The Necessity of Average Rewards in Cooperative Multirobot Learning

Poj Tangamchit [1] John M. Dolan [2] Pradeep K. Khosla [1]
e-mail: poj@andrew.cmu.edu jmd@cs.cmu.edu pkk@cs.cmu.edu
Dept. of Electrical and Computer Engineering [1], The Robotics Institute [2]
Carnegie Mellon University, 5000 Forbes Ave. Pittsburgh, PA 15213, USA

Abstract

Learning can be an effective way for robot systems to deal with dynamic environments and changing task conditions. However, popular single-robot learning algorithms based on discounted rewards, such as Q learning, do not achieve cooperation (i.e., purposeful division of labor) when applied to task-level multirobot systems. A task-level system is defined as one performing a mission that is decomposed into subtasks shared among robots. In this paper, we demonstrate the superiority of average-reward-based learning such as the Monte Carlo algorithm for task-level multirobot systems, and suggest an explanation for this superiority.

1. Introduction

Robot learning is the ability of robots to adjust to their environment. It can increase flexibility by enabling robots to deal with different and unexpected situations. Early research in robot learning began with one robot and one learning entity. Recent improvements in computer speed and cost have made multirobot systems a promising research topic. A key feature of multirobot systems is the potential to cooperate: several robots can help each other to accomplish a task faster or better, and they can compensate for each other's weaknesses. We define cooperation as a purposeful division of labor according to function and/or location. Cooperation generally results in higher efficiency. In this paper, we differentiate between action-level and task-level systems. Action-level systems perform missions based on reactive behaviors, whereas task-level systems perform missions at a higher level by decomposing them into subtasks shared among robots. The key result of this paper is the insight that learning techniques based on cumulative discounted rewards, such as the popular Q learning method [5], are unable to induce cooperation and therefore give suboptimal results in task-level systems, whereas learning methods based on average reward, such as the Monte Carlo algorithm, are capable of achieving the optimal result through cooperation.

2. Previous Work

Early research in multirobot learning began with artificial intelligence concepts and no physical implementation. In the last decade, there have been many real-robot learning experiments. Reinforcement learning [4] [9] has been a successful learning method for robot systems [1] [6] [9]. One of the popular reinforcement learning algorithms is Q learning by Watkins [4]. There has been a lot of successful use of Q learning on a single robot. However, there have only been a small number of learning experiments with multiple robots to date. Mataric [1] used reinforcement learning to control the behaviors of a robot group. Balch [6] performed an experiment with different types of tasks to explore the effect of diversity in robot groups. Both researchers used modified rewards (i.e. shaped reinforcement signals or progress estimators) to give feedback on progress or specific behavior. However, there were variations in performance that depended on several factors. In our work, we did not use special types of rewards to induce specific behaviors. Instead, we performed experiments using traditional rewards (rewards at the goal) to see if the robots could achieve the optimal result.

3. Approach

We investigated the behavior of a decentralized multirobot system performing puck collection, using delayed rather than instant rewards. We compared performance using local rewards to that using global rewards. We also compared performance using Q learning, which is based on cumulative discounted rewards, to that using Monte Carlo learning, which is based on average rewards. The remainder of this section justifies these choices.

3.1 Centralized vs. Decentralized Multirobot Systems

Multirobot systems can be designed based on two types of architecture: centralized and decentralized. A centralized architecture has a central unit that plans for and controls all robots in the system. A decentralized architecture does not have a central unit. Instead, each robot plans for and controls itself. Centralized systems are easy to implement, but they lack the robustness and scalability of decentralized architectures, which have recently received a lot of attention from researchers due to these properties.

Learning in centralized systems requires a single learning entity at the central unit. Because the central unit receives all data and commands all actions, learning in centralized systems is equivalent to single-robot learning. In a decentralized system, each robot needs its own learning entity, and the multiple learning entities may indirectly influence one another by rewards and robot actions. We chose a decentralized architecture because it leads to a multirobot, distributed learning problem, which is of intrinsic interest, and because it yields a more scalable and robust system.

3.2 Cooperation and Level of Robots' Hierarchy

There are six levels in a robot's hierarchy: mission, task, action, robot, joint and physical [11]. Learning can be done at any of these levels, but it is often performed at the task and action levels. The task level is where a mission is decomposed into several subtasks. This is where division of labor and planning take place. For example, in order to build a car, robots have to assemble the engine, the doors, the wheels and so on. The action level is where robots take low-level actions based on reactive behavior. If a robot system is designed with the action level as the highest level, the robots will build a car by individually assembling everything that they can without any plan. The introduction of the task level makes the robots' interaction more efficient by enabling robots to effectively share resources and duties (i.e., to cooperate). We define cooperation at the task level as a purposeful division of labor according to function and/or location.

