ANT-ViBRA: Using a Swarm algorithm to coordinate assembly tasks in a MAS

Reinaldo A. C. Bianchi and Anna H. Reali-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.
{rbianchi@usp.br, anna.reali@poli.usp.br
http://www.lti.pcs.usp.br/

Abstract. The purpose of this work is to use a Swarm Algorithm that combines Reinforcement Learning (RL) approach with Heuristic Search to coordinate agent actions in a MAS, creating plans that minimize the execution time of an assembly, and also reducing the learning time of each new plan.

To achieve this goal, a swarm algorithm that combines Heuristic Search with RL – the Ant Colony System algorithm (ACS) – was modified to be able to cope with planning when several agents are involved. This algorithm was modified to solve a combinatorial optimization problem where interleaved execution is needed. In the ACS, the knowledge acquired during experimentation is stored in one pheromone table. The modification proposed here consists of using several pheromone tables, one for each possible operation, instead of only one table.

This modification allows the use of *a priori* domain knowledge to decompose the assembly problem in subtasks, reducing the search space and the learning time of each new plan.

The domain on which the system is applied consists of visually guided assembly tasks (pick up, put down and clean), performed by a manipulator working in an assembly cell.

Experiments were carried out in a simulated domain. Results show that the combination of RL, Heuristic Search and the use of explicit domain information about states and actions to minimize the search space used in the proposed algorithm presents better results than any of the techniques alone.

1 Introduction

In the last years the use of Swarm Intelligence for solving several kinds of problems has attracted an increasing attention [1, 2, 4] of the AI community. Being an approach that studies the emergence of collective intelligence in groups of simple agents, it emphasizes the flexibility, robustness, distributedness, autonomy and direct or indirect interactions among relatively simple agents.

As a promissing way of designing intelligent systems, researchers are applying this technique to problems such as: communication networks, combinatorial optimization, robotics, on-line learning to achieve robot coordination, adaptative task allocation and data clustering.

The purpose of this work is to use a Swarm Algorithm that combines Reinforcement Learning (RL) approach with Heuristic Search to:

- coordinate agent actions in a MAS used in an assembly domain, creating plans that minimize the execution time, reducing the number of movements made by the robotic manipulator.
- reduce the learning time of each new plan.

To be able to learn the best assembly plan in the shortest possible time a well known Swarm Algorithm – the Ant Colony System (ACS) Algorithm [6] – was modified to be able to cope with planning when several agents are involved. The ACS algorithm is based on the metaphor of ant colonies and was initially proposed to solve the TSP problem. In it, several ants are allowed to travel between the cities of a TSP problem, and the path of the ant that have the shortest lenght is reinforced. It is considered one of the faster algorithms to solve TSP problems [6] and has been sucessfully applied to several optimisation problems, such as Asymetric TSPs, Network and Vehicle Routing and Graph Colouring.

The reminder of this paper is organized as follows. Section 2 describes the assembly task domain used in the experiments. Section 3 reviews some key concepts concerning Swarm Intelligence algorithms. Section 4 presents the ACS algorithm. Section 5 describes the porposed algorithm and 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

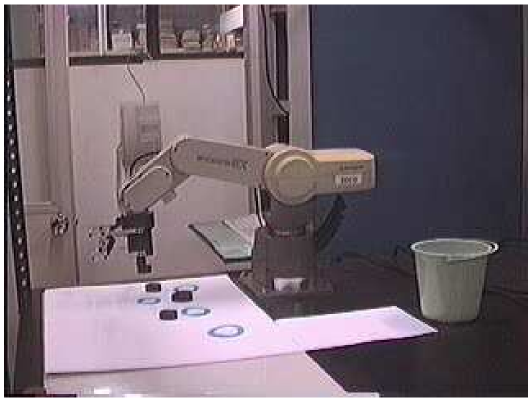

Fig. 1. One of the Assembly Cell manipulators.

been the subject of previous group work in the LSI Flexible Assembly Cell shown in Figure 1

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: 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) and the collision avoidance.

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.

One of the problems to be solved when a problem is decomposed in several tasks is how to coordinate the task allocation process in the system.

