Multiagent Coordination in Roombas: From the perspective of Reinforcement Learning and Neuroevolution

Jimmy Xin Lin
Department of Computer Science
the University of Texas at Austin
Austin, TX 78712
jimmylin@cs.utexas.edu

Barry Feigenbaum
The University of Texas at Austin
Address
email

***Abstract*—This paper presents our research about the reinforcement learning approach and the neuroevolution approach, by which the crumb collection task can be more effectively and efficiently solved with communication and coordination between multiple agents under the simulated Roombas environment. The preliminary literature part gives a brief overview about how existing works fulfill the coordination under general multiagent environments. The initial setup experimentation shows our works about the learned agents that simulate greedy strategies. The key things ... are shown in the following multiagent experiments. It is observed from our experiments that .**

I. Introduction

In the past years, Roomba Vacuum has gained its popularity in the industry of domestic services. Most of existing studies about Roomba (iRobots) is to qualitatively investigate its utility in the home as a single autonomous domestic service provider. In the contrast, our interests focus on the working efficacy of Roomba agents under a decentralized system.

The decentralized decision making has a long history, originated from the team thoery ([1]–[5]), where the decisions made by team members need to contribute to the fulfillment of global objectives. However, the individual members have only partial information about the entire system, i.e. limited knowledge of common goals and global states. This motivates the need for coordination because agents have to share resources and expertise required to achieve their goals. Researchers in the field of Distributed Artificial Intelligence (DAI) have been developing efficient mechanisms to coordinate the activities of multiple autonomous agents ([6], [7]). Specifically, previous works for the multiagent coordination include using sophisticated information to exchange protocols, investigating heuristics for negotiation, and developing formal models of possibilities of conflict and cooperation among agent interests.

Reinforcement learning has been widely used as the most useful techniques for autonomous game playing (Atari, Pacman, and Angry Bird) and intelligent task fulfillment either for single-agent or multi-agent environment.

Neuroevolution.

In this paper, we investigate various multiagent coordination techniques under the framework of reinforcemnet learning

Fig. 1. A exemplar Roomba robot in the real world.

and neuroevolution could improve the working efficacy of Roomba in the task of cleaning the floor. Our contributions include: (i) set up the raw Roomba System. (ii) implemented reinforcement learning mechanism and fixed up the default neuroevolution mechanism. (iii) design sensors and their representations for multiagent communication and coordination. (iv) compare performances of the agents learned through various approaches and settings.

The remained part of this paper is organized as follows. In section II, a detailed presentation about the existing reinforcement learning treatments and the neuroevolution treatments for multiagent systems. Section III describes in detail the virtual Roomba environment, under which our experiments proceed. Preliminary experiments that both serve as baseline intelligent agents and verify the correct system setup are indicated in the section V. The experiments that demonstrate our achievements on multiagent coordination are articulated in the section **??**. We summarize our conclusions and discuss some promising future works in the section VI. TODO: need to update!

II. Technical Literatures

This section focuses on the summary of previous works about how multiagent coordination can be incorporated in the reinforcement learning technique and the neuroevolution tehnique.

A. *Multiagent Planning as Optimization*

Distributed Constraint Optimization Problems (DCOPs) are problems where agents need to coordinate their value assignments to maximize the sum of the resulting constraint utilities.

Dynamic Distributed Constraint Optimization Problems (Dynamic DCOPs) are introduced to model dynamically changing multiagent coordination problems, where a dynamic DCOP is a sequence of static DCOPs, each of which partially different from the DCOP preceding it. The formulation of Dynamic DCOPs allow us to model problems that can not be assumed to be static. Dynamic DCOPs are especially advantageous for those problems that change so frequently that by the time a DCOP solver has found a solution it is already obsolete. A series of asynchronous algorithms have been proposed to solve the Dynamic DCOP: Abstract Distributed Constraint Optimization (DCOP) [**?**], ADOPT [**?**], Support-Based Distributed Optimisation (SBDO) [11],

B. *Reinforcement Learning Based Coordination*

Factored Markov Decision Process (Factored MDP) is a common model-based reinforcement learning formulation for cooperative multiagent dynamic systems. This framework simply views the entire multiagent system as a single, large MDP, which can be represented in a factored way using a dynamic Bayesian network (DBN) [8]. It implies that the action space of the resulting MDP is the exponential union of action space of the entire set of agents. One effective approach is to employ factored linear value functions as an approximation to the joint value function, which can be solved simply by a single linear program [8]. A large collection of algorithms are developed based on the factored MDP model. These algorithms have a parameterized, structured representation of a policy or value function, by which agents coordinate both their action selection activities and their parameter updates and then determine a jointly optimal action without explicitly considering every possible action in their exponentially large joint action space [9]. However, the factored MDP may not be applicable to the real application due to its large search space and associated complexity. To address this problem, an accelerated gradient method comes up with $\mathcal{O}(\epsilon^{-1})$ rate of convergence for solving the distributed multi-agent planning with Factored MDPs [10].

