LEARNING COOPERATION IN HUNTER-PREY PROBLEM VIA STATE ABSTRACTION

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

ATIL İŞÇEN

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
COMPUTER ENGINEERING

JUNE 2009

Approval of the thesis:

LEARNING COOPERATION IN HUNTER-PREY PROBLEM VIA STATE ABSTRACTION

submitted by **ATIL İŞÇEN** in partial fulfillment of the requirements for the degree of **Master of Science in Computer Engineering Department, Middle East Technical University** by,

Prof. Dr. Canan Özgen
Dean, Graduate School of **Natural and Applied Sciences** _____

Prof. Dr. Müslim Bozyiğit
Head of Department, **Computer Engineering** _____

Prof. Dr. Faruk Polat
Supervisor, **Computer Engineering** _____

Examining Committee Members:

Prof. Dr. Kemal Leblebicioğlu
Electrical and Electronics Engineering, METU _____

Prof. Dr. Faruk Polat
Computer Engineering, METU _____

Prof. Dr. Göktürk Üçoluk
Computer Engineering, METU _____

Assoc.Prof.Dr. Ahmet Coşar
Computer Engineering, METU _____

Dr. Sertan Girgin
GATA _____

Date: _____

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last Name: ATIL İŞÇEN

Signature :

ABSTRACT

LEARNING COOPERATION IN HUNTER-PREY PROBLEM VIA STATE ABSTRACTION

İşçen, Atıl

M.S., Department of Computer Engineering

Supervisor : Prof. Dr. Faruk Polat

June 2009, 41 pages

Hunter-Prey or Prey-Pursuit problem is a common toy domain for Reinforcement Learning, but the size of the state space is exponential in the parameters such as size of the grid or number of agents. As the size of the state space makes the flat Q-learning impossible to use for different scenarios, this thesis presents an approach to make the size of the state space constant by producing agents that use previously learned knowledge to perform on bigger scenarios containing more agents. Inspired from HRL methods, the method is composed of a parallel subtasks schema dividing the task into choices of simpler subtasks, a state representation technique convenient for this schema and its extension for bigger grids. Experiment results show that proposed method successfully provides agents that perform near to hand-coded agents by using constant sized state space independent from parameters of the domain.

Keywords: Reinforcement Learning, Hunter-Prey Problem, State Abstraction, Hierarchical Reinforcement Learning

ÖZ

AV AVCI PROBLEMİNDE DURUM SOYUTLAMA YOLUYLA İŞBİRLİĞİ ÖĞRENME

İşçen, Atıl
Yüksek Lisans, Bilgisayar Mühendisliği Bölümü
Tez Yöneticisi : Prof. Dr. Faruk Polat

Haziran 2009, 41 sayfa

Avcı-av problemi Pekiştirmeli Öğrenme yöntemi için sıkça kullanılan bir deney alanıdır, ancak durum uzayı hacminin büyüklüğü ajan sayısına ve ortam büyüklüğüne üstel bağlı olarak değişmektedir. Durum uzayının bu büyüklüğü standart Q-öğrenme algoritmasının kullanımını imkansız kıldığından, bu tez daha önce öğrenilmiş bilgiyi kullanıp daha büyük deney ortamlarında çalışabilen ajanlar üreterek, durum uzayı büyüklüğünün sabit tutmayı sağlayan bir yöntem tanıtmaktadır. Bu metot, Hiyerarşik Takviyeli Öğrenme yöntemlerinden esinlenerek görevi daha basit alt görev seçimlerine bölen paralel alt görev mekanizmasından, bu yönteme yönelik bir durum gösterim tekniğinden ve bunun daha büyük alanlar için genişleti-minden oluşmaktadır. Deneysel sonuçlar önerilen yöntemin alanın parametrelerinden bağım-sız, sabit büyüklükte bir durum uzayı kullanarak, el ile yazılmış algoritma kullanan ajanlara yakın, başarılı sonuçlar elde ettiğini göstermektedir.

Anahtar Kelimeler: Pekiştirmeli Öğrenme, Durum Soyutlama, Avcı-av problemi, Hiyerarşik Takviyeli Öğrenme

ACKNOWLEDGMENTS

First, I would like to thank my advisor Faruk Polat for his advices, suggestions and encouragements. His advices will definitely play an important role in my life and future research career. I would like to thank Umut Eroğul and Çelebi Kocair for their ideas on the topic and for making the office an enjoyable place to work. I would also like to thank Gökçe Över for her encouragement and understanding throughout this hard process. Lastly, I am thankful to my parents for their support.

This work was supported by TÜBİTAK National Scholarship Programme for MSc Students.

TABLE OF CONTENTS

ABSTRACT iv

ÖZ v

ACKNOWLEDGMENTS vi

TABLE OF CONTENTS vii

LIST OF TABLES ix

LIST OF FIGURES x

CHAPTERS

1 INTRODUCTION 1

2 BACKGROUND 3

2.1 Reinforcement Learning 3

2.1.1 Temporal Difference Learning 5

2.1.2 Hierarchical Reinforcement Learning 7

2.1.3 Transfer Knowledge on RL 10

2.2 Multi-Agent Reinforcement Learning 10

3 PROBLEM DESCRIPTION 12

3.1 Multi-Agent Learning 12

3.2 Hunter Prey Domain and Variants 13

3.2.1 Different Domain Parameters 14

3.2.2 Chosen Domain Configuration 15

3.2.3 Classic State Representations and Size of The State Space 17

4 PROPOSED METHODS 20

4.1 Prey-Oriented State Representation 20

4.2 Expanding State Space with Mapping 22

4.3 Hierarchical Reinforcement Learning 24

5 EXPERIMENTS AND RESULTS 28
5.1 State Representation 28
5.2 Bigger Map 33
5.3 Multi Prey 35
6 CONCLUSIONS AND FUTURE WORK 38
REFERENCES 40

LIST OF TABLES

TABLES

Table 3.1 Size of The State Space for Three Methods 19

Table 5.1 Hunters with $11 * 11$ state space on $15 * 15$ grid 29

Table 5.2 Different Agent Types trying to capture 5 preys 35

Table 5.3 Average Capture Steps for Different Number of Preys 36

Table 5.4 Size of the State Space for Different Grid Sizes and Number of Preys 36

LIST OF FIGURES

FIGURES

Figure 2.1 Reinforcement Learning Framework 3

Figure 2.2 3x3 gridworld as a RL domain 5

Figure 2.3 Taxi Domain and MAXQ task decomposition 9

Figure 2.4 Optimality Problem in Two Rooms Maze Domain 10

Figure 3.1 Hunter-Prey (Prey-Pursuit) Problem 13

Figure 3.2 Hand-Coded Prey Escape Algorithm 16

Figure 3.3 Standard State Representation for Prey-Pursuit Domain 18

Figure 3.4 Two equivalent map states on a toroidal grid 18

Figure 3.5 Relative Positions State Representation for Prey-Pursuit Domain 18

Figure 3.6 45 Degree Partitioned Polar State Representation 19

Figure 4.1 Prey Oriented State Representation 21

Figure 4.2 Rings Around an Agent 22

Figure 4.3 Mapped Square Examples for the Mapping Used 23

Figure 4.4 Two Rooms Maze Problem Divided to Two Subtasks 25

Figure 4.5 Subtasks for a Multi-Prey Problem 26

Figure 5.1 Agents with Same Size of State Space on a 15x15 Grid 29

Figure 5.2 Guaranteed Capture Strategy for Hunters 31

Figure 5.3 Hunters Trying to Capture the Prey by Chasing 31

Figure 5.4 Performances of Agents on a 11x11 Grid 32

Figure 5.5 Performances of Agents on a 13x13 Grid 33

Figure 5.6 Performance of the agents on 25x25 grid 34

CHAPTER 1

INTRODUCTION

Reinforcement Learning is a subfield of Artificial Intelligence where an agent tries to adopt to the environment by learning from its experiences. Inspired from psychological theories of human learning, it differs from supervised learning methods by the absence of correct actions to be learned, instead of it, it uses reinforcement signals that the environment provides to the agent to find correct policy.

The limitation of the RL is the size of the state space which makes it currently applicable to only toy problems instead of complex ones. Hunter-Prey problem (also known as Prey-Pursuit domain) is a good example to this limitation. The studies on this domain work on small sized grids having few agents because of the size of the state space which increases exponentially according to the grid size and number of agents.

To overcome this limitation, there are many studies on Hierarchical Reinforcement Learning which divides the problem into smaller subtasks to make the problem suitable to abstraction methods. These methods use sequential subtasks such as picking up a passenger and putting down for a taxi problem. On the other hand, some problems containing different choices for the agent, are composed of parallel subtasks instead of sequential ones. Hunter-Prey problem with multiple preys has such a problem structure by forcing the hunter agents to choose one of the preys to capture.

This thesis uses mentioned parallel subtasks structure of the Hunter Prey domain to remove the dependency of the size of the state space to the size of the grid and to the number of the agents. The method proposed is a three step approach to be able to work on more complex versions of the problem: Changing state representation, mapping to bigger state spaces and using parallel subtasks.

This thesis is composed of six chapters. This chapter introduces the study, the second chapter gives necessary background information on the domain and the third chapter describes the problem in details. The fourth chapter explains the methods proposed and the fifth chapter contains the experiments and their results. The last part contains discussion about conclusions and future works.

CHAPTER 2

BACKGROUND

2.1 Reinforcement Learning

Reinforcement Learning (RL) is a machine learning method where the agent learns by interaction with its surrounding environment. Its difference from supervised learning methods is that the agent does not learn from given training set containing correct actions. RL is a goal directed learning method, which means that the designer of the problem does not define correct behavior or actions for the agent, he defines goals of the agents by designing reinforcement signals. The agent gets reinforcement signals from the environment according to its actions' consequences and it autonomously finds a policy to maximize these signals.

Figure 2.1 shows the interaction of the agent and the environment in a RL framework. On a discrete time problem, at each step, this interaction is composed of three signals representing the state, action and reward at time t. If we need to define it in details, the basic concepts in RL are as follows:

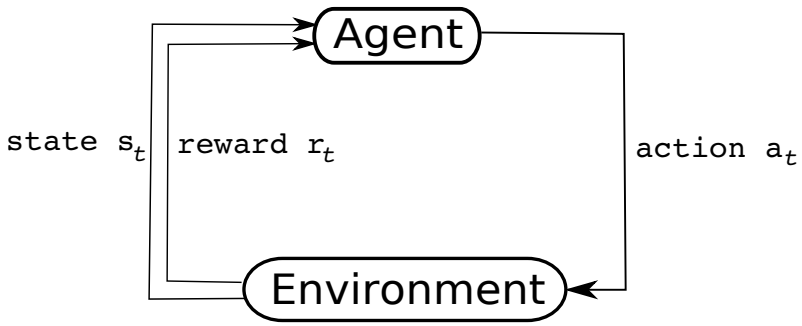

Figure 2.1: Reinforcement Learning Framework

$s_t \in \mathbb{S}$ is the signal used to define current situation of the environment at time t, and $\mathbb{S}$ is the set of all possible states. This state signal can be consisted of one or more parameters according to the state space used.

