Comparing Distributed Reinforcement Learning approaches to learn agent coordination

Reinaldo A. C. Bianchi and Anna H. R. Costa

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.
{reinaldo.bianchi, anna.reali}@poli.usp.br
http://www.lti.pcs.usp.br/

Abstract. This work compares the performance of the Ant-ViBRA system to approaches based on Distributed Q-learning and Q-learning, when they are applied to learn coordination among agent actions in a Multi Agent System. Ant-ViBRA is a modified version of a Swarm Intelligence Algorithm called the Ant Colony System algorithm (ACS), which combines a Reinforcement Learning (RL) approach with Heuristic Search. Ant-ViBRA uses *a priori* domain knowledge to decompose the domain task into subtasks and to define the relationship between actions and states based on interactions among subtasks. In this way, Ant-ViBRA is able to cope with planning when several agents are involved in a combinatorial optimization problem where interleaved execution is needed. The domain in which the comparison is made is that of a manipulator performing visually-guided *pick-and-place* tasks in an assembly cell. The experiments carried out are encouraging, showing that Ant-ViBRA presents better results than the Distributed Q-learning and the Q-learning algorithms.

1 Introduction

Based on the social insect metaphor for solving problems, the use of Swarm Intelligence for solving several kinds of problems has attracted an increasing attention of the AI community [2, 3]. It is an approach that studies the emergence of collective intelligence in groups of simple agents, and emphasizes the flexibility, robustness, distributedness, autonomy and direct or indirect interactions among agents.

The most common Swarm Methods are based on the observation of ant colonies behavior. In these methods, a set of simple agents, called ants, cooperate to find good solutions to combinatorial optimization problems. As a promising way of designing intelligent systems, researchers are applying this technique to solve problems such as: communication networks, combinatorial optimization, robotics, on-line learning to achieve robot coordination, adaptive task allocation and data clustering.

One possible manner of coordinating agent actions in a Multi Agent System is using the Ant-ViBRA system [1]. It is a Swarm Algorithm that combines the

Reinforcement Learning (RL) approach with Heuristic Search to solve combinatorial optimization problems where interleaved actions need to be performed. The Ant-ViBRA is a modification of a well known Swarm Algorithm – the Ant Colony System (ACS) Algorithm [6] – which was adapted to cope with planning when several agents are involved.

In order to better evaluate the results of the Ant-ViBRA system, this work compares the system performance with the ones obtained using a Distributed Q-learning (DQL) algorithm proposed by [7] and the Q-learning algorithm [8].

The domain in which this comparison is made is that of a manipulator performing visually guided *pick-and-place* tasks in an assembly cell. The system optimization goal is to minimize the execution time by reducing the number of movements made by the robotic manipulator, adapting in a timely fashion to new domain configurations.

The remainder of this paper is organized as follows. Section 2 describes the pick-and-place task domain used in this work. Section 3, 4, and 5 present the Q-Learning, the Distributed Q-Learning, and the Ant-ViBRA algorithm, respectively. 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 pick-and-place domain can be characterized as a complex planning task, where agents have to generate and execute plans, to coordinate its activities to achieve a common goal, and to perform online resource allocation. The difficulty in the execution of the pick-and-place 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 work [4, 5] in a flexible assembly cell.

In the pick-and-place 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 presents during their manufacturing process. To clean a piece means to remove these unwanted edges or other objects that obstruct packing. In this way, there is no need to clean all pieces before packing them, but only the ones that will be packed and are not clean. In this work, pieces to be packed (and eventually cleaned) are named tenons and the desired place to pack (and eventually clean) are called mortises.

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 pick-and-place domain is a typical case of a task that can be decomposed into a set of independent tasks: packing (if a tenon on the table is clean, pick it up with the manipulator and put it on a free mortise); cleaning (if a tenon or mortise have sharp edges, clean it before packing) and collision avoidance.

One of the problems to be solved when a task is decomposed into several tasks is how to coordinate the task execution. Since collision avoidance is an extremely reactive task, its precedence over cleaning and assembly tasks is preserved. This way, only interactions among packing and cleaning are considered. The packing subtask is performed by a sequence of two actions – *Pick-Up* followed by *Put-Down* – and the cleaning subtask applies the action *Clean*. Actions and relations among them are:

- *Pick-Up*: to pick up a tenon. After this operation only the *Put-Down* operation can be used.
- *Put-Down*: to put down a tenon over a free mortise. In the domain, the manipulator never puts down a piece in a place that is not a free mortise. After this operation both *Pick-Up* and *Clean* can be used.
- *Clean*: to clean a tenon or a mortise, removing unwanted material to the trash can and maintaining the manipulator stopped over it. After this operation both *Pick-Up* and *Clean* can be used.