[12] This paper presents a model-free, scalable learning approach that synthesizes multi-agent reinforcement learning (MARL) and distributed constraint optimization (DCOP). By exploiting structured interaction in ND-POMDPs, our approach distributes the learning of the joint policy and employs DCOP techniques to coordinate distributed learning to ensure the global learning performance. Our approach can learn a globally optimal policy for ND-POMDPs with a property called groupwise observability. Experimental results show that, with communication during learning and execution, our approach significantly outperforms the nearly-optimal non-communication policies computed offline.

[13] Existing work typically assumes that the prob- lem in each time step is decoupled from the problems in other time steps, which might not hold in some applications. Therefore, in this paper, we make the following contributions: (i) We introduce a new model, called Markovian Dynamic DCOPs (MD-DCOPs), where the DCOP in the next time step is a function of the value assignments in the current time step; (ii) We introduce two distributed reinforcement learning algorithms, the Distributed RVI Q-learning algorithm and the Distributed R-learning algorithm, that balance exploration and exploitation to solve MD-DCOPs in an online manner; and (iii) We empirically evaluate them against an existing multi-arm bandit DCOP algorithm on dynamic DCOPs.

[14]

[15]

[16]

[17] Complex problems involving multiple agents exhibit varying degrees of cooperation. The levels of cooperation might reflect both differences in information as well as differences in goals. In this research, we develop a general mathematical model for distributed, semi-cooperative planning and suggest a solution strategy which involves decomposing the system into subproblems, each of which is specified at a certain period in time and controlled by an agent. The agents communicate marginal values of resources to each other, possibly with distortion. We design experiments to demonstrate the benefits of communication between the agents and show that, with communication, the solution quality approaches that of the ideal situation where the entire problem is controlled by a single agent.

[18]

C. *Neuroevolution for Multiagent Coordination*

Barry: Add literatures of Neuroevolution here...
ATA: barbarian

III. The Roomba Environment

To the best of our knowledge, the OpenNERO platform is the best choice for cheap software simulations. The OpenNERO is an open-source platform for artificial intelligence research and education [19]. The public release of the OpenNERO contains an underdeveloped implementation of Roomba environment, which supports the communication and coordination of multiple robots.

A. *Invariant Structures*

The Roomba environment is a virtual computer lab with crumbs distributed on the floor. In this virtual lab, there are four classes of objects: agents, crumbs, walls, decorations. Vacuum cleaner agents, shown as grey cylinders in Fig. 2, are supposed to collect the static crumbs that are labelled as blue cubes. The agents will be rewarded if they move to a place where there are some crumbs. Walls are also set up as the boundaries of the computer lab, such that agents are not allowed to move beyond the walls and no pellets can be placed outside the walls. Other decorative objects within the virtual environment include tables, chairs, and computers. For simplicity at the moment, these decorations only serve as physically transparent decorations, which means they do not block agents' movements.

As shown in the Fig. 2, the small window floating on the top right is the controller that masters the type of scripted AI agents to load in, the number of agents, and particular commands that impact the progress of the Roombas simulation (Pause/Resume, Add/Remove Robots, Exit). On each run of the simulation, only one particular type of AI agents are allowed to be loaded. Note also that the type of AI agents employed cannot be switched to the other one during the intermediate process of crumb collection task. If one needs to switch to the other type of AI agents, the running agents have to be removed and then the user is able to effectively load the desired AI agents.

In the Roombas environment, the movements of vacuum cleaner agents are not constrained by four directions (left, right, forward, backward). Instead, agents are able to move towards all directions and each moving action is denoted as a continuous radius value. Besides, all agents move in a synchronous way. Agents are allowed to make their movements of the following step if and only if all agents have completed their movements in the last step.

