The Thesis Committee for Ruohan Zhang

Certifies that this is the approved version of the following thesis

Action Selection in Modular Reinforcement Learning

APPROVED BY

SUPERVISING COMMITTEE

Supervisor: _____

Dana Ballard

Co-Supervisor: _____

Mary Hayhoe

Action Selection in Modular Reinforcement Learning

by

Ruohan Zhang, B.A.

Thesis

Presented to the Faculty of the Graduate School
of the University of Texas at Austin
in Partial Fulfillment
of the Requirements
for the Degree of

Master of Science in Computer Sciences

The University of Texas at Austin
August 2014

Acknowledgment

I owe the greatest thank to my advisor, Professor Dana Ballard, and Professor Mary Hayhoe. I came to UT Computer Science Department as a master student with a bachelor's degree in psychology, and I struggled to find my place. I joined your Embodied Cognition and Virtual Reality Lab in Spring 2013. I am happy to become a member of this lab, where I can pursue my interests in both CS and psychology. You have been always supportive for my research and career plans.

I want to thank Professor Peter Stone, who taught a wonderful class in reinforcement learning, which has become the focus of my master thesis research.

I also thank my parents and my family. I came to the United States 6 years ago, and only visited home a few times since then. Your emotional support is very important to me, so I don't feel lonely when away from home.

Finally, to my fiancee Edith, who has been through everything with me in a country far away from our home, I would not be able to do all these without you.

Ruohan Zhang

The University of Texas at Austin
August 2014

Action Selection in Modular Reinforcement Learning

Ruohan Zhang, M.S.Comp.Sci.
The University of Texas at Austin, 2014

Supervisor: Dana H. Ballard

Modular reinforcement learning is an approach to resolve the curse of dimensionality problem in traditional reinforcement learning. We design and implement a modular reinforcement learning algorithm, which is based on three major components: Markov decision process decomposition, module training, and global action selection. We define and formalize module class and module instance concepts in decomposition step. Under our framework of decomposition, we train each modules efficiently using SARSA(λ) algorithm. Then we design, implement, test, and compare three action selection algorithms based on different heuristics: Module Combination, Module Selection, and Module Voting. For last two algorithms, we propose a method to calculate module weights efficiently, by using standard deviation of Q-values of each module. We show that Module Combination and Module Voting algorithms produce satisfactory performance in our test domain.

Contents

Chapter 1

Introduction

Reinforcement learning (RL) is an active research field in machine learning and cognitive science. Reinforcement learning is a class of problems, where a learning agent interacts with its environment, receives feedback, and learns to maximize cumulative reward by systematic trial and error. In most cases, we assume the feedback is delayed. Markov Decision Process (MDP) is a common model for such a problem.

General machine learning algorithms fall into two categories: supervised learning and unsupervised learning. Reinforcement learning algorithm is considered to be a type in between [22]. It is different from unsupervised learning since feedback is provided to the agent. But unlike supervised learning, where the feedback is *instructive*, the feedback in RL is *evaluative* [22]. Such property provides reinforcement learning certain advantages over other machine learning methods, especially when it is difficult for the agent's designer to instruct exactly how to accomplish a task, but easy to evaluate agent's performance. For this reason, reinforcement learning is more common than supervised learning for intelligent creatures in the nature. Much effort has been spent to understand its biological mechanism, and to develop biologically inspired computer algorithms.

1 Markov Decision Process

First we introduce Markov Decision Process (MDP), the basic model of reinforcement learning problem. An MDP can be defined formally as a tuple $\langle S, A, T, R, \gamma \rangle$[11], where:

- S is a finite set of environment states.
- A is a finite set of agent's available actions.
- T is the state transition function, which describes the probability $P(s'|s, a)$, i.e., the probability of entering state s' when agent takes action a in state s.
- $R : S \mapsto \mathbb{R}$ is a reward function. $R(s, a)$ denotes scalar reward received of

taking action a in state s.

• $\gamma \in [0, 1)$ is a discount factor. We assume the agent values future rewards less than immediate reward, therefore future rewards are discounted by parameter γ.

• $\pi : S \mapsto A$ is called a policy of the agent, which specifies which action to choose in each state. The purpose of the agent, is to find an optimal policy that maximizes its cumulative reward.

2 Reinforcement Learning Algorithms

Any method attempts to solve reinforcement learning problem can be considered as a reinforcement learning algorithm [22]. These algorithms can be roughly categorized into two classes, *policy search* and *value function estimation*. The former searches directly in the policy space, such as policy gradient method [23]. While value function approach estimates the utility function of a state [22]:

$$V^{\pi}(s) = E[R|s, \pi] \tag{1}$$

is called state-value function, which gives expected value of following policy π in state s.

$$Q^{\pi}(s, a) = E[R|s, a, \pi] \tag{2}$$

is called action-value function, which gives expected value of taking action a in state s, and following policy π afterwards.

The optimal policy is defined as the policy that obtains maximum value at each state [22]:

$$V^{*}(s) = \max_{\pi} V^{\pi}(s) \tag{3}$$

$$Q^{*}(s, a) = \max_{\pi} Q^{\pi}(s, a) \tag{4}$$

Hence, to find the optimal policy for a finite MDP, the agent should first calculate optimal state-value or action-value function. Temporal difference (TD) learning, based on recursive Bellman equation, is a popular class of algorithms to estimate value functions. The idea is to use *temporal difference error* to update estimated action-value function during learning. Two TD algorithms, SARSA [16]

and Q-learning [28] are widely used in different reinforcement learning problems. The update rule for SARSA is:

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma Q(s',a') - Q(s,a)] \tag{5}$$

The update rule for Q-learning is:

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma max_{a'} Q(s',a') - Q(s,a)] \tag{6}$$

where α is learning rate, and terms $(r+\gamma Q(s',a')-Q(s,a))$ and $(r+\gamma max_{a'} Q(s',a')-Q(s,a))$ are temporal difference errors. The difference here makes SARSA an *on-policy* learning algorithm, which means it learns action-value function of its current policy. In contrast, Q-learning is *off-policy*, which means it learns action-value function of the greedy policy.

3 Curse of Dimensionality

Plain SARSA and Q-learning algorithms do not scale up well to many practical problem with large state space. One reason is that plain reinforcement learning algorithms suffer from the problem called *curse of dimensionality*, a term first coined by Richard Bellman: computational cost grows exponentially as number of state variables grows [3, 22]. Although RL theories guarantee convergence and optimality of plain RL algorithms, due to this problem, they are often practically too slow to converge to the desired results. RL community has spent much effort to study heuristics and develop approximation algorithms to increase computational efficiency of RL algorithms.

The overall research purpose of this thesis is to develop a computationally efficient reinforcement learning algorithm, which can achieve satisfactory performance in a RL problem with large state space. We propose to take a *modular reinforcement learning* approach, which we will discuss in Chapter 2.

4 Thesis Organization

The rest of thesis is organized as follows: Chapter 2 summarizes related reinforcement learning research to resolve curse of dimensionality problem, and proposes

research goals for the thesis. Chapter 3 introduces a test domain, and demonstrates our modular reinforcement learning algorithm. In Chapter 4, we present two alternative algorithms, and show experimental results for comparison. At last, Chapter 5 we conclude and discuss potential future works.

Chapter 2

Literature Review

As we mentioned earlier, reinforcement learning is "bedeviled by curse of dimensionality" [2]. One approach to this problem is value function approximation. This approach has achieved remarkable performance in large state reinforcement learning problem, such as very large 2D grid world [4], robot navigation and recharge [14], Backgammon game [25], and job-shop scheduling [29].