$a_t \in \mathbb{A}$ represents the action taken by the agent at time t, after perceiving the state s_t. A is the action space containing all the possible actions.

$r_{t+1} \in \mathbb{R}$ is the reward sent by the environment at time $t+1$ in response to a_t, the action taken by the agent at time t.

$\pi(s, a)$ is the policy used by the agent. It maps the states to the actions by defining the probability to choose action a at state s. The optimal policy that the agent tries to learn is the policy that maps the states to actions such that expected cumulative reward is maximized.

$V^\pi(s_t)$, $Q^\pi(s_t, a)$ are the value functions. $V^\pi(s)$ represents the total reward that the agent expects starting from the state s_t and follows the policy π. $Q^\pi(s, a)$ is the same concept associated with state, action pairs (expected value of choosing action a at state s and then following the policy π.).

If we define the value function formally:

$$V^\pi(s) = E_\pi\{r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ...\} = E_\pi\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s\right\} \tag{2.1}$$

where γ is discount factor which determines the effect of the future rewards on the value estimation and E_π is expected value that the agent follows policy π. When the task is episodic, the agent starts each episode at one of the start states (s_{start}), executes actions until one of the terminal states ($s_{terminal}$) then a new episode starts. In this case, this sum goes until end of the episode instead of ∞.

To define optimal policies, we define $\pi \geq \pi'$ if and only if $V^\pi(s) \geq V^{\pi'}(s)$ for all $s \in S$. This means that the policy π is better than or equal to the policy π' if and only if the value function of following policy π returns better results for each state of the environment. In other words, the optimal policy π^* is the policy which is better than or equal to all other policies. Using same method, the optimal state value function $V^*(s)$ is defined as

$$V^*(s) = \max_\pi V^\pi(s) \tag{2.2}$$

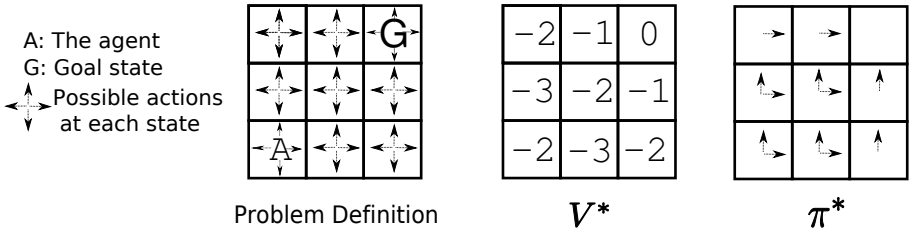

Figure 2.2: 3x3 gridworld as a RL domain

for all $s \in S$, and the optimal state-action value function Q^* is defined as

$$Q^*(s,a) = \max_{\pi} Q^{\pi}(s,a) \tag{2.3}$$

To clarify these concepts, we can consider the classic gridworld problem where an agent tries to go to the goal square located at the top right corner of the $3 * 3$ grid (Figure 2.2). The agent is a robot that moves on the grid, the environment is the grid. The state s_t is defined as the position of the agent at time t , the action a_t is the action that the agent chooses from possible actions (north,east,south,west) and the reward is 0 when the agent passes to the goal state and -1 elsewhere. Figure 2.2 shows the task, optimal state value function, the optimal state-action value function and the optimal policy for the task. Looking at state values, you can see that the goal state is valued as 0 and it decreases gradiently with the states distance to the goal state. For state-action values actions that lead to the goal state are valued as 0 and the other actions are valued according to the state they lead to.

2.1.1 Temporal Difference Learning

For a RL problem, Bellman [1] had shown that if the environment state transition probabilities and the reward functions were available to the agent, it could easily calculate V^* by using dynamic programming methods. On the other hand, considering most of the RL problems, this information is not available to the agent, the agent tries to estimate Q^* without knowledge of the environment model. Monte Carlo methods have this ability to estimate Q^* without requiring the knowledge of the environment model, but they are not suitable as a step by step method. Temporal Difference (TD) methods [15] provides the ability to learn Q^* by combining Dynamic Programming methods which need complete knowledge of the environment and Monte Carlo methods which can learn the optimal state values but not step by step, only after

the final outcome. By combining these methods TD algorithms estimate the value function and update its estimation after each timestep according to its error on previous timestep. One of the popular TD algorithms, Q-learning [21] uses the formula 2.4 to update value of state action pair at time t.

$$Q(s_t, a_t) \leftarrow (1 - \alpha)Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_a Q(s_{t-1}, a_{t-1}) \right] \tag{2.4}$$

According to this formula, Q-learning agent updates its estimation of $Q(s_t, a_t)$ so that it becomes closer to the new estimation $r_{t+1} + \gamma \max_a Q(s_{t-1}, a_{t-1})$. The $\alpha \in [0, 1]$ parameter is the learning rate signifying the amount of the update applied to the old value estimation. If it is 1, the agent instantly replaces the old estimation with the new one.

Algorithm 1: Algorithm for Q-LEARNING

Initialize $Q(s, a)$;

foreach *episode* **do**

  Initialize s;

  foreach *step* **do**

    choose action $a \in A$ using policy derived from Q;

    do action a and obtain: $s^{'} \in S$,$r \in R$;

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

    $s \leftarrow s^{'}$;

  end

end

.

Using the methods explained above, estimation of the optimal state values is updated at each step according to the states and rewards that the agent encounters. On the other hand, this update from experience mechanism brings out the exploration-exploitation dilemma of the RL agent. The quality of the estimation of the agent increases with the amount of exploration, but the agent has to choose best actions to maximize its reward. A common example given on this subject is k one armed bandit problem presented by Thompson W.R. [20]. The agent has to choose one of k-arms of a slot machine (bandit). The agent gets reward according to a probability distribution of the chosen arm and the goal of the agent is to maximize total

reward. To maximize the reward it gets, the agent has to choose between exploration (trying other arms) and exploitation (choosing according to the current estimation of the expected rewards). This problem can be represented as a single state RL problem with k actions. The agent gets the reward r_t according to its action a_{t-1}. The simplest method in exploration-exploitation dilemma is the ϵ-greedy selection [21]. The agent chooses random action with ϵ probability or choose a best profit action with $1 - \epsilon$ probability.

2.1.2 Hierarchical Reinforcement Learning

Hierarchical Reinforcement Learning (HRL) signifies RL algorithms that use a task structure instead of learning a flat policy. They are applied on RL problems with repeated subtasks to reuse previously learned knowledge. They aim to both reduce the size of the state space and shorten the learning time. HRL is based on temporally extended actions which are formally defined as Semi Markov Decision Processes (SMDP). In HRL, the agent learns the problem using a task schema defined by the author, but the definition of these tasks has three different approaches. Sutton and Precup [16] proposed the option method which defines the subtasks as fixed policies described by the author. Parr and Russell [12] proposed The Hierarchy of Abstract Machines (HAM) which defines permitted actions for each task like a partial policy. Third method is MAXQ proposed by Dietterich [4], it represents tasks as policies by defining terminal states and reward functions for each of them. Algorithm 2 gives more detailed information on this method.

Taxi domain is a good example to clarify the HRL methods. The domain is composed of passengers spawning from fixed points and a taxi carrying the passengers. Figure 2.3 illustrates a commonly used example on $5 * 5$ grid with four spawning points: Red, Greed, Blue and Yellow. The taxi must go to the passengers location, pick up the passenger, go to the destination and put down the passenger. The agent's state variables are the source of the passenger, its destination, its own location and whether the passenger is in the taxi. The primitive actions are north, east, west, south, pick up and put down. Looking at the domain structure, one can easily divide the task into two subtasks: "pick up" and "put down" the passenger. Pick up passanger is composed of navigating to the source and pick up action, put down subtask is composed of

Algorithm 2: Algorithm for MAXQ Q-LEARNING

function MAXQ-Q(state s,subtask p)

while p *is not terminated* **do**

$a \leftarrow \pi(s)$; /* Take action a according to the policy π */

if a *is primitive* **then**

$Reward \leftarrow r$

else

$Reward \leftarrow MAXQ - Q(s, a)$; /* Invoke subroutine a */

$TotalReward \leftarrow TotalReward + Reward$

if a *is primitive* **then**

$V(a, s) \leftarrow (1 - \alpha)V(a, s) + \alpha r$

else

$C(p, s, a) \leftarrow (1 - \alpha)C(p, s, a) + \alpha \max_{a'}[V(a^{'}, s^{'}) + C(p, s^{'}, a^{'})]$

end

return $TotalReward$

navigating to the destination and put down action. When dividing the task into these subtasks, we use temporal abstraction, because finishing these subtasks take more than one steps. In addition, we can use state abstraction, because some of the state variables are irrelevant to the subtasks like the destination position for the subtask "pick up the passenger". Moreover we can use subtask sharing between these two tasks which repeatedly require navigating to one of the four points (Red,Green,Blue,Yellow). We can reuse the learned navigation knowledge by sharing this task which will surely shorten learning time. Figure 2.3 shows the complete task decomposition tree for this task, root is the overall task and the leaf nodes are primitive actions.

By dividing the task into subtasks, an important question arises on the optimality of these methods: "Are these methods converge to the optimal policies like flat learning algorithms?". Two types of optimality criteria are considered for these methods: Hierarchical optimality and recursive optimality. When the agent learns the task such that all the subtasks have optimal policies, it is called recursively optimal. In this case, although all the subtasks are optimal, that does not guarantee overall optimality. On the other hand, Hierarchical Optimality means that following policy and its sub-policies will lead to the optimal actions for each state. Most of the HRL research provides recursive optimality, because their main concern is speeding

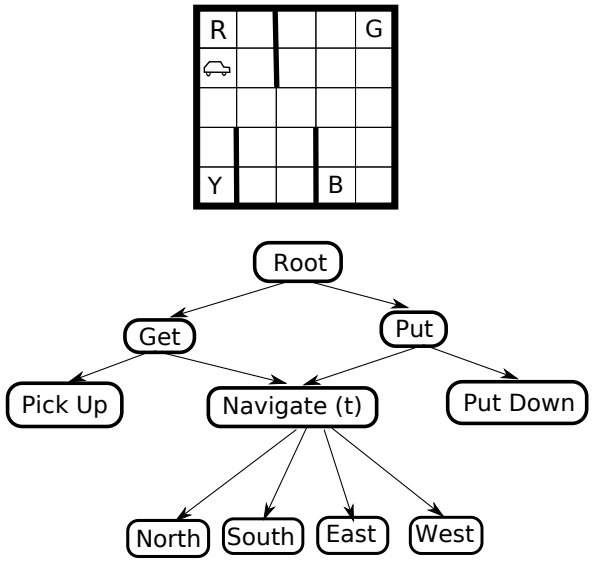

Figure 2.3: Taxi Domain and MAXQ task decomposition

up learning and using abstractions. Ghavamzadeh et al. [7] proposed a method for achieving Hierarchical Optimality in the MAXQ framework by adding expected reward outside the current subtask.