To illustrate this difference, consider two real robot tasks: exploration and robot soccer. Both tasks can be solved by going only as high as the action level. However, greater efficiency requires the task level. In exploration, robots can be programmed to have a behavior of wandering around and sensing the environment. The more robots, the larger the area likely to be covered, although duplication of effort is also likely. Although the performance may increase, we do not classify this as cooperation because it is not a result of purposeful division of labor. Exploration can be made more efficient by dividing the area into subareas and having the robots disperse to explore those subareas. This is classified as cooperation because the robots are aware of their actions and effects on other robots by choosing to explore different subareas. Another example is robot soccer. It can be designed at the action level by simply having robots find the ball and try to kick it into the goal. However, it can be improved by introducing goalkeeping, team tactics, passing, and a dribbling mechanism. This division of labor occurs at the task level.

We are interested in the ability of learning to induce cooperation. Cooperation understood as purposeful division of labor can only occur at the task level, so we consider task-level, rather than action-level, systems in our research.

3.3 Rewards

Rewards are an important component in reinforcement learning. They are used as feedback signals that tell robots how good their actions are. Based on Dudek's Taxonomy [7], rewards can be classified into delayed vs. instant and global vs. local. At the task level, because a mission is decomposed into a series of subtasks, rewards should generally be delayed. An action from a robot may have to be combined with subsequent actions from other robots to accomplish the mission. Therefore, it will take some time for the action from the first robot to get a reward.

The other classification of rewards is local vs. global. Local rewards propagate only to the robot responsible for that action. Global rewards, on the other hand, propagate to all robots in the group. Consider an example in robot soccer. A robot scores a goal and receives a reward. If its teammates get no rewards, then the reward is local. If its teammates also get rewards because of this goal, then the reward is global. Unlike the instant vs. delayed rewards issue, in which rewards are necessarily delayed in task-level systems, we built our system to

use and compare both local and global reward schemes.

3.4 Learning Algorithms: Q learning and Monte Carlo Learning

We tested two learning algorithms on a task-level system. The first was Q learning, which is based on a cumulative discounted reward framework. The second was Monte Carlo learning, which is based on an average reward framework. Q learning is a commonly used robot learning method due to several advantages it has over others. First, it is fast and requires no world model. Second, it can handle delayed rewards. Q learning has been successfully used with single-robot and action-level multirobot systems. Q learning is designed to optimize a robot policy (π) that is based on cumulative discounted rewards (V^{π}). The cumulative discounted reward is the sum of rewards that a robot expects to receive after entering into a particular state. The discount factor (γ) makes rewards that are received in the future fade over time.

$$V^{\pi}(t) = r_t + \gamma\ r_{t+1} + \gamma^2 r_{t+2} + \ldots = \sum_{i=0}^{\infty} \gamma^i r_{t+i}$$

where $0 < \gamma < 1$

Q learning defines an evaluation function *Q(s,a)*. This function is the maximum cumulative discounted reward that can be achieved by starting from state *s* and applying action *a* as the first action. Using Q learning, robots learn and update the Q value by the following equation:

$$Q(s,a) \longleftarrow\!\!\!\!\!\!- r(s,a) + \gamma \max_{a'} Q(s',a')$$

where s' and a' are the next state and the next possible action.

The second learning algorithm tested was the Monte Carlo algorithm (MC). We studied the effect of the Monte Carlo algorithm, which is based on the average reward framework, because Q learning did not give good results on task-level systems. Research on average-reward learning has been minimal. There are few algorithms known to date. We chose the Monte Carlo algorithm because it is not complex to analyze. Monte Carlo learning was invented at the beginning of robot learning and has the advantage of the average reward framework. However, it has rarely been used because it is slow. It uses probability theory to estimate the value of actions from experience. Monte Carlo learning is used in episodic tasks. The algorithm traces the states that have been visited until the end of episode. It then gives credits to those states according to rewards that the robots receive. There are two versions of Monte Carlo learning: first-visit MC and every-visit MC. First-visit MC records average rewards after the first visit to each state. Every-visit averages all rewards after every visit to each state. The first-visit MC algorithm looks like the following.