Another way to resolve curse of dimensionality problem is to use divide-and-conquer strategy. We break original reinforcement learning problem into manageable sub-problems. This strategy is embedded in modular reinforcement learning and hierarchical reinforcement learning, two different but closely related approaches.

Modular reinforcement learning [7, 10, 12, 20] decomposes original RL problem into *modules*. The global task goal is divided into subgoals and one module is responsible for achieving one subgoal. Each module is a sub-MDP with its own state space, but action space is shared among modules. Since sub-MDPs are much smaller, modules can be trained using plain reinforcement learning algorithm. However, a centralized arbitrator is required to combine trained modules and form global policy. Humphrys and Karlsson both explore this approach [10, 12], and their work is further extended by Sprague and Ballard [20]. An algorithm similar to Karlsson's is used in Russell and Zimdars's Q-Decomposition algorithm [17]. The difference between Humphrys and Karlsson is how arbitrator forms global policy, a topic we will discuss later.

Evidences from neuroscience studies also support a modular model in human reinforcement learning [13]. Neuroimaging and lesion study results suggest that, human brain modularizes a complex learning problem to make it more computationally tractable. Activities of important brain areas that are responsible for learning, are found to be correlated to mechanisms in machine reinforcement learning, such as reward estimation in Markov decision making process, temporal-

difference error, and discount factor [9, 13]. In addition, modular reinforcement algorithms have been applied to several practical domains, such as wheel robot soccer playing [26], a simple driving task [12], pendulum swing-up task [7, 18], and fungus eater game [8]. These successful results suggest modular reinforcement learning might be a promising approach to curse of dimensionality problem.

A close relative to modular reinforcement learning is hierarchical reinforcement learning [2, 5, 6, 15], which introduces hierarchy into RL framework. Most hierarchical RL methods are based on the theoretical work of Sutton, Precup, and Singh in temporal abstraction and semi-MDP [24], which defines a concept called *option*. Options are temporally extended courses of primitive actions [24]. Some hierarchical RL methods attempt to reduce the original large MDP problem to be a *single* smaller semi-MDP problem with options [15, 24]. Another type of approach is MAXQ value function decomposition [6]. MAXQ decomposes original tasks into *multiple* semi-MDPs, and create a hierarchy upon them [6], which makes MAXQ more similar to modular RL approach.

Although modular and hierarchical RL methods are shown to be supported by biological studies or be successful in several practical domains, they raise many new issues. The first problem of previously mentioned modular reinforcement learning algorithms is *global action selection*. Given multiple modules, it is nontrivial for arbitrator to compose their individual policies and select an action that maximizes global longterm payoff. One way to do so is to select global action based on weighted outcome of all modules. Hence, the problem here is to determine weights for modules who participate in global action selection. Humphrys shows that module weights can be learned using his W-learning algorithm [10], but this approach introduces an additional learning problem. Our first goal is to bypass this learning procedure and find a more computationally efficient method to determine module weights.

Given multiple modules with their weights, different heuristics are assumed for global action selection. GM-Q or GM-SARSA [12, 20] and Q-Decomposition algorithm [17] are based on the idea of maximizing collective happiness, i.e., max-

imizing the *total* payoff of all modules. However, Humphrys argues that doing so may result in a global action that is no good for any of the module, so he attempts to maximize payoff of a *single*, most important module at each step [10]. Intuitively, there could be action selection algorithms in between: the algorithm can favor some important modules more, and attempt to maximize their utility as its priority. Therefore, the second goal of this thesis is to compare different action selection heuristics, and explore new methods in between.

The second problem with modules and hierarchies is credit assignment. Suppose the agent observed its reward, it is challenging to decide how to assign credit to each module. This is known to be the inter-module credit assignment problem [18]. Hierarchical credit assignment is also challenging. In MAXQ algorithm, designer must decompose reward function while implement the hierarchy, and relevant responsible subtasks must be marked to solve this problem [6].

The cause for credit assignment problem is that, most modular RL and hierarchical RL algorithms assumes a *top-down* learning structure. A centralized arbitrator is responsible for the division of labor, calculating or learning module weights, combining components to produce global policy, and providing feedback to each module. With such structure, the agent is trained using the global task, hence arbitrator needs to perform credit assignment to the modules when giving them feedbacks. On the contrary, we might choose to use a more *bottom-up* approach for modular reinforcement learning. The arbitrator should only define modules. Modules are trained independently in their own sub-MDPs thus no global credit assignment is required. It is possible that module weights could be provided by themselves and passed to the arbitrator, then arbitrator should only perform very simple decision making to select global action. Hence, the third goal of this thesis is to develop a *bottom-up* modular reinforcement learning architecture to reduce computational costs of the arbitrator.

Chapter 3

Modular Reinforcement Learning Algorithm

1 Test Domain

Before describing the algorithm, we first introduce a test domain. Our test domain is a 2D grid world (Figure 1), which is commonly used in reinforcement learning research. Object position is given by its 2D-position $(Row, Column)$. In a single experiment *trial*, the agent (red dot) navigates in the 2D grid world to collect prizes (yellow circles). A prize will be removed if collected. Once all prizes are collected, trial terminates (success). The agent needs to keep away from a predator (black dot), who chases the agent. If agent is captured, current trial terminates (fail). The predator and agent share the same action space: they can go up, down, left, or right. They select their actions simultaneously. If agent steps on an obstacle (blue square), it receives certain amount of penalty. In addition, if agents navigates for too many steps without being able to collect all prizes, trial also terminates (fail).

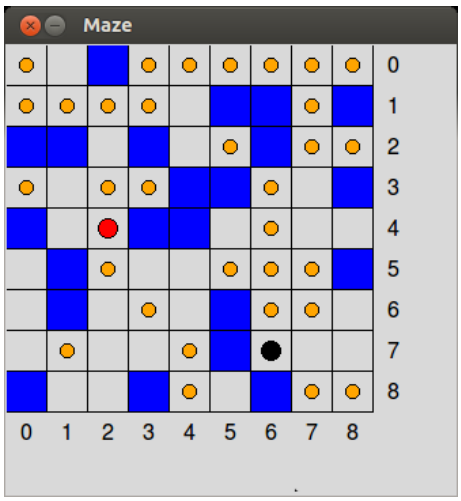


Figure 1: Test domain.

Parameters for our test domain are:

• m : size of the grid world is m by m.

• n_{prize} : initial total number of prizes.

• $n_{obstacle}$: total number of obstacles.

• p_{chase} : predator has certain level of randomness in its action selection. With probability p_{chase}, predator will choose the best action (break tie evenly) to chase the agent; with probability $1 - p_{chase}$, it will select a random action instead.

Intuitively, if maze size is fixed, $n_{obstacle}$ and p_{chase} define the difficulty of the task: the larger these two parameters, the more difficult the task is for the agent.

We also need to establish performance criteria for the test domain. We do not expect the agent to learn the shortest path to collect all the prizes while avoiding obstacles and predator. Four criteria we choose are:

• *cum_reward* : cumulative reward received in one trial, including prizes collected, obstacle hit, and being captured by predator.

• *success_rate* : rate of success in multiple trials. A successful trial means that the agent collects all prizes without being captured by predator. It is OK for the agent to step on the obstacle.

For all successful trials, we want to know:

• *number_steps* : number of steps to complete a successful trial.

For failed trails, we want to know:

• *perc_collected* : percentage of prizes collected before being captured by predator, in a failed trial.

