L-VIBRA: Learning in the VIBRA Architecture

Anna H. Reali-Costa and Reinaldo A. C. Bianchi

Laboratório de Técnicas Inteligentes - LTI/PCS
Escola Politécnica da Universidade de São Paulo
Av. Prof. Luciano Gualberto, trav. 3, 158. 05508-900 São Paulo - SP, Brazil.
{anna,rbianchi}@pcs.usp.br
http://www.pcs.usp.br/~lti

Abstract. Research in Multi-Agents Systems (MAS) has been, from its outset, concerned with coordinating intelligent behavior among a collection of autonomous intelligent agents. In the last years the use of on-line learning approaches to achieve coordination has attracted an increasing attention. The purpose of this work is to use a Reinforcement Learning approach in the job of learning how to coordinate agent actions in a MAS, aiming to minimize the task execution time. To achieve this goal, a control agent with learning capabilities is introduced in an agent society. The domain on which the system is applied consists of visually guided assembly tasks such as picking up pieces, performed by a manipulator working in an assembly cell. Since RL requires a large amount of learning trials, the approach was tested in a simulated domain. From the experiments carried out we conclude that RL is a feasible approach leading to encouraging results.

1 Introduction

Research in Multi-Agents Systems (MAS) has been, from its outset, concerned with coordinating intelligent behavior among a collection of autonomous intelligent agents (Bond and Gasser 1988). Systems have been traditionally organized in ways to achieve this coordination: in centralized and hierarchical organizations, in authority structures, market-like structures or in communities with rules of behavior.

Whereas previous research on MAS focused on off-line design of agent coordination mechanisms (Boutilier and Brafman 1997, Decker and Lesser 1995, Goldman and Rosenschein 1994), in the last years the use of on-line learning approaches to achieve coordination has attracted an increasing attention.

One of the most successful approaches applied to the multi-agent coordination problem is Reinforcement Learning - RL (Sen and Sekaran 1998). In an RL scenario, an agent learns on-line by trial-and-error performing the following basic steps: *(i)* based on its perceptions, the agent chooses and performs an action on the environment; *(ii)* the agent receives back from the environment a scalar feedback based on past actions; *(iii)* the agent then updates its internal mapping from perceptions to actions based on the rewards and punishments it received from the environment, aiming at a feedback maximization.

The purpose of this work is to use an RL approach in the job of learning how to coordinate agent actions in a MAS, aiming to minimize the task execution time. To achieve this goal, a control agent with learning capabilities is introduced in the agent society.

The MAS used in this work is based on the VIBRA architecture that was developed in previous group work with the objective of integrating visual perception, planning, reaction and execution to solve real world problems (Reali-Costa et al. 1998). VIBRA is organized with authority structures and rules of behavior and has been applied in the assembly domain, which consists of visually guided tasks such as picking up pieces, performed by a manipulator working in an assembly cell.

The reminder of this paper is organized as follows. Section 2 describes the assembly task domain used in the experiments. An overview of the VIBRA Architecture is given in Section 3, which also describes the problems with its initial implementation and the solution adopted. Section 4 reviews some key concepts concerning reinforcement learning. Section 5 presents highlights of the related approaches to the multi-agent coordination problem. Section 6 presents the experimental setup, the experiments performed in the simulated domain and the results obtained. Finally, Section 7 summarizes some important points learned from this research and outlines future work.

2 The Application Domain

The assembly domain can be characterized as a complex and reactive planning task, where agents have to generate and execute plans, coordinate its activities to achieve a common goal and perform online resource allocation. The difficulty in the execution of the assembly task rests on possessing adequate image processing and understanding capabilities and appropriately dealing with interruptions and human interactions with the configuration of the work table. This domain has been the subject of previous group work in the LSI Flexible Assembly Cell shown in Figure 1 (Reali-Costa et al. 1998).