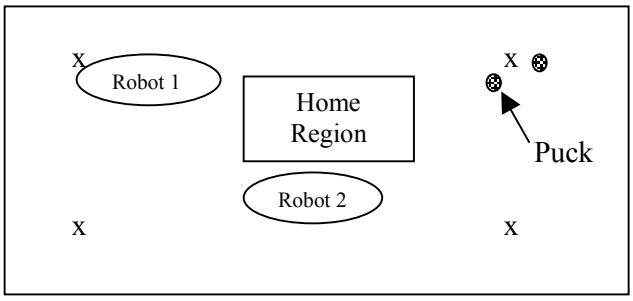
```
Q(s,a) ← arbitrary          % Q(s,a) is an average reward after the first visit in state s, action a
π(s) ← arbitrary            % π (s) is the policy and decision at state s
Rewards(s,a) ← Empty list

Repeat Forever:
    - Generate an episode using π
    - For each pair s,a appearing in the episode:
            R ← reward following the first occurrence of s,a

            Append R to Rewards(s,a)
            Q(s,a) ← average(Rewards(s,a))

    - For each s in the episode
            π(s) ← argmax_a Q(s,a)
```

4. Experiments and Results

Our test problem is the puck-collecting problem, consisting of two robots and a rectangular field. Pucks are distributed randomly at four predefined points at the corners of the field. The robots are ordered to investigate and find a puck around these points. There is a home region in the middle of the field with a bin inside. The task for the robots is to move all pucks to the home region and deposit them in the bin. Both robots can sense a puck, pick up a puck, or drop a puck. The first robot can move to and investigate around the points, or it can move to the home region and deposit a puck. The second robot is restricted to move only in the home region,

but it can still sense a puck, pick up a puck, or deposit a puck in the bin. Depositing a puck in the bin is time-consuming for the first robot, but it is easy for the second robot. Therefore, although the second robot cannot move around, it can play an important role by depositing pucks in the bin. The optimal complete sequence is that the first robot picks up a puck, comes back to the home region, and drops the puck. Then, the second robot picks up the puck, and deposits it in the bin.

The puck-collecting problem is inherently designed at the task level because it is divided into series of subtasks required from both robots. The first robot has to intentionally drop a puck at the home region in order to hand over the task to the second robot.



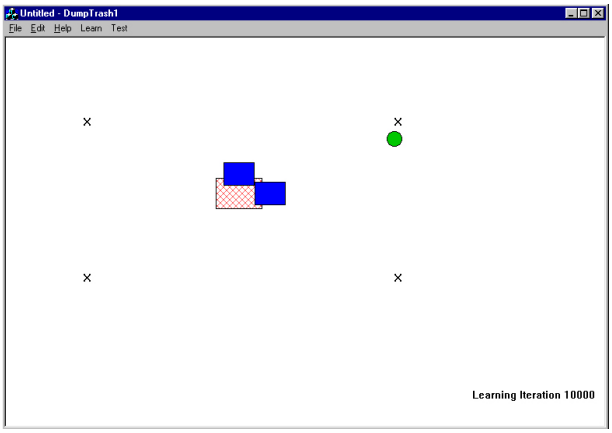
State =
{At?, HavePuck, SensePuck}

Action =
{Goto?, PickPuck, DropPuck, Store, DoNothing}

Parameter values of rewards and costs are shown in the table below. All robot actions result in negative rewards (cost) except depositing a puck, which gives a big positive reward because it is the final goal. These values are based on the relative difficulty of the actions. For example, dropping a puck is relatively easy compared to picking up a puck, which requires sensing and manipulation. These reward values can be varied within reasonable bounds without changing the result as long as the relative magnitudes are preserved (e.g., picking up a puck should not become easier than dropping one). The table below shows the reward values used in our experiment. All values of both robots are the same, except when depositing a puck. It costs Robot 1 ten times more than it costs Robot 2. This will encourage Robot 1 to drop a puck and let Robot 2 carry out the task. Depositing a puck by Robot 1 will cost 2000. If it let Robot 2 handle it, the cost will be 10(drop) + 200(deposit by Robot 2) = 210. However, if we set the cost of depositing a puck to be equal, Robot 1 will do the task all by itself. This is because the total cost will be higher if it passes the task to Robot 2 (overhead from dropping a puck).