It is necessary to discuss the size of state space of this test domain. The state space is

$$O(m^2 m^2 \binom{m^2}{n_{prize}} \binom{m^2}{n_{obstacle}} 2^{n_{prize}}) \tag{7}$$

. The proof is trivial, each factor accounts for one dimension of the state space: agent position, predator position, possible layout of prizes, possible layout of obstacles, and each prize being collected or not. For a given grid world map, where prize and obstacle positions are fixed, state space is still

$$O(m^2 m^2 2^{n_{prize}}) \tag{8}$$

The agent encounter the curse of dimensionality problem: state space could be very large, plain RL algorithms are computationally intractable.

2 Basic Modular Reinforcement Learning Algorithm

2.1 Defining Module: MDP Decomposition

The first step of modular RL approach is to define a module, which specifies a formal method to decompose original MDP, referred as meta-MDP, into simpler sub-MDPs. We give two key definitions for our modular approach:

• A *module*, or more precisely, a *module class* defines a module MDP based on task subgoals and reward sources. In our test domain, we have three module classes: prize, obstacle, and predator. Each module class i has its own MDP $\langle S_i, A_i, T_i, R_i, \gamma_i \rangle$.

• An *instance of a module class* is an object. Every object in the grid world is an instance of its corresponding module class. For example, in Figure 2, prize module class has two instances: prizes at $(2, 1)$ and $(3, 2)$; obstacle module class has three instances; predator module has one instance. Each instance of a module class has its own state.

Based on these concepts we define module MDPs for prize, obstacle, and

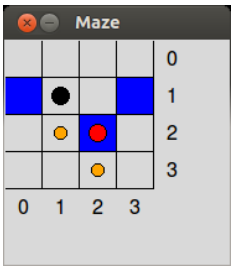


Figure 2: Instances of Module Classes

predator module classes:

• $S_{prize} = prize\_position - agent\_position$ is the set of a single prize's all possible relative 2D-positions to the agent. For example, in Figure 2, agent is at $(2, 2)$, thus prizes at $(2, 1)$ and $(3, 2)$ have states $(0, -1)$ and $(+1, 0)$, respectively.

• $A_{prize} = \{Up, Down, Left, Right\}$.

• T_{prize} : all actions will succeed with probability 1; except if agent tries to move out of the boundaries, it stays at its current position.

• R_{prize} : is numerical reward given to the agent for collecting a prize.

- $\gamma_{prize} \in [0, 1)$ is the discount factor for prize module.

The other two module MDPs are defined in analogy to prize module MDP. An important property of such definition is that, *action space* are completely shared between modules. For reward, we choose $R_{prize} = +10$, $R_{obstacle} = -10$, and $R_{predator} = -100$.

Notice that, under this modular RL setting, it is intuitive to choose discount factor γ. We choose $\gamma_{prize} = 0.7$, $\gamma_{obstacle} = 0.0$, and $\gamma_{predator} = 0.1$.

2.2 Module Training

Decomposed modules have much smaller state space, and can be easily trained using plain reinforcement learning algorithm. Figure 3 shows the training environment for each module. The agent is trained to approach and collect a single prize, to wander without stepping onto an obstacle, and to escape from a predator. Algorithm of training the agent in the prize module world is shown in Algorithm 1. The algorithms for other two modules are very similar except the reward function. In predator module, $p_{chase} = 1.0$, i.e., predator is set to always chase the agent in training.

Multiple techniques are used to improve efficiency of module training. First,

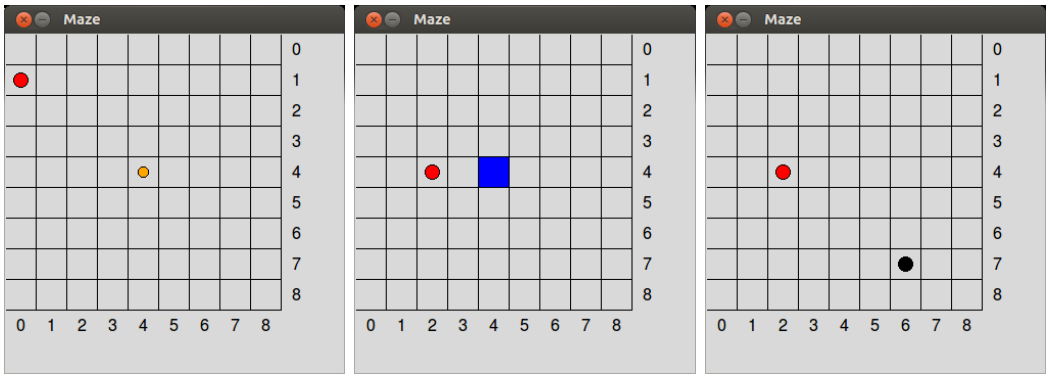


Figure 3: Individual module training: prize, obstacle, and predator.

we use TD-learning with eligibility traces. The algorithm is based on SARSA(λ), with replacing eligibility traces [19] [22]. We choose to use replacing traces rather than accumulating traces, since several states are revisited before their traces decay to 0, causing these traces to be greater than 1. This is a problem described in

Algorithm 1 Module Training (Price Module)

Require: maze size m, untrained Q table

Initialize $Q(s, a) = 0$ and $e(s, a) = 0$ for all s, a

for $agentPos$ in (range(m),range(m)) **do**

for $prizePos$ in (range(m),range(m)) **do**

Place agent at $agentPos$

Place prize at $prizePos$

for episode in range($MAX\_NUM\_EPISODE$) **do**

calculate initial state $s \leftarrow prizePos - agentPos$

select action a from $Q(s)$ using ϵ-greedy algorithm

while prize not collected and $stepCount < MAX\_NUM\_STEP$ **do**

agent takes action a, calculate new agent position $agentPos$

$s' \leftarrow prizePos - agentPos$; calculate reward r

select a' from $Q(s')$ using ϵ-greedy algorithm

$\delta \leftarrow r + \gamma_{prize} Q(s', a') - Q(s, a)$

for all s, a **do**

$e(s, a) \leftarrow 1 + \gamma_{prize} \lambda e(s, a)$ for current state s and current action a

$e(s, a) \leftarrow 0$ for all other actions of current s

$e(s, a) \leftarrow \gamma_{prize} \lambda e(s, a)$ for all other states

end for

$s \leftarrow s'; a \leftarrow a'$

$stepCount \leftarrow stepCount + 1$

end while

end for

end for

end for

return learned $Q(s, a)$

[22] and is shown to be improved by replacing traces.

Second, to encourage better exploration of state space, we place agent and prize at all possible relative locations. In addition, to facilitate convergence of SARSA learning, we increase exploration parameter ϵ gradually towards 0 using rule:

$$\epsilon = 1 - \frac{current\_episode}{total\ \#\ of\ episode} \tag{9}$$

We anneal learning rate α gradually towards 0 using the same rule:

$$\alpha = 1 - \frac{current\_episode}{total\ \#\ episode} \tag{10}$$

2.3 Action Selection: Module Combination

The key question to modular RL approach is how to combine modules to produce a good global policy. A simple way to approximate global optimality is an extended version of GM-SARSA algorithm and Q-Decomposition algorithm for our setting, as we previously mentioned in discusion of [12, 17, 20]. We modified these algorithms to be our Module Combination algorithm. The global Q-value is obtained by adding Q-values of all instances of all module classes: $Q_{global}(s, a) = \sum_i^n Q_i(s_i, a)$, and the action with highest global Q-value is selected. The algorithm is shown in Algorithm 2.

