# IMPROVING REINFORCEMENT LEARNING USING DISTINCTIVE CLUES OF THE ENVIRONMENT

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

BY

ALPER DEMİR

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF DOCTOR OF PHILOSOPHY
IN
COMPUTER ENGINEERING

NOVEMBER 2019

Approval of the thesis:

# IMPROVING REINFORCEMENT LEARNING USING DISTINCTIVE CLUES OF THE ENVIRONMENT

submitted by **ALPER DEMİR** in partial fulfillment of the requirements for the degree of **Doctor of Philosophy in Computer Engineering Department, Middle East Technical University** by,

Prof. Dr. Halil Kalıpçılar
Dean, Graduate School of **Natural and Applied Sciences**

Prof. Dr. Halit Oğuztüzün
Head of Department, **Computer Engineering**

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

Dr. Erkin Çilden
Co-supervisor, **STM Defense Tech. Eng. and Trade Inc.**

**Examining Committee Members:**

Prof. Dr. H. Altay Güvenir
Computer Engineering, Bilkent University

Prof. Dr. Faruk Polat
Computer Engineering, METU

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

Assoc. Prof. Dr. Erol Şahin
Computer Engineering, METU

Assoc. Prof. Dr. Mehmet Tan
Computer Engineering, TOBB ETU

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, Surname: Alper Demir

Signature :

# ABSTRACT

## IMPROVING REINFORCEMENT LEARNING USING DISTINCTIVE CLUES OF THE ENVIRONMENT

Demir, Alper
Ph.D., Department of Computer Engineering
Supervisor: Prof. Dr. Faruk Polat
Co-Supervisor : Dr. Erkin Çilden

November 2019, 120 pages

Effective decomposition and abstraction has been shown to improve the performance of Reinforcement Learning. An agent can use the clues from the environment to either partition the problem into sub-problems or get informed about its progress in a given task. In a fully observable environment such clues may come from subgoals while in a partially observable environment they may be provided by unique experiences. The contribution of this thesis is two fold; first improvements over automatic subgoal identification and option generation in fully observable environments is proposed, then an automatic landmark identification and an anchor based guiding mechanism in partially observable environments is introduced. Moreover, for both type of problems, the thesis proposes an overall framework that is shown to outperform baseline learning algorithms on several benchmark domains.

Keywords: Reinforcement Learning, Automatic Subgoal Identification, Options Framework, Automatic Landmark Identification, Anchor Based Guiding

# ÖZ

## ÇEVREDEN GELEN BELİRGİN İPUÇLARINI KULLANARAK PEKİŞTİRMELİ ÖĞRENMEYİ GELİŞTİRME

Demir, Alper
Doktora, Bilgisayar Mühendisliği Bölümü
Tez Yöneticisi: Prof. Dr. Faruk Polat
Ortak Tez Yöneticisi : Dr. Erkin Çilden

Kasım 2019 , 120 sayfa

Etkili ayrıştırma ve soyutlamanın Pekiştirmeli Öğrenme performansını arttırdığı birçok çalışmada gösterilmiştir. Bir etmen, çevrenin ipuçlarını ya sorunu alt sorunlara bölmek ya da verilen bir görevdeki ilerleyişi hakkında bilgilenmek için kullanabilir. Tamamen gözlenebilir bir ortamda bu tür ipuçları, alt hedeflerden gelebilirken, kısmen gözlenebilir bir ortamda ender gözlenen tecrübeler ile sağlanabilir. Bu tezde iki aşamalı bir katkı sunulmuştur; tamamen gözlemlenebilir ortamlarda otomatik alt hedef tanımlama ve seçenek oluşturma konusunda iyileştirmeler önerilirken, otomatik olarak bir yer işareti tanımlaması ve kısmen gözlenebilir ortamlardaki destek noktalarına dayanan bir yönlendirme mekanizması da tanıtılmıştır. Ayrıca, her iki model türü için de tez, birkaç ölçüt problemdeki temel öğrenme algoritmalarından daha iyi performans gösteren genel bir çerçeve önermektedir.

Anahtar Kelimeler: Pekiştirmeli Öğrenme, Otomatik Alt Hedef Tespiti, Seçenekler Çatısı, Otomatik Yer İşareti Tespiti, Destek Nokta Temelli Yönlendirme

*Dedicated to my dear family*

# ACKNOWLEDGMENTS

I would like to start by thanking my supervisor, Prof. Dr. Faruk Polat. His wisdom and passion for research have guided me throughout my academic career. He acted as a father figure, supporting me in every way he could. His professionalism and rational viewpoint have always impressed me. I hope that one day, I could be an academic in his standards.