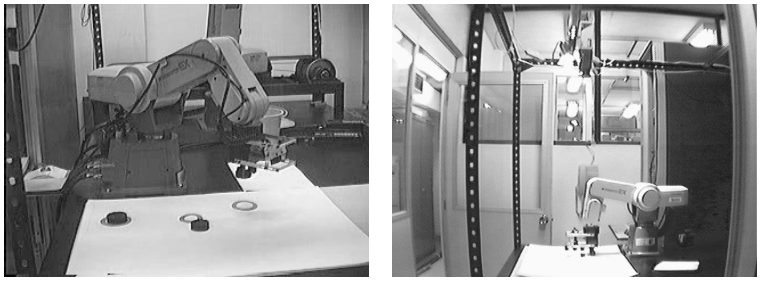

Fig. 1. One of the Assembly Cell manipulators.

In the assembly task, given a number of parts arriving on the table (from a conveyor belt, for example), the goal is to select pieces from the table, clean and pack them. The pieces can have sharp edges as molded metal or plastic objects usually do in their manufacturing process. To clean a piece means to remove these unwanted edges or other objects that obstructs packing. In this way, there is no need to clean all the pieces before packing them, but only the ones that will be packed and are not clean.

While the main task is being executed, unexpected human interactions can happen. A human can change the table configuration by adding (or removing) new parts to it. In order to avoid collisions, both the cleaning and packing tasks can have their execution interrupted until the work area is free of collision contingencies.

The assembly domain is a typical case of a task that can be decomposed into a set of independent tasks. To reduce the complexity of the problem, it can be assumed that each task of the solution corresponds to a desired behavior which is independent and interacts with other behaviors. Therefore, the solution of this assembly task can be decomposed into three subtasks:

- the assembly: if a piece on the table is clean, pick it up with the manipulator and put it on a desired location, packing it;
- the cleaning: if a piece have sharp edges, clean it before packing;
- the collision avoidance: to avoid collisions of the manipulator with objects that move in the workspace (other manipulators or humans), aiming the preservation of the system's physical integrity.

In the assembly cell the goal of the assembly and the type of pieces involved in it can change, e.g., from the packing task to the assembly of a known object or the selection of pieces by shape or color. Frequent applications are the assembly of small industrial products as mechanical parts or toys. It's worthy noting that an assembly cell can be part of a larger assembly line, accomplishing part of a complex production. Whereas the chosen domain is the one of an assembly cell, the architecture can be applied to other domains, e.g., autonomous mobile robots.

While existing research has not yet produced an ultimate paradigm for the distribution and coordination of tasks that an intelligent robotic system must posses, we propose a Multi-Agent approach where tasks, as well as the relationship among them, are translated into autonomous agents which communicates with each other, composing a society of autonomous agents. This model is described in the next section.

3 The VIBRA Architecture

The VIBRA - VIsion Based Reactive Architecture can be viewed as a society of Autonomous Agents (AAs), each of them depicting a problem-solving behavior due to its specific competence, and collaborating with each other in order to orchestrate the process of achieving its goals (Reali-Costa et al. 1998).

Multi-Agent Systems (MAS) define the agent's organization in a society, where relationships of authority, communication, control, and information flow are described. The VIBRA architecture is a multi-agent system, which processes incoming asynchronous goal requests dictated by sensory data, prioritizes them, temporally suspends lower priority actions, and interleaves compatible behaviors.

In this architecture an AA is composed of eight components (see Figure 2): a *Communicator module*, responsible for interactions between AAs and the planner-executor module; a *Planner-Executor module*, responsible for generating and executing a plan to solve the task that will exhibit the behavior expected from the AA; *Primitive Agents (PAs)*, that executes simple tasks, including sensing and acting; *Protocols and languages of interaction*, which define the communication capability of the agent; the *Authority Structure*, that consists of the relation resource-agent priority level; the *Conducting Rules of the society* define priority levels for each pair resource-agent; the *Set of AAs in the society*, containing the list of AAs in the society; and a *Symbolic representation of the world*, that represents the knowledge about the environment needed by each agent.

