Selective Perception Learning for Tasks Allocation

Sébastien Paquet, Nicolas Bernier and Brahim Chaib-draa
DAMAS Laboratory, Laval University, Canada
{spaquet;bernier;chaib}@damas.ift.ulaval.ca

Abstract

This paper presents a learning algorithm used to allocate tasks to agents in an uncertain real-time environment. In such environment, tasks have to be analyzed and allocated really fast for the multiagent system to be effective. To analyze those tasks, described by a lot of attributes, we have used a selective perception technique to enable agents to narrow down the description of each task by choosing the attributes that it should be considering in each situation. By doing so, we have obtained a drastic reduction of the number of possible states. We have used this algorithm at two different levels for the problem of choosing the best fire to extinguish for each firefighter agent in the RoboCupRescue simulation environment. First, a center agent is using the algorithm to allocate a zone on fire for each firefighter agent. Then, those agents are choosing the best fire to extinguish in this zone. Our results show a good improvement in the agents capability to extinguish fires, as the agents become better at distinguishing the world states.

1. Introduction

Analyzing many possible goals and choosing the best one is obviously a difficult task 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.

In those difficult settings, we present in this paper an allocation algorithm based on the information learned by a selective perception learning algorithm. In our approach, each agent evaluates the number of agents needed to accomplish a particular task and it uses this information to allocate itself to a task, knowing the other agent are doing the same thing. Thus, our agents are coordinating themselves without any communication about their intentions. The selective perception technique enables the agent to learn by itself which is the level of precision it needs to efficiently describe states in all possible situations [3].

We have tested our approach in the RoboCupRescue simulation environment. Our results show that the agents' performances improved with the learning algorithm. They obtained good results with a compact representation of the state space.

In the remaining of this article we explain in more details the approach we have used, but first we describe, in the following section, our test environment, the RoboCupRescue simulation.

2. The RoboCupRescue Environment

The goal of the RoboCupRescue simulation project is to build a simulator of rescue teams acting in large urban disasters [1, 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 There are 10 to 15 agents of this type. Their goal is to extinguish fires. Each *FireBrigade* agent is in contact by radio with all other *FireBrigade* agents as well as with the *FireStation*.

PoliceForce There are 10 to 15 agents of this type. Their goal is to clear roads to enable agents to circulate. Each *PoliceForce* agent is in contact by radio with all other *PoliceForce* agents as well as with the *PoliceOffice*.

AmbulanceTeam There are 5 to 8 agents of this type. Their goal is to search in shattered buildings for buried civilians and to transport injured agents to hospitals. Each *AmbulanceTeam* agent is in contact by radio with all other *AmbulanceTeam* agents as well as with the *AmbulanceCenter*.

Center agents There are three types of center agents: *FireStation*, *PoliceOffice* and *AmbulanceCenter*. The only actions those agents can make are to send and receive messages. They are in contact by radio with all their platoon agents as well as with the other center agents. Platoon agents are the moving agents in the

simulation, which are the first three mentioned in this list. A center agent can read more messages than a platoon agent, so center agents can serve as information centers and coordinators for their platoon agents.

In the simulation, each individual agent receives visual information of only the region surrounding it. Thus, no agent has a complete knowledge of the global state of the environment. Therefore the RoboCupRescue domain is in general, collectively partially observable [4]. This means that even if the agents are putting all their perceptions together, they do not have a perfect perception of the environment. This uncertainty complicates the problem greatly. Agents have to explore the environment, it is not enough to work only on the visible problems. They also have to communicate to help each other to have a better knowledge of the situation.

3. Problem Description

In this article, we focus only on the work of the *FireBrigade* and *FireStation* agents. Those agents are faced with the problem of choosing a fire to extinguish between a list of buildings on fire, which are described with some attributes. 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 regroupments of near buildings on fire.

To stop a zone from spreading, the *FireBrigade* agents have to extinguish all fires at the border of the zone. Also, it is pointless to constantly change from one fire zone to another, because by doing so, all zones would spread. A better strategy is to choose a zone and stop the spreading or really slow it down, before choosing another zone.

Therefore, when an agent has to choose a building to extinguish, it firstly has to choose the fire zone. In other words, agents are using a two level decision making process. First of all, they look at the global view of the situation, looking only at groups of buildings on fire. Afterwards, they use more detailed information to choose which specific building to extinguish in the chosen fire zone.