Reward Table	Robot 1	Robot 2
Move to point	(Distance) x (-100)	(Distance) x (-100)
Pick up a puck	-100	-100
Drop a puck	-10	-10
Deposit a puck to the bin	-2000	-200
After the puck is deposited (reward)	+20000	+20000

Our simulation is shown below. It was written with Microsoft Visual C++. Circles represent pucks. Two dark rectangles represent two robots.



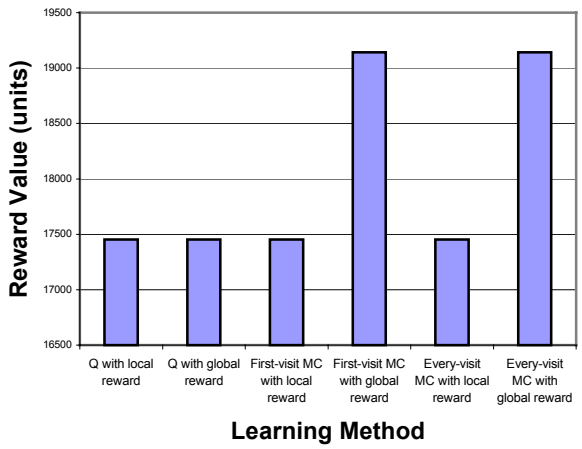
We performed experiments with both Q-learning and Monte Carlo learning on this problem. In addition, we used both global reward and local reward schemes. In all cases, the robots achieve stable results. However, there are two types of results. The first type is the optimal result described previously, in which both robots cooperate in placing a puck in the bin. The second type is a non-cooperative situation, in which the first robot does not drop the puck. Instead, it does everything by itself. The first type was only achievable by using

the Monte Carlo learning with global reward. This result supports our assumption described previously.

Learning Algorithm	Global Reward	Local Reward
Q-Learning	Not Cooperate	Not Cooperate
First-visit Monte Carlo method	Cooperate	Not Cooperate
Every-visit Monte Carlo method	Cooperate	Not Cooperate

The chart below shows a record of total rewards that Robot 1 got in each cycle. These values are non-discounted and do not include the global rewards generated by Robot 2.

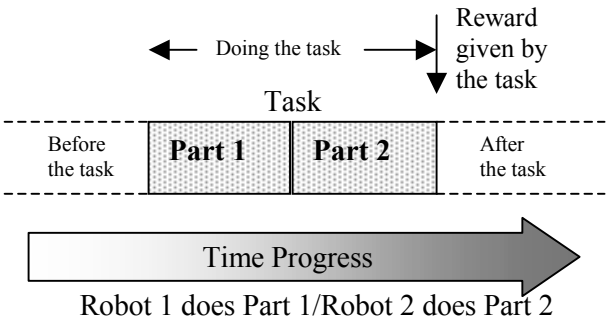
Total Local Rewards (non-discounted) of Robot 1 in each Learning Cycle (at stable point)



5. Discussion

The experimental results indicate that only Monte Carlo learning with a global reward scheme can achieve cooperation. In this paper, we claim that learning algorithms that are based on cumulative discounted rewards, such as Q learning and TD(γ), do not induce cooperation and therefore give suboptimal results in task-level systems. When there are multiple learning entities in a task-level system, they will have asynchronous learning time frames. An event that benefits the whole system usually occurs after the actions of all robots are performed, but it is often observed by only one robot. This robot will get a reward immediately. It will then take some time to propagate to other robots. Because of this delay in the cumulative discounted reward framework, the other robots will get a smaller reward for their actions. This phenomenon encourages the other robots to only choose actions that yield an immediate reward. However, cooperation requires the robots to divide their duties and do sequential actions. If all robots compete for actions that have immediate rewards, the learning space is limited and the system is unlikely to learn the best solution.

To illustrate this phenomenon, consider the example of two robots with a sequential task. The task consists of two parts in strict order. Only after the first part is finished can the second part begin. Rewards are given to the robots at the end of the second part. Both robots use a global reward scheme.



