A learning layered agent model

Assignment in the agent programming course at DSV

(Stockholm University and KTH) 2I1235, 2003

Group 8:

Yet Another Chess Playing Multi-Agent System
(YACP-MAS)

28th May 2003

Table 1: Group members

Pedro Navarette Estrella
Christian Fernandes
Ronnie Johansson
Krister Ljung
Anders Windelhed

Abstract

Designing agent behaviors is rarely straight forward, especially if the natural environment of the agent is open and complex (as is the case of the Internet). In this paper, we describe our behavior discriminative agent (BDA) model which aims at supporting designers of such agents. The behaviors are here themselves represented by agents so the BDA agent comprises a multi-agent system, where behavior agents interact with a coordinator agent which decides the actions of the BDA agent.

The BDA agent uses the Q-learning technique and learns by sensing the consequences of its actions and updates its belief in the behaviors continually.

We implement our agent model in a chess game setting and present some early results.

Contents

1 Introduction

Even though there appear to be many definitions of *agent* in use, most agree that an agent is a situated entity that can *perceive* its environment and *act* in it to achieve some objectives [Wei99, RN95, Nil98].

Over the last couple of decades, researchers have exhibited a tremendous interest in *software agents* and the related *agent theory*. Some researchers believe that software agents will lead to improvements in computing, especially in complex, open, distributed systems[WJ99, Syc98], and consider it to be a new programming paradigm comparable to object-oriented programming.

As a tool for programming, agents provide programmers with a concept that is easy to relate to – facilitating a means of managing system complexity and communicating design between programmers – and also let them explicitly implement system behaviors. However, as pointed out by [WJ99], the success of the agent paradigm is dependent on our ability to make agents amenable to efficient implementation and execution.

One apparent problem, faced by the agent programmer, that originates from the often open and, hence, uncertain and complex agent environment (e.g., the Internet) is to assess the effectiveness of the agent implementation and design. In this, work we present a *layered agent model* [Woo99] for detecting inefficient agent design.

There are really two types of layered agent architectures: *vertical* and *horizontal*.[Woo99] In layered architectures, two or more layers, representing behaviors, can contribute to the *action selection* of the agent. There are typically at least one *reactive* and one *pro-active* layer. The reactive behavior is designed to respond to sudden changes in the environment (e.g., in robotics, obstacle avoidance) and the pro-active (or deliberative) to deal with mission goals.

In vertically layered architectures, control flow (from perception to action) is passed through all layers to force a consensus. In horizontally layered architectures, however, each layer separately perceives the environment and proposes an action. Horizontally layered architectures, thus, requires a centralized mechanism to determine the output action.

We have chosen to study a horizontally layered architecture where the layers represent various competences. In the horizontally layered architecture, a layer can easily be removed and possibly replaced with another competence if it is recognized to be inefficient. Contrary to a vertically layered architecture, this can be done with minimal affect on the other layers.

Notice that the layers in a horizontal architecture all perceive and suggest actions. The layers are, thus, agents in that sense. Hence, what we have in our model is actually an agent architecture composed of simpler agents, a *multi-agent system*. The architecture includes at least one agent that suggests actions, and one agent that is responsible for selecting which action to use.

As mentioned above, the purpose of our study is to support design of agent behaviors. In accordance with [Bro90], we believe that "the world is its own best model" and, hence, we want inefficient behaviors to be singled out by *learning* from the environment. In effect, we propose a learning agent model called *behavior discriminative agent model* (BDA).

Even though the general horizontal architecture supports continual action selection, we here, for simplicity, just study the situation where several behaviors (all actually) suggest an action each at the same time.

Our work is related to that of [Lin93], but have a different aim. In [Lin93], Q-learning is used to learn to use which behavior (agent) and when, just as in our model.

The difference, is however, that the problem of the agent is assumed to be decomposable into subtasks and that each behavior agent is specialised on one of the subtasks. In contrast, in our work, we do not assume that a behavior agent knows how to perform its task efficiently.

Section 2 describes our multi-agent architecture in more detail. An extensive explanation of the learning techniques used for isolating inefficient behaviors is explained in Section 3. In Section 4 we show that inefficient behaviors can actually be isolated by learning, and, finally, in Section 5 we conclude our work.