Since each mobile agent has only a local view of the situation, it is often hard for a *FireBrigade* agent to have a good view of the global situation. Therefore, it is difficult for them to choose between the fire zones since they don't have a good knowledge of the fire zones that are far from them. It is why it is the responsibility of the *FireStation* agent to allocate fire zones to *FireBrigade* agents, because it can receive more messages, so it normally has a better knowledge of the global situation compared to the *FireBrigade* agents.

Our goal is to enable the *FireBrigade* agents to learn how many agents are needed to extinguish all kinds of fires. In addition, the learning algorithm has to be resistant to noise because of the uncertainty present in the RoboCupRescue simulation environment. For example, two buildings on fire could be identically described and give different rewards.

4. Selective Perception

To learn the expected reward of choosing one building on fire, we have used a selective perception technique [3], because the description of our states is too big. With this technique, the agent learns by itself to reduce the number of possible states. In fact, the agent regroups all similar states together and it does not distinguish between states of the same group. It considers states to be similar if they have similar expected rewards. Since they have similar expected rewards, the agent does not have to distinguish them since it would take the same decision in all of those situations.

To be more precise, it is worth mentioning that the algorithm is not really trying to regroup states, it is trying to divide them. To do that, the algorithm 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. To divide a leaf, it tries to divide its experiences, stored in that leaf, by making a test on an attribute. By doing so, the agent creates new states, by expanding a leaf of the tree, and thus it refines its view of the state space.

An advantage of this algorithm is that it distinguishes only states that really need to be distinguished. This has the effect of reducing the state space of the learning algorithm and thus facilitating the learning process. The following subsections describe the algorithm in more details.

4.1. Recording of the Agent's Experiences

At each time step t, the agent records its experience captured as an "instance" that contains 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. Consequently, an instance at time t is defined as:

$$i_t = \langle i_{t-1}, o_t, r_t, i_{t+1} \rangle \quad (1)$$

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 agent records many instances organized in many instance chains. It keeps all those instances until the end of the simulation. There is no learning taking place during the simulation since it would take too much time. Agents are

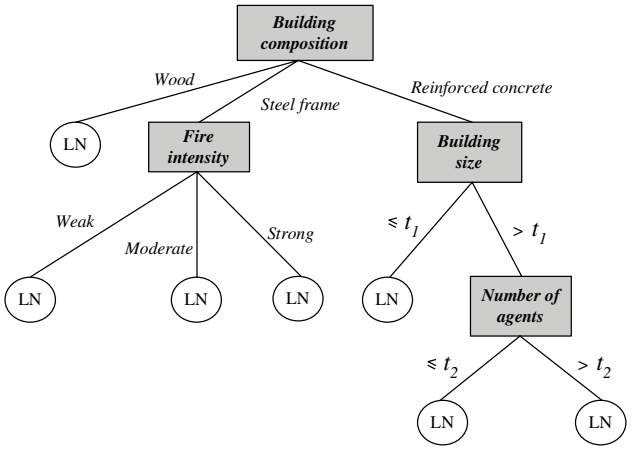

Figure 1. Structure of a tree.

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. By regrouping their experiences, agents can accelerate the learning process. The next sections explain the tree structure and how it is updated with the instances recorded by all agents.

4.2. Tree Structure

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 is to regroup all instances having similar expected rewards.

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. 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's attribute value. Each leaf of the tree also contains the expected reward if a fire that belongs to this leaf is chosen.

An example of a tree is shown in Figure 1. Each rectangular node represents a test on the specified attribute. The words on the links represent possible values for discrete variables. A test on a continuous attribute has always two possible results, it is either less or equal to the threshold or greater than the threshold. The oval nodes (LN) are the leaf nodes of the tree, where the expected rewards are stored.