Two rooms maze problem is a simple and clear illustration of these two optimality types. The grid is composed of left and right rooms, and there are two gates between them. The agent starts at the left room, and the goal state is at the right top square. Figure 2.4 shows the subpolicy for passing from the left room to the right room . Policy shown on the figure is recursively optimal, the agent exits from the left room in minimum number of steps, on the other hand marked row causes problems for overall optimality. Considering that the agent will go to the top-right square after the execution of the sub-policy, to exit from south gate increases the number of steps required to reach the goal state. Hierarchically optimal policy will differ from recursive optimal one with these marked squares.

In addition to the optimality criteria, there are various HRL methods differing according to their reward mechanism. Instead of discounted reward system explained above, Ghavamzadeh et al. [8] worked on average reward HRL methods and represented a successful Hierarchically Optimal Average Reward Reinforcement Learning Method [7]. They also extended their work to continuous time problems in addition to discrete time problems[6].

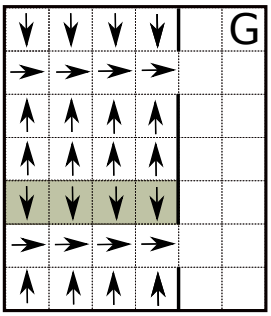

Figure 2.4: Optimality Problem in Two Rooms Maze Domain

2.1.3 Transfer Knowledge on RL

Psychological theories that humans and animals benefit from transfer learning inspired research on Transfer on Reinforcement Learning. Transfer on RL is the notion of using previously learned knowledge on a task (the source), to speed up learning on another task (the target). Given two tasks which possibly have different state or action spaces, the designer finds a mapping between two tasks' value function to use weights learned during training on the source task [10]. For example, consider the keepaway game on robocup soccer where some agents try to hold possession of the ball and the others try to get the ball. While learning 4 vs 3 keepaway game, the agent can use its previous knowledge on 3 vs 2 keepaway version of the game if we can find a mapping between state representations of these two different problems, [19]. On the other hand, giving this mapping manually requires the knowledge on these two domains which is not always available. To make our agent able to discover a good mapping, Taylor et al. [18] directed the research on Transfer RL towards autonomous discovery of the mapping between different tasks.

2.2 Multi-Agent Reinforcement Learning

Multi Agent Systems (MAS) is a subfield of AI that works on complex systems composed of multiple interacting agents. The agents are autonomous, they can share a common goal to cooperate like ant colonies or they can care about their own interests like in stock market economy. As most of the problems' requires a multi agent system representation, the research on MAS attracted much attention in recent years. Despite the research on MAS is not concentrated only on learning, we will analyze learning (specifically RL) perspective of the domain.

Further information on the topic can be found in survey of Stone and Veloso on MAS and Learning [14].

When we consider Reinforcement Learing perspective, the difference between MAS and single agent system is more than number of the agents. Even one additional agent increases the problems complexity dramatically, because the dynamics of the environment changes with the other agents actions. For single agent domains, we explained optimal policies and the algorithms converging to the optimal policies, but when multiple agents are involved in the problem, the optimal policy is not only dependent on the learning agent, but also on other agents' choices. In addition to learning the propoerties of the domain, the agent has to adopt to other agents behaviors, learn to cooperate with or compete against them. Moreover when other agents policies are dynamic (i.e. learning agents) the level of difficulty increases.

Multi-Agent RL algorithms can be classified according to the problems settings. As an example one important setting is the agents controlling mechanism. The controlling mechanism can be centralized. As this type of problems can be treated as single agent problems by using single decision mechanism controlling multiple agents (taking multiple actions instead of one action), ongoing research is concentrated on decentralized problems which forces the agents to give decisions by themselves [3]. In addition, the agents ability to share their knowledge, their vision or their ability to communicate (and cost of the communication) creates completely different problem structures, but the commonly used problem setting is where the agents are independent and they have limited or no communication with other agents.

For these Multi-Agent domains, to develop RL algorithms applicable to cooperative/competitive tasks, at first, the researchers like M.L.Littman [9] and M.Tan [17] worked on producing multi-agent RL algorithms by expanding standard selfish RL algorithms. With the research on hierarchical reinforcement learning decomposing the task into subtasks, layered learning methods with cooperative subtasks or schemas are developed to improve the agents ability to cooperate with others. For example, Stone [13] applied layered learning to robocup soccer and Erus and Polat [5] developed layered learning for hunter prey problem. As a generalized Multi-Agent HRL method Makar et al. [11] extended MAXQ framework to multi agent case to develop Hierarchical Multi Agent Reinforcement Learning algorithm.

CHAPTER 3

PROBLEM DESCRIPTION

3.1 Multi-Agent Learning

Current research on Multi-Agent Learning is about learning cooperative or competitive behavior of agents against each other. The environment provides complex tasks to the agents and the agents try to learn required cooperative or competitive behavior. As explained in previous chapter, the complex part of the problem is that the ”correct action” corresponding to a state is not always same, it changes according to other agents actions. To provide the variety of choices to the learning agents, used testbed must contain a significant number of agents and a significantly big environment for different scenarios. On the other hand, while using the standard state representations, the agent can quickly have a huge state space by increasing the number of agents and the size of the environment. In addition to these problems, as the number of agents increase, complexity of the environment, and difficulty of cooperation increase.

To overcome these problems, the ongoing research is focused on state abstraction and temporal abstraction on multi agent learning algorithms. One solution is dividing the task into subtasks and applying abstractions according to these subtasks. For the domains easily separable to subtasks, the Hierarchical Reinforcement Learning methods are applied and became successfull on reducing learning time and state space. These methods are successfully applied on problems like Taxi domain which is divided to subtasks such as picking up and dropping down. These subtasks need to be done one after the other causing a division of the domain into sequential subtasks. On the other hand, on some multi-agent domains, the agent has multiple choices leading to different goal states. For example it must choose the agent to cooperate

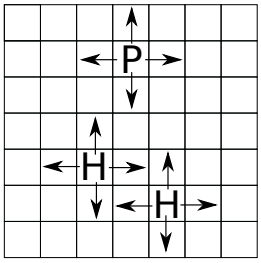

Figure 3.1: Hunter-Prey (Prey-Pursuit) Problem

with or to compete against. In this case, if a task has more than one ways to finish correctly, the standard HRL methods do not consider these choices as subtasks. For the multi-agent case, the structure of the problem makes these choices more important, because they do not only change optimality of the solution, they also change the ability to learn cooperation of the agents. One agent choosing wrong or different way will prevent the agents to reach to the goal state and they will slow down the learning process. Hunter-Prey domain presented in following section is a good example to such problems, when the problem contains numerous hunters and preys.

3.2 Hunter Prey Domain and Variants

Hunter-Prey or Prey-Pursuit Problem is one of the commonly used toy domains for Multi-Agent Systems, because it is suitable to many different approaches and it is capable of illustrating many different scenarios. In addition to non learning multi-agent studies, it is widely used in machine learning, because it addresses different aspects of Multi-Agent Learning with the ability of simulating cooperative or competitive problems with different settings. After it is first introduced by Benda et al. [2], it is used as a multi-agent learning testbed by Tan [17] to test cooperative and individual agents.

The domain is generally composed of a grid and some hunters that try to capture one or more preys by moving in four directions (Figure 3.1), but its various parameters can produce completely different scenarios. For example, the famous gridworld problem can be accepted as a special case of hunter-prey domain where there is only one hunter and the prey is fixed. We will now examine the domain, its parameters and the scenario used in this work.

3.2.1 Different Domain Parameters

The Grid is the main part of the Pursuit Domain that obstacles and agents stand on. As it is the body of the simulation, its properties can significantly change the environment. First and the most important parameter of the grid is whether it is toroidal or not. When the grid is toroidal, the agents move like on an infinite plane. This setting is essential if you want to simulate scenarios where the preys can not be squeezed at the corners or captured by a simple chase. Another important setting is the size of the grid, because as the grid gets bigger, the hunters spend more time on exploring instead of squeezing the prey. Moreover, although bigger grids provide more interesting and difficult challenges, they are not preferred because of the resulting size of the state space. Grid shape is also important, for example using hexagonal grids completely changes the problem, because the number of neighbor squares and corresponding actions increase.

The walls are the obstacles on the grid. For their placement on the grid, there are two main possibilities. First, there can be walls which are randomly positioned at the start of each episode. As they are not fixed, the learning agents state representation must include the positions of the walls. In this case, the problem will get harder, because in addition to the capturing behavior, the agents will be forced to learn navigation on a grid with obstacles. Second, we can use fixed walls to analyze the ability of the agent to learn navigation on a static specific environment. For example, four rooms maze problem is an example to these type of domains. Although the walls are not included in state representation, the agents learn behaviors such as navigation to different rooms.

Considering different hunter and prey versions, they can have many different parameters such as their vision range, moving speeds, etc. As an example, the learning agents vision range is a key parameter of the simulation because it determines whether the domain is fully or partially observable. One classical setting of the agents is having four hunters aiming to surround one random moving prey. This setting is designed to work on learning cooperative behavior. Another commonly used setting is constructed on hunters that try to capture all of the preys on minimum timestep possible. This type of studies test another aspect of cooperation, partitioning the goals to the agents.

3.2.2 Chosen Domain Configuration

As explained in previous section, prey pursuit domain can simulate many different scenarios, but this study prefers specific parameters so that the problem requires strict cooperation of the agents. Unlike other RL studies that prefer small grids with 3 or 4 agents, I aimed to use more agents and grids that are big enough for the number of agents used.

The shape of the grid is chosen toroidal and it does not contain any walls, because the aim of this domain is to test the cooperation of the hunters, the walls are not required for these tests, moreover they will provide selfish hunters the ability to capture the preys without cooperation.