2 The Behavior Discriminative Agent Model

As mentioned in Section 1, we propose a learning agent model (the higher-level agent), composed of lower-level agents, i.e., *behavior agents* and a *coordinator agent* (see Figure 1). All agents perceive their environment. Behavior agents suggest suitable actions for the higher-level agent to perform in the environment, and the coordinator decides a resulting action considering the suggestions of the behavior agents.

Selecting a suitable action (or, in this case, a suitable agent) requires an *action selection mechanism*. In [Pir99], such mechanisms are divided into two groups, one called *command fusion* and the other *arbitration*. Command fusion mechanisms allow multiple behaviors to contribute to the resulting action of the higher-level agent. The arbitration mechanism, on the other hand, only allows one behavior to be active at a time. To make the actions of the higher-level agent more "pure", we prefer the latter mechanism which avoids making vain compromised actions. The arbitration mechanism will also allow the coordinator to learn which behavior is appropriate and inappropriate, respectively.

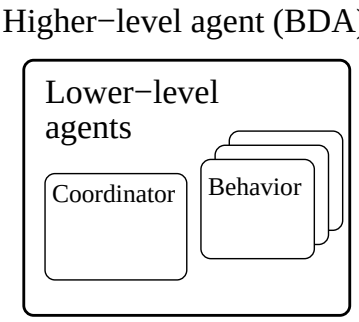

Figure 1: Our agent model (the higher-level agent which we call BDA) is composed of lower-level agents.

An agent assigns a high value to its suggested action if it believes it will bring the overall system closer to its objectives, or a low value if it believes the action to have just a small impact or if it is unsuitable.

Communication among the lower-level agents primarily concerns actions that behavior agents suggest to the coordinator agent. Behavior agents may communicate among each other, but it is not required, and may sometimes be unsuitable since it increases the coupling between the agents (i.e., the interdependence). There are no special restrictions on the behavior agents, how they come up with a suggestion for an action.

It becomes the responsibility of the coordinator agent to detect inefficient behaviors. By carefully observing the results of the actions of the different behaviors, the coordinator will learn that some behaviors may be more beneficial for some environment states. It will also be able to single out behaviors that are always less efficient than others.

Among various learning techniques, for instance supervised learning, we have selected *reinforcement learning* for our model.[KLM96] Unlike supervised learning, reinforcement learning does not rely on examples, but learns directly from the environment. In the next section, we will describe reinforcement learning in more detail.

3 Reinforcement learning

Reinforcement learning addresses the problem of how an acting and sensing agent can learn to perform the right action given the state that it is in. However, for the agent to be able to know when it has done something right it also needs a *reward* or *reinforcement*. Given this information, the agent can learn to choose an action in a given sensed state so that the reward will be maximized.

The idea is illustrated in Figure 2, where the agent perceives the state to generate both and interpretation of the current state (the I-module) and a reward (the R-module) to evaluate the consequences of previous actions.

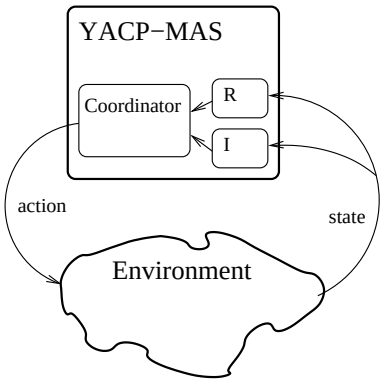

Figure 2: Reinforcement learning i YACP-MAS

The method used to learn which action to take in YACP-MAS is Q-learning (see for instance [Mit97, KLM96]), where the agent learns an action-value function, or Q-function. This Q-function returns the expected reward of taking a given action in a given state. Q-learning agents have the advantage that they do not need to have a model of the state space, i.e., they do not need to know the outcome of their actions, however this also means they cannot look ahead.

In the Q-learning algorithm a function $Q(s,a)$ is learned that maps every action a and the current state s to an expected reward gained from choosing action a.