4.3. Update of the Tree

After a simulation, all agents put their new experiences together. This set of new experiences is then used to update the tree. First, all instances are added to the tree in their respective leaf. Afterwards, we update the expected reward of each leaf with the following equation:

$$Q(l) \leftarrow R(l) + \gamma \sum_{l'} Pr(l'|l)Q(l') \tag{2}$$

where $Q(l)$ is the expected reward if the agent tries to extinguish a building belonging to the leaf l, $R(l)$ 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. $R(l)$ is the average reward obtained when a fire belonging to this leaf was chosen and $Pr(l'|l)$ is the proportion of next instances that are in leaf l':

$$R(l) = \frac{\sum_{i_t \in I_l} r_{t+1}}{|I_l|} \tag{3}$$

$$Pr(l'|l) = \frac{|\{\forall i_t \in I_l | L(i_{t+1}) = l'\}|}{|I_l|} \tag{4}$$

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 all new instances have been added to the tree, we check all leaf nodes to see if it would be useful to expand some leafs and replace them with new center nodes containing a new test, thus dividing the instances more finely. To find the best test to divide the instances at a leaf node, we try all possible tests, i.e. we try to divide the instances according to each attribute describing a state. After all attributes have been tested, we choose the attribute that maximizes the error reduction as shown in equation 5 [7]. In fact, the test is chosen only if the expected error reduction is greater than a certain threshold, if not, it means that the test does not add enough distinction, so the leaf is not expanded. The error measure considered is the standard deviation ($sd(I_l)$) on the instances' 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 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:

$$\Delta error = sd(I_l) - \sum_{d} \frac{|I_d|}{|I_l|} \times sd(I_d) \tag{5}$$

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

$$Q_I(i_t) = r_t + \gamma Pr(L(i_{t+1})|L(i_t)) \times Q(L(i_{t+1})) \quad (6)$$

where $Pr(L(i_{t+1})|L(i_t))$ is calculated using equation 4 and $Q(L(i_{t+1}))$ is the value returned by equation 2.

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 for this attribute. For instance, if an attribute has three possible values, it generates three subsets, thus adding three children nodes to the tree. We then use the equation 5 and record the error reduction for this test. For a continuous attribute, we have to test different thresholds to find the best one. A continuous attribute always divides the instances in two subsets, the first one is for the instances with a value less or equal to the threshold for the specified attribute and the second subset is for the instances with a value greater than the threshold.

To find the best threshold, we have used the technique described by Quinlan [6]. The instances are first sorted according to their value for the attribute being considered. Afterwards, we examine all $m-1$ possible splits, where m is the number of different values. For example, with an ordered list of values $\{v_1, v_2, ..., v_m\}$, we try all possible thresholds. So, we try the value v_1 as a threshold, thus dividing the instances in two subsets, those less or equal and those greater than v_1. We calculate and record the error reduction for this division. Then, we do the same thing for the other possible values, v_2 to v_{m-1}. At the end, we keep only the threshold with the best error reduction value.

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

5. Use of the Tree

During the simulation, the agents are using the tree created offline to choose the best fire zone and the best building on fire to extinguish. Since the *FireStation* agent has a better global view of the situation, it is its responsibility to suggest fire zones to *FireBrigade* agents. However, those agents have a better local view, so they are choosing which particular building on fire to extinguish. By doing so, we can take advantage of the better global view of the *FireStation* agent and the better local view of the *FireBrigade* agent at the same time. This process is illustrated on Figure 2.

5.1. Number of Agents Estimation

The tree learned is used to get an estimation on the number of agents that are needed to extinguish a fire. The situations that are stored in the tree contain some attributes describing a fire and the number of agents. However, when we use the tree, we do not know the number of agents, it is what we want to estimate. More precisely, what we want is a lower bound, i.e. the minimum number of agents needed to extinguish a fire. To find this estimation, we find in the tree all expected rewards for all number of agents until we reach a certain threshold, see the algorithm on Figure 3. For example, if we have to estimate the number of agents for a fire f, we find the tree's leaf corresponding at the description of f with one agent extinguishing it. Suppose we obtain 10 and it is under the threshold, then we find the leaf for two agents and the same fire f. We continue like this adding one agent each time and we stop if the expected reward becomes greater or equal to the specified threshold. This threshold represents the limit over which we think that the agents are able to extinguish the fire. By doing so, we can estimate the less number of agents that are needed to extinguish a fire, even if the number of agents is an attribute describing a state.