I would also like to thank my co-supervisor, Dr. Erkin Çilden, to starting me off by giving his code base, examining every minor issue and contributing to this thesis. It is a blessing to have someone like Erkin as he always had the time to discuss on the work, patiently correct my mistakes and help with my academic writing.

I am lucky to meet Hüseyin Aydın, who became a dearest friend over the years. We had many good moments besides beating our brains out on research topics. I have always enjoyed his companion through life. I sincerely hope that we could publish a paper together in the future.

I am grateful to be the child of a two excellent educators, Necmeddin and Emel Demir. My father have always pushed me forward by providing an amazing leadership. Having a Ph.D. was his vision for me and without him I wouldn't know what to do. My courage and passion originates from my mother. As a strong woman, she set a great example for pursuing my goals. I know that she will never give up on me. As art teachers, my parents inspired me by their touching on people's lives. Also, I am so lucky to have a fun sister Başak, that I know, will accomplish great things in life. I love them so much.

Finally, I want to thank my girlfriend Melek. We started dating at the beginning of this Ph.D. Although we lived in different cities, I have always felt at home when we were together. Over the years, we studied, travelled and shared a happy life together. Living in the same city with her was one of the main motivations to complete my thesis in such a short time.

# TABLE OF CONTENTS

ABSTRACT . . . . . . . . . . v

ÖZ . . . . . . . . . . vi

ACKNOWLEDGMENTS . . . . . . . . . . viii

TABLE OF CONTENTS . . . . . . . . . . ix

LIST OF TABLES . . . . . . . . . . xiii

LIST OF FIGURES . . . . . . . . . . xv

LIST OF ABBREVIATIONS . . . . . . . . . . xix

CHAPTERS

1 INTRODUCTION . . . . . . . . . . 1

1.1 Motivation and Problem Definition . . . . . . . . . . 1

1.1.1 Clues of the Environment . . . . . . . . . . 2

1.1.1.1 Subgoals . . . . . . . . . . 2

1.1.1.2 Landmarks . . . . . . . . . . 3

1.1.1.3 Anchors . . . . . . . . . . 4

1.2 Proposed Methods . . . . . . . . . . 5

1.3 Contributions and Novelties . . . . . . . . . . 6

1.4 The Outline of the Thesis . . . . . . . . . . 7

2 BACKGROUND AND RELATED WORK . . . . . . . . . . 9

2.1 Markov Decision Processes . . . . . . . . . . . . . . . . . . . . . . . 9
2.2 Semi Markov Decision Processes . . . . . . . . . . . . . . . . . . . . 10
2.3 Partially Observable Markov Decision Processes . . . . . . . . . . . 10
2.4 Reinforcement Learning . . . . . . . . . . . . . . . . . . . . . . . . . 11
  2.4.1 Q-Learning . . . . . . . . . . . . . . . . . . . . . . . . . . 12
  2.4.2 Sarsa . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 13
  2.4.3 Eligibility Traces . . . . . . . . . . . . . . . . . . . . . . 13
    2.4.3.1 Q($\lambda$) . . . . . . . . . . . . . . . . . . . . . . . 14
    2.4.3.2 Sarsa($\lambda$) . . . . . . . . . . . . . . . . . . . . . 14
2.5 Abstractions in Reinforcement Learning . . . . . . . . . . . . . . . . 15
  2.5.1 Options . . . . . . . . . . . . . . . . . . . . . . . . . . . 15
  2.5.2 Macro-Q Learning . . . . . . . . . . . . . . . . . . . . . . . 16
2.6 Reinforcement Learning with Hidden States . . . . . . . . . . . . . . 16
3 GENERATING EFFECTIVE INITIATION SETS . . . . . . . . . . . . . . . 19
3.1 Automatic Subgoal Identification . . . . . . . . . . . . . . . . . . . 20
3.2 Automatic Option Generation . . . . . . . . . . . . . . . . . . . . . 23
3.3 History Tree Heuristic . . . . . . . . . . . . . . . . . . . . . . . . . 25
3.4 Experiments . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 30
  3.4.1 Problem Domains . . . . . . . . . . . . . . . . . . . . . . . . 30
  3.4.2 Settings . . . . . . . . . . . . . . . . . . . . . . . . . . . 32
  3.4.3 Results and Discussion . . . . . . . . . . . . . . . . . . . . 35
3.5 Summary and Discussion . . . . . . . . . . . . . . . . . . . . . . . . 39
4 A CONCEPT FILTERING APPROACH FOR DIVERSE DENSITY . . . . 41