This algorithm is executed at every step of agent's action selection. The agent first identify its own position, observe all surrounding objects' positions, initialize a module instance for all observed objects, then execute the above algorithm to decide its action. Notice we use softmax action selection rule instead of greedy rule $a_{global} = \operatorname{argmax}_a Q_{global}(s, a)$. The reason is that, in our preliminary test, the agent could get stuck in a infinite loop between several positions, due to map layout and greedy action selection rule. Softmax action selection introduces randomness into agent's behavior to avoid this problem, we found it significantly boosts task success rate.

Algorithm 2 Module Combination

Require: trained Q_{module} tables $Q_{prize}, Q_{obstacle}, Q_{predator}$; current agent position $agentPos$; list of all prizes/obstacles/predator positions: $\{prizePos\}, \{obstaclePos\}, \{predatorPos\}$

$Q_{global}(s) = [0, 0, 0, 0]$

for $objPos$ in $\{prizePos\} \cup \{obstaclePos\} \cup \{predatorPos\}$ **do**

$s_{obj} \leftarrow objPos - agentPos$

for a in $\{Up, Down, Left, Right\}$ **do**

read $Q(s_{obj}, a)$ from corresponding Q_{module}

$Q_{global}(s, a) += Q(s_{obj}, a)$

end for

end for

choose a_i with softmax probability $\frac{e^{Q_{global}(s,a_i)}}{\sum_a e^{Q_{global}(s,a)}}$

return chosen action a_{global}

3 Results

The test domain we use is a grid world of $m = 9$. At the beginning of each trial, agent starts at the center of the map, while predator starts at upper left corner. We randomly initialize map cells to contain a prize or an obstacle with probability p_{prize} and $p_{obstacle}$. We choose $p_{prize} = 0.2$. As we mentioned before, $p_{obstacle}$ and predator chasing action probability p_{chase} define task difficulty. A easy task with $p_{obstacle} = 0.2$ and a difficult task with $p_{obstacle} = 0.5$ are shown in Figure 4.

Maximum number of steps the agent can navigate is 1,000. After 1,000 steps, the current trial is counted as failure. We choose $p_{obstacle} = [0.0, 0.5]$ of step size 0.05 (11 values), and $p_{chase} = [0.5, 1.0]$ of step size 0.05 (11 values). We test all the combinations of selected $p_{obstacle}$ and p_{chase} values ($11 * 11 = 121$ data points). For each data point, we randomly generate 300 maps, agent navigates each map once, and we calculate the average performance of these 300 trials. Total of 36,300 trials of data are collected for this experiment.

The algorithm's performance on previous defined criteria is shown in Figure 5. For easiest task $p_{obstacle} = 0.0$ and $p_{chase} = 0.5$, the agent is able to complete a trial

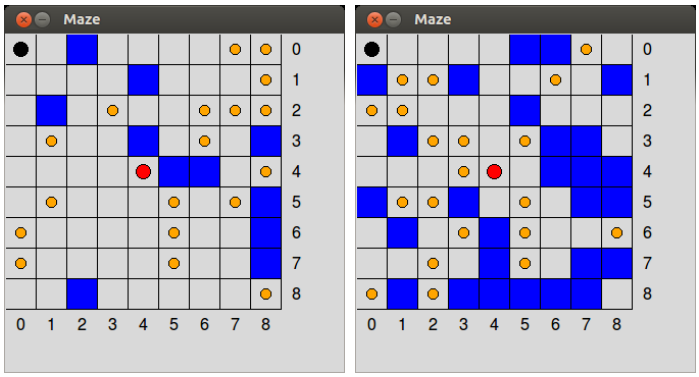


Figure 4: An easy task (left) and a difficult task (right).

successfully at rate 92.5% within 100 steps. With $p_{obstacle} = 0.0$ and $p_{chase} = 1.0$, agent can complete a successful trial even faster with 69.14 steps, but at lower success rate overall. We observe that agent's performance degrades as $p_{obstacle}$ and p_{chase} increase. With $p_{obstacle} = 0.5$ and $p_{chase} = 1.0$, success rate is only 2.5%. However, even for those failed trials the agent is able to collect at least 59.5% prizes before trial failed. Notice with high p_{chase}, the agent can hardly focus on anything else other than escaping from predator, and trial often terminates with failure because agent exceeds maximum number of steps it can navigate.

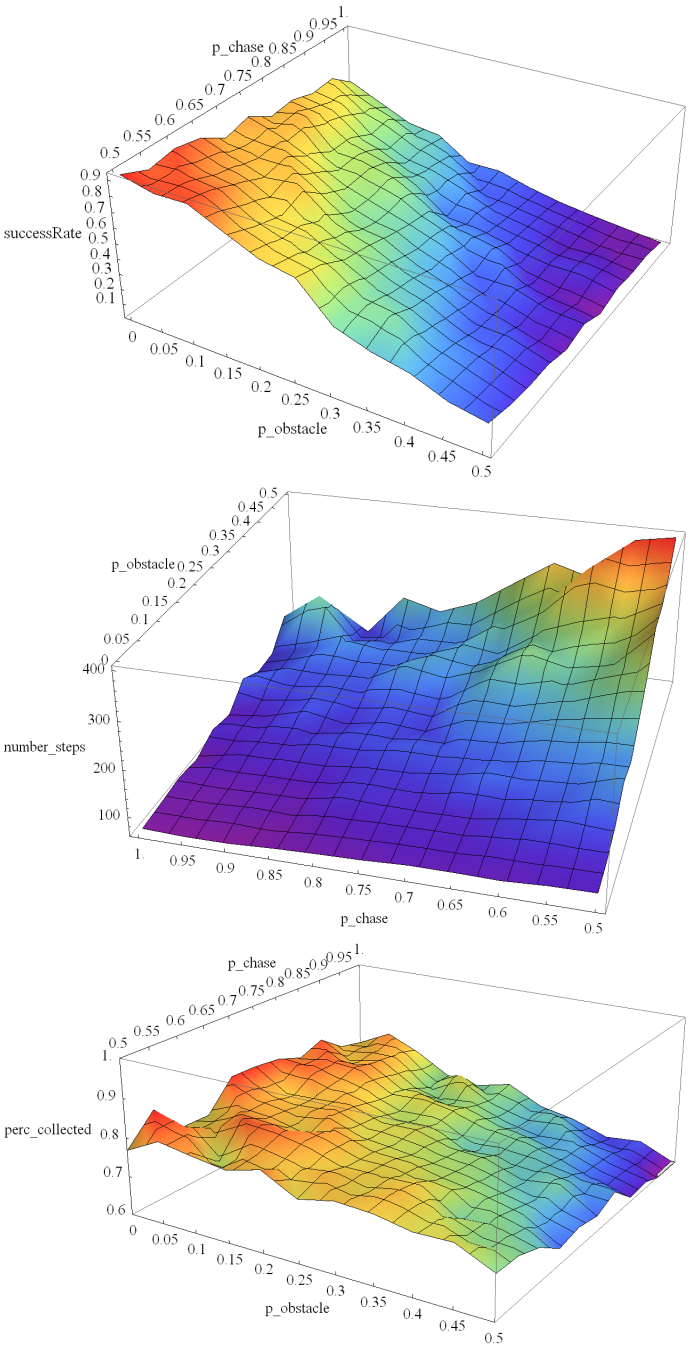


Figure 5: Module Combination Algorithm success rate (top), number of steps to succeed (middle), percentage of prizes collected before failed (bottom).