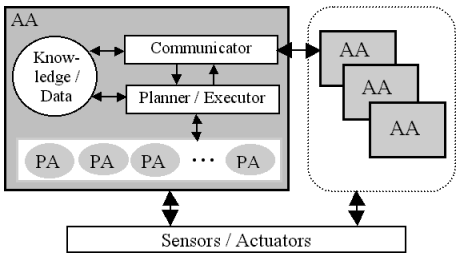

Fig. 2. Structure of the Autonomous Agent.

This model is used to define all AAs in the society, no matter what their behavior is. A special agent in the society can create, change or destroy the society, by adding or deleting agents, and controlling the resources at the initialization or termination phase of the society. As conditions change, new agents can be implemented and activated in the society during the system execution. On the other hand, those agents that are not performing well can be disabled.

An important way by which agents interact is through resource allocation, performed on line by the system. A resource is defined as a part of the system that can be time-shared by different agents. The resources in this assembly application are the camera and the manipulator, which are shared by the concurrent tasks assembly, cleaning and collision avoidance.

No conflicts arises due to a request for the camera resource by any of them. On the other hand, the manipulator is highly disputed and hence the tasks have to obey a specific policy for conflict resolution, since only one task should control the manipulator in a given moment. This policy, which is deeply related with the coordination of the behaviors, involves conducting rules and an authority

structure. This policy enables the agents to decide which agent should have the control of a resource at each moment.

The authority structure defines the agent's priority level in the use of a specific resource. The authority structure is domain dependent: the priority levels vary for each agent and each resource. In general, reactive tasks should have a precedence over deliberative tasks, and in this way the authority structure can be a rank ranging from the most reactive agent to the most deliberative one. In the assembly domain, the collision avoidance task should have the highest priority in order to prevent accidents, with the cleaning task second with a higher priority than the assembly task.

The explicit use of social rules in the definition of an agent enables it to achieve its dynamically-acquired goals without interfering with others. These rules define how the authority structure can be used to control the communication and share resources among agents. In the VIBRA architecture we adopt the following three simple rules:

- **Rule # 1:** Only one agent can control a resource in a given moment.
- **Rule # 2:** At any moment any agent can request control of a resource from an agent with lower authority than itself.
- **Rule # 3:** An agent can only request control of a resource from a higher authority agent if that agent is releasing control.

3.1 Problems in the former solution

One of the drawbacks of the described solution is that VIBRA uses a fixed, predefined authority structure. Once established that one agent has precedence over another, the system will always behave in the same way, no matter if it results in an efficient performance. However, in a real application, if an unwanted object is not preventing an assembly action, it is not necessary to perform a previous cleaning action.

To solve this problem we replaced the fixed authority structure by a dynamic one, through the addiction of a learning task to the architecture, resulting in L-VIBRA. The learning task aims to minimize the execution time of assembly tasks, selecting the best order in which agents should perform their actions.

This learning task is introduced in the architecture in the form of a control agent that takes part in the society and learns the best action policy based on the task to be performed and the perceived state of the environment. Since collision avoidance is an extremely reactive task, L-VIBRA preserves its precedence over cleaning and assembly tasks.

We use Reinforcement Learning to learn how to coordinate agent actuation, deciding among cleaning and assembly actions. The next section reviews some key concepts concerning reinforcement learning.

4 Reinforcement Learning

Let us consider an autonomous agent interacting with its environment via perception and action. On each interaction step the agent senses the current state

s of the environment, and chooses an action a to perform. The action a alters the state s of the environment, and a scalar reinforcement signal r (a reward or penalty) is provided to the agent to indicate the desirability of the resulting state. The goal of the agent in an RL problem is to learn an action policy that maximizes the long-run sum of values of the reinforcement signal from any starting state.

One of the most difficult problems facing an RL agent is *temporal credit assignment*, where the agent must be able to determine which one of its actions is desirable based on delayed rewards. Here we define one general formulation of the RL problem, based on a discrete time, finite state, finite action Markov Decision Process (MDP), since problems with delayed reinforcement are well modeled as MDPs.