4.1 Diverse Density . . . . . . . . . . . . . . . . . . . . . . . . . . . 41
4.2 Diverse Density for Automatic Subgoal Identification . . . . . . . . . 42
4.3 Diverse Density with Concept Filtering . . . . . . . . . . . . . . . . 43
  4.3.1 Bridging and Clustering Coefficients . . . . . . . . . . . . . . . 44
  4.3.2 Congestion Ratio . . . . . . . . . . . . . . . . . . . . . . . . 44
  4.3.3 Concept Filtering with Congestion Ratio . . . . . . . . . . . . 46
4.6 Experiments . . . . . . . . . . . . . . . . . . . . . . . . . . . 49
  4.6.1 Problem Domains . . . . . . . . . . . . . . . . . . . . . . . . 49
  4.6.2 Settings . . . . . . . . . . . . . . . . . . . . . . . . . . . 51
  4.6.3 Results and Discussion . . . . . . . . . . . . . . . . . . . . . 51
4.7 Summary and Discussion . . . . . . . . . . . . . . . . . . . . . . 54
5 AUTOMATIC LANDMARK DISCOVERY . . . . . . . . . . . . . . . . . 57
5.1 Landmarks and Landmark-POMDPs . . . . . . . . . . . . . . . . . . 59
5.2 SarsaLandmark . . . . . . . . . . . . . . . . . . . . . . . . . . 59
5.3 Diverse Density with Concept Filtering . . . . . . . . . . . . . . . . 60
5.4 The Extended SarsaLandmark Framework . . . . . . . . . . . . . . . 61
5.5 Experiments . . . . . . . . . . . . . . . . . . . . . . . . . . . 63
  5.5.1 Problem Domains . . . . . . . . . . . . . . . . . . . . . . . . 63
  5.5.2 Settings . . . . . . . . . . . . . . . . . . . . . . . . . . . 69
  5.5.3 Results and Discussion . . . . . . . . . . . . . . . . . . . . . 70
5.6 Summary and Discussion . . . . . . . . . . . . . . . . . . . . . . 73
6 ANCHOR BASED GUIDANCE . . . . . . . . . . . . . . . . . . . . . 75
6.1 State Estimation . . . . . . . . . . . . . . . . . . . . . . . . . 76

6.2 Anchors . . . . . . . . . . . . . . . . . . . . . . . . . . . . 77

6.4 Reward Shaping . . . . . . . . . . . . . . . . . . . . . . . . . 79

6.5 Intrinsic Motivation . . . . . . . . . . . . . . . . . . . . . . . 81

6.6 Anchor Based Guidance . . . . . . . . . . . . . . . . . . . . . 82

6.8 Automatic Anchor Discovery . . . . . . . . . . . . . . . . . . . 86

6.9 Experiments . . . . . . . . . . . . . . . . . . . . . . . . . . . 87

6.9.1 Problem Domains . . . . . . . . . . . . . . . . . . . . . 88

6.9.2 Settings . . . . . . . . . . . . . . . . . . . . . . . . . . 92

6.9.3 Learning Performances . . . . . . . . . . . . . . . . . . 93

6.9.4 Analysis on Guidance with Subsets of Anchors . . . . . . . . 98

6.10 Summary and Discussion . . . . . . . . . . . . . . . . . . . . . 99

7 CONCLUSION . . . . . . . . . . . . . . . . . . . . . . . . . . . 101

7.1 Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 101

7.2 Future Work . . . . . . . . . . . . . . . . . . . . . . . . . . . 103

7.3 Final Remarks . . . . . . . . . . . . . . . . . . . . . . . . . . 104

REFERENCES . . . . . . . . . . . . . . . . . . . . . . . . . . . . 107

CURRICULUM VITAE . . . . . . . . . . . . . . . . . . . . . . . . 119

# LIST OF TABLES

TABLES

Table 3.1 Sizes and reference publications given with determinism status of the action's outcomes for the environments used in this chapter. The subscript $F$ represents full observability. . . . . . . . . . . . . . . . . . . . . . 30