One possible solution to this problem is to use 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 inefficient performance.

This solution was adopted in the ViBRA - VIsion Based Reactive Architecture [10], which was developed in previous group work with the objective of integrating visual perception, planning, reaction and execution to solve real world problems, such as the assembly domain.

The ViBRA architecture proposes that a system 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. ViBRA is organized with authority structures and rules of behavior

However, this solution have several drawbacks, e.g., in a real application, if an unwanted object is not preventing an assembly action, it is not necessary to perform a previous cleaning action.

Another solution to the task allocation problem is to use a Reinforcement Learning Algorithm to learn the best route to do the assembly, taking into account the assembly and the cleaning tasks an thus selecting the best order in which this agents should perform their actions. This solution was adopted in the L-ViBRA [11], where a control agent that implemented the Q-Learning algorithm was introduced in the agents society.

The use of the Q-Learning algorithm in L-ViBRA resulted in a system that was able to create the optimized assembly plan needed, but that was not fast in producing these plans. As each time the workspace configuration is changed the system must learn again the assembly plan, a high performance learning algorithm is needed.

As this routing problem is similar to a combinatorial Travelling Salesman Problem, a new system – the ANT-ViBRA – was created by modifying the ACS algorithm, one of the faster algorithms used to solve TSP problems [6], and using it to plan the best route do perform the assembly.

Finally, since collision avoidance is an extremely reactive task, its precedence over cleaning and assembly tasks is preserved. The next section reviews some key concepts concerning Swarm Algorithms.

3 Swarm Intelligence

Based on the social insect metaphor for solving problems, Swarm Intelligence has become a exciting topic to researchers in the last five years. [1, 2, 4].

The most common Swarm Methods are based on the observation of ant colonies behavior. In this methods, a set of simple agents, called ants, cooperate to find good solutions to combinatorial optimization problems.

Swarm Intelligence can be viewed as a major new paradigm in control and optimization, that can be compared to the artificial neural network paradigm. "An ant colony is a 'connectionist' system, that is, one in which individual units are connected to each other according to a certain pattern"[2]. Some differences that can be noted between ANNs and Swarm Algorithms are [2]:

- the mobility of the units, which can be a mobile robot or a Softbot moving on the Internet, is an essential feature of Swarm Algorithms;

– the dynamic nature of the connectivity pattern;
– the use of feedback from the environment as a medium of co-ordination and communication;
– and the use of pheromone – ants which discover new paths leave traces which informs the other ants whether the path is a good one or not – which facilitates the design of distributed optimization systems.

Researchers are applying this Swarm Intelligence techniques in the most varied fields, from automation systems to the management of production processes. Some of them are:

– Routing problems [12]: using the Swarm Intelligence paradigm is possible to feed in artificial ants which moves in communications networks and identify congested nodes. For example, if an ant has been delayed a long time because it went through a highly congested part of the network, it will update the corresponding routing-table entries with a warning. The use of Ant Algorithms in communication networks or vehicle routing and logistics problems is now called Ant Colony Routing – ACR .
– Combinatorial optimization problems such as the Travelling Salesman Problem [6], the Quadratic Assignment Problem [9] and Graph Colouring [3]. The techniques to solve these problems were inspired by food retrieval in ants and are called Ant Colony Optimization – ACO.
– In several problems involving robotics, on-line learning to achieve robot coordination and transport [7], Adaptative Task Allocation [8].
– Data Clustering.

In the next section one of the most used Ant algorithm - the ACS - is described.

4 The Ant Colony System Algorithm

The ACS - Ant Colony System 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 [9]. In the remaining of this section the TSP is used to describe the algorithm.

The most important concept of the ACS is the $\tau(r, s)$, called *pheromone*, which is positive real value associated to the edge (r, s). It is the ACS counterpart of Q-learning Q-values, and indicates how useful it is to move to the city s when in city r. $\tau(r, s)$ values are updated at run time by the artificial ants. The pheromone acts as a memory, allowing the ants to cooperate indirectly.