Three different modes (MANUAL, RANDOM, and CLUSTER) are employed to specify the placing position of each individual crumb. In the MANUAL mode, one pellet is deterministically placed at a user-specified position. In the RANDOM mode, the placement of one pellet is totally randomized; the environment will throw a rejection and repeat the random generation, if an invalid position is yielded. The last one is the CLUSTER mode, where the position of pellets are partially randomized. In this mode, environment samples the position of one pellet from a gaussian distribution whose centers and spreads at x and y coordinates are specified. In this paper, the specifications for the placement of crumbs are generated from the starting CLUSTER-mode experiment and remained invariant by employing MANUAL mode in all experiments afterwards.

The learning process designed in the Roomba environment is as follows. The physically dynamic component, consisting of robots and pellets, will be reset to their initial status once one episode expires. Except the spatial positions, all other properties of agents are allowed to preserve through two adjacent episodes, for example, the accumulated Q values from previous episodes. Note that the default configuration for one episode is 100 steps. That is, for every 100 steps, all agents will be placed back to their birth places no matter how many pellets they have collected and all crumbs are set at their specified locations, regardless of whether these crumbs exist at the end of last episode.

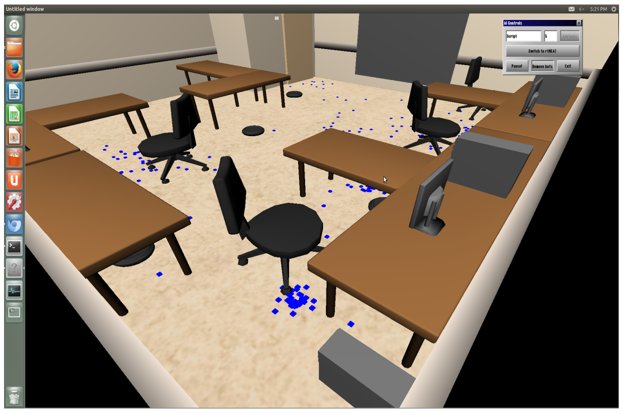

Fig. 2. Overall Picture of the Roomba Environment

B. Practical Issues

What agents are allowed to perceive from their local views is one practical issue that matters the most. This is important because both of the design of sensors and their representations have significant impact on the learning outcome of a team. The default implementation of Roomba environment allows each agent to sense all crumbs on the floor about their positions, existence status, and even rewards. In this sense, agents are able to perceive limitless information from the computer lab. In addition, the spatial information of other collaborated robots is available for each individual agent, as well as the user-defined working status of these teammates (e.g. a history of previous positions and movements). Although the Roomba environment provides a large design space for sensors, it is a practical treatment to start from a small number of simple sensors. For example, a combination of the bumping status, the position of its own, and the location of closest crumb suffice for an agent to learn a greedy strategy, as illustrated in the initial setup experimentation. A balance is struck between the amount of information available to each agent and the associated cost.

The reward design is another big issue for configuring the Roomba environment. By default, the only reward being set up for agents is when they successfully collect some pellets on the floor. In order to facilitate the learning of agents, an penalty for being alive is supposed to be incorporated to the reward system. That is, agents should receive some negative rewards, typically a very small quantity, for each step they move. Similarly, penalties can also be granted to the collisions between roombas, bumping of agents towards the world boundaries, and repetitive movements around an area.

C. Expected Multiagent Behaviors

TODO: add discussion about the Expected Multiagent Behaviors here...

Work Balance / competition avoidance.

Collision avoidance.

IV. Our Approaches

This section articulates, in a formal way, our original ideas or design priciples to advance the cleaning efficacy of the

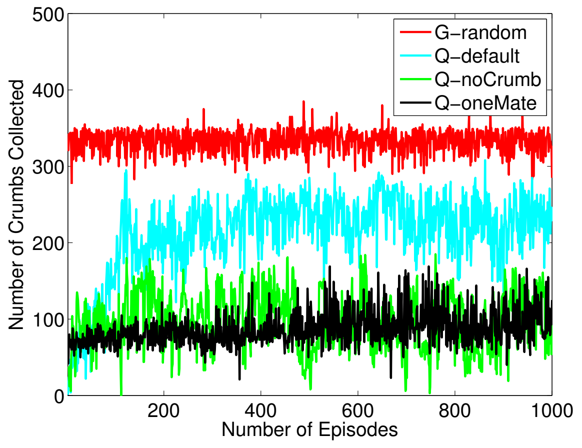

Fig. 3. Q-learning with various simple sensors. (i) Red: G-random curve represents the greedy agent with 0.1 probability to make a random decision, which serve as the benchmark agents. (ii) (iii) (iv) agents receive various collection of sensors for their decision making of each step. Q-noCrumb, Q-default, and Q-oneMate represent the learning performances of agents that receive S_{nc}, S_d, and S_{om} respectively.