The coordination problem of the task execution, given a certain table configuration in the pick-and-place domain, becomes a routing problem that can be modeled as a combinatorial TSP Problem, since the goal here consists in minimize the total amount of displacement performed by the manipulator during the task execution. We have used three different reinforcement learning algorithms to learn the agent coordination policy that minimizes the routing problem for a given table configuration: Q-Learning, DQL and Ant-ViBRA. These algorithms are discussed in the next sections.

3 Q-Learning

Reinforcement Learning (RL) algorithms have been applied successfully to the on-line learning of optimal control policies in Markov Decision Processes (MDPs). In RL, learning is carried out on-line, through trial-and-error interactions of the agent with the environment. 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 task of an RL agent is to learn a policy $\pi : S \rightarrow A$ that maps the current state s into the desirable action a to be performed in s. One strategy to learn the optimal policy π^* is to allow the agent to learn the evaluation function $Q : S \times A \rightarrow \mathcal{R}$. Each $Q(s, a)$ value represents the expected cost incurred by the agent when taking action a at state s and following an optimal policy thereafter.

The Q-learning algorithm [8] iteratively approximates Q, provided the system can be modeled as an MDP, the reinforcement function is bounded, and actions are chosen so that every state-action pair is visited an infinite number of times. The Q learning rule is:

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r(s,a) + \gamma \max_{a'} Q(s',a') - Q(s,a)] \qquad (1)$$

where: s is the current state, a is the action performed in s, $r(s,a)$ is the reinforcement received after performing a in s, s' is the new state, γ is a discount factor ($0 \leq \gamma < 1$), and α is the learning rate ($\alpha > 0$).

4 Distributed Q-Learning

A recent Distributed Reinforcement Learning algorithm is the Distributed Q-learning algorithm (DQL), proposed by Mariano and Morales [7]. It is a generalization of the traditional Q-learning algorithm described in the previous section where, instead of a single agent, several independent agents are used to learn a single policy.

In DQL, all the agents have a temporary copy of the state-action pair evaluation functions of the environment, which is used to decide which action to perform and that are updated according to the Q-Learning update rule.

These agents explore different options in a common environment and when all agents have completed a solution, their solutions are evaluated and the best one receives a reward. The DQL algorithm is presented in table 1.

Table 1. The general DQL algorithm [7].

```
Initialize $Q(s,a)$ arbitrarily
Repeat (for $n$ episodes)
    Repeat (for $m$ agents)
        Initialize $s$, copy $Q(s,a)$ to $Qc^m(s,a)$
        Repeat (for each step of the episode)
            Take action $a$, observe $r$, $s'$
            Update $Qc^m(s,a) \leftarrow Qc^m(s,a) + \alpha[\gamma \max_{a'} Qc^m(s',a') - Qc^m(s,a)]$
            $s \leftarrow s'$
        Until $s$ is terminal
    Evaluate the $m$ solutions
    Assign reward to the best solution found
    Update $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$
```

5 Ant-ViBRA

Ant-ViBRA is a modification of the ACS algorithm in order to cope with different sub-tasks, and to plan the route that minimizes the total amount of displacement performed by the manipulator during its movements to execute the pick-and-place task.

5.1 The ACS Algorithm

The ACS Algorithm is a Swarm Intelligence algorithm proposed by Dorigo and Gambardella [6] for combinatorial optimization based on the observation of ant colonies behavior. It has been applied to various combinatorial optimization problems like the symmetric and asymmetric traveling salesman problems (TSP and ATSP respectively), and the quadratic assignment problem. The ACS can be interpreted as a particular kind of distributed RL technique, in particular a distributed approach applied to Q-learning [8]. In the remaining of this section TSP is used to describe the algorithm.

The ACS represents the usefullness of moving to the city s when in city r in the $\tau(r,s)$, called *pheromone*, which is a positive real value associated to the edge (r,s) in a graph. It is the ACS counterpart of Q-learning Q-values. There is also a *heuristic* $\eta(r,s)$ associated to edge (r,s). It represents an heuristic evaluation of which moves are better. In the TSP $\eta(r,s)$ is the inverse of the distance δ from r to s, $\delta(r,s)$.