Another important value is the *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 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:

$$s = \begin{cases} \arg \max\limits_{u \in J_k(r)} \tau(r,u) * \eta(r,u)^{\beta} & if\ q \leq q_0 \\ S & otherwise \end{cases} \tag{1}$$

where:

- β is a parameter which weigh 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 defined 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:

$$p_k(r,s) = \begin{cases} \dfrac{[\tau(r,u)] \cdot [\eta(r,u)]^{\beta}}{\sum\limits_{u \in J_k(r)} [\tau(r,u)] \cdot [\eta(r,u)]^{\beta}} & if\ s \in J_k(r) \\ 0 & otherwise \end{cases} \tag{2}$$

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

In order to learn the pheromone values, the ants in ACS update the values of $\tau(r,s)$ in two situations: the local update step and the global update step.

The ACS local updating rule is applied at each step of the construction of the solution, while the ants visit edges and change theis pheromone levels using the following rule:

$$\tau(r,s) \leftarrow (1-\rho) \cdot \tau(r,s) + \rho \cdot \Delta\tau(r,s) \tag{3}$$

where $0 < \rho < 1$ is a parameter, the learning step.

The term $\Delta\tau(r,s)$ can be defined in two different ways:

1. $\Delta\tau(r,s) = \gamma \cdot \max_{z \in J_k(s)} \tau(s,z)$
   Using this equation the local update rule becames similar to the Q-learning update, being composed by a reinforcement term and the discounted evaluation of the next state (with γ as the discount factor). The only difference is that the set of available actions in state s, (the set $J_k(s)$) is a function of the previous history of agent k. When the ACS uses this update it is called Ant-Q.
2. The other possibility to implement the local update is to use $\Delta\tau(r,s) = \tau_0$, where τ_0 is the initial pheromone level. The authors states that both $\Delta\tau(r,s)$ resulted in similar performance.

Once the ants have completed the tour, the pheromone level τ is updated by the following global update rule:

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

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 lenght 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 feromone on the shortest tours, achieving this by simulating the adition of new pheromone deposited by ants and evaporation.

In short, the system works as follows: after the ants are positioned in the cities, each ant builds a tour. During the construction of the tour, the local updating rule is applied modifying the pheromone level of the edges. When the ants have finished their tours, the global updating rule is applyed, modifying again the pheromone levels. This cycle is repeated until no improovement is obtained or a fixed number of iteractions were reached. The ACS algorithm is presented below. Finally, the Dorigo and Di Caro proposed a more abstraction version of this algorithm (without implementation details), called the Ant Colony Optimization – ACO – Meta Heuristic, which outlines the working model of any system that "uses a colony of artificial ants to cooperate in finding good solutions to dificult discrete optimization problems"[5].

The ACS algorithm

```
Initialize the pheromone table, the ants and the list of cities.
Loop /* an iteration */
   Position each ant in the starting node.
   Loop /* a step */
      Chose next state using equation (1).
      Update list of visited cities Jk.
      Apply local update to pheromones using equation (3).
   Until (ants have a complete tour).
Apply global pheromone update using equation (4).
Until (Final Condition is reached).
```

Common critic related to this work is that the ACS algorithm is not an MDP. In this way, it cannot be prooved that the system converges to the optimal policy.

5 The Proposed Algorithm

To be able to to cope with a combinatorial optimization problem where interleaved execution is needed, the ACS algorithm was modified by the introduction of several pheromone tables, one for each operation that the system can perform, and of several J_k lists, that are used to record completed tasks.

This modification allows the use of *a priori* domain knowledge to decompose the assembly problem in subtasks, explicitando as acoes e estados.

This modification is possible because the *a priori* domain knowledge includes: how to decompose the problem in subtasks; the relation between operation and states; and the relation between any two operations, including the order in which they can be performed. In the assembly domain the operations 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 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 certain position, removing unwanted material to the trash can and staying over it. After this operation both *Pick-Up* and *Clean* can be used.

The use of knowledge about the relations between operations and states reduces the learning time because it makes explicit which part of the search space must be analised before making a transition: instead of searching one table with all the possibilities, the modified algorithm search only the tables of the operations which are possible to be performed from the present state.