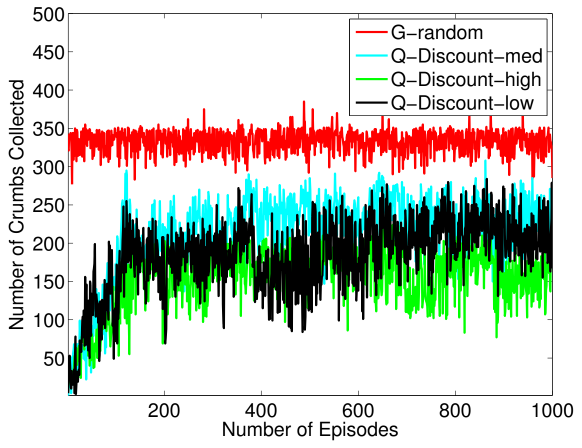

Fig. 4. Q-learning with various discounting factors. (i) Red: G-random curve represents the greedy agent with 0.1 probability to make a random decision, which serve as the benchmark agents. (ii) (iii) (iv) the tilted tabular Q-learning with discounting factors $\gamma = 0.1, 0.5, 0.8$ for Q-Discount-low, Q-Discount-med, and Q-Discount-high respectively.

Roomba environment from the multiagent perspective.

A. *Baseline Agents*

Agents with greedy strategy simply approaches to the direction where the closest pellet to it is there.

TODO: explain Random greedy

B. *Tiled Tabular Q-learning*

TODO: description of TTQ

The motives to use Tiled Tabular Q-learning include (i) verifying that we have set up the environment correctly for general reinforcement learning techniques. (ii) illustrating the benchmark (reinforcement learning) intelligence for the crumb collection problem.

C. *High-level Reinforcement Learning*

TODO: any original idea goes here.

D. *Neuroevolution*

TODO: any original idea goes here.

V. Experiments

This section presents our research investigation in two threads: the reinforcement learning thread and the neuroevolution thread.

A. *Tiled Tabular Q-learning*

Before diving into the investigation of multiagent coordination, we take preliminary experiments to investigate the impact of various collections of simple sensors and the effects of various discounting factors on the outcome of the tiled tabular Q-learning.

TODO: explain some other constans (Number of agents, pellets,)

The sensors designed for the local perspective of each agent are as follows.

$$S_{nc} = \begin{pmatrix} self.position.x \\ self.position.y \end{pmatrix} S_d = \begin{pmatrix} self.position.x \\ self.position.y \\ closestCrumb.x \\ closestCrumb.y \end{pmatrix}$$

$$S_{om} = \begin{pmatrix} self.position.x \\ self.position.y \\ closestCrumb.x \\ closestCrumb.y \\ DistToClosestMate.x \\ DistToClosestMate.y \end{pmatrix}$$

Note that we discretize all real-value sensors by using the tiling techniques mentioned in the previous section.

The learning outcomes derived from these sensors are compared in the Fig. 3. Observations tell us that the positional information of the closest crumb tremendously contributes to the quality of agents' learned policies. In addition to that, the relative position to the closest collaborated agent indeed hinder the Q-learning process, such that the communicative agents (black curve) performs worse than the team that do not monitor collaborators' positions (light-blue curve) during the observed episodes. One exciting discovery behind the figure is the narrow performance gap between the team of tiled tabular Q-learning agents and that of greedy-random agents. It turns out that the collective works made by naive Q-learning agents even outperform the team with greedy strategies.

The Fig. 4 shows the effects that various discounting factors will bring to the outcome of tilted tabular Q-learning. It turns out that the best learning outcome came from the discounting factor $\gamma = 0.5$ under the given setting. On top of that, this chart also indicates the negative effects of too large or too small discounting factors, by which worse learning outcome arises. Nevertheless, it can be concluded that the team formed by tabular Q-learning agents cannot never outperform the team whose members employ the greedy strategy, regardless of how the discounting factor is set.