Chapter 4

Action Selection Algorithms

1 Module Weight

We have shown that our Module Training and Module Combination algorithm produce reasonable performance in given test domain. As we discussed before, we want to explore and compare different action selection heuristics. Module Combination algorithm tries to maximize *collective happiness* for all module instances, while [10] suggests a winner-takes-all approach. Hence, to determine a winning module, our next goal is to assign *weight*, or priority, to each module instance, and we simply select the module with highest weight. As we mentioned in Chapter 2, we do not want to introduce weight learning. These weights should be easily computed, and reflects the relative importance of a module instance under its current state. The global action selection algorithm should be able to make decision quickly based on the weights provided by module instances.

2 Flatness of Q-Values as Module Weight

An intuitive way to determine the weight of module instances is the magnitude of its optimal action's Q-value in its current state. For example, if Q-values for a module instance under current state are $[10, 0, 0, 0]$, and another module instance's Q-values are $[0, 1, 0, 0]$, the weight for the first module instance should be 10 and the second should be 1. We hence select the optimal action of first module instance. However, consider a module instance with Q-values $[10, 10, 10, 10]$ and another one with $[0, 1, 0, 0]$. Although the first one's optimal Q-value has larger magnitude, it is *indifference* about its action selection. We should assign second module instance higher weight.

Our observation for action-value function is that, it not only tells us the optimal action and its magnitude. Like probability distribution, it can also be analyzed using statistical properties. For a module instance at a given state, the indifference

in action selection can be measured by the *flatness* of Q-values. Such flatness can be calculated using kurtosis (fourth momentum of a distribution). Since we have only four values (action values for Up, Down, Left, Right), we can use standard deviation instead.

The way Q-values are calculated and learned supports this approach. Figure 6 shows Q-values (a) for agent at different states relative to a prize (b). Due to the fact that discount factor γ is less than 1, as agent moves closer to the prize, the shape of its Q-values becomes more peaky, the standard deviation becomes larger, and we assign higher weight to the module instance.

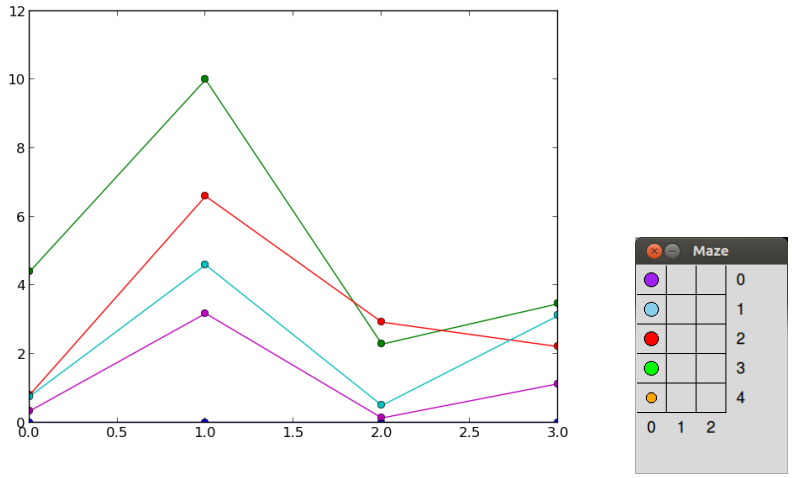


Figure 6: (a) Q-values of actions Up, Down, Left, and Right. (b) agent at different states. Colors are matched.

3 Module Selection Algorithm

Similar to Humphrys's algorithm [10], we might choose to maximize the payoff of a single module instance, rather than total payoff of all module instances. An algorithm derived from this idea is to select a single module with highest weight (standard deviation) to take over control at every given global state. We choose the optimal action of selected module as global action. We refer this algorithm to be Module Selection, as shown in Algorithm 3.

Notice that, comparing to Module Combination, in which we must sum Q-

Algorithm 3 Module Selection

Require: trained Q_{module} tables $Q_{prize}, Q_{obstacle}, Q_{predator}$; current agent position $agentPos$; list of all prizes/obstacles/predator positions: $\{prizePos\}, \{obstaclePos\}, \{predatorPos\}$

for $objPos$ in $\{prizePos\} \cup \{obstaclePos\} \cup \{predatorPos\}$ **do**

$s_{obj} \leftarrow objPos - agentPos$

$weight_{obj} \leftarrow$ standard deviation of $Q_{module}(s_{obj})$

end for

choose obj^* with highest $weight_{obj}$

return $a_{global} = \underset{a}{\text{argmax}}\, Q_{obj^*}(s_{obj^*}, a)$

values at each step, weights of Q-values can be precomputed and stored to further reduce computational costs during navigation.

4 Module Voting Algorithm

An algorithm in between Module Combination and Module Selection is Module Voting. Instead of selecting a single module instance to make decision, we can allow each module instance to vote for its optimal action. However, voters are not equal, we use weighted voting rather than one-man-one-vote system. Like in shareholder meeting, we value important module instances' decision more. Hence, each module instance's vote on its optimal action is weighted, and we use the same method to calculate weight as in Module Selection algorithm. The Module Voting algorithm is shown in Algorithm 4. Although we can use softmax action selection based on vote count of each action, preliminary results show greedy action selection produces better performance.

5 Results

We find that Module Selection and Module Voting are slightly faster because they can pre-calculate and store weights, while Module Combination must sum Q-values of all module instances at every step.

Algorithm 4 Module Voting

Require: trained Q_{module} tables $Q_{prize}, Q_{obstacle}, Q_{predator}$; current agent position $agentPos$; list of all prizes/obstacles/predator positions: $\{prizePos\}, \{obstaclePos\}, \{predatorPos\}$

$Vote\_Count(s) = [0, 0, 0, 0]$

for $objPos$ in $\{prizePos\} \cup \{obstaclePos\} \cup \{predatorPos\}$ **do**

$s_{obj} \leftarrow objPos - agentPos$

$a^*_{obj} = \underset{a}{\operatorname{argmax}}\, Q_{obj}(s_{obj}, a)$

$weight_{obj} \leftarrow$ standard deviation of $Q_{module}(s_{obj})$

$Vote\_Count(s, a^*_{obj}) += weight_{obj}$

end for

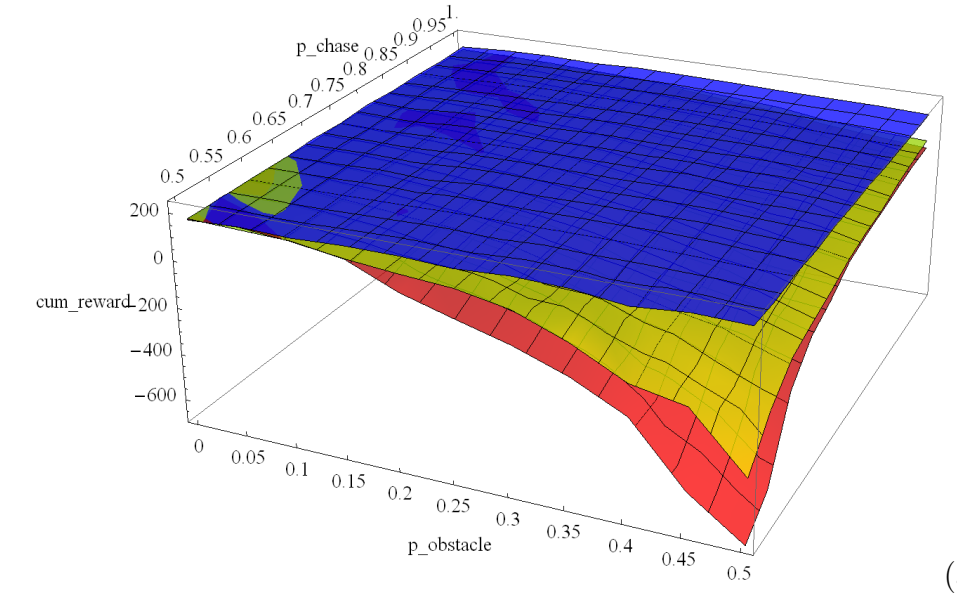
return chosen action $a_{global} = \underset{a}{\operatorname{argmax}}\, Vote\_Count(s, a)$

In Figure 7, 8, 9, and 10 we show comparative results of Module Combination (blue), Module Voting (yellow), and Module Selection (red) algorithms on cumulative reward, success rate, number of steps needed to success, and percentage of prizes collected before failed. All figure (a) show data similar to Figure 5, but with three algorithms together. Again, each data point is the average of 300 trials, and 36300 trials of data are collected for each module. In all Figure (b) we convert Figure (a) into 2D to better visualize the relation between task difficulty and performance, where $task\_difficulty = p_{obstacle} + p_{chase}$.