The modified algorithm is the similar to the presented in the last section, with the following changes:

- Initialization takes care of several pheromone tables, the ants and the lists of completed tasks.
- Instead of directly choosing the next state using the state transition rule (equation 1), the next state is choosen among the possible operations. This is done using a precondition check list that is not on the table. If there are several possible operations, then equation (1) is used.
- The local update is applied to pheromone table of the executed operation.
- Instead of updating the list of visited cities J_k, the list corresponding to the executed operation is updated.

For example, in an assembly task where one manipulator is used to pick and place pieces, there are 2 tables, one for picking up tenons and another for placing them over the mortises. There are also two lists, one for tenons remaining to be picked and another to free positions. In this case the two operations are mutually excludent, so the system works one step picking up and the next step placing the tenon.

A more complex system is one with assembly and cleaning tasks. In this case, at the start the system can choose between 2 operations: pick up a tenon or remove an unwanted object. Which one will be done depends on the result of the state transition rule (equation 1).

The next section presents these examples in detail and also the results of the implemented system.

6 Experimental Description and Results

The approach was tested in a simulated domain, a discrete workspace where each cell in this grid can have one piece, one tenon or pieces with trash on it.

Experiments were performed considering different numbers of workspace cells, learning successfully an optimal action policy in each experiment under the assembly task domain. In order to illustrate the results we present three examples. In all of them, the goal to find sequence in which the assembly actions must be performed in order to minimize execution the distance the manipulator grip travels to make the assembly (which is the same as the time to perform the assembly). One iteraction finishes when there is no more piece left to be assembled, and the system stops when the result does not improove for a certain number of iteractions or reaches a maximum number of iteractions.

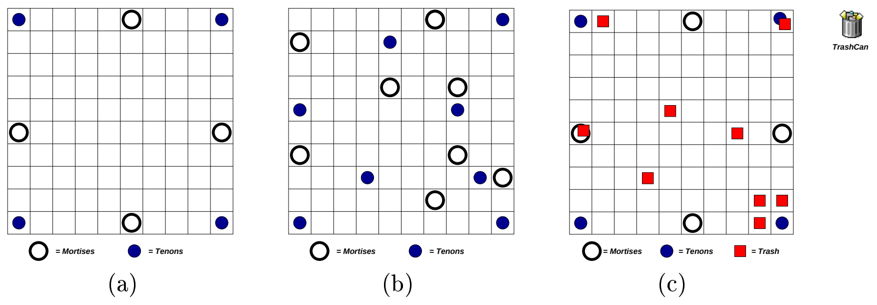

Fig. 2. Configuration of example 1 to 3 (from left to right).

In the first example (figure 2-a) there are initially 4 pieces and 4 tenons on the border of a square with side 9. As there is no thrash, the operations that can be performed are to pick up a tenon or put it down over a mortise.

The result (figure 3-a) shows that the modified ACS algorithm took less than 6 iterations to converge to the optimal solution, which is 36 (the total distance between pieces and tenons). The same problem took 332 steps to achieve the same result using the Q-learning algorithm. This shows that the combination of both reinforcement and heuristics yelds good results.

The second example (figure 2-b) is similar to the first, but now there are 8 tenons and 8 mortises spread in a random disposition on the grid. Rhe result (figure 3-b) of the Modified ACS is also the best one, compared with Q-learning.

Finally, example 3 (figure 2-c) presents a configuration where the system must clean some grids before performing the assembly. 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 (line = 1, colunm = 10) and over the mortise (6, 1). The operations are pick up, put down and clean. this last one moves the manipulator over the position to be cleaned, picks the undesired object and puts