Robot 1 does Part 1/Robot 2 does Part 2
Reward = 10

Robot 1 does Part 2/Robot 2 does Part 1
Reward = 8

We assume that Robot 1 is more suited to do Part 1 than Robot 2. In the best case, Robot 1 chooses Part 1 and Robot 2 chooses Part 2, which will provide a reward of 10 units. The other case is when Robot 1 chooses Part 2 and Robot 2 chooses Part 1, which will provide a reward of 8 units. Suppose the length of Part 2 is three time-steps and the discount factor (γ) is 0.9. In the first case, Robot 1 chooses Part 1 and gets a reward three time-steps later of $10*(0.9)^3 = 7.3$. Robot 2 chooses Part 2 and gets a full 10-unit reward immediately. In the second case, Robot 2 chooses Part 1 and gets a reward three time-steps later of $8*(0.9)^3 = 5.8$. Robot 1 chooses Part 2 and gets an immediate reward of 8 units. The total reward of the first case is 7.3+10 = 17.3 and the total reward of the second case is 5.8+8 = 13.8. The

second case seems inferior, but Robot 1 gets a bigger reward (8 instead of 7.3). Therefore, using the cumulative discount reward framework, Robot 1 will learn the second case, which is a selfish behavior.

Learning algorithms that are based on an average reward framework, such as the Monte Carlo algorithm, can solve this problem. We used the Monte Carlo algorithm in our experiment due to its simplicity. With average rewards, it does not matter who gets the reward first, since the reward will not be discounted. The reward that each robot receives is the sum of all rewards divided by the number of time-steps. Therefore, all robots receive equal rewards. From the previous example, if the total number of time-steps is five, all robots receive an average reward of 10/5 = 2.0 units in the first case and 8/5 = 1.6 units in the second case.

When the robots are homogeneous with cumulative discounted learning, both robots still get a different amount of reward depending on who goes first. Consider the example described previously, but with a final reward of 10 units in both cases. Using Q learning, the robot that does Part 1 will get a reward of 7.3 units and the robot that does Part 2 will get a reward of 10 units. Since Part 2 gives a bigger reward, both robots will compete for doing Part 2. If they can wait or do some useless actions to make the other choose Part 1, they will get a bigger reward. Therefore, both robots will learn to wait and let the other go first. Again, Monte Carlo learning can solve this problem because it yields an equal reward for both robots.

Global and local reward systems are also an important factor affecting learning in task-level systems. Our experiment indicates that robots cannot learn cooperation if we use a local reward system. The reason for this result is intuitive: a robot will not help other robots if it does not get a reward for doing so. Without global reward, instead of cooperating, every robot will compete for the goal.

6. Conclusions and Future Work

We have studied different learning algorithms on a multirobot system. Our multirobot system is fully decentralized, and our learning entities are distributed and independent on each robot. Multirobot systems can be designed based on the action level or the task level. Popular non-average-reward-based learning techniques such as Q learning are effective at the action level, but not at the task level, because they do not induce cooperation, understood as the division of labor according to function and/or location. The main reason is that the values of rewards fade over time, causing all robots to prefer actions that have immediate rewards. We demonstrated that using Monte Carlo learning with a global reward scheme solves this problem and induces cooperation. Although Monte Carlo learning is simple, it is very slow, and makes weak use of training samples. In future work, we will include the implementation of Sutton's Dyna architecture [4] to speed up the learning process.

References

[1] Mataric M.J., "Interaction and intelligent Behavior", Ph.D. thesis, MIT EECS, 1994.

[2] Parker L.E., "Heterogeneous Multi-Robot Cooperation", Ph.D. thesis, MIT EECS, 1994.

[3] Tangamchit P., Dolan J.M. and Khosla P.K., "Dynamic Task Selection: A Simple Structure for Multirobot Systems", DARS 2000, pp.483-484.

[4] Sutton R.S. and Barto A.G., "Reinforcement Learning: An Introduction", MIT Press, Cambridge, MA, 1998.

[5] Watkins C.J.C.H., "Learning from Delayed Rewards", Ph.D. thesis, King's College, Cambridge, UK, 1989.

[6] Balch T., "Behavioral Diversity in Learning Robot Teams", Ph.D. thesis, Dept. of Computer Science, Georgia Tech., 1998.

[7] Dudek G., Jenkin M.R., Milios E. and Wilkes D., "A Taxonomy for Multi-Agent Robotics", Autonomous Robots 3 (4):375-397, December 1996,Kluwer Academic Publishers.

[8] Balch T., "Taxonomies of Multirobot Task and Reward", Technical Report Robotic Institute, CMU, 1998.

[9] Kaelbling L., Littman M. and Moore A., "Reinforcement Learning : A Survey", Journal of AI Research 4, pp.237-285, 1996.

[10] Schwartz A., "A reinforcement learning for maximizing undiscounted rewards", Proceedings of Tenth International Conference on Machine Learning, pp.298-305, 1993.

[11] McKerrow P.J., "Introduction to Robotics", Addison-Wesley, chapter 9 pp. 483-543, 1991.