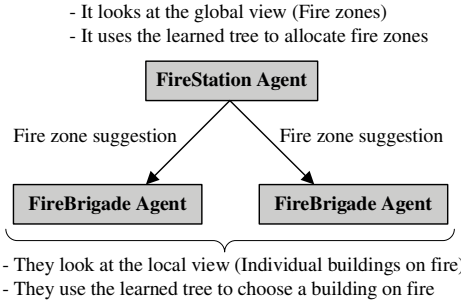

Figure 2. Illustrates how the agents collaborate to choose the fires to extinguish.

If for one particular situation, the is no node "number of agents" on the path from the root to the leaf, that means that the expected reward is independent of the number of agents. In this case, there are two possibilities: the expected reward is greater or equal to the threshold or less. If it is greater or equal, the agent considers that one agent is enough to extinguish this fire. But, if the expected reward is less then the threshold, the agent considers that it is impossible to extinguish the fire. In this last option, they will all go to the same building, if there is no "possible" fires left.

5.2. Fire Zones Allocation

To allocate the fire zones, the *FireStation* agent has a list of all fire zones. For each fire zone, it has to estimate the number of agents that are needed to extinguish this zone. To do so, it makes a list of all the buildings that are at the border of the zone. For each of those buildings, the agent uses the algorithm on Figure 3 to get an estimate of the number of agents needed to extinguish each fire. It then estimates the number of agents needed to extinguish the fire

function NUMBER-AGENTS(*tree*, *fireDescription*, $totalAgents$) **returns** *numberAgents*, the estimated number of agents.

Inputs: *tree*, the learned tree.
fireDescription, a vector of attribute values describing a fire.
totalAgents, the total number of agents available.

for $numberAgents = 1$ to $totalAgents$ **do**
  $expectedReward \leftarrow$ FIND-EXPECTED-REWARD($tree$, $fireDescription$, $numberAgents$)
  if $expectedReward \geq THRESHOLD$ **then**
    return *numberAgents*
  end if
end for
return -1 {It is impossible to extinguish this fire with the available agents.}

Figure 3. Algorithm used to estimate the number of agents needed to extinguish a fire.

zone as the maximum number of agents returned for one building in the zone. The *FireStation* agent does the same thing with all fire zones, ending up with a number of agents for each zone. With this information, it then chooses the zone for which the needed *FireBrigade* agents are the closest. When a zone is chosen, the assigned agents are those that are closer to this zone. Afterwards, it removes this zone and the assigned agents from its lists and continues the process with the remaining agents and the remaining fire zones. It continues until there is no agent or fire zone left. When it is done, the center agent sends a message to each *FireBrigade* agent to inform them about their new fire zone assignment.

5.3. Fires Allocation

Therefore, each *FireBrigade* agent receives a fire zone to extinguish from the *FireStation* agent. Since all messages are broadcasted, each agent also knows the assigned fire zone of all other agents. When choosing a fire to extinguish, the *FireBrigade* agent has a list of buildings on fire it knows about in the specified fire zone. More precisely, this list is a sorted list of buildings on the border of the zone in which each building at the position i is the closest building, not already in the list, of the building in position $i - 1$. The first building is a reference building given by the *FireStation*.

All *FireBrigade* agents have approximately the same list of buildings on fire. To choose their building on fire they go through the list, one building at a time. For each building, they use the tree to find the expected number of agents needed to extinguish the fire by using the algorithm presented on Figure 3. Afterwards, each agent uses its predefined position in the list of agents to find the building it has to extinguish. For example, if five agents are needed for the first building, the first five agent in the list of agents would choose this building and the other agents would choose between the other buildings. In short, the agents are using their predefined order to allocate themselves to a fire and thus the coordination is obtained without any communication about their respective choices. They can do that because they know which agents are with them to extinguish their assigned fire zone.

Figure 4. Evolution of the *FireBrigade* agents efficiency over 90 simulations.

6. Experimentations

In our experiments, we have started with an empty tree and we have let the agents learn a tree to distinguish the states. The graphic on figure 4 presents the evolution of the agents efficiency over 90 simulations. The graphic shows the percentage of intact buildings in average on 10 simulations. As we can see the agents' performances improved gradually as the tree become better at dividing the expected rewards.