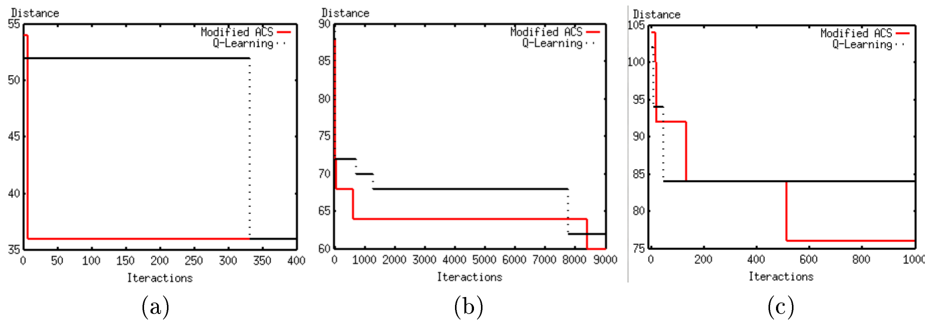

Fig. 3. Result of the Modified ACS for examples 1 to 3 (from left to right).

it on the thrashcan, located at position (1, 11). Again, we can see in the result (figure 3-c) that the modified ACS is the best result.

In the 3 examples above the parameters used were the same: the local update rule used was the Ant-Q rule; the exploitation/exploration rate is 0.9; the learning step ρ is set at 0.1; the discount factor α is 0.3; the maximum number of iteractions allowed was set to 1000 and the results were build during 15 epochs.

The system was implemented on a AMD K6-II-500MHz, with 256 MB RAM memory, using Linux and GNU gcc. The time to run each iteraction is less than 0.5 seconds for examples 1 and 3. Increasing the number of tenons require an increasing iteration time in the learning algorithms.

7 Conclusion and Future Works

The implementation of the ViBRA architecture with the Modified ACS Algorithm led to a system where agents exists at two levels: on a higher level are the autonomous agents of the ViBRA Architecture (assembler, cleaner and collision avoider) and on a lower level, the agents based on the ACS ants, which are used to optimize the work of the higher level agents.

From the experiments carried out we conclude that the combination of RL, Heuristic Search and the use of explicit domain information about states and actions to minimize the search space used in the proposed algorithm presents better results than any of the techniques alone.

The results obtained showed that the system was able to minimize the task execution time in several configurations, also minimizing the learning times when compared to other RL techniques.

Future works includes the implementation of this architecture in a Flexible Assembly Cell with a robotic manipulator and the extension of the system to control teams of mobile robots performing foraging tasks.

References

[1] Eric Bonabeau, Marco Dorigo, and Guy Theraulaz. *Swarm Intelligence: From Natural to Artificial Systems.* Oxford University Press, New York, 1999.

[2] Eric Bonabeau, Marco Dorigo, and Guy Theraulaz. Inspiration for optimization from social insect behaviour. *Nature 406 [6791]*, 2000.

[3] D Costa and A Hertz. Ants can colour graphs. *Journal of the Operational Research Society*, 48:295–305, 1997.

[4] Marco Dorigo. Ant algorithms and swarm intelligence. *Proceedings of the Seventeen International Joint Conference on Artificial Intelligence, Tutorial MP-1*, 2001.

[5] Marco Dorigo, Gianni Di Caro, and Luca Maria Gambardella. Ant algorithms for discrete optimization. *Artificial Life*, 5(3):137–172, 1999.

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

[7] C R Kube and H Zhang. Collective robotics: from social insects to robots. *Adaptive Behavior*, 2:189–218, 1994.

[8] C R Kube and H Zhang. Task modelling in collective robotics. *Autonomous Robots*, 4:53–72, 1997.

[9] V Maniezzo, Marco Dorigo, and A Colorni. Algodesk: an experimental comparison of eight evolutionary heuristics applied to the qap problem. *European Journal of Operational Research*, 81:188–204, 1995.

[10] Anna Helena Reali-Costa, Leliane Nunes Barros, and Reinaldo 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.

[11] Anna Helena Reali-Costa and Reinaldo A. C. Bianchi. L-vibra: Learning in the vibra architecture. *Lecture Notes in Artificial Intelligence*, 1952:280–289, 2000.

[12] R Schoonderwoerd, O Holland, J Bruten, and L Rothkrantz. Ant-based load balancing in telecommunications networks. *Adapt. Behav.*, 5:169–207, 1997.