The learner's environment can be modeled by a MDP represented by a 4-tuple $\langle S, A, R, T \rangle$ (see Kaelbling et al. 1996, Mitchell 1997), where:

- S is a set of states
- A is a set of actions
- R is a scalar reward function, $R : S \times A \rightarrow \mathcal{R}$
- T is a state transition function, $T : S \times A \rightarrow \Pi(S)$, where a member of $\Pi(S)$ is a probability distribution over S. $T(s, a, s')$ represents the probability of moving from state s to s' by performing action a.

The model is *Markov* if the state transition function and the reward function are independent of any previous environment states or agent actions. As defined, the state transition function and the reward function can be nondeterministic functions.

The task of an RL agent is to learn a policy $\pi : S \rightarrow A$ that maps the current state s into the desirable action(s) a to be performed in s. In RL, the policy π should be learned through trial-and-error interactions of the agent with a dynamic environment, that is, the RL learner must explicitly explore its environment. This way, the learner faces the fundamental tradeoff between *exploration* to gather new information and *exploitation* to maximize its cumulative reward. A strategy for exploration should be defined and used in the RL problem to balance between exploration and exploitation.

In L-VIBRA, we adopted the infinite horizon model to define the expected cumulative value $V^{\pi}(s_t)$ achieved by following an arbitrary policy π from an arbitrary initial state s_t. The Infinite horizon discounted model takes the long-run reward of the agent into account, and also includes a constant γ (where $0 \leq \gamma < 1$) that determines the relative value of delayed versus immediate rewards (Mitchell 1997, Kaelbling et al. 1996):

$$V^{\pi}(s_t) \equiv E\left[\sum_{i=0}^{\infty} \gamma^i r_{t+i}\right].$$

One strategy to learn the optimal policy π^* when the model (T and R) is not known in advance is known as Q Learning. It allows the agent to learn the evaluation function Q, instead of V^*.

Let $Q^*(s,a)$ be the reward received upon performing action a in state s, plus the discounted value of following the optimal policy thereafter:

$$Q^*(s,a) \equiv R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') V^*(s').$$

Since $V^*(s) \equiv \max_a Q^*(s,a)$, the optimal policy π^* is $\pi^* \equiv \arg\max_a Q^*(s,a)$. Rewriting $Q^*(s,a)$ in a recursive form:

$$Q^*(s,a) \equiv R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') \max_{a'} Q^*(s',a').$$

Let $\hat{Q}$ be the learner's estimate of $Q^*(s,a)$. The Q learning algorithm iteratively approximates $\hat{Q}$, i.e., the $\hat{Q}$ values will converge with probability 1 to Q^*, provided the system can be modeled as a MDP, the reward function is bounded $(\exists c \in \mathcal{R}; (\forall s,a), |R(s,a)| < c)$, and actions are chosen so that every state-action pair is visited an infinite number of times. The Q learning rule is:

$$\hat{Q}(s,a) \leftarrow \hat{Q}(s,a) + \alpha \left[r + \gamma \max_{a'} \hat{Q}(s',a') - \hat{Q}(s,a) \right],$$

where s is the current state; a is the action performed in s; r is the reward received; s' is the new state; γ is the discount factor $(0 \leq \gamma < 1)$; $\alpha = 1/(1 + visits(s,a))$, where $visits(s,a)$ is the total number of times this state-action pair has been visited up to and including the current iteration.

An interesting property of the Q learning is that, although the exploration-exploitation tradeoff must be addressed in Q learning, the $\hat{Q}$ values will converge to Q^*, independent of the strategy of exploration employed, providing all state-action pairs are visited often enough. (Mitchell 1997).

5 Related Work

Some researchers have used RL for developing effective behavior coordination in physical robots (Maes and Brooks 1991, Matarić 1997, Matarić 1998).

Maes and Brooks (1991) applied a statistical RL technique using immediate positive and negative feedback in order to learn when to activate behaviors for walking on a six-legged robot. The learning algorithm employed was completely distributed, so that each behavior tried to learn when it should become active.