When choosing an action however, it's not as simple as maximizing $Q(s,a)$ as this would lead to choosing actions that have been found early in training to have high $Q(s,a)$ values while it might miss other actions with even higher values. Therefore a

probabilistic approach is used to select actions

$$P(a_i|s) = \frac{k^{Q(s,a_i)}}{\sum_j k^{Q(s,a_j)}}$$

where $P(a_i|s)$ is the probability of choosing action i when in state s, and $k \geq 1$ is a constant that determines how strongly the selection favors higher valued actions. Thus increasing k as time progresses will cause the agent to go from exploring different actions during the early stages of learning, and then as k increases the agent will exploit the $Q(s,a)$ values more by choosing the higher valued actions.

The Q-function is updated according to the following rule when it performs an action a leading it to a new state s_{t+1} from the old state s_t and receives the reward $R(s_t)$.

$$Q(s_t,a) \leftarrow Q(s_t,a) + \alpha(R(s_t) + \gamma \max_{a'} Q(s_{t+1},a') - Q(s_t,a)) \quad (1)$$

The $0 \leq \gamma \leq 1$ parameter indicates how important future rewards are. $\gamma = 0$ means that the agent is only interested in immediate rewards. $0 \leq \alpha \leq 1$ is the learning rate parameter. It determines how much the learning process should react to rewards. A low value on α will lead to a slow convergence of the Q-function, while a high value may lead to exaggerated updates of the Q-function.

An extension, TD-learning, which we used, updates the Q-function not just in the previous state and action, but also in some of the previous states and actions (as shown in Equation 2). An extra memory, the *eligibility trace* $e(a,s)$, keeps track of previous states and actions.

$$Q(s,a) \leftarrow Q(s,a) + \alpha(R(s_t) + \gamma \max_{a'} Q(s_{t+1},a') - Q(s,a))e(s,a) \quad (2)$$

where the eligibility trace is updated in the following way

$$e(s,a) = \begin{cases} \gamma e(s,a) + 1 & \text{if } s \text{ is current state} \\ 0 & \text{if previous } e(s,a) \text{ is below some threshold} \\ \gamma e(s,a) & \text{otherwise} \end{cases} \quad (3)$$

4 Experiments

To test our behavior discriminative agent (BDA) model, we attempt to run an agent based on BDA in an environment using a set of behaviors where one is expected to be much less efficient than the others. Our approach is to monitor the learning process to see that the coordinator learns to avoid the inappropriate behavior.

The following subsection provides detailed information about the experiment and may be skipped. The results of the experiments can be found in Section 4.6.

4.1 Experiment setup

The selected environment is a chess game. This setting provides us with a discrete action and state space to facilitate the construction of fairly simple agents. Also, For a designer of a chess playing agent it might be difficult to design suitable behaviors since the outcome of the game is heavily dependent on the skill and techniques of the opponent.

Our BDA is provided with three behavior agents (explained in Section 4.2). One is a specialist in opening moves (only responding to initial states of chess game, otherwise

idle). A second agent is using a min-max search of states with alpha-beta pruning. The third agent uses a modified min-max search which makes sure that it makes the worst move possible (from the perspective of the BDA agent). Furthermore, the BDA has a learning coordinator that implements the reinforcement learning technique described in Section 3.

The implemented BDA competes repeatedly against a sparring partner which is composed of a coordinator that does not learn but selects randomly between a min-max search agent and one that makes random moves. The purpose of letting the sparring agent use a random agent is to allow the BDA agent to experience a greater variety of states. The idea is depicted in Figure 3.

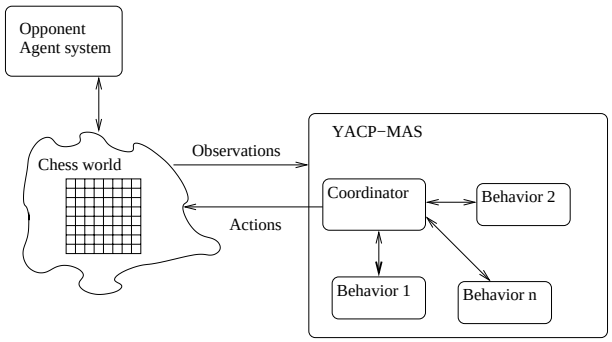

Figure 3: The experiment setup

4.2 Behavior agents

The purpose of the behavior agents is to select a behavior consistent action given the current environment state. We have developed a few behavior agents as explained below.