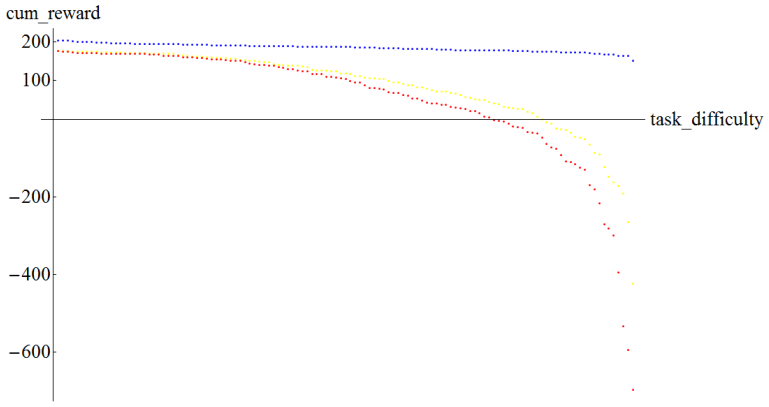
Overall, Module Combination and Module Voting algorithms have better performance than Module Selection algorithm. For cumulative reward in Figure 7, Module Combination has the highest average reward under most settings. However, for another important criterion success rate in Figure 8, although Module Combination (blue) has best performance for easy tasks, Module Voting (yellow) achieves higher overall success rate, for medium and difficult tasks. Module Selection (red) does the best when $p_{obstacle}$ is very low while p_{chase} is high. Because in such situation, Module Selection algorithm forces the agent to always focus on predator module instance, which has highest weight when predator is close to the agent. Such strategy allows agent to survive in a world without many obstacles,

but performance deteriorates in a world with more obstacles.

Figure 9 shows average number of steps needed to complete a successful trial. Module Combination (blue) takes the fewest steps in most situations. However, for difficult tasks, Module Voting algorithm requires fewer steps. Figure 10 shows in failed trails, average percentage of prizes collected before being captured by predator or agent exceeds maximum number of steps. Module Combination collects more prizes than other two algorithms under most settings. In general, Module Selection algorithm has the poorest performance in these two performance criteria.

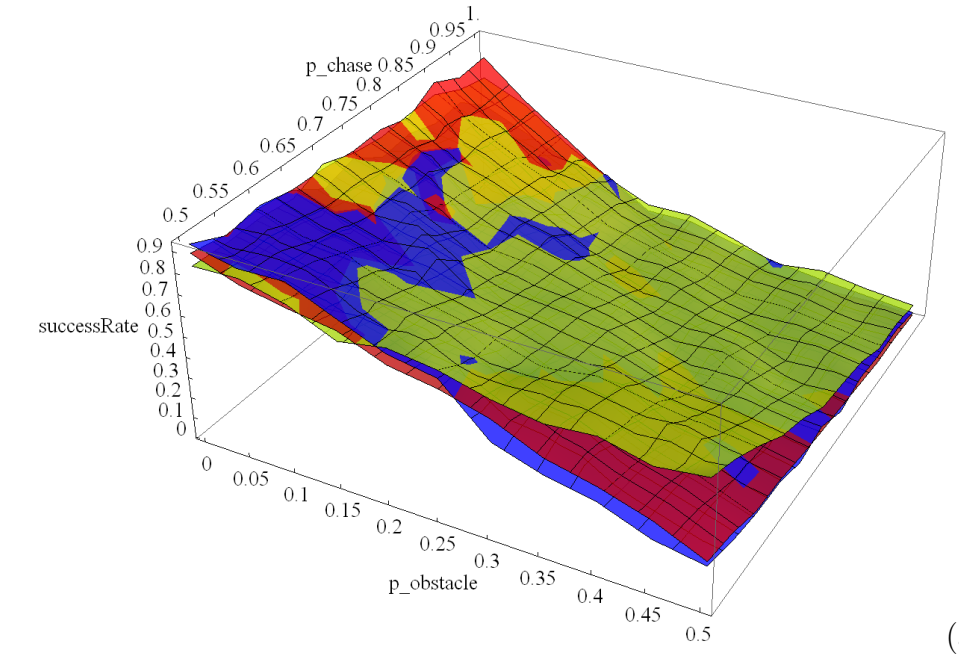


(a)

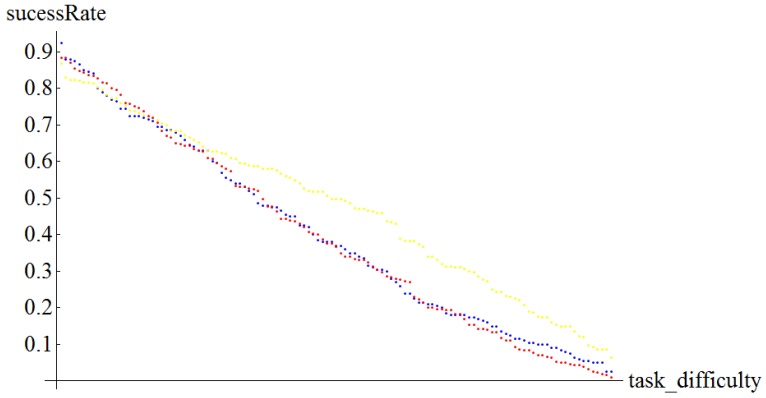


(b)

Figure 7: Performance comparison of three algorithms: cumulative reward. Blue: Module Combination; Yellow: Module Voting; Red: Module Selection.



(a)



(b)

Figure 8: Performance comparison of three algorithms: success rate. Blue: Module Combination; Yellow: Module Voting; Red: Module Selection.

