acting in large urban disasters [2]. More precisely, this project takes the form of an annual competition in which participants are designing rescue agents trying to minimize damages, caused by a big earthquake, such as civilians buried, buildings on fire and blocked roads. In the simulation, participants have approximately 30 to 40 agents of six different types to manage: *FireBrigade*, *PoliceForce*, *AmbulanceTeam*, *FireStation*, *PoliceOffice* and *AmbulanceCenter*.

In this article, we focus only on *FireBrigade* and *FireStation* agents. The task of those agents is to extinguish fires. Furthermore, in order to be effective, they have to coordinate themselves on the same buildings on fire, because more than one agent is often needed to extinguish a building on fire.

A *FireBrigade* agent is faced with the problem of choosing a fire to extinguish between a list of buildings on fire. Since there could be a lot of fires, agents do not consider all fires at once. They separately choose which fire zone to extinguish and which specific building in the chosen fire zone to extinguish. Fire zones are simply groupings of near buildings on fire.

Since the *FireStation* agent can receive more messages, it normally has a better knowledge of the global situation compared to the *FireBrigade* agents. Therefore, it is the responsibility of the *FireStation* agent to allocate fire zones to *FireBrigade* agents. After they have received their fire zone, *FireBrigade* agents have the possibility to learn how to coordinate their efforts to extinguish the more important fires in a given fire zone.

To reduce the learning task complexity, each agent considers only one possible task at a time, i.e. agents are only considering one building at a time when choosing a building on fire. This enables us to have a fix state description. When an agent wants to choose a building on fire to extinguish, it goes through the list of all buildings on fire, it evaluates them independently by calculating the utility and the expected reward of each. The utility is an estimation of the importance to extinguish a given building, calculated by considering the number of civilians and buildings on danger. The expected reward, learned with the algorithm presented in section 2, can be seen as an estimate of the capacity to extinguish a given fire.

2 Selective Perception

To learn the expected reward of choosing one fire, we have used a selective perception technique [1], because the description of our states is too big. With this technique, the agent learns by itself to reduce the number of possible states. To do that, it uses a tree structure similar to a decision tree. At the beginning all states are considered to be the same, so there is only the root of the tree. After some experiences, the agent tests if it would be interesting to divide the different states, represented as the leaves of the tree. An advantage of this algorithm is that it can drastically reduce the number of states distinguished by the agent.

At each time step t, the agent records its experience captured as an "instance" that contains the action it made the step before (a_{t-1}), the observation it perceives (o_t) and the reward it obtains (r_t). Each instance also has a link to the preceding instance and the next one, thus making a chain of instances. In our case, we have one chain for each building that an agent chooses to extinguish. A chain contains all instances from the time an agent chooses to extinguish a building until it changes to another building.

Therefore, during the simulation, the agents record many instances organized in many instance chains and they keep all those instances until the end of the simulation. There is no learning taking place during the simulation, because the agents are evolving in a real-time environment and they cannot afford to take time to learn during the simulation. After the simulation, the agents are regrouping all their experiences together, the tree is updated with all those new instances and the resulting tree is returned to each agent.

To learn how to classify the states, we use a tree structure similar to a decision tree. The tree divides the instances in clusters depending on their expected reward. The objective here is to regroup all instances having similar expected rewards, because it means that the agent does not have to distinguish between them, since it should act the same way in all those situations.

The algorithm presented here is an instance-based algorithm in which a tree is used to store all instances which are kept in the leaves of the tree. The other nodes of the tree, called center nodes, are used to divide the instances with a test on a specific attribute. To find the leaf to which an instance belongs, we simply start at the root of the tree and head down the tree choosing at each center node the branch indicated by the result of the test on the instance attribute value. In addition, each leaf of the tree also contains a Q-value indicating the expected reward if a fire that belongs to this leaf is chosen. In our approach, a leaf of the tree l is considered to be a state for the reinforcement learning algorithm.

After a simulation, all agents put their new experiences together. This set of new experiences is then used to update the tree. Firstly, all the new experiences are added to their respective leaf nodes. Afterwards, the Q-values of each leaf node are updated to take into consideration the new instances which were just added. The updates are done with the following equation:

$$Q(l) \leftarrow R(l) + \gamma \sum_{l'} Pr(l'|l) Q(l') \tag{1}$$

where $R(l)$ is the estimated immediate reward if a fire that belongs to the leaf l is chosen, $Pr(l'|l)$ is the estimated probability that the next instance would be stored in leaf l' given that the current instance is stored in leaf l. Those values are calculated directly from the recorded instances:

$$R(l) = \frac{\sum\limits_{i_t \in I_l} r_{t+1}}{|I_l|},\ Pr(l'|l) = \frac{|\{\forall i_t \in I_l | L(i_{t+1}) = l'\}|}{|I_l|} \tag{2}$$

where $L(i)$ is a function returning the leaf l of an instance i, I_l represents the set of all instances stored in leaf l, $|I_l|$ is the number of instances in leaf l and r_{t+1} is the reward obtained after the instance i_t was chosen.