Random agent The random agent is our most simple agent. It receives a list of all currently possible moves from the chess board object, picks one of them randomly and sends it as its suggestion to the blackboard. It was created primarily because we quickly wanted a simple agent to test our system with, but was later used in experiments to create variation in the games.

Opening agent This opening agent has only one task and that is to deliver the first five moves to the coordinator. The five moves are a static sequence, which will move the pieces to a good strategic position and then leave the move-suggestions to the other agents. There is one opening sequence for white and one for black. If the sequence should be broken by the opponent taking one of the pieces included in the sequence, the agent will make no more move-suggestions.

Min-max agents The min-max agents uses a modified version of min-max alpha-beta search algorithm (see for instance [RN95]). The difference from the ordinary algorithm is that it does not assume that both players have the same chess board evaluation function.

4.3 Coordinator agents

The BDA coordinator can not propose an action by itself. That would be a violation of the abstraction used in BDA. Instead, they rely on the behavior agents to generate appropriate actions. The coordinator of the sparring partner is implement in a similar way for convenience.

BDA coordinator Has a learning coordinator that implements the reinforcement learning technique described in Section 3.

Sparring coordinator To evaluate our chess playing system, we created a new, much simpler system to spar against. The sparring system consists of one random agent, one min-max alpha-beta agent and a simple coordinator that by 80% chance picks the move that the min-max agent suggests and 20% chance picks the move that the random agent suggests. The random agent was included to create more variation to the game and to simulate an opponent the makes unpredicted moves.

4.4 Simplified state representation

The number of states in a chess game is enormous. It would not be feasible to maintain a Q-function containing all possible states of the chess game. Instead we interpret the state of the chess board in a summarized form.

The state is interpreted as a nine bit word (i.e., a maximum of $2^9 = 512$ groups of states). The two most significant bits represent the current time of the game; is it early in the game, about the middle of the game, is it late or is the game over.

The following three bits represent the *material balance* between the two chess players. Material balance simply reflects the strength of pieces of a player compared to its opponent. Figure 4 shows how the difference in material value translates to one of seven values (ranging from -3 to 3).

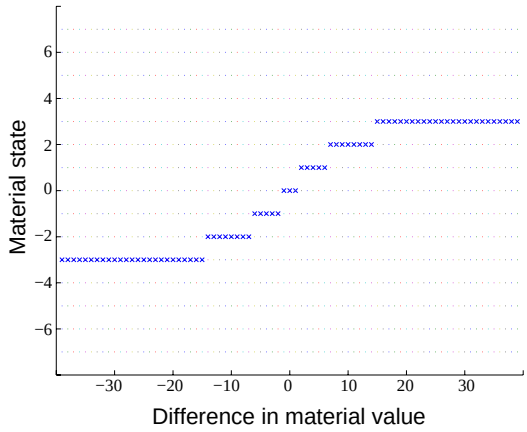

Figure 4: Simplified description of the difference between the current player and its opponent.

The following two bits reflect a *tropism* value for each of the two chess players. If one agent is close to the king with his rooks or queen its bit will be one, otherwise zero. The same thing goes for the other agent.

The final two bits reflect the current board control of the game, i.e., which player controls most squares on the chess board. A player controls a square if it can move more pieces to that square than its opponent.

4.5 Calculating reinforcement signals

The R-module in Figure 2, also known as *critic*, calculates the reinforcement signal (i.e., the reward) that the coordinator uses to update the Q-function. If the BDA agent was in state s_t when it made it previous move, and it finds itself in state s_{t+1} when it is time to make the next move, the reinforcement signal $R(s_t)$ becomes

$$R(s_t) = V(s_{t+1}) - V(s_t), \tag{4}$$

where $V(s)$ is the value of the chess board according to the BDA coordinator. When calculating $V(s)$, we consider material value, board control, tropism, and development. The first three were explained in Section 4.4. The fourth gives negatives values to states where some pieces are blocking other pieces. The selection of these four is somewhat arbitrary and reflects the objectives of the BDA agent. If the new state was a win or a loss of the game, the reinforcement signal will be a great positive or great negative value, respectively.

4.6 Results

To get an idea of the consequences of the implemented learning process and some early results, we let the BDA agent play about a hundred games of chess. Figure 5 visualizes the results.