An agent k positioned in the city r moves to city s using the following rule, called state transition rule [6]:

$$s = \begin{cases} \arg\max\limits_{u \in J_k(r)} \tau(r,u) \cdot \eta(r,u)^{\beta} & if\ q \leq q_0 \\ S & otherwise \end{cases} \tag{2}$$

where:

- β is a parameter which weighs the relative importance of the learned pheromone and the heuristic distance values ($\beta > 0$).
- $J_k(r)$ is the list of cities still to be visited by the ant k, where r is the current city. This list is used to constrain agents to visit cities only once.
- q is a value chosen randomly with uniform probability in [0,1] and q_0 ($0 \leq q_0 \leq 1$) is a parameter that defines the exploitation/exploration rate: the higher q_0 the smaller the probability to make a random choice.
- S is a random variable selected according to a probability distribution given by:

$$p_k(r,s) = \begin{cases} \dfrac{[\tau(r,u)] \cdot [\eta(r,u)]^{\beta}}{\sum\limits_{u \in J_k(r)} [\tau(r,u)] \cdot [\eta(r,u)]^{\beta}} & if\ s \in J_k(r) \\ 0 & otherwise \end{cases} \tag{3}$$

This transition rule is meant to favor transition using edges with a large amount of pheromone and which are short.

Ants in ACS update the values of $\tau(r,s)$ in two situations: the local update step (applied when ants visit edges) and the global update step (applied when ants complete the tour).

The ACS local updating rule is:

$$\tau(r,s) \leftarrow (1-\rho) \cdot \tau(r,s) + \rho \cdot \Delta\tau(r,s) \tag{4}$$

where: $0 < \rho < 1$ is a parameter (the learning step), and $\Delta\tau(r,s) = \gamma \cdot \max_{z \in J_k(s)} \tau(s,z)$.

The ACS global update rule is:

$$\tau(r,s) \leftarrow (1-\alpha) \cdot \tau(r,s) + \alpha \cdot \Delta\tau(r,s) \tag{5}$$

where: α is the pheromone decay parameter (similar to the discount factor in Q-Learning), and $\Delta\tau(r,s)$ is a delayed reinforcement, usually the inverse of the length of the best tour. The delayed reinforcement is given only to the tour done by the best agent – only the edges belonging to the best tour will receive more pheromones (reinforcement).

The pheromone updating formulas intends to place a greater amount of pheromone on the shortest tours, achieving this by simulating the addition of new pheromone deposited by ants and evaporation. The ACS algorithm is presented in table 2.

Table 2. The ACS algorithm (in the TSP Problem).

```
Initialize the pheromone table, the ants and the list of cities.
Repeat (for $n$ episodes) /* an Ant Colony iteration */
    Repeat (for $m$ ants) /* an ant iteration */
        Put each ant at a starting city.
        Repeat (for each step of the episode)
            Chose next city using equation (2).
            Update list $J_k$ of yet to be visited cities for ant $k$.
            Apply local update to pheromones using equation (4).
        Until (ants have a complete tour).
    Apply global pheromone update using equation (5).
```

5.2 The Ant-ViBRA algorithm

To be able to cope with a combinatorial optimization problem where interleaved execution is needed, the ACS algorithm was modified by introducing: (i) several pheromone tables, one for each operation that the system can perform, and; (ii) an extended $J_k(s,a)$ list, corresponding to the pair state/action that can be applied in the next transition.

A priori domain knowledge is intensively used in order to decompose the pick-and-place problem into subtasks, and to define possible interactions among subtasks. Subtasks are related to pick-and-place actions (*Pick-Up*, *Put-Down* and *Clean* – see section 2), which can only be applied to different (disjunct) sets of states of the domain.

The use of knowledge about the conditions under which every action can be applied reduces the learning time, since it makes explicit which part of the state space must be analyzed before performing a state transition.

In Ant-ViBRA the pheromone value space is decomposed into three subspaces, each one related to an action, reducing the search space. The pheromone space is discretized in "actual position" (of the manipulator) and "next position" for each action. The assembly workspace configuration perceived by the vision system defines the position of all objects and also the dimensions of the pheromone tables.

The pheromone table corresponding to the *Pick-Up* action has entries "actual position" corresponding to the position of the trash can and of all the mortises, and entries "next position" corresponding to the position of all tenons. This means that to perform a pick-up, the manipulator is initially over a mortise (or the trash can) and will pick up a tenon in another place of the workspace.

In a similar way, the pheromone table corresponding to the *Put-Down* action has entries "actual position" corresponding to the position of the tenons and entries "next position" corresponding to the position of all the mortises. The pheromone table corresponding to the *Clean* action has entries "actual position" corresponding to the position of the trash can and of all the mortises, and entries "next position" corresponding to the position of all tenons and all mortises.