Table 3.2 Learning parameters. . . . . . . . . . . . . . . . . . . . . . . . . . 33

Table 3.3 Parameter values used for automatic subgoal identification methods. 34

Table 3.4 Average number of steps to goal per episode for automatic subgoal identification experiments with 95% confidence interval. Standard deviation of the values are given in the parentheses. . . . . . . . . . . . . . . . 36

Table 3.5 Average initiation set sizes with Segmented Q-Cut algorithm given with 95% confidence interval. Standard deviations are given in the parentheses. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 37

Table 3.6 Average memory usage per episode (KB) for automatic subgoal identification experiments with 95% confidence interval. Standard deviation of the values are given in the parentheses. . . . . . . . . . . . . . . . 37

Table 3.7 Average CPU time consumption (msec) per episode for automated experiments with 95% confidence interval. Standard deviation of the values are given in the parentheses. . . . . . . . . . . . . . . . . . . . . . . . 38

Table 4.1 Sizes and reference publications given with determinism status of the action's outcomes for the environments used in this chapter. The subscripts $F$ and $P$ represents full and partial observability. . . . . . . . . . 49

Table 4.2 Reward per step results averaged over 50 experiments. . . . . . . . . 51

Table 4.3 Number of used concepts and time results averaged over 50 experiments. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 54

Table 5.1 Problem sizes, the number of landmarks in the problems (including the goal states) and their reference publications. The problems marked with * are modified versions of their originals and the sub-script $P$ represents partial observability. . . . . . . . . . . . . . . . . . . . . . . . . . 63

Table 5.2 Average number of steps taken in the problem domains. The values are given with their %95 confidence interval. . . . . . . . . . . . . . . . 70

Table 5.3 Number of concepts used for automatic landmark identification. . . 71

Table 6.1 Details of the domains used in the experiments for Anchor Based Guidance. The sub-script $P$ represents partial observability. . . . . . . . . 88

Table 6.2 Details of the estimated state space with different state estimation forms where $\kappa$ is the average aliasing ratio. . . . . . . . . . . . . . . . . . 92

Table 6.3 Anchor identification performance of DDCF under different learning algorithms. Values are given with their lower and upper bound of confidence intervals. . . . . . . . . . . . . . . . . . . . . . . . . . . 94

# LIST OF FIGURES

FIGURES

Figure 1.1 An example task of taxi driving where picking up a passenger is a subgoal to reach the goal of carrying a passenger to a destination. . . . 3

Figure 1.2 An example task of taxi driving in a partially observable environment where the location when the museum is seen on the left is a landmark state. . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4

Figure 1.3 An example task of taxi driving in a partially observable environment where observing a museum on the left is not unique but observing a second one after going straight is an anchor. . . . . . . . . . . . . . . 5

Figure 2.1 Reinforcement learning sketch. . . . . . . . . . . . . . . . . . . . 11

Figure 3.1 `2Rooms2Doors`$_{\mathtt{F}}$ domain with state names where the sub-script $F$ represents full observability. The doorways, $s_{100}$ and $s_{101}$, are subgoal states and the goal state is named as G. . . . . . . . . . . . . . . . 26

Figure 3.2 Comparison of option lag and history tree heuristics for initiation set generation on an example episode. . . . . . . . . . . . . . . . . . . 27

Figure 3.3 Domains used in the experiments. The goal states are marked as $G$ in the grid world domains (except `Taxi`$_{\mathtt{F}}$). . . . . . . . . . . . . . 31

Figure 3.4 Number of steps to reach the goal state for each subgoal identification method with different heuristics in `4Rooms4Doors`$_{\mathtt{F}}$ domain. . 35

Figure 4.1 Sample graphs colored by (a) $BC$ (b) $CR^{(3)}$ (c) $1/CC^{(2)}$ (d) $CR^{(2)}$ values. Brighter is better. . . . . . . . . . . . . . . . . . . . . . 45

Figure 4.2 Domains used in the experiments where the goal states are marked as G. Numbers in `2Rooms1Door`$_{\text{P}}$ indicate the identifiers of observations gathered by the agent when it resides in the corresponding grid cell. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 50

Figure 4.3 Subgoals found in `2Rooms1Door`$_{\text{F}}$ domain. Brighter color means higher number of identification as a subgoal. . . . . . . . . . . . 52