Matarić (1997, 1998) chooses the concurrent multi-robot learning domain for her experiments. She uses conditions and behaviors to effectively diminish the otherwise prohibitively large learning space. In order to provide richer and more continuous reinforcement and to deal with the credit assignment problem, shaped reinforcement in the form of heterogeneous reward functions and progress estimators were used. Her approach is demonstrated on two multi-robot learning experiments.

Sen & Sekaran (1998) focus on reinforcement learning coordination resulted from individual and concurrent learning by multiple, autonomous, and non-communicating agents. They show that an uniform RL algorithm suffices as

a coordination mechanism in both cooperative and non-cooperative domains, using immediate as well as delayed feedback. They demonstrate that agents can develop effective policies to coordinate their actions without explicit information sharing. A particular limitation identified in the used RL schemes is the inability of independent learning to develop effective global coordination when agent actions are strongly coupled, feedback is delayed, and there is one or few optimal behavior combinations.

A significant percentage of the research in evolving global coordination in multi-agent scenarios have concentrated on cooperative learning between communicating agents where agents share their knowledge or experiences. In our solution we adopted a centralized approach, where a specialized learning agent is added to the agent society, and it is responsible for the action control.

6 Experimental Description and Results

Since RL requires a large amount of learning trials, which is prohibitive to be executed in physical robots, the approach was tested in a simulated domain. We adopted a discrete workspace where each cell in this grid can have one state of the following configuration set $S_i = \{*, P, T, PT\}$, where:

- S_i is the set of possible states of the *i-th* cell;
- $*$ represents an empty cell;
- P represents a cell with a piece ready to be packed;
- T represents a cell with trash (unwanted objects, loose edges, etc.);
- PT represents a cell with a piece that should be cleaned before packing (piece and trash).

In this system, the actions that can be executed are: (i) $moveTo(X,Y)$ - place the manipulator over the cell at position X, Y, (ii) $assemble$ - pick up and pack the piece at the current position and (iii) $clean$ - clean the current cell. The learning agent performing the Q learning builds a $Q - table(s, a)$ recording the reward received upon performing action a in the state s, plus the discounted value of following an action policy thereafter. The size of the table is:

$$Q\_tableSize = N \times C^N \times N_a, \text{ where:}$$

- N is the number of cells;
- C is the number of possible configurations of each cell;
- N_a is the number of possible actions.

Experiments were performed considering different numbers of workspace cells, learning successfully an optimal action policy in each experiment under the assembly task domain. The goal of the task is reached when there is no more piece left to be assembled. Increasing cell numbers require an increasing iteration time in the learning algorithms.

In order to illustrate the results we present a simple example where a two cell workspace ($first$ and $last$) is considered. In this example, the possible actions

to be performed can be simplified to *move-to-first*, *move-to-last*, *assemble* and *clean*. We use this example as it is the smallest possible configuration, producing a Q-table of 128 entries ($N = 2$, $C = 4$, $N_a = 4$).

On each time step the agent chooses one action from the set of four actions to be performed. We encourage exploration at the beginning of the learning process (20% of actions) and then this rate is decreased over time (rate of 0.99). The rewards received depend on the perceived configuration of the world and the executed action. The learner is penalized every time it chooses: *(i)* a *moving* action to be performed and the manipulator is over a P or PT cell; *(ii)* a *cleaning* action and it is over a P or an empty cell; *(iii)* an *assembly* action and the current cell configuration is T, PT or empty. On the other hand, the learner is rewarded every time it chooses: *(i)* a *moving* action that brings the manipulator over a P or PT cell; *(ii)* a *cleaning* action and it is over a PT cell; *(iii)* an *assembly* action and the current cell configuration is P.

Table 1 presents the results of this learning for three states: in the first state ($[P, T]$) the first cell contains a piece and the last contains an unwanted object; the second state ($[PT, T]$) the first cell contains a piece and some trash and the last contains an unwanted object; and in the last state ($[T, P]$) the first cell contains unwanted object and the last contains an a piece. In all these states the manipulator is over the first cell.