B. High-level Reinforcement Learning

After the preliminary experiments, it is natural to investigate the problem of how to incorporate effective coordinations for this particular multiagent system.

C. Neuroevolution

Barry: Add implementation details and experimental results of Neuroevolution here...

VI. Conclusions

The conclusion goes here. In this paper, we investigate. Promising future Works go here.

References

[1] J. Marschak, "Elements for a theory of teams," *Management Science*, vol. 1, no. 2, pp. 127–137, 1955.

[2] R. Radner, "Team decision problems," *The Annals of Mathematical Statistics*, pp. 857–881, 1962.

[3] ——, "The application of linear programming to team decision problems," *Management Science*, vol. 5, no. 2, pp. 143–150, 1959.

[4] Y.-C. Ho *et al.*, "Team decision theory and information structures in optimal control problems–part i," *Automatic Control, IEEE Transactions on*, vol. 17, no. 1, pp. 15–22, 1972.

[5] J. N. Tsitsiklis and M. Athans, "On the complexity of decentralized decision making and detection problems," *Automatic Control, IEEE Transactions on*, vol. 30, no. 5, pp. 440–446, 1985.

[6] G. Weiss, *Multiagent systems: a modern approach to distributed artificial intelligence*. MIT press, 1999.

[7] M. N. Huhns, *Distributed artificial intelligence*. Elsevier, 2012, vol. 1.

[8] R. T. Maheswaran, M. Tambe, E. Bowring, J. P. Pearce, and P. Varakantham, "Taking dcop to the real world: Efficient complete solutions for distributed multi-event scheduling," in *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems-Volume 1*. IEEE Computer Society, 2004, pp. 310–317.

[9] P. J. Modi, W.-M. Shen, M. Tambe, and M. Yokoo, "Adopt: Asynchronous distributed constraint optimization with quality guarantees," *Artificial Intelligence*, vol. 161, no. 1, pp. 149–180, 2005.

[10] G. Billiau, C. F. Chang, and A. Ghose, "Sbdo: A new robust approach to dynamic distributed constraint optimisation," in *Principles and Practice of Multi-Agent Systems*. Springer, 2012, pp. 11–26.

[11] C. Guestrin, D. Koller, and R. Parr, "Multiagent planning with factored mdps." in *NIPS*, vol. 1, 2001, pp. 1523–1530.

[12] C. Guestrin, M. Lagoudakis, and R. Parr, "Coordinated reinforcement learning," in *ICML*, vol. 2, 2002, pp. 227–234.

[13] S. A. Hong and G. Gordon, "An accelerated gradient method for distributed multi-agent planning with factored mdps." in *4th NIPS Workshop on Optimization for Machine Learning*, 2011.

[14] C. Zhang and V. R. Lesser, "Coordinated multi-agent reinforcement learning in networked distributed pomdps." in *AAAI*, 2011.

[15] D. T. Nguyen, W. Yeoh, H. C. Lau, S. Zilberstein, and C. Zhang, "Decentralized multi-agent reinforcement learning in average-reward dynamic dcops," in *Proceedings of the 2014 international conference on Autonomous agents and multi-agent systems*. International Foundation for Autonomous Agents and Multiagent Systems, 2014, pp. 1341–1342.

[16] C. Zhang and V. Lesser, "Coordinating multi-agent reinforcement learning with limited communication," in *Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems*. International Foundation for Autonomous Agents and Multiagent Systems, 2013, pp. 1101–1108.

[17] B. Banerjee, J. Lyle, L. Kraemer, and R. Yellamraju, "Sample bounded distributed reinforcement learning for decentralized pomdps." in *AAAI*, 2012.

[18] L. Kraemer and B. Banerjee, "Informed initial policies for learning in dec-pomdps." in *AAAI*, 2012.

[19] A. Boukhtouta, J. Berger, W. B. Powell, and A. George, "An adaptive-learning framework for semi-cooperative multi-agent coordination," in *Adaptive Dynamic Programming And Reinforcement Learning (ADPRL), 2011 IEEE Symposium on*. IEEE, 2011, pp. 324–331.

[20] S. Sen, M. Sekaran, J. Hale *et al.*, "Learning to coordinate without sharing information," in *AAAI*, 1994, pp. 426–431.

[21] I. Karpov, J. Sheblak, and R. Miikkulainen, "Opennero: A game platform for ai research and education." in *AIIDE*, 2008.