Algorithm 3: Algorithm for Hand-Coded Preys

foreach *square s* **do**
  $danger_s = MAX$
  foreach *hunter h in vision* **do**
    if $distance(s, h) < danger_s$ **then**
      $danger_s \leftarrow distance(s, h)$
  end
end
foreach *direction dir* **do**
  $value_{dir} \leftarrow danger_{s1} + danger_{s2}$ for s1,s2 squares in direction dir
end
$action \leftarrow \underset{dir}{\text{argmax}}(value_{dir})$
return $action$

The selected hunters are individual learners that are not able to communicate, they do not share any information and their goal is to capture one of the preys. A prey is captured when one of its neighboring positions is occupied by a hunter agent. The preys are intelligent, they follow a hand-coded policy instead of a random one. If possible, they try to avoid hunters through a designed nearly optimal hand coded policy. Using this algorithm, at each step, each prey agent determines the direction with safest squares which are furthest to all hunters. Figure 3.2 illustrates an example to this decision process which is given in Algorithm 3. The

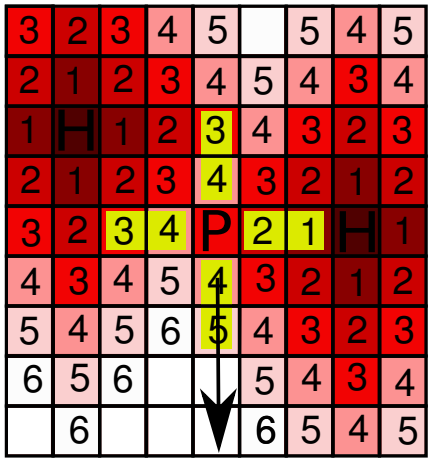

Figure 3.2: Hand-Coded Prey Escape Algorithm

highlighted squares in the figure are the possible escape directions of the prey and the numbers seen are the minimum distance of the hunters. However, the grid is toroidal and the prey that use this algorithm will be invulnerable against hunters. To avoid this, the prey agents have an handicap over hunters: their vision is limited, they can not perceive squares with four or more distance. With these settings the individual hunters can not capture a prey on their own without cooperation with other hunters, but they will be able to via using strategies such as squeezing the prey symmetrically.

One interesting point with these intelligent preys is that they are also difficult to capture when they are chased by a single hunter to make it escape towards another hunter, because their moves are unpredictable when they see only one hunter. For example, if the prey perceives only one hunter placed at 2 squares away in one direction, this hunter will have equal distance to other three directions. This will result in three equally weighted escape directions, making the prey's future position unpredictable for other hunters wanting to cross the prey's path. This topic is further explained in next chapter where hand coded hunters are explained in details.

In addition, as we said earlier the simulation must contain more than one preys to add another dimension to the cooperation of the hunters: They must choose same prey to be able to capture one. In addition to choosing the target prey, we can add a new challenge for the learning agents by adding more hunters to the simulation which forces them to make a decision on who to cooperate with. To sum up, we will have a multi-hunter and multi-prey domain, in which

hunters must cooperate and focus on a specific prey to capture it.

3.2.3 Classic State Representations and Size of The State Space

We saw that although it has many different variations, the aim of the version of the pursuit domain chosen in this study is to develop cooperative learning hunters against preys. To effectively test cooperation level of the agents, we need to provide our agents many different choices by using significant number of hunters and preys. As you will see in following sections, to build this perfect test environment, state space is the most important problem for a flat Q-learning, because we need to include many details about all the other agents for ability of the learning agent to classify current state of the simulation.

While working on a fixed gridworld, the most obvious and the easiest state representation is storing global positions of the agents as illustrated in Figure 3.3. This state representation is mostly used on settings with fixed walls, so that states of the agents can contain information about the environment. For example, if third square is occupied by a wall, instead of enlarging the state space by representing this wall in agents state, the agent will learn that its state variable representing its global position does not change while trying to go to the third square. The walls are not represented but the agent has ability to learn the environment by looking at the combination of its state and action and next state.

While this global position is suitable for single-agent cases or small grid sizes, it causes problems for toroidal or partially observable environments. When we consider a toroidal world without walls, shifting all the agents one square to the same direction results in exactly same situation for the agent, but the state representation gives a completely different state. Figure 3.4 includes an example case to this problem. Of course this is not a barrier for the learning process but it slows it down by representing same situation with $height \times width$ different states.

One alternative state representation to overcome this problem is representing other objects using their relative positions to the agent instead of global positions. Figure 3.4 shows an example to this method. Using such a representation provides the ability to represent the objects easily, but the size of the state space increases polynomially with possible objects. If there is x possible objects for a square (i.e. empty, hunter, prey, wall) and y squares that the

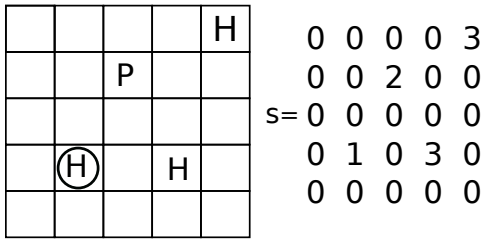

Figure 3.3: Standard State Representation for Prey-Pursuit Domain

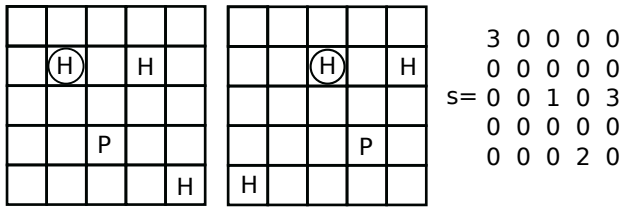

Figure 3.4: Two equivalent map states on a toroidal grid

agent sees (i.e. $11 * 11 = 121$ for an agent with vision limit 5), we have x^y number of states to store (5^{121} in this case). This state representations size is proportional with the vision of the agent which avoids using big grid sizes.

As the domain used do not contain any walls, considering that the agents will be interested only with other agents, not any other objects, we can represent each one of agents using integers representing their relative coordinates according to the learning agent. Figure 3.5 shows the states for the previous example. Using this representation, size of the state space is not proportional with the number of squares that the agent sees, it is proportional with the number of agents on the simulation (n). The size of the state space used is now y^{n-1} instead of x^y (for an agent perceiving up to five distance squares, instead of 3^{121} the size changes to

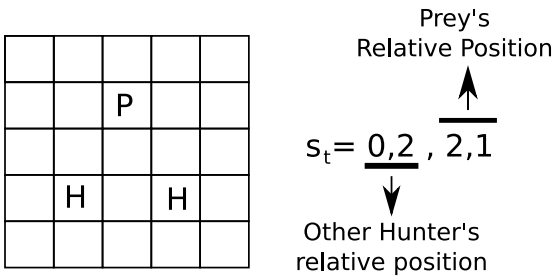

Figure 3.5: Relative Positions State Representation for Prey-Pursuit Domain

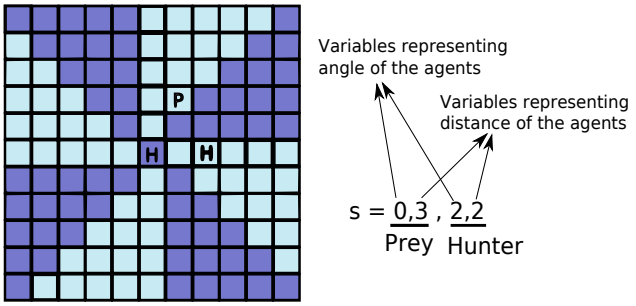

Figure 3.6: 45 Degree Partitioned Polar State Representation

121^{n-1}).

To decrease further size of the state space, instead of using integers for each of the other agents, we can use another commonly used coordination system: polar coordinates. By using polar coordinates, for each agent, we use one variable for the angle and one variable for its distance to the learning agent as seen in Figure 3.6. This representation has the ability to be used as a basic function approximation, because if we reduce the number of partitions, we automatically make the agent approximate close areas. For example, if the area around the agent is divided into 4 90-degree slices, the first state variable will be same for two agents standing on upper left part of the agent.

Table 3.1: Size of The State Space for Three Methods

Size of the state space	Standard	Relative Positions	45-Degree Polar
11x11 Grid with 3 agents	3^{11}	11^4	$(8*10)^2$
25x25 Grid with 3 agents	3^{25}	25^4	$(8*24)^2$
11x11 Grid with 7 agents	3^{11}	11^{12}	$(8*10)^6$
25x25 Grid with 7 agents	3^{25}	25^{12}	$(8*24)^6$

Table 3.1 shows the size of the state spaces for discussed representation alternatives. By using different methods, there is a significant decrease in numbers but still all the methods used have state space exponentially related to the number of agents or size of the grid. Even for 2 hunters and 5 preys, these state representation are not applicable due to their sizes.

CHAPTER 4

PROPOSED METHODS

In previous section, we explained the properties of the problem we have, the domain that we will use and possible problems to encounter. In this section, we will present the method to obtain cooperative agents on big-sized hunter prey domain with multiple preys and hunters. As training the agents directly in the described domain is impossible, this chapter starts with a small pursuit problem containing one prey and reaches in three steps to the goal of multiple agents. First a goal oriented state representation is represented, then this representation is extended for big grids, finally a layered approach is introduced to have successfull hunters against multiple preys. Although the general idea of our approach is not specific to a single domain, first two techniques are developed to make the Hunter Prey domain and size of the state space suitable for the third method used.

4.1 Prey-Oriented State Representation

For reinforcement learning, correctly representing the state is as important as the learning part. We analyzed different state representations for Hunter-Prey domain and their advantages, but we need a new one that is applicable to bigger problems. The previous methods have one thing in common, they are self-oriented which means that the agent perceives the area around itself and the coordinates of other objects are relative to the agent. Here, we propose a prey-oriented state representation which does not represent position of the prey, but represent the hunters around this prey. Figure 4.1 shows an example for this representation. This representation changes the problem structure for the hunters because they must always choose a prey to focus on, but this does not cause any problem because the hunter's goal is to capture one of the preys.

