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. In this work we compare the performance of the Ant-ViBRA system to aproaches based on Distributed Q-learning and Q-learning, when they are applies to learn coordination among agent actions in a Multi Agent System. Ant-ViBRA uses a Swarm Intelligence Algorithm that combines a Reinforcement Learning (RL) approach with Heuristic Search. The goal of Ant-ViBRA is to create plans that minimize the execution time of assembly tasks.
Ant-ViBRA is a modified version of a swarm algorithm called the Ant Colony System algorithm (ACS), so that it could be able to cope with planning when several agents are involved in a combinatorial optimization problem where interleaved execution is needed.
Aiming at the reduction of the learning time, Ant-ViBRA uses *a priori* domain knowledge to decompose the assembly problem into subtasks and to define the relationship between actions and states based on interactions among subtasks.
Results acquired using Ant-ViBRA are encouraging and show that the combination of RL, Heuristic Search and the use of explicit domain knowledge presents better results than the Distributed Q-learning and the Q-learning algorithms.

Keywords: Reinforcement Learning, Robotics, Multi Agent Systems and Distributed AI.
Conference Topics: Automatic Learning, Robotics, Multi Agent Systems and Distributed AI.
Submitted for PAPER TRACK

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. In this work we compare the performance of the Ant-ViBRA system to aproaches based on Distributed Q-learning and Q-learning, when they are applies to learn coordination among agent actions in a Multi Agent System. Ant-ViBRA uses a Swarm Intelligence Algorithm that combines a Reinforcement Learning (RL) approach with Heuristic Search. The goal of Ant-ViBRA is to create plans that minimize the execution time of assembly tasks.
Ant-ViBRA is a modified version of a swarm algorithm called the Ant Colony System algorithm (ACS), so that it could be able to cope with planning when several agents are involved in a combinatorial optimization problem where interleaved execution is needed.
Aiming at the reduction of the learning time, Ant-ViBRA uses *a priori* domain knowledge to decompose the assembly problem into subtasks and to define the relationship between actions and states based on interactions among subtasks.
Results acquired using Ant-ViBRA are encouraging and show that the combination of RL, Heuristic Search and the use of explicit domain knowledge 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 [1, 2]. 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.

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, adaptative task allocation and data clustering.

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.

The purpose of this work is to use a Swarm Algorithm that combines the Reinforcement Learning (RL) approach with Heuristic Search to: (i) coordinate agent actions in a Multi Agent System (MAS) used in an assembly domain, creating plans that minimize the execution time, by reducing the number of movements made by a robotic manipulator; (ii) 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 [5] – was adapted 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 Traveling Salesman Problem (TSP), where several ants are allowed to travel between cities, and the path of the ant that have the shortest length is reinforced. ACS is a combination of distributed algorithms and Q-Learning [7] (a well known RL algorithm). It is considered one of the faster algorithms to solve TSP problems [5] and has been successfully applied to several optimization problems, such as Asymmetric TSPs, Network and Vehicle Routing and Graph Coloring.

In order to better evaluate the results of the modified ACS algorithm implemented, the system performance is compared with the ones obtained using a Distributed Q-learning (DQL) algorithm proposed by [6] and the Q-learning algorithm [7].

The remainder of this paper is organized as follows. Section 2 presents the ACS algorithm and section 3 presents the DQL algorithm. Section 4 describes the assembly task domain used in the experiments. Section 5 describes the proposed approach to solve the assembly problem 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 Ant Colony System Algorithm

The ACS Algorithm is a Swarm Intelligence algorithm proposed by Dorigo and Gambardella [5] 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 reinforcement learning (RL) technique, in particular a distributed approach applied to Q-learning [7]. In the remaining of this section TSP is used to describe the algorithm.

The most important concept of the ACS is 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, 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 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 [5]:

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

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:

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

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 their 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 as: $\Delta\tau(r,s) = \gamma \cdot \max_{z \in J_k(s)} \tau(s,z)$. Using this equation the local update rule becomes similar to the Q-learning update, being composed of a reinforcement term and the discounted evaluation of the next state (with γ as a 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.

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

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

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 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.

In short, the system works as follows: after the ants are positioned in initial cities, each ant builds a tour. During the construction of the tour, the local updating rule is applied and modifies the pheromone level of the edges. When the ants have finished their tours, the global updating rule is applied, modifying again the pheromone levels. This cycle is repeated until no improvement is obtained or a fixed number of iterations were reached. The ACS algorithm is presented in table 1.

Table 1. 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 (1).
            Update list $J_k$ of yet to be visited cities for ant $k$.
            Apply local update to pheromones using equation (3).
        Until (ants have a complete tour).
    Apply global pheromone update using equation (4).
```

3 Distributed Q Learning

Another recent Distributed Reinforcement Learning algorithm is the Distributed Q-learning algorithm, proposed by Mariano and Morales [6]. It is a generalization of the traditional Q-learning algorithm proposed by Watkins [7] where, instead of a single agent, several independent agents are used to learn a single policy. 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 2.

In this work, we propose to compare the performance of the DQL to a modified version of the ACS algorithm in the assembly domain, which is described in the next section.

Table 2. The general DQL algorithm [6].