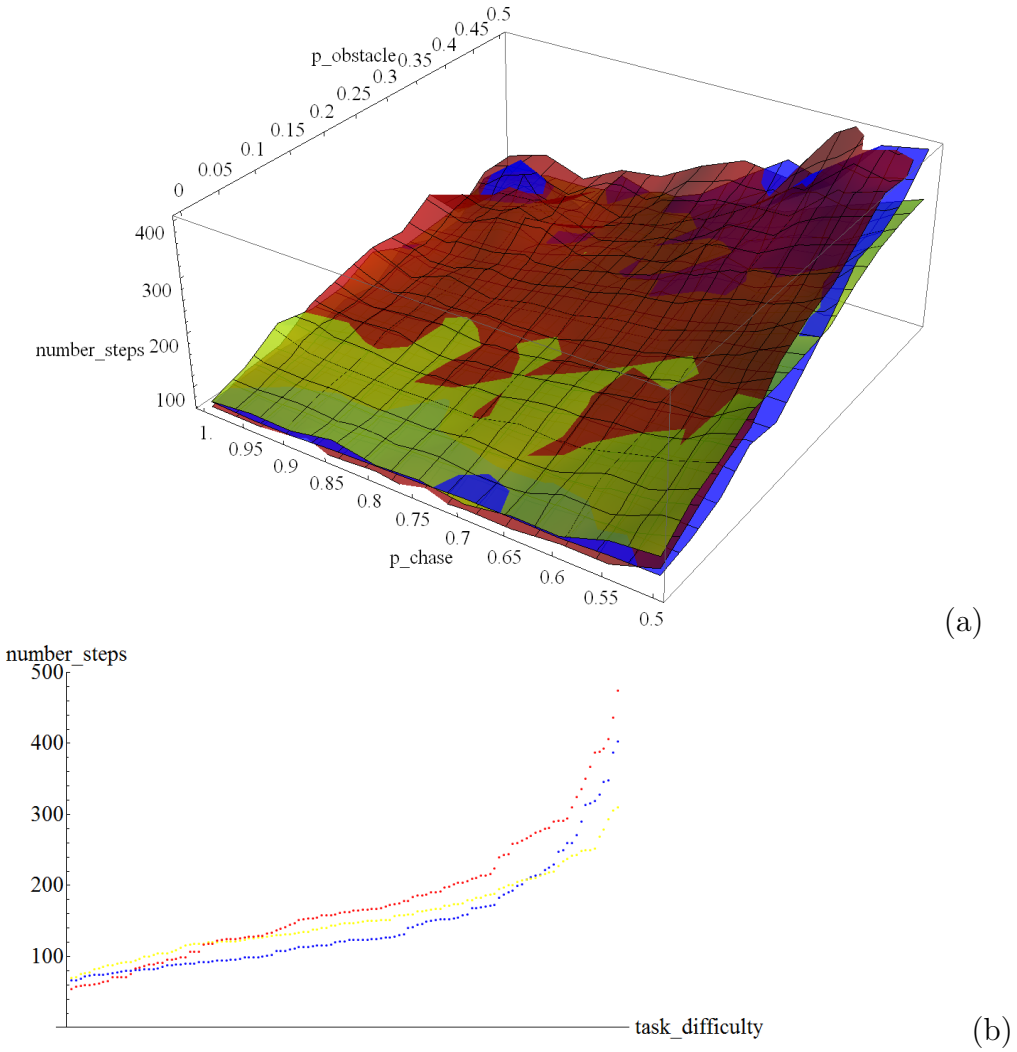
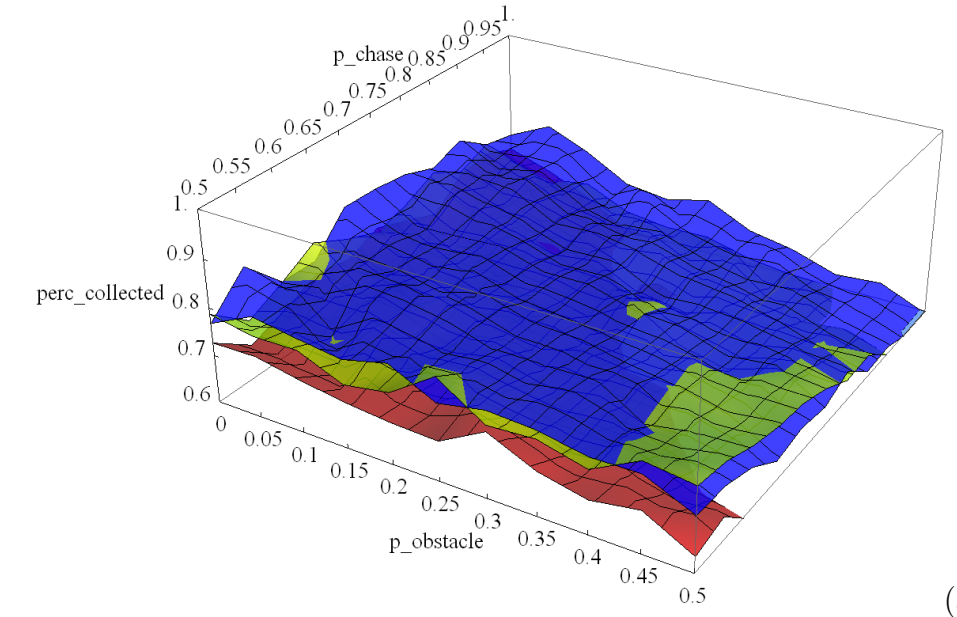
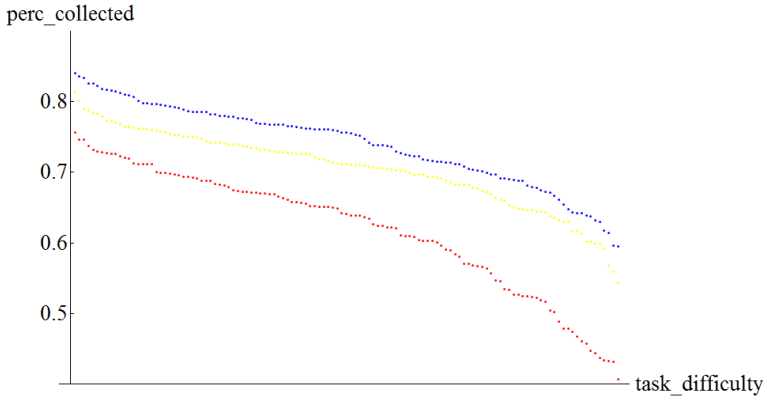


Figure 9: Performance comparison of three algorithms: average number of steps required to complete a trial. Blue: Module Combination; Yellow: Module Voting; Red: Module Selection.



(a)



(b)

Figure 10: Performance comparison of three algorithms: in failed trials, average percentage of prizes collected before failed. Blue: Module Combination; Yellow: Module Voting; Red: Module Selection.

Chapter 5

Conclusion

In this thesis we presented a modular reinforcement learning algorithm. Comparing to plain RL algorithms, such modular approach aims at developing computationally tractable algorithm to cope with curse of dimensionality problem. One contribution of this thesis is to formalize and distinguish concepts of *module class* and *module instance*, when decompose original MDP problem into module-MDPs. Such decomposition allows us to train each module class efficiently using SARSA(λ). Module Training algorithm is finished in prior to the actual navigation, hence will not affect the efficiency of agent's action selection algorithm.

The main focus of this research is to select global action from many module instances. We design and test three action selection algorithms, and compare the performances on our designed test domain. The first algorithm, Module Combination, derived from [12, 17, 20], which attempts to maximizes collective payoff of all module instances, resulted in satisfactory performance.

Another contribution of this thesis is that we develop a fast mechanism to determine the weights for module instances, by using standard deviation of Q-values. Then we design our Module Selection algorithm, based on maximizing an individual module instance's utility at each step. It performs considerably worse than Module Combination. At last, we design the Module Voting algorithm, based on shareholder meeting heuristic. This algorithm yields competitive results, especially on difficult tasks.

The discussion on maximizing individual module utility versus collective utility relates our modular reinforcement learning approach to multi-agent reinforcement learning. Modules with different goals are similar to heterogeneous learning agents [21]. One research focus of multi-agent system is the cooperation and competition between agents. Similarly, modules can be cooperative or competitive with other modules. Research results in multi-agent RL may provide us insights on this topic in the future.