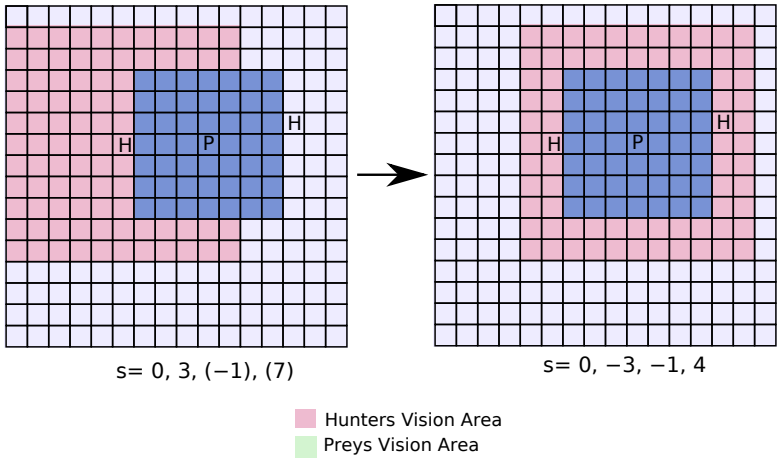

Figure 4.1: Prey Oriented State Representation

The most important benefit of this method is the distribution of the squares represented in the state, they are homogeneously distributed around the target. This benefit plays an important role when the hunters squeeze the prey from two opposite sides. If we look at the case in Figure 4.1, the learning hunter has to coordinate and see the other hunter which is at the other side of the target prey. For this case, assuming that prey's vision is x, the hunters vision range must be at least $2x+5$. As the map size increases, for the agents to see each other from longer distance, this number increases. For this case, the two state variables indicating the other agents position varies between $-(2x+5)$ and $+(2x+5)$, and the agent unnecessarily considers the squares that are on the left side of it where nothing important happens at this timestep. On the other hand, when using prey oriented state representation, an interest limit $i < (2x+5)$ is chosen. This variable indicates how far from the targeted prey the agent will be interested in, and the state variables vary between $-i$ and $+i$. For example, if we consider the case expressed in Figure 4.1, to cover the area that the other hunter stands the state variables interval must be $[-8,8]$ but new state variables must be in interval $[-5,5]$. The number of possible values of each state variable change from 17 to 11 which means a significant decrease of the state space from 17^{2*n} to 11^{2*n} with n representing the number of agents.

Another advantage of this method is its capability to limiting maximum values of state variables using this interest limit and making the size of the state space independent from the agents vision. For greater grid sizes, this is very important because the agent can have full vision of the grid while size of the state space remains same.

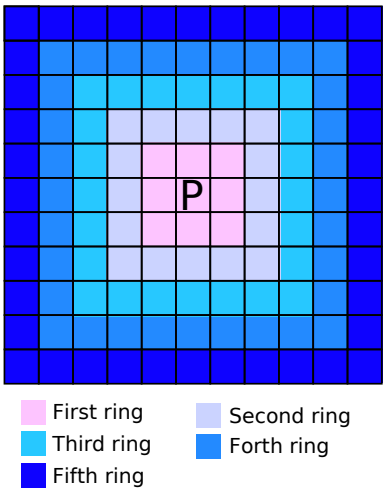

Figure 4.2: Rings Around an Agent

In addition to its benefits explained above, this prey oriented state representation is required for the second part where it is extended for greater grid sizes. Moreover the notion of focusing on a target prey introduced in this section will play an essential role in the third part of this method where the environment will contain multiple preys instead of one.

4.2 Expanding State Space with Mapping

In previous section, it is shown that using a prey oriented state representation reduces the number of possible states, however its benefits are more than that. The learning agents can now decide according to the surrounding squares of target prey but the state space still gets very big when interest limit used increases. To sum up, our agent is capable of learning when it considers a small area around the prey, but problems arise when this area is enlarged. Now, instead of learning on bigger interest rates, this section proposes using the knowledge learned on smaller interest rates.

Considering the problem structure, when the hunters are more distant from the prey, their exact positions are not as important as when they are one or two square away from the target. In this case, knowledge of approximate positions can be enough to decide on the action. As an example, lets consider the case where the prey is far from the agent with relative coordinates 4 North, 12 East. Instead of exact positions, if the agent gets an approximate coordinate for this prey such as somewhere between 4,10 and 4,16, its strategy and chosen action will probably

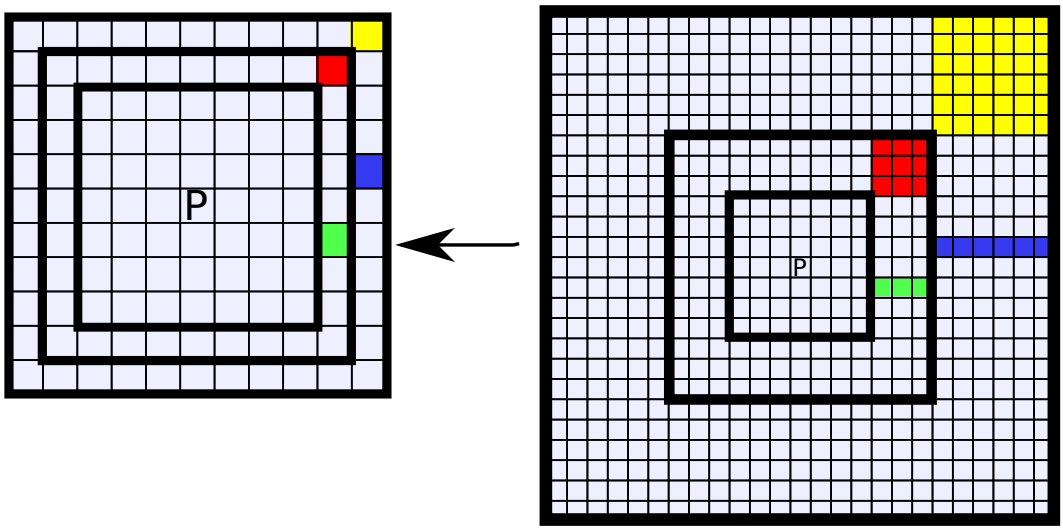

Figure 4.3: Mapped Square Examples for the Mapping Used

be the same. This approximation can be used to create a mapping between two state spaces with lower interest limits and higher interest limits to provide agents ability to move on bigger grids by using mapped state space which is smaller than the required one.

Constructing the perfect mapping is not easy, but one simple method is using virtual rings constructed around the prey. The set of squares that have same distance to the prey (according to 8-neighborhood) are called a ring. Figure 4.2 illustrates the different rings around a prey. To map two state spaces, lets choose 11×11 map and 21×21 map as an example. We must map a field composed of five rings to a field composed of ten rings. For inner rings, the correct positions are important, because shifting an agent one square can change all the agents actions, because of this, first three rings are mapped to themselves, in other words, there is no change for these squares. For outer rings fourth, fifth and sixth rings are mapped to forth ring, and the outer ones are mapped to the fifth ring.

Figure 4.3 shows how the area is partitioned according to this mapping. For example, an agent which has relative coordinates 6 South, 8 East (+6,+8) will assume its state as +4,+5. Indeed for all the cases where the coordinates are between +4,+7 and +6,+10 the agent will use this state. This is like assuming all these squares as one state representing somewhere between +4,+7 and +6,+10.

On the other hand, this method changes the structure of the domain, because when an agent located in one of these squares decides to move closer to the prey, it will move to the closer

square, but its state may not change if it still remains on same partition. This nondeterminism can cause a confusion for a learning agent but considering that the grid is empty it will not cause a problem here, because as explained earlier, the agents will use this mapping after completion of learning process on smaller grid. So, if the agent wants to get closer, it will get closer, and if its state does not change, it will apply same action until it exits its current approximation area causing a change in the state. It can be explained as the agent moves $1/3$ of the difference between two states but the state representation ignores this fraction by flooring the number to an integer.

4.3 Hierarchical Reinforcement Learning

After ability to work on bigger grids, third method will allow the agents to work with multiple preys without increasing size of the state space. The problem is same but it includes more preys and the hunters' goal is to capture one of the preys (not all of them). Looking at the structure of the domain presented in previous chapter, it is explained that to capture one of the preys, two hunters must choose to attack on same prey as their strict cooperation is required to capture the prey. For a domain with multiple preys, from the agents perspective, the task of capturing one of the preys is completely independent of other preys. So, instead of disrupting the agent by adding other preys into its state representation, we can divide the problem into subproblems of capturing each of the preys and reuse capturing knowledge as we saw in Hierarchical Reinforcement Learning methods.

As in HRL methods, ability to reuse the knowledge of a repeating task can be used to speed up learning, but these methods are used with sequential subtasks which must be done in order. For example taxi domain has two subtasks: "pick the passenger" and "put the passenger", and they must be done one after the other. In our case these sub-duties must not be done in sequential order, the agent must choose one of the subtasks. Consider two room maze problem used by Dietterich [4] to explain recursive optimality (Figure 4.4). The agent starts at the left room and the goal is at the right top square. Hierarchical Schema used is composed by two tasks: exiting left room and reaching to the goal. Although exiting the left room is defined as a single task, it has two terminal states and two successfull ends. Here in our problem, we divide such a task into two subtasks: exiting from top and bottom. From these two choices, the agent must choose one according to its benefits.

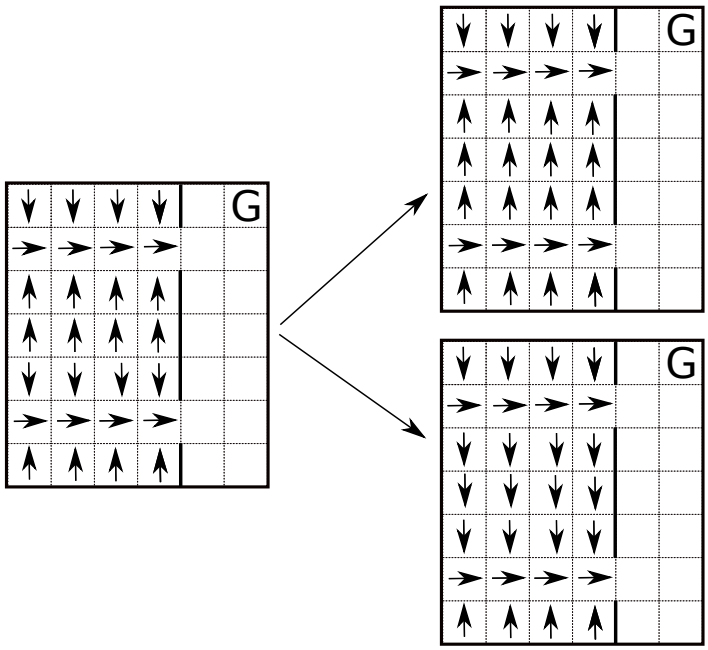

Figure 4.4: Two Rooms Maze Problem Divided to Two Subtasks

This decision of choosing a subtask is related with two notions: the expected duration of the subtask and the value of the subtask for overall task. For the two room maze example, these notions are the distance to each exit square and the exit squares distance to the goal state. When the values of the subtasks are same for the entire task (in the Hunter-Prey case capturing one of the preys ends the episode with a positive reward to all of the hunters), this choice is only dependent to the response of the question 'What will be my position according to its local goal state if I choose one of these subtasks?'. Proving that this decision flow leads to the optimality is not difficult. With the assumption that the agent has the optimal policy and its discounted value table for the subproblems, the agent has a value for each state-action pair of the domain. From the definition of the RL algorithms, the value of the state-action pairs leading to the goal state will have maximum value, and the values of the state-action pairs leading to these states will be the $m * \gamma$. If the agent is n step away from the goal, this value will be multiplied by γ^n. When the agent has multiple choices, it now has a very important information on each of the choices: how many steps they are away from the goal. Despite the possible error, we still get an important information on the situation, current states value for each of the subtask choices.