The $J_k(s,a)$ list is an extension of the $J_k(r)$ list described in the ACS. The difference is that the ACS $J_k(r)$ list was used to record the cities to be visited, assuming that the only action possible was to move from city r to one of the cities in the list.

To be able to deal with several actions, the $J_k(s,a)$ list records pairs $(state/actions)$, which represent possible actions to be performed at each state. The Ant-ViBRA algorithm introduces the following modifications to the ACS algorithm:

- Initialization takes care of several pheromone tables, the ants and the $J_k(s,a)$ list of possible actions to be performed at every state.
- Instead of directly choosing the next state by using the state transition rule (equation 2), the next state is chosen among the possible operations, using the $J_k(s,a)$ list and equation (2).
- The local update is applied to pheromone table of the executed operation.
- When cleaning operations are performed the computation of the distance δ takes into account the distance from the actual position of the manipulator to the tenon or mortise to be cleaned, added by the distance to the trash can.
- At each iteration the list $J_K(s,a)$ is updated, pairs of $(state/actions)$ already performed are removed, and new possible pairs $(state/actions)$ are added.

The next section presents experiments of the implemented system, and results where the performance of Ant-ViBRA, DQL and Q-learning are compared.

6 Experimental Description and Results

Ant-ViBRA was tested in a simulated domain, which is represented by a discrete 10x10 workspace where each cell in this grid presents one of the following six configurations: one tenon, one mortise, only trash, one tenon with trash on it, one mortise with trash on it, one tenon packed on one mortise, or a free cell.

Experiments were performed considering different configurations of the workspace, learning successfully action policies in each experiment under the assembly task domain. In order to illustrate the results we present three examples. In all of them, the goal is to find a sequence in which assembly actions should be performed in order to minimize the distance traveled by the manipulator grip during the execution of the assembly task. One iteration finishes when there is no more piece left to be packed, and the learning process stops when the result becomes stable or a maximum number of iterations is reached. All three algorithms implemented *a priori* domain knowledge about the position of the pieces in order to reduce the state space representation, reducing the search space.

In the first example (figure 1) there are initially 4 pieces and 4 tenons on the border of a 10x10 grid. Since there is no trash, the operations that can be performed are to pick up a tenon or put it down over a mortise. The initial (and final) position of the manipulator is over the tenon located at (1,1).

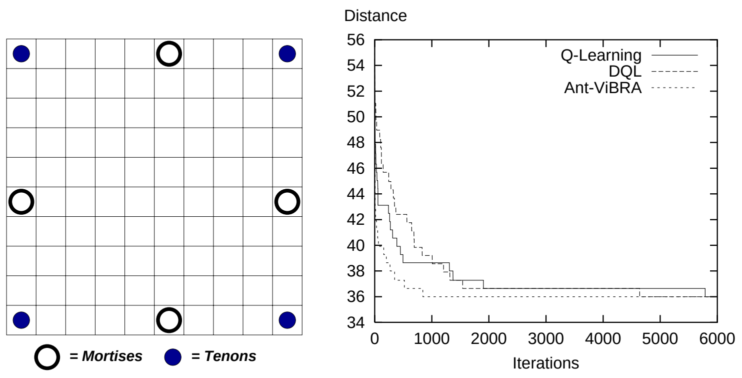

Fig. 1. Configuration of example 1 (left) and its results (right).

In this example, the average of 25 episodes of the Ant-ViBRA algorithm took 844 iterations to converge to the optimal solution, which is 36 (the total distance between pieces and tenons). The same problem took 4641 steps in average to achieve the same result using the Distributed Q-learning and 5787 steps using the Q-learning algorithm. This shows that the combination of both reinforcement learning and heuristics yields good results.

The second example (figure 2) is similar to the first one, but now there are 8 tenons and 8 mortises spread in a random disposition on the grid. The initial position of the manipulator is over the tenon located at (10,1). The result (see figure 2-right) is also better than that performed by both the DQL and the Q-learning algorithm.

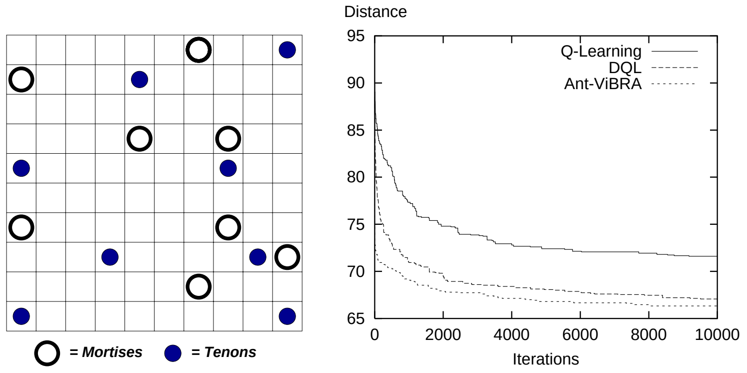

Fig. 2. Configuration of example 2 (left) and its results (right).