By looking at the simulations, we could see that the *FireStation* agent become better at estimating the number of agents needed to extinguish a fire zone. At the same

time, the *FireBrigade* agents become better at estimating the number of agents needed for one building. Because of that, they were better coordinated and they were able to extinguish more than one fire at a time. It is the reason why they were faster at extinguishing fire zones. At the beginning, they were all extinguishing the same fire, but after some time, they learned a better estimate of the number of agents needed, so they were able to split themselves efficiently on different fires.

Furthermore, the agents were able to attain such good performances with trees having only approximately 1500 leafs. Therefore, they were just distinguishing 1500 states out of the 30 000 possible states. In other word, they were able to perform efficiently with an intern state space of only 5% of the complete state space. This shows a very good reduction of the state space, enabling the learning algorithm to work on an easier state space. In our tests, the states were described by:

- the intensity of the fire (3 values),
- the building's composition (3 values),
- the building's size (continuous attribute),
- the building's degree of deterioration (4 values),
- the number of *FireBrigade* agents (15 values).

7. Related works

Other researchers have used tree structures to represent and learn the state space of the agent. One of them is McCallum with its U-tree algorithm [3]. It used a tree structure to store Q-values for every possible basic actions that an agent can take. In our work, we use a similar tree structure to calculate the expected reward of a particular goal decision of the agent. It still has to find the actions to accomplish this goal. In other words, the tree is used at the goal decision level, not at the action decision level. Also, instead of generating all possible subtrees, we use an error reduction estimation measure, borrowed from the decision tree theory [7], that enables us to find good tests. In our case, the generation of subtrees would really be expensive since our state space is very large.

Like Uther and Veloso [8] with their Continuous U-Tree algorithm, we also support continuous attributes, but with a different splitting criterion. Pyeatt has also tried other splitting criteria [5].

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 [9].

8. Conclusion

This paper has presented a learning algorithm that enabled the agents to learn efficient expected rewards in a big state space. The approach we have presented enabled the agent to reduce the state space by distinguishing only states that really need to be distinguished. The agents are using the tree, with the expected rewards learned, in their allocation algorithm, enabling them to be well coordinated. Furthermore, the coordination process we have presented requires very few communications therefore it is effective in real-time environments with limited bandwidth. The results show that the agents were able to obtain good results with trees having a very small number of leaves compared to the total number of states before learning.

References

[1] H. Kitano. Robocup rescue: A grand challenge for multi-agent systems. In *Proceedings of ICMAS 2000*, Boston, MA, 2000.

[2] H. Kitano, S. Tadokor, H. Noda, I. Matsubara, T. Takhasi, A. Shinjou, and S. Shimada. Robocup-rescue: Search and rescue for large scale disasters as a domain for multi-agent research. In *Proceedings of the IEEE Conference on Systems, Man, and Cybernetics (SMC-99)*, 1999.

[3] A. K. McCallum. *Reinforcement Learning with Selective Perception and Hidden State*. PhD thesis, University of Rochester, Rochester, New-York, 1996.

[4] R. Nair, M. Tambe, and S. Marsella. Team Formation for Reformation in Multiagent Domains like RoboCupRescue. In G. Kaminka, P. Lima, and R. Roja, editors, *Proceedings of RoboCup-2002 International Symposium*, Lecture Notes in Computer Science. Springer Verlag, 2003.

[5] L. D. Pyeatt and A. E. Howe. Decision tree function approximation in reinforcement learning. Technical Report TR CS-98-112, Colorado State University, Fort Collins, Colorado, 1995.

[6] J. R. Quinlan. *C4.5 Programs for Machine Learning*. Morgan Kaufmann, San Mateo, CA, 1993.

[7] J. R. Quinlan. Combining instance-based and model-based learning. In *Proceedings of the Tenth International Conference on Machine Learning*, pages 236–243, Amherst, Massachusetts, 1993. Morgan Kaufmann.

[8] W. T. B. Uther and M. M. Veloso. Tree based discretization for continuous state space reinforcement learning. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, pages 769–774, Menlo Park, CA, 1998. AAAI-Press/MIT-Press.

[9] P. Xuan, V. Lesser, and S. Zilberstein. Modeling Cooperative Multiagent Problem Solving as Decentralized Decision Processes. *Autonomous Agents and Multi-Agent Systems*, 2004. (under review).