One important advantage of using this subtask system is its convenience for reducing state

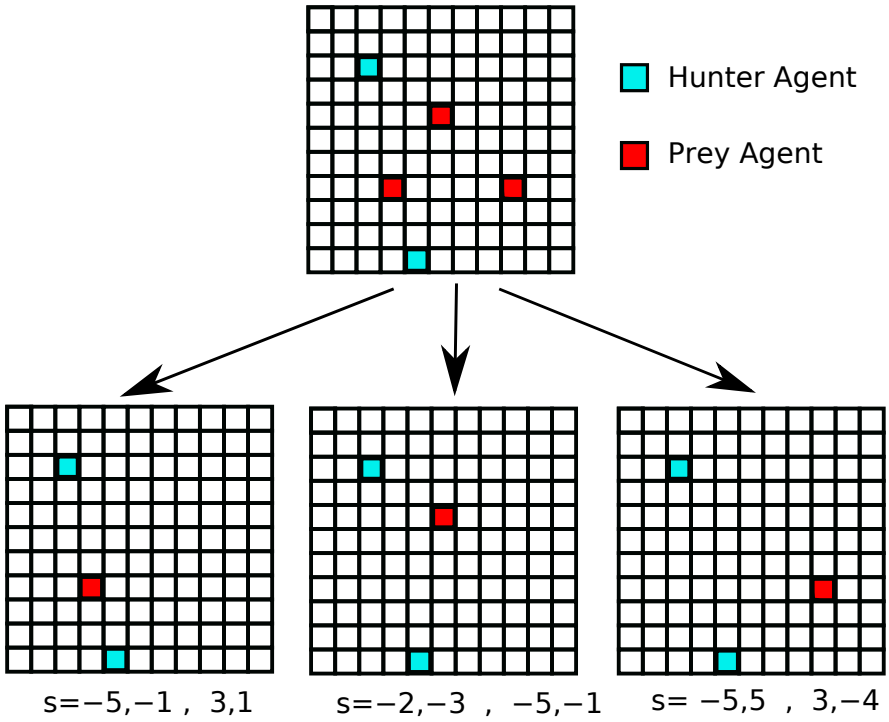

Figure 4.5: Subtasks for a Multi-Prey Problem

space via state abstractions. If the subgoals are independent, and they only need part of the information, the agent uses an automatic state abstraction as it does not need these information on subtasks state representation. Two rooms maze problem provides a good example to this abstraction, when the state representation includes positions of the passages between two rooms. Instead of one policy containing all the passages as state variables, the agent will choose between policies each one using only one of the passage points.

Second, it is explained that in HRL methods, the subtasks have terminal states and continue until the states that you define. Here, the agent does not have to stick to its first choice until the end of the subtask, instead, at each step it has the ability to change its selection via estimating each subtasks value according to the relative current state. This means that while working on a subtask, at intermediate steps if the agent gets a more favorable choice (because of indeterminism or external effects such as other agents) it will definitely switch to this subtask. Moreover if the agent is successful on the subtask independently, the success on its choice is guaranteed.

The Hunter-Prey domain is very suitable for this method, and it changes completely structure and difficulty of the problem. Considering a simulation with two hunters and five preys, the

hunters aim is to capture one of the preys as quickly as possible. To use a flat Q-learning for this problem, one have to include the other hunter and all the five preys in state representation. This does not only result in a huge state space, also learning time and converged policy is far from optimal policy because of the unnecessary information given to the agent at each step. If state abstraction is used to consider position of only one of the preys, the agent has to give a decision on which prey to concentrate, which requires information on all the agents positions. Instead of this, the method explained above divides the problem into sub-problems of capturing each one of the preys as illustrated in Figure 4.5. By using learned knowledge on two hunters and one prey scenario, the agents will have the information about approximately how many steps are required to catch a prey, which gives the ability to determine which prey is easier to capture. Moreover, this method guarantees that two hunters will attack to same prey, because the learned knowledge on 2 vs 1 scenario is symmetrical for two hunters (because they get same reward) and state value for each prey will be same for two hunters.

CHAPTER 5

EXPERIMENTS AND RESULTS

In this chapter, Hunter-Prey simulation experiments are conducted to test the methods proposed in previous chapter. The general domain settings chosen are described in Chapter 3. The task is accepted as episodic, and each episode starts with randomly positioned agents and lasts maximum 120 steps or until the hunters captures one of the preys. As learning algorithm, all the learning agents use Q-learning with ϵ-greedy exploration with parameters $\alpha = 0.5$ and $\gamma = 0.9$. Although different values for these parameters may change learning speed and convergence, they do not affect the results that this work is interested in.

The experiments are separated into three parts. First part uses a small grid to test the state representation method proposed and analyzes the tests to obtain the policy required for later stages. The second part shows the results of the mapping proposed to use this policy for bigger grid sizes. Third part presents results of subtasks approach to extend same policy to multiple preys pursuit scenario.

5.1 State Representation

As explained in previous chapter, Prey-Oriented State Representations aim is reducing size of the state space. Hence, for the agents having full vision, it is meaningless to compare the state representations effects on the learning performance. The learning results will not change, because in both representations each different state of the environment will be represented by unique states, they will only be shifted according to the preys position. On the other hand, if agents have smaller vision, things may change. To show the consequences of these different state representations, I used two different experiment setups.

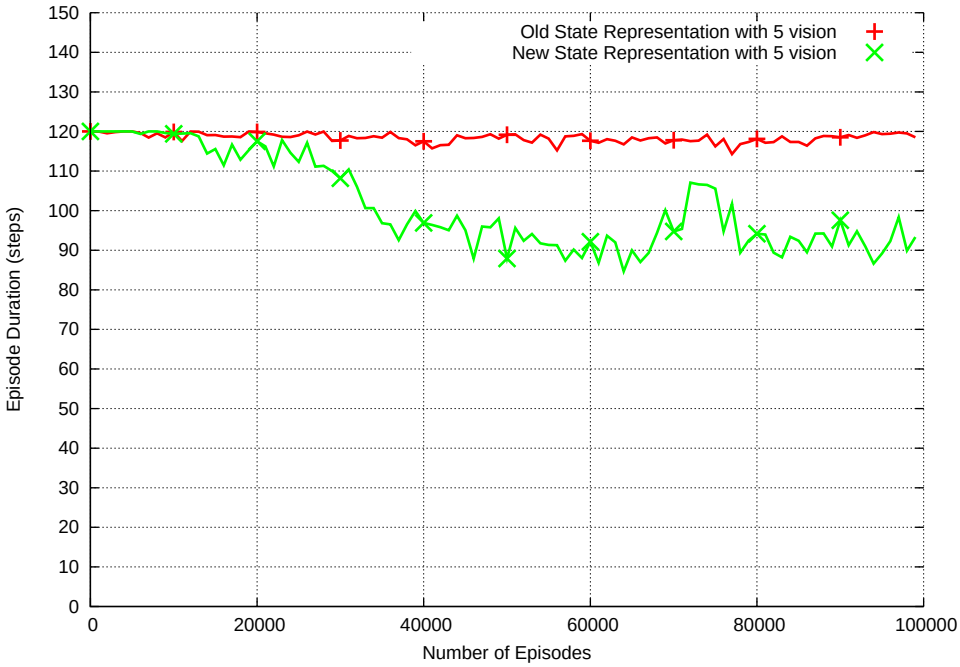

Figure 5.1: Agents with Same Size of State Space on a 15x15 Grid

First experiment aims to show the effects of the new state representation in required state space size. We test the agents ability to learn on partially observable domain using different state spaces with same size. The test environment is composed of a 15x15 grid, one hand coded prey and two hunters restricted to represent 121 of 225 squares in their state representation. The classic hunters use 121 squares around itself, the second type of hunters use the 121 squares around the prey instead.

Table 5.1: Hunters with $11 * 11$ state space on $15 * 15$ grid

	Average	Capture %	Av. Steps for Successful Episodes
Standard State Representation	117.9	2%	N/A
Prey Centered State Representation	93.2	32.2%	37.2

Figure 5.1 shows learning graphs of two types of hunters and reveals that classic hunters are not able to decrease their average episode time. On the other hand newly designed hunters learn to capture and decrease their average to 95 steps. This difference between two agents are consequence of the distribution of the squares effecting the hunters states, when they are

distributed around the prey, the state space with same size allows the agent to perceive two hunters placed at opposite sides of the prey.

At first, this result of 93 steps may seem vary bad for two hunters on a 15×15 grid, but to compare these results with the agents having full vision (which require 10-20 steps to catch), one should definitely consider that this agent can only see 54% of the grid which avoids perceiving either the other hunter or the prey in most of the cases. Table 5.1 shows that this loss of information causes the agents to capture the prey with 32% percentage, but in full vision case, this ratio is 100%. Considering only the episodes that the hunters have success, this average time reduces to 37 steps which is not a bad result for the conditions discussed. Besides, the aim of this state representation is not reducing the steps that learning converges to, it is the ability to capture the prey using smaller state space.

Second experiment is about testing learning capabilities of the learning agent when it has full vision on small grids. This experiment is required to verify learning agents performance on the designed domain and to obtain learned policy needed for the second and third parts of this work. The environment is composed of a 11×11 taurus grid, two learning hunters with 5 cell vision range (covers the grid) and one hand coded prey with 3 vision (covers 7 rows and 7 columns).

This experiment compares four types of learning agents. The first one is the standard Q-learning agent which balances exploitation and exploration by using ϵ-greedy action selection. As explained earlier, obtaining a better policy in this step is important, because this policy will be used in different scenarios to test success of the method proposed. To improve the resultant policy, second agent tested prefers exploration over exploitation. This exploring agent uses full random action selection until episode number 25000, then it linearly decreases ϵ to 0, finally it always chooses to exploit after 50000^{th} episode.