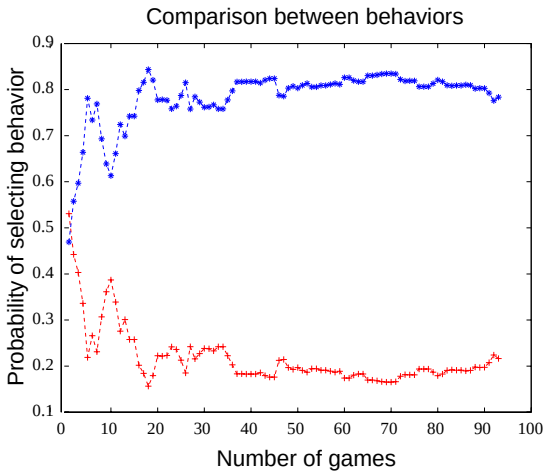

Figure 5: Probability of selecting the behaviors

For each game, the probabilities of the coordinator selecting the inappropriate behavior (the lower curve) and the other behavior are plotted. The probability values

summarize the probability of selecting each of the behaviors over all *visited* states. A state has been visited if it has been considered during the learning process. The probabilities of the opening agent are not considered as it is only active in a few states. The probabilities are calculated in this way

$$P(a) = \frac{\Sigma_{s \in \{\text{visited states}\}} P(a|s)}{\text{number of visited states}}$$

As can be seen in the figure, the coordinator quickly learns to avoid the inappropriate behavior.

5 Summary and Conclusion

Designing efficient behaviors for agents acting in an open and complex environment is rarely straight forward. In this paper, we present our behavior discriminative agent (BDA) model, which should support such design. The idea is to let the agent learn which behaviors are appropriate and which are not.

We implemented and tested our model in a chess game setting, and provided our BDA agent with both "good" and "bad" behaviors. Even though the coordinator, in our experiment, learned to "dislike" the inappropriate behavior fairly quickly, we had expected a more distinct isolation of that behavior. It could be that a longer learning time is required to see such result, or perhaps a more careful tuning of the learning parameters.

In Section 4.6, we tentatively propose a measurement of efficiency (i.e., the summarized probability of selection). However, many others are conceivable. In our case, one might want to find a suitable threshold (of probability of selection), and discard a behavior that plunges below this threshold.

For future research, it would be interesting to try more learning settings, finding suitable thresholds for discarding behaviors, and testing other measures of efficiency. It would also be interesting to study how different behaviors can complement each other.

References

[Bro90] Rodney A. Brooks. Elephants don't play chess. *Robotics and Autonomous Systems*, (6):3–15, 1990.

[KLM96] Leslie Pack Kaelbling, Michael L. Littman, and Andrew P. Moore. Reinforcement learning: A survey. *Journal of AI Research*, 4:237–285, 1996.

[Lin93] Long-Ji Lin. Scaling up reinforcement learning for robot control. In *Proceedings of the Tenth international conference on Machine learning*, 1993.

[Mit97] Tom Mitchell. *Machine learning*. McGraw-Hill, 1997.

[Nil98] Nils J. Nilsson. *Artificial Intelligence - A New Synthesis*. Morgan Kaufmann Publishers Inc., 1998.

[Pir99] Paolo Pirjanian. *Multi-objective Action Selection and Behavior fusion using voting*. PhD thesis, Aalborg University, 1999.

[RN95] Stuart Russell and Peter Norvig. *Artificial Intelligence - A Modern Approach*. Prentice Hall, Englewood Cliffs, New Jersey, 1995.

[Syc98] Katia P. Sycara. Multiagent systems. *AI Magazine*, 19(2):79–92, Summer 1998.

[Wei99] Gerhard I. Weiss, editor. *Multiagent Systems - A Modern Approach to Distributed Artificial Intelligence*. MIT Press, 1999.

[WJ99] Michael Wooldridge and Nicholas Jennings. Software engineering with agents: pitfalls and pratfalls. *IEEE Internet Computing*, 3(2):20–28, May/June 1999.

[Woo99] Michael Wooldridge. *Multiagent Systems - A Modern Approach to Distributed Artificial Intelligence*, chapter 1, pages 27–78. In Weiss [Wei99], 1999.