Table 1. Results of the Q Learning for three different states where the manipulator is over the first cell.

Action	$[P, T]$	$[PT, T]$	$[T, P]$
assemble	1.00	0.70	0.78
clean	0.89	1.90	0.79
move to first	– 0.10	0.69	0.79
move to last	0.81	1.49	0.90

We can see in the table that the Q learning produced a result where the appropriate action is the one with higher value in the Q-table, i.e., *assemble* in the first case, *clean* in the second one and *move-to-last* in the third case. It can also be noticed that *move-to-first* is the action with lower value in the first and second case, since the manipulator is already in that position. In this example the Q learning algorithm took less than 20000 iterations to converge to the optimal solution.

7 Conclusion and Future Work

From the experiments carried out we conclude that the use of a control agent based on a Reinforcement Learning approach in the L-VIBRA architecture al-

lowed a dynamic control structure for the coordination of agent actions in a MAS.

The results obtained showed that the system was able to minimize the task execution time in several configurations under acceptable learning times. However, the size of the search space needed grows exponentially with the number of cells. To cope with real world problems an alternative to the lookup tables is the use of some compact form to represent the value function. It is widely known that a Multilayer Perceptron Neural Network, trained with backpropagation, is an universal function approximator and it seems to be a good solution for this problem.

Finally, future works include the distribution of the learning and the control processes among agents. We intend to study the use of Markov games as a framework for multi-agent reinforcement learning.

8 Acknowledgements

The authors would like to thank Professor Manuela Veloso for her suggestions and comments. This work is part of the project MAPPEL ProTeM-CC CNPq-NFS and FINEP RECOPE project under grant no. 77970937.00. Anna H. Reali Costa has been partially supported by FAPESP under grant n. 98/06417-9.

References

Bond, A. H.; Gasser, L. (1988). *Readings in Distributed AI.* Morgan Kaufmann, San Mateo, CA, 1988.

Boutilier, C.; Brafman, R. I. (1997). Planning with Concurrent Interacting Actions. In *Proceedings of the 14th National Conference on Artificial Intelligence.* AAAI Press.

Decker, K.; Lesser, V. (1995). Designing a Family of Coordination Algorithms. UMASS Computer Science Technical Report 94-14. University of Massachussets.

Goldman, C. V.; Rosenschein, J. S. (1994). Emergent coordination through the use of cooperative state-changing rules. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, Seattle, Washington, pp.408-413. AAAI Press.

Kaelbling, L. P.; Littman, M. L.; Moore, A. W. (1996). Reinforcement Learning: A Survey. *Journal of Artificial Intelligence Research 4*, pp. 237-285.

Maes, P. and Brooks, R. A. (1991). Learning to Coordinate behaviors. In *Proceedings of the National Conference on Artificial Intelligence*, Boston, MA, pp.796-802. AAAI Press.

Matarić, M. J. (1997). Using communication to reduce locality in distributed multi-agent learning. *Autonomous Robots*, n. 4, vol. 1, pp. 73-83.

Matarić, M. J. (1998). Reinforcement Learning in the Multi-Robot Domain. *Journal of Experimental & Theoretical Artificial Intelligence*,n. 10, vol. 3, pp. 357-369.

Mitchell, T. M. (1997). *Machine Learning.* WCB/McGraw-Hill, Boston.

Reali-Costa, A. H.; Barros, L. N.; Bianchi, R. A. C. (1998). Integrating Purposive Vision with Deliberative and Reactive Planning: An Engineering Support on Robotics Applications. *Journal of the Brazilian Computer Society - Special Issue on Robotics.* Sociedade Brasileira de Computação, n. 3, vol. 4, Campinas, April 1998. pp. 52-60.

Sen, S.; Sekaran, M. (1998). Individual learning of coordination knowledge. *Journal of Experimental & Theoretical Artificial Intelligence*, n. 10, vol. 3, pp. 333-356.