There are also another alternatives to look for a better policy. Although these agents can observe all the environment, and the other agents' actions at each step, these actions are not explicit in state representation. Because of this, taking same action at the same state can lead to many different states according to the actions taken by the prey and the other hunter. This effect can be reduced by altering Q-table with joint actions of hunters instead of actions of the only learning agent. This type of Q-table will store values for every combination of the hunters actions, and each agent$_i$ will find the joint action $A = (a_0, a_1, ..., a_n)$ giving maximum

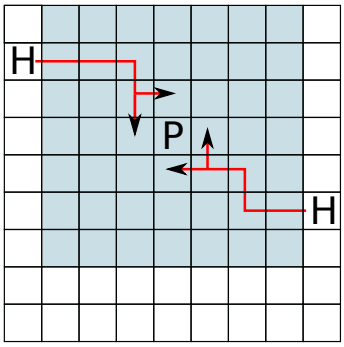

Figure 5.2: Guaranteed Capture Strategy for Hunters

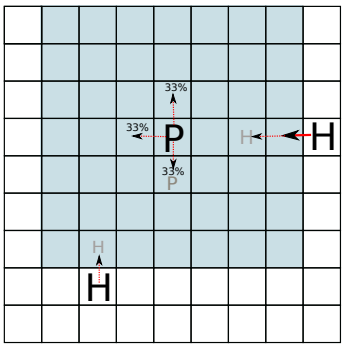

Figure 5.3: Hunters Trying to Capture the Prey by Chasing

value for $Q[s][A]$ and will take the action $a_i \in A$ (they assume that each teammate will choose appropriate action for the joint action A). Third and Fourth agents used in the experiment are joint action versions of the first two types of agents.

In addition to learning ones, we needed a hand-coded agent to compare our results. For the hand coded agent, there are two different possibilities. First one is coordinating the agents outside preys vision, and getting in a position that they will definitely capture the prey in three or four steps. This algorithm requires positioning symmetrically around the prey at the points seen in Figure 5.2 and squeezing the prey cooperatively. Second one is chasing by one of the hunters and hoping that prey moves towards the other agent which stands outside its vision range. Figure 5.3 shows an example to this situation. The hunters has 33% chance to get a capture position, but if they do not succeed, they loose their positioning advantage by getting positioned on same side of the prey, which is far from the positioning that the agents need. Instead of the agents that try this possibility, I used the type of hand coded agents to compare learning agents with deterministic, always capturing hunters. To implement first algorithm,

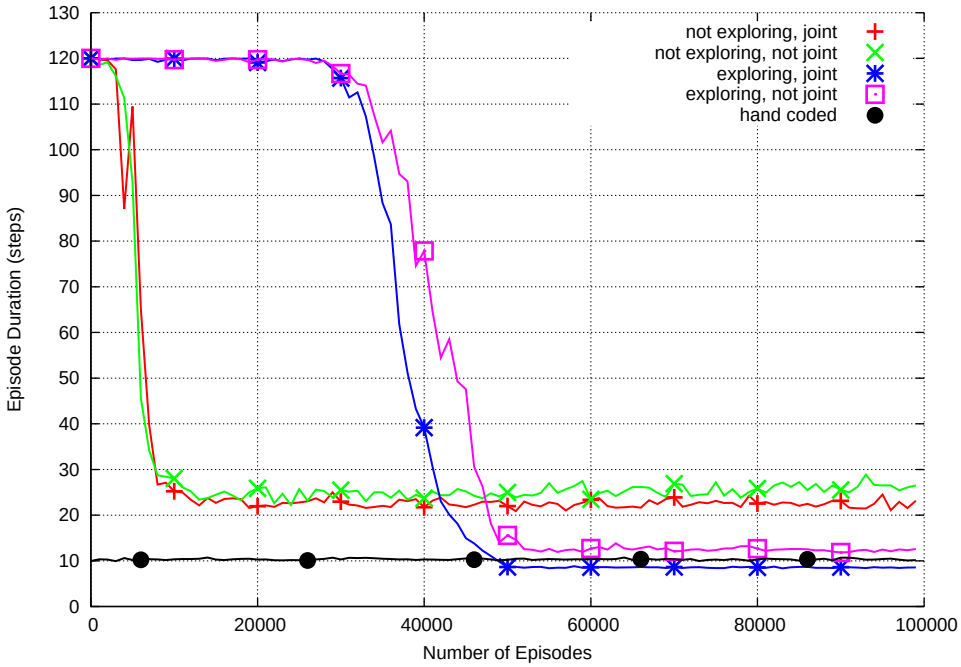

Figure 5.4: Performances of Agents on a 11x11 Grid

the agent finds the shortest path to the symmetric square of other agent and it moves towards this position, and after they get symmetrical in quickest way, they move to the squares seen in 5.2 and catch the prey certainly.

As you can see in Figure 5.4 the ϵ-greedy learning agents can learn to hunt the prey on 11×11 grid. They converge to an average of 25 steps to capture the prey. On the other hand, considering that the grid is very small, 25 steps is not a good result. The exhaustively exploring agents get a better result by reducing their average to 13 steps. Here, as expected, the best result is achieved by exploring agents with joint actions. These agents compete with hand-coded agents, they got an amazing result with an average of 8.5 steps. The most important point of this result is that the agents now have a policy and q-table for the small sized hunter prey problem. We will see its importance clearly when we will use this policy to create learned agents for bigger grid sizes.

To verify these agents performances on a bigger grid, same experience is repeated for 13×13 grid. Figure 5.5 shows that the scores of the agents are normally increased one or two steps, but this increase is the expected effect of the grid size, which also effects the hand-coded agent.

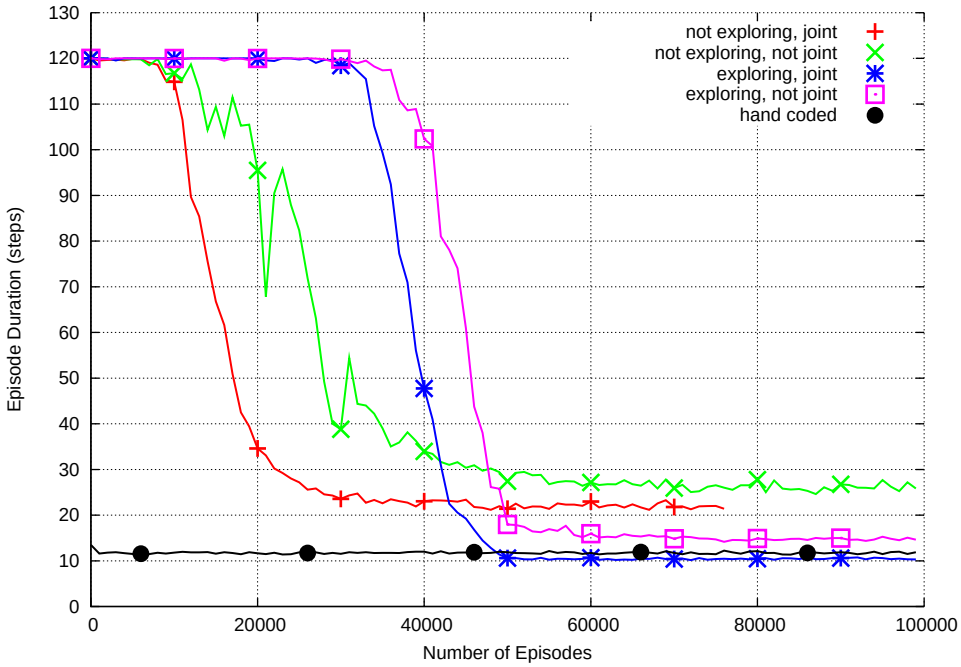

Figure 5.5: Performances of Agents on a 13x13 Grid

5.2 Bigger Map

This section tests and analyzes the results of the method proposed to use small map knowledge on bigger maps. The first experiment is made on 25x25 grid, using agents with full vision of the environment. There are four agents. First one is the agent using pre-loaded state values learned on 11x11 grid tests. The second agent uses same method with 13x13 knowledge. The mapping used for these agents is same as the one explained in Chapter 4. To use 11x11 grid knowledge, first three rings map to themselves, following three rings map to the fourth ring and outer ones map to the fifth ring. For the 13x13 knowledge the state variables range from -6 to 6 instead of -5 to 5 which means one extra ring that can be assigned for the most distant squares.

The main effect of this method is reducing the size of the state space required to use a learning agent on 25x25 grid. On the other hand, we would like to compare these results with flat q-learning covering all the grid, but the size of the state space (25^4) requires a 26 times larger memory to allocate, which prevents us from experimenting flat q-learning agents. Instead, to compare with the learning ones, the results of the hand-coded agents are taken into consideration. The Figure 5.6 shows satisfying results in which extended agents score near to the

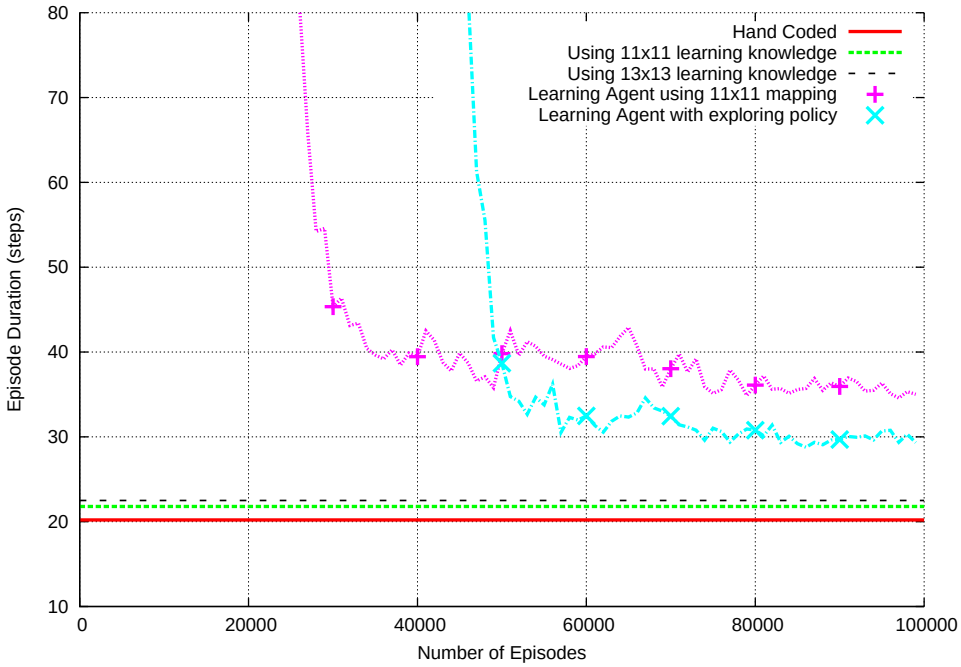

Figure 5.6: Performance of the agents on 25x25 grid

hand coded agent. As the agents loose precision because of the mapping explained, the two or three steps difference between them and hand-coded agents is acceptable.

During the development stage, it is explained that these mappings are not designed to use with learning agents, that they are designed to use previously learned knowledge on smaller maps. They can not be used to reduce size of the state space for first-time learning agents. Despite this statement, to compare learning times, I used same learning agents with an empty q-table and its exploration versions. Figure 5.6 shows that these agents are able to capture the prey but the results are not close to others. They score nearly 30 steps, while the agents that are trained on smaller maps score near 20 steps. The main cause of this difference is the learning algorithm's unsuccess at handling confusion caused by partitioning the state space which was mentioned in Section 4.2.