Finally, example 3 (figure 3) presents a configuration where the system must clean some pieces before performing the packing task. The tenons and mortises are on the same position as example 1, but there are trashes that must be removed over the tenon in the position (1, 10) and over the mortise (6, 1). The initial position of the manipulator is over the tenon located at (1,1). The operations are pick up, put down and clean. The clean action moves the manipulator over the position to be cleaned, picks the undesired object and puts it on the trash can, located at position (1, 11). Again, we can see in the result shown in figure 4-right that the Ant-ViBRA presents better results.

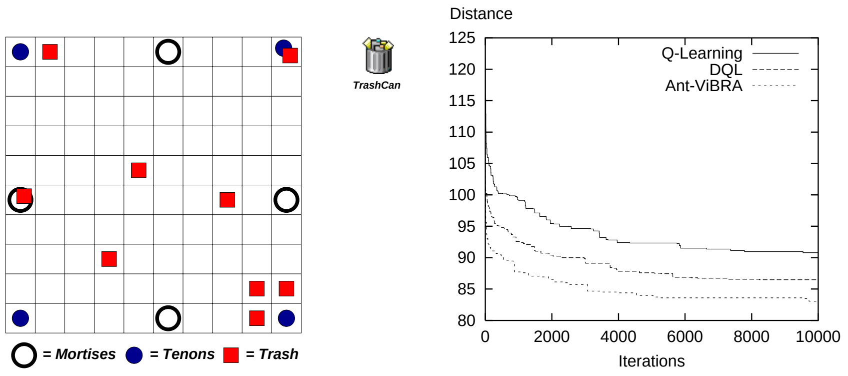

Fig. 3. Configuration of example 3 (left) and its results (right).

In the 3 examples above the parameters used were the same: exploitation/exploration rate is 0.9, the discount factor γ is set at 0.3, the maximum number of iterations allowed was set to 10000 and the results are the average of

25 episodes. For both the Q-learning and the DQL, the learning rate α is set at 0.1. In a similar way, the learning step ρ is set at 0.1 for the Ant-ViBRA. The experiments were implemented on a AMD K6-II-500MHz, with 256 MB RAM memory, using Linux and GNU gcc.

7 Conclusion

The experiments carried out show that the Ant-ViBRA algorithm was able to minimize the task execution time (or the total distance traveled by the manipulator) in several configurations of the pick-and-place workspace. Besides that, the learning time of the Ant-ViBRA was also small when compared to the Distributed Q Learning and Q-Learning techniques.

Future works include the implementation of an extension of the Ant-ViBRA algorithm in a system to control teams of mobile robots performing foraging tasks, and the exploration of new forms of composing the experience of each agent to update the pheromone table after each iteration.

Acknowledgements. This research was conducted under the NSF/CNPq-ProTeM CC Project MAPPEL (grant no. 68003399-8) and FAPESP Project AACROM (grant no. 2001/14588-2).

References

[1] R. A. C. Bianchi and A. H. R. Costa. Ant-vibra: a swarm intelligence approach to learn task coordination. *Lecture Notes in Artificial Intelligence*, 2002.

[2] E. Bonabeau, M. Dorigo, and G. Theraulaz. *Swarm Intelligence: From Natural to Artificial Systems.* Oxford University Press, New York, 1999.

[3] E. Bonabeau, M. Dorigo, and G. Theraulaz. Inspiration for optimization from social insect behaviour. *Nature 406 [6791]*, 2000.

[4] A. H. R. Costa, L. N. Barros, and R. A. C. Bianchi. Integrating purposive vision with deliberative and reactive planning: An engineering support on robotics applications. *Journal of the Brazilian Computer Society*, 4(3):52–60, April 1998.

[5] A. H. R. Costa and R. A. C. Bianchi. L-vibra: Learning in the vibra architecture. *Lecture Notes in Artificial Intelligence*, 1952:280–289, 2000.

[6] M. Dorigo and L. M. Gambardella. Ant colony system: A cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation*, 1(1), 1997.

[7] C. Mariano and E. Morales. A new distributed reinforcement learning algorithm for multiple objective optimization problems. *Lecture Notes in Artificial Intelligence*, 1952:290–299, 2000.

[8] C. J. C. H. Watkins. *Learning from Delayed Rewards.* PhD Thesis, University of Cambridge, 1989.