Our modular reinforcement learning approach provides us several advantages over plain reinforcement learning. Module class trainings are off-line so they do not introduce computational costs to actual experiment. Three action selection algorithms have the same bound on computational costs, which is linear on number of module instances:

$$O(n_{prizes} + n_{obstacles} + n_{predators}) \tag{11}$$

Recall in equation (7) and (8), the size of state space for plain reinforcement learning algorithm grows exponentially with environment parameters. The major advantage of our algorithm is its computational efficiency, which allows us to test it in a world with large state space.

An important feature of our modular approach is that, we can choose different discount factors γ for different module classes, where in plain reinforcement learning a global γ is defined. For prize module class, γ is high because we care about future rewards. For obstacle and predator module class, γ is low because we does not need to care about an obstacle or a predator far away. Such flexibility allows us to interpret γ better, and train each module class more efficiently.

Comparing to other modular and hierarchical RL algorithms, our approach emphasizes *bottom-up* control structure rather than top-down structure. We resolve weight learning and credit assignment problems by letting modules calculate their weights themselves. Central arbitrator only performs small amount of calculation to determine global action.

One drawback of our algorithm is that, it does not guarantee global optimality for resulted policy. Modules are trained independently so they cannot account for the effects of executing other modules. It is computationally impractical to test how close the policies derived from our algorithms are to the optimal policy trained by plain reinforcement learning, since plain RL does not scale up to our test world. We sacrifice theoretical guarantee of optimality for computational efficiency. One reason is that we hope to apply our algorithm to practical real-time decision making problems with even larger state space, for example, autonomous driving. Driving emphasizes fast decision making, and requires a close-to-optimal

policy. In addition, driving environment is a relatively 'easy' world, in terms of $p_{obstacle}$, and it is also friendly due to the absence of predator. Our algorithms are shown to achieve high success rate in such an environment. Previous research has been done on modularizing the driving task [27], and module selection in driving [1], but not under MDP and reinforcement learning framework. We hope to incorporate our modular reinforcement learning approach with these works.

References

[1] Dana H Ballard, Dmitry Kit, Constantin A Rothkopf, and Brian Sullivan. A hierarchical modular architecture for embodied cognition. *Multisensory research*, 26:177–204, 2013.

[2] Andrew G Barto and Sridhar Mahadevan. Recent advances in hierarchical reinforcement learning. *Discrete Event Dynamic Systems*, 13(4):341–379, 2003.

[3] R. Bellman and Rand Corporation. *Dynamic Programming*. Rand Corporation research study. Princeton University Press, 1957.

[4] Justin Boyan and Andrew W Moore. Generalization in reinforcement learning: Safely approximating the value function. *Advances in neural information processing systems*, pages 369–376, 1995.

[5] Peter Dayan and Geoffrey E Hinton. Feudal reinforcement learning. In *Advances in neural information processing systems*, pages 271–271. Morgan Kaufmann Publishers, 1993.

[6] Thomas G Dietterich. Hierarchical reinforcement learning with the maxq value function decomposition. *arXiv preprint cs/9905014*, 1999.

[7] Kenji Doya, Kazuyuki Samejima, Ken-ichi Katagiri, and Mitsuo Kawato. Multiple model-based reinforcement learning. *Neural computation*, 14(6):1347–1369, 2002.

[8] Christopher J Hanna, Ray J Hickey, Darryl K Charles, and Michaela M Black. Modular reinforcement learning architectures for artificially intelligent agents in complex game environments. In *Computational Intelligence and Games (CIG), 2010 IEEE Symposium on*, pages 380–387. IEEE, 2010.

[9] Masahiko Haruno, Tomoe Kuroda, Kenji Doya, Keisuke Toyama, Minoru Kimura, Kazuyuki Samejima, Hiroshi Imamizu, and Mitsuo Kawato. A neural correlate of reward-based behavioral learning in caudate nucleus: a functional

magnetic resonance imaging study of a stochastic decision task. *The Journal of Neuroscience*, 24(7):1660–1665, 2004.

[10] Mark Humphrys. Action selection methods using reinforcement learning. *From Animals to Animats*, 4:135–144, 1996.

[11] Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. Reinforcement learning: A survey. *arXiv preprint cs/9605103*, 1996.

[12] Jonas Karlsson. *Learning to solve multiple goals*. PhD thesis, Citeseer, 1997.

[13] Mitsuo Kawato and Kazuyuki Samejima. Efficient reinforcement learning: computational theories, neuroscience and robotics. *Current opinion in neurobiology*, 17(2):205–212, 2007.

[14] Long Ji Lin. Programming robots using reinforcement learning and teaching. In *AAAI*, pages 781–786, 1991.

[15] Ronald Parr and Stuart Russell. Reinforcement learning with hierarchies of machines. *Advances in neural information processing systems*, pages 1043–1049, 1998.

[16] Gavin A Rummery and Mahesan Niranjan. *On-line Q-learning using connectionist systems*. University of Cambridge, Department of Engineering, 1994.

[17] Stuart Russell and Andrew Zimdars. Q-decomposition for reinforcement learning agents. In *ICML*, pages 656–663, 2003.

[18] Kazuyuki Samejima, Kenji Doya, and Mitsuo Kawato. Inter-module credit assignment in modular reinforcement learning. *Neural Networks*, 16(7):985–994, 2003.

[19] Satinder P Singh and Richard S Sutton. Reinforcement learning with replacing eligibility traces. *Machine learning*, 22(1-3):123–158, 1996.

[20] Nathan Sprague and Dana Ballard. Multiple-goal reinforcement learning with modular sarsa (0). In *IJCAI*, pages 1445–1447, 2003.

[21] Peter Stone and Manuela Veloso. Multiagent systems: A survey from a machine learning perspective. *Autonomous Robots*, 8(3):345–383, 2000.

[22] Richard S Sutton and Andrew G Barto. *Introduction to reinforcement learning*. MIT Press, 1998.

[23] Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *NIPS*, volume 99, pages 1057–1063. Citeseer, 1999.

[24] Richard S Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1):181–211, 1999.

[25] Gerald Tesauro. Temporal difference learning and td-gammon. *Communications of the ACM*, 38(3):58–68, 1995.

[26] Eiji Uchibe, Minoru Asada, and Koh Hosoda. Behavior coordination for a mobile robot using modular reinforcement learning. In *Intelligent Robots and Systems' 96, IROS 96, Proceedings of the 1996 IEEE/RSJ International Conference on*, volume 3, pages 1329–1336. IEEE, 1996.

[27] Chris Urmson, Joshua Anhalt, Drew Bagnell, Christopher Baker, Robert Bittner, MN Clark, John Dolan, Dave Duggins, Tugrul Galatali, Chris Geyer, et al. Autonomous driving in urban environments: Boss and the urban challenge. *Journal of Field Robotics*, 25(8):425–466, 2008.

[28] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

[29] Wei Zhang and Thomas G Dietterich. A reinforcement learning approach to job-shop scheduling. In *IJCAI*, volume 95, pages 1114–1120. Citeseer, 1995.

Vita

Ruohan Zhang was born in Chengdu, China on July 6, 1988. He graduated from Chengdu No.7 High School. In 2008 he came to the United States to study at Rhodes College, Memphis, TN. In May 2012 he received the B.A. degree in Psychology, with a Minor in Computer Science, and a Minor in Economics. In August 2012, he joined the graduate program in the Department of Computer Science at The University of Texas at Austin. His research interests are Machine Learning and Robotics.

Permanent Address:

Department of Science and Technology
Sichuan University
Chengdu, Sichuan, P.R.China - 610065
zharug@gmail.com

This thesis was typeset with LaTeXby the author.