After the Q-values have been updated, the algorithm checks all leaf nodes to see if it would be useful to expand a leaf and replace it with a new center node containing a new test, thus dividing the instances more finely. To find the best test to divide the instances, we try all possible tests, i.e. we try to divide the instances according to each attribute describing an observation. After all attributes have been tested, we choose the attribute that maximizes the error reduction as shown in equation 3 [3]. The error measure considered is the standard deviation ($sd(I_l)$) on the instances' expected rewards. If the standard deviation is reduced, it means that the rewards are closer to one another. Thus, the tree is moving toward its goal of dividing the instances in groups with similar expected rewards. The expected error reduction obtained when dividing the instances I_l of leaf l is calculated using the following equation where I_d denotes the subset of instances in I_l that have the d^{th} outcome for the potential test:

$$\Delta error = sd(I_l) - \sum_d \frac{|I_d|}{|I_l|} \times sd(I_d) \tag{3}$$

The standard deviation is calculated on the expected reward of each instances which is defined as:

$$Q_I(i_t) = r_t + \gamma Pr(L(i_{t+1})|L(i_t)) \times Q(L(i_{t+1})) \tag{4}$$

where $Pr(L(i_{t+1})|L(i_t))$ is calculated using equation 2 and $Q(L(i_{t+1}))$ is the value returned by equation 1.

As mentioned earlier, one test is tried for each possible instance's attribute. For a discrete attribute, we divide the instances according to their value and for a continuous attribute, we test different thresholds to find the best one.

Finally, after the tree has been updated, we update the Q-values again to take into consideration the new state space.

During the simulation, the agents are using the tree created offline to choose the best fire to extinguish. When choosing a fire to extinguish, the *FireBrigade* agent has a list of buildings on fire it knows about. For each building in the list, the agent finds the leaf of the tree to which this building belongs and record the expected reward associated with this leaf. The chosen building is the one that has the greatest expected reward.

3 Experimentations

As said before, experimentations have been done in the RoboCupRescue simulation environment. We have implemented the algorithm presented in this paper for the task of choosing a fire to extinguish by the *FireBrigade* agents. Each agent independently evaluates the list of buildings on fire it knows about and make its decision regarding which one to extinguish.

At first, the agent looks at the fires from a higher point of view. It only considers zones on fire, which are clusters of near buildings on fire. To make its decision, it should consider some attributes describing a zone on fire. In our experiments, there were almost 4×10^{13} possible states. After the *FireBrigade* agent has chosen a zone on fire, it should choose which fire to extinguish. To this end, this agent can examine the attributes describing a fire. In our experiments, there were more than 265×10^6 possible states. The results we have obtained show a drastic reduction of the state space. The agents have learned a tree with 998 leaves for the problem of choosing a zone on fire and 571 leaves for the problem of choosing a building on fire.

With those trees that are relatively small compared to the possible number of states, the agents were able to learn some interesting behaviors. In our preliminary tests, the agents were choosing useful zones and buildings on fire to extinguish. However, we also observed an unwanted variation of performances from one simulation to another. One hypothesis is that the variation is due to the uncertainty of the environment. There are a lot of hidden states because the agents are not able to perceive everything.

4 Related work

The approach presented is inspired by the work of McCallum in his thesis [1]. In our work, we use approximately the same tree structure to which we have made some modifications. Firstly, we do not use the tree to calculate Q-values for every possible basic actions that an agent can take. We use the tree to calculate the expected reward if the agent chooses a certain task. It still has to find the actions to accomplish this task.

Another difference is in the way the leaves are expanded. Instead of generating all possible subtrees, we use an error reduction estimation measure, borrowed from the decision tree theory [3], that enables us to find good tests. Like Uther and Veloso [4] with their Continuous U-Tree algorithm, we also support continuous attributes, but with a different splitting criteria.

The way we manage our instance chains is also quite different. In our work, there is not only one chain, but many chains, one for each attempt to extinguish a fire or a fire zone. Our concept of instance chain is closer to the concept of *episode* described by Xuan, Lesser and Zilberstein [5].

5 Conclusion

This article has presented an efficient algorithm to enable the agent to learn the best task to choose, i.e. which building to extinguish. We have shown how we can adapt a reinforcement learning algorithm to the problems of task allocation and multi-attribute decision making in a complex application. Our algorithm enables the agents to manage a large state space by letting them learn by themselves to distinguish only the states that need to be distinguished. By doing so, the agent drastically reduces the number of states, thus facilitating the calculation of the Q-values. Our results show that agents are able to obtain good results with trees having a small number of leaves compared to the total number of states before learning. In our future work, we will adjust the agents behavior to take more into account the uncertainty in the environment and thus diminishing the variation in the results.

References

1. McCallum, A.K.: Reinforcement Learning with Selective Perception and Hidden State. PhD thesis, University of Rochester, Rochester, New-York (1996)
2. Kitano, H.: Robocup rescue: A grand challenge for multi-agent systems. In: Proceedings of ICMAS 2000, Boston, MA (2000)
3. Quinlan, J.R.: Combining instance-based and model-based learning. In: Proceedings of the Tenth International Conference on Machine Learning, Amherst, Massachusetts, Morgan Kaufmann (1993) 236–243
4. Uther, W.T.B., Veloso, M.M.: Tree based discretization for continuous state space reinforcement learning. In: Proceedings of the Fifteenth National Conference on Artificial Intelligence, Menlo Park, CA, AAAI-Press/MIT-Press (1998) 769–774
5. Xuan, P., Lesser, V., Zilberstein, S.: Modeling Cooperative Multiagent Problem Solving as Decentralized Decision Processes. Autonomous Agents and Multi-Agent Systems (2004) (under review).