These results validates that the agents trained in previous section can perform successfully on bigger grids via a basic mapping between state representations. The mapping functions used are logical but probably not optimal, these results can be further improved by testing different mapping functions or combining multiple learning datasets autonomously.

5.3 Multi Prey

This part of our work tests parallel subtasks mechanism described in Section 4.3 and it requires a domain providing the agents with more than one parallel options. Increasing the number of preys is the evident option for such a setup. All the experiments are done on 25x25 grid with multiple hand coded preys and two hunter agents using mapping to previously learned data.

First experiment shows the designed agents performance against five preys. To measure the effect of the task choosing mechanism, it is compared with agents using same learned knowledge combined with different task selection mechanisms. First task choosing mechanism is random task selection at each step, second one randomly selects and insists on the decision until end of the episode. On the other hand, it is shown that the domain and the prey algorithm chosen requires two agents strict concentration on one of the preys. Because of randomness, these two types of agents will have trouble on selecting same prey which will cause unsuccess. To have a more logical comparison, third type of agents select together a random prey to attack. Indeed these agents cheat by communicating but they will provide a better challenge against the designed agents.

Table 5.2: Different Agent Types trying to capture 5 preys

	Average Steps
Random Select	116.9
Random Select and Insist	95.6
Cooperative Random Select	27.2
One Time Select	23.8
Proposed Agent	23.1

Table 5.2 shows a clear victory of the proposed agents. As expected, the agents that select individually and randomly have problems on cooperating. Their score is near to the 120 steps maximum limit, far from the proposed agents. On the other hand communicating random agents score close, but not as good as the designed agents, because their scenario is same as having one prey on the simulation, they do not look for more advantageous choices. Another important result is the difference between the proposed agent which renews its choice at each

step, and the agents that choose only at the start of each episode. The difference between these two agents is very small, but it reflects the designed algorithms ability to change its decision when it gets more advantageous one. This is an important feature, because although they are not designed for parallel subtasks, most HRL algorithms continue to the subtasks until pre-defined terminal states.

Table 5.3: Average Capture Steps for Different Number of Preys

	1 Prey	3 Preys	5 Preys	10 Preys
Proposed Agent	26.8	23.5	23.1	20.2
One Time Choosing Agent	27.3	24.3	23.8	20.8
35*35	39.2	35.0	32.7	30.7
Learning	30.2	N/A	N/A	N/A

In addition, a series of experiments are conducted to analyze the change in average steps required to capture one prey, according to the number of preys on the simulation. This experience will signify the ability of our agents to exploit the possibility of a better positioned prey, when the number of the preys increase. As we see in Table 5.3, the proposed agents are able to capture one of the preys in less time, when the number of the prey increases. This difference is caused by the probability of an advantageous position at the start of the episode, it is not a consequence of the probability to encounter a prey while navigating, because the prey used are not random, and they can easily escape if the hunters do not squeeze them properly. Even if there are too many preys according to the grid size, the hunters can not capture another prey accidentally.

Table 5.4: Size of the State Space for Different Grid Sizes and Number of Preys

Grid Size / Number of Preys	1	3	5
11x11	14641	214358881	3138428376721
15x15	50625	2562890625	129746337890625
25x25	390625	152587890625	59604644775390625

Another important information about the designed agents, is the possible size of the state

space related to the number of the preys if the agents were using flat q-learning as the learning algorithm. Table 5.4 shows us how much progress we made so far by having a successfully learned agent on a multi-agent domain requiring such a big state space. Previous section provided the ability to use 14641 states instead of 390625 (vertical progress by increasing the size of the grid) and this method of choosing subtasks provided the ability to use 14641 instead of 3138428376721 (horizontal progress by increasing the number of agents). In the end, combining these two methods, we obtained successfull agents with 14641 states performing on a domain requiring 59604644775390625 states for a flat Q-learning.

CHAPTER 6

CONCLUSIONS AND FUTURE WORK

This thesis discusses the Hunter Prey problem as a Reinforcement learning domain and a three step approach proposed to reduce the required size of the state space. The agents are trained for a simpler scenario and learned knowledge is used as a subtask knowledge in a HRL method to produce agents performing on more complex tasks.

Experiments presented analyzes success of each part of the proposed method. As expected, prey-oriented state representation reduced the size required for partial observers and prepared the framework for other parts. Mapping the learned knowledge to bigger state spaces increased the size of the grid that can be used with same state space. Finally, the hierarchical learning approach used same state space and learned knowledge to produce agents that can perform on scenarios containing more agents. In conclusion, the state space for this problem became independent of the domain parameters like size of the grid or number of agents. This is an important result because the example given in previous chapter shows that, the agents can use a q-table with size of 14641 instead of 59604644775390625.

In addition to the decreased size of the state space required for learning, despite it can not be tested, learning time of the problem is also decreased. Instead of learning a complex task such as capturing one of the preys in a big environment full of agents, the agent learns to capture the only prey in a smaller environment, and this knowledge is optimally reflected to the overall problem, by using learned policy for each prey as a subtask to be choosen.

Moreover, the third part of the method which uses a layered approach composed of subtasks that use same policy, is not a method strictly specific to this domain. If the problem requires choices between similar subtasks, the method provides a hierarchically optimal policies when the optimal policies for each of the subtasks is provided to the agent. This part of the work can

be analyzed further and be applied to different problems (such as two rooms maze problem) as an extension to this study.

The aim of this work is extending a small scenario to a multiple-prey scenario, but the task is capturing one of these multiple preys. On the other hand, the problem turns into a scenario where there are multiple preys, and the hunters can capture the easiest prey to capture, but the task ends here and the other preys are not taken into consideration after first capture. The hunters learned to capture one of the preys but if the task requires to capture all of the preys, the method used is not sufficient for the problem. For a flat q-learning algorithm, this problem is not applicable due to the reason explained before, but now the agents are capable of capturing the preys, they only require to learn which prey to choose first which transforms the scenario into a goal sharing problem. This can be a further research topic for the hunter-prey domain.

In addition to the scenario discussed above, the scenario used can also be extended to forming groups to cooperate by including more than two hunter agents. The proposed method is perfect for two hunters, because the policy used to capture one prey is symmetrical. This means that current configuration of the grid may be represented by different states for each of the hunters but these different states have same value for these hunters if they use same or similar policy. On the other hand, when there are more than two hunters, the agent must choose the hunter to cooperate with. The same method of using the policy learned on each couple of preys and hunters can be applied but this can cause problems for the scenarios requiring to capture all of the preys. Using this algorithm, each hunter will choose one hunter and one prey such that these three agents form the easiest capturing scenario, but the other hunter chosen may have better options and may not choose this hunter. The agents must also consider these factors while deciding whom to cooperate with. Like previous ones, the solution to this scenario can be an extension to the method proposed.

REFERENCES

[1] R. Bellman. A problem in the sequential design of experiments. *Sankhyā*, 16:221–229, 1956.

[2] M. Benda, V. Jagannathan, and R. Dodhiawala. On optimal cooperation of knowledge sources - an empirical investigation. Technical Report BCS–G2010–28, Boeing Advanced Technology Center, Boeing Computing Services, Seattle, Washington, July 1986.

[3] Daniel S. Bernstein, Robert Givan, Neil Immerman, and Shlomo Zilberstein. The complexity of decentralized control of markov decision processes. In *Mathematics of Operations Research*, page 2002, 2000.

[4] Thomas G. Dietterich. The maxq method for hierarchical reinforcement learning. In *ICML '98: Proceedings of the Fifteenth International Conference on Machine Learning*, pages 118–126, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc.

[5] Guray Erus and Faruk Polat. A layered approach to learning coordination knowledge in multiagent environments. *Applied Intelligence*, 27(3):249–267, 2007.

[6] Mohammad Ghavamzadeh and Sridhar Mahadevan. Continuous-time hierarchical reinforcement learning. In *ICML '01: Proceedings of the Eighteenth International Conference on Machine Learning*, pages 186–193, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

[7] Mohammad Ghavamzadeh and Sridhar Mahadevan. Hierarchically optimal average reward reinforcement learning. In *ICML '02: Proceedings of the Nineteenth International Conference on Machine Learning*, pages 195–202, San Francisco, CA, USA, 2002. Morgan Kaufmann Publishers Inc.

[8] Mohammad Ghavamzadeh and Sridhar Mahadevan. Hierarchical average reward reinforcement learning. *J. Mach. Learn. Res.*, 8:2629–2669, 2007.

[9] Michael L. Littman. Friend-or-foe q-learning in general-sum games. In *ICML '01: Proceedings of the Eighteenth International Conference on Machine Learning*, pages 322–328, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

[10] Yaxin Liu and Peter Stone. Value-function-based transfer for reinforcement learning using structure mapping. In *Proceedings of the Twenty-First National Conference on Artificial Intelligence*, pages 415–20, July 2006.

[11] Rajbala Makar, Sridhar Mahadevan, and Mohammad Ghavamzadeh. Hierarchical multi-agent reinforcement learning. In *AGENTS '01: Proceedings of the fifth international conference on Autonomous agents*, pages 246–253, New York, NY, USA, 2001. ACM.

[12] Ronald Parr and Stuart Russell. Reinforcement learning with hierarchies of machines. In *NIPS '97: Proceedings of the 1997 conference on Advances in neural information processing systems 10*, pages 1043–1049, Cambridge, MA, USA, 1998. MIT Press.

[13] Peter Stone. *Layered Learning in Multiagent Systems: A Winning Approach to Robotic Soccer*. MIT Press, 2000.

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

[15] Richard S. Sutton and Andrew G. Barto. *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1998.

[16] Richard S. Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: a framework for temporal abstraction in reinforcement learning. *Artif. Intell.*, 112(1-2):181–211, 1999.

[17] Ming Tan. Multi-agent reinforcement learning: Independent vs. cooperative agents. In *In Proceedings of the Tenth International Conference on Machine Learning*, pages 330–337. Morgan Kaufmann, 1993.

[18] Matthew E. Taylor, Gregory Kuhlmann, and Peter Stone. Autonomous transfer for reinforcement learning. In *In The Seventh International Joint Conference on Autonomous Agents and Multiagent Systems*, 2008.

[19] Matthew E. Taylor and Peter Stone. Behavior transfer for value-function-based reinforcement learning. In Frank Dignum, Virginia Dignum, Sven Koenig, Sarit Kraus, Munindar P. Singh, and Michael Wooldridge, editors, *The Fourth International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 53–59, New York, NY, July 2005. ACM Press.

[20] W. R. Thompson. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25:285–294, 1933.

[21] C.J.C.H. Watkins and P. Dayan. Q-learning. *Machine learning*, 8:279–292, 1992.