```
Initialize $Q(s, a)$ arbitrarily
Repeat (for $n$ episodes)
    Repeat (for $m$ agents)
        Initialize $s$, copy $Q(s, a)$ to $Qc(s, a)$
        Repeat (for each step of the episode)
            Take action $a$, observe $r$, $s'$
            Update $Qc(s, a)$ according to the Q-learning rule
            $s \leftarrow s'$
        Until $s$ is terminal
    Evaluate the $m$ solutions
    Assign reward to the best solution found as in the Q-learning
```

4 The Application Domain

The assembly domain can be characterized as a complex 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 work [3, 4] in the flexible assembly cell shown in Figure 1.

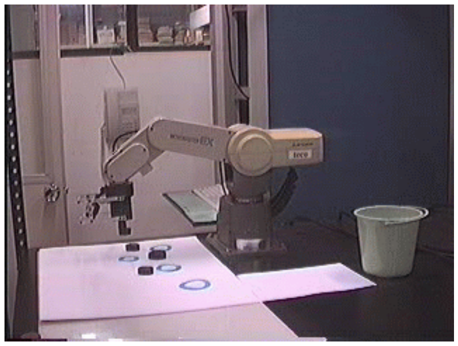

Fig. 1. One of the Assembly Cell manipulators.

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 presents during 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 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 assembly 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. In our approach, each task is assigned to an autonomous agent. One of the problems to be solved when a task is decomposed in several tasks is how to coordinate the task execution.

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 ViBRA - Vision Based Reactive Architecture [3]. However, this solution has several drawbacks, e.g., in a real application, if an unwanted object is not preventing a packing action, it is not necessary to perform a previous cleaning action, but the fixed ViBRA authority structure imposes that a cleaning action should always be executed before a packing action.

Another solution to the task allocation problem is to use a Reinforcement Learning Algorithm to learn the coordination among agents, taking into account the packing and the cleaning agents and thus selecting the best order in which this agents should perform their actions, based on the table configuration perceived by the vision system. This solution was adopted in the L-ViBRA [4], where a control agent using the Q-Learning algorithm was introduced in the agent society.

The use of the Q-Learning algorithm in L-ViBRA resulted in a system that was able to learn how to coordinate agents properly, but that was not fast enough in the learning process. Every time the workspace configuration is changed, the system must learn a new coordination procedure. This way, a high performance learning algorithm is needed.

As this routing problem can be modeled as a combinatorial TSP Problem, a new system – the Ant-ViBRA – is proposed by adapting the ACS algorithm to cope with different sub-tasks, and using it to plan the route that minimizes the total amount of displacement done by the manipulator during its movements to perform the assembly task. This approach is compared to the general DQL approach.

The next section describes the proposed adaptation of the ACS Algorithm to the assembly domain.

5 The Ant-ViBRA System

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 assembly problem into subtasks, and to define possible interactions among subtasks. Subtasks are related to assembly actions, which can only be applied to different (disjunct) sets of states of the assembly domain.

The assembly task is decomposed into three independent subtasks: packing, cleaning and collision avoidance. 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 use of knowledge about the states in which every action can be applied reduces the learning time, since it makes explicit which part of the state space must be analyzed before making a state transition.

In the 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 is similar to the ACS algorithm presented in section 2, with the following modifications:

- 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 1), the next state is chosen among the possible operations, using the $J_k(s, a)$ list and equation (1).
- 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-VBRA, 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 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 numbers of workspace cells, 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 was used in order to reduce the state pace representation, reducing the search space.

In the first example (figure 2-a) 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

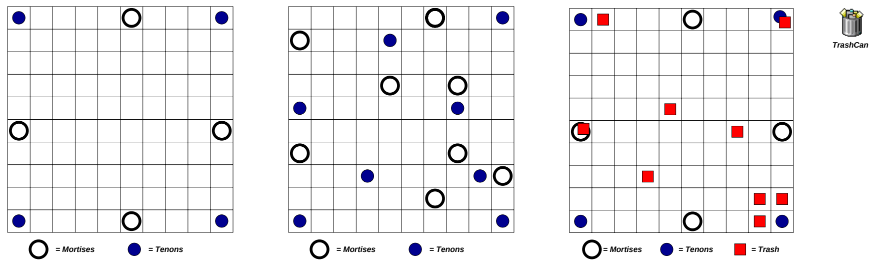

Fig. 2. Configuration of example 1 to 3 (from left to right).

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).

In this example, the average of 25 runs 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.

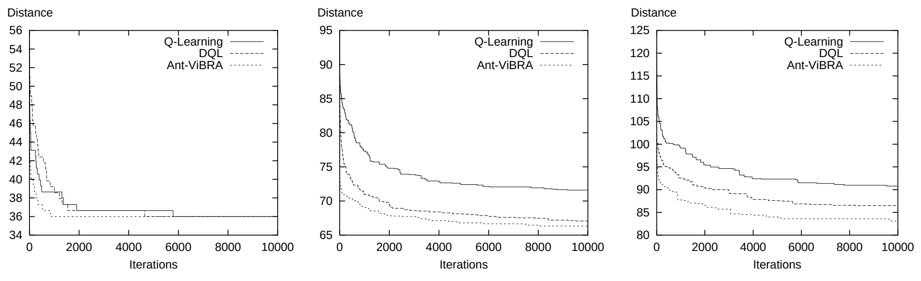

Fig. 3. Result for examples 1 to 3 (from left to right), using Q-learning, DQL and Ant-ViBRA algorithms.

The second example (figure 2-b) 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 3-b) is also better than that performed by both the DQL and the Q-learning algorithm.

Finally, example 3 (figure 2-c) 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 3-c that the Ant-ViBRA presents the best result.

In the 3 examples above the parameters used were the same: the local update rule used was the Ant-Q rule (equation 3); 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 iterations allowed was set to 10000 and the results are the average of 25 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 iteration is less than 0.5 seconds for examples 1 and 3. Increasing the number of pieces require an increasing iteration time in the learning algorithms.

7 Conclusion

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

The results obtained show that the Ant-ViBRA was able to minimize the task execution time (or the total distance traveled by the manipulator) in several configurations. Besides that, the learning time was also reduced when compared to the Distributed Q Learning and Q-Learning techniques.

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

References

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

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

[3] 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.

[4] 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.

[5] 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.

[6] 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.

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