Figure 4.4 Subgoals found in `2Rooms1Door`$_{\text{P}}$ domain. Brighter color means higher number of identification as a subgoal. . . . . . . . . . . . 53

Figure 5.1 SarsaLandmark with MDD/DDCF workflow. Shaded parts are the steps that DDCF introduces to MDD. . . . . . . . . . . . . . . . . . 61

Figure 5.2 Domain sketches. In `ElevatorEscape`$_{\text{P}}$, the light switch, the button and the elevator positions are marked with S, B, E and the goal states and the landmark states are marked with G and L in the other domains. The states marked as L* in `LargeLandmarkGrid`$_{\text{P}}$ are the additional landmark states, emerging due to the observation semantics of the problem. . . . . . . . . . . . . . . . . . . . . . . . . . . . . 64

Figure 5.3 Observation transition graphs of the smaller domains. The goal states and landmark states are labeled as G and L, respectively. . . . . . 66

Figure 5.4 Observation transition graphs of the larger domains. The goal states and landmark states are labeled as G and L, respectively. . . . . . 67

Figure 5.5 Average CPU time usage for DD approaches. . . . . . . . . . . 71

Figure 5.6 Precision and recall comparisons of DDCF against MDD. . . . . 72

Figure 6.1 An example mapping from the set of estimated states $X$ to the set of true states $S$ in a POMDP with hidden states. . . . . . . . . . . 76

Figure 6.2 Example 2D grid world domain where the agent can take four navigational actions and gets observations based on the presence of a wall in those directions. The goal state is marked as G. . . . . . . . . . 78

Figure 6.3 An illustration of an abstract model for a sample grid world domain (`6Rooms`$_{\mathrm{P}}$, see Figure 6.5a for the original sketch). The anchors correspond to the doorways and the goal state is marked with $G$ where circles and dashed lines represent the anchors and transitions between anchors in the abstract model. . . . . . . . . . . . . . . . . . . . . . . . 84

Figure 6.4 The workflow of Reinforcement Learning when the agent is equipped with the coupling of DDCF and ABG. . . . . . . . . . . . . . 86

Figure 6.5 Sketches of the domains used in the experiments for Anchor Based Guidance. The goal states are marked with G in the grid world domains. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 89

Figure 6.6 A simple example in `ToH3`$_{\mathrm{P}}$ problem where seeing only one rod may correspond to multiple configurations but a transition from one to another with *left* action is specific to a single state, creating an anchor formed as $x = o_{t-1}a_{t-1}o_t$. . . . . . . . . . . . . . . . . . . . . . . 90

Figure 6.7 Observation transition graphs of the domains. The goal states are labeled as G. . . . . . . . . . . . . . . . . . . . . . . . . . . . 91

Figure 6.8 Average number of steps taken to reach the goal state in `6-Rooms`$_{\mathrm{P}}$ domain where the state estimation has the form of $x = o_t$. The dashed line represents the best value from the MDP version of the problem and shaded areas are the $95\%$ bootstrapped confidence intervals. 95

Figure 6.9 Average number of steps taken to reach the goal state in `4-Rooms4Hallways`$_{\mathrm{P}}$ domain with different state estimation forms. Shaded areas are the $95\%$ bootstrapped confidence intervals. . . . . . . . . . . 95

Figure 6.10 Average number of steps taken to reach the goal state in `4-Rooms4Hallways`$_{\mathrm{P}}$ domain where the state estimation has the form of $x = o_{t-1}o_t$. The dashed line represents the best value from the MDP version of the problem and shaded areas are the $95\%$ bootstrapped confidence intervals. . . . . . . . . . . . . . . . . . . . . . . . . . . 96

Figure 6.11 Average number of steps taken to reach the goal state in `ToH3`$_{\mathrm{P}}$ domain with different state estimation forms. Shaded areas are the $95\%$ bootstrapped confidence intervals. . . . . . . . . . . . . . . . . . . . . . 97

Figure 6.12 Average number of steps taken to reach the goal state in `ToH3`$_{\mathrm{P}}$ domain where the state estimation has the form of $x = o_{t-1}a_{t-1}o_t$. The dashed line represents the best value from the MDP version of the problem and shaded areas are the $95\%$ bootstrapped confidence intervals. 97

Figure 6.13 Average number of steps to goal in `4Rooms4Hallways`$_{\mathrm{P}}$ when ABG is fed with the different subsets of the anchor set. The percentage of the provided subsets are given in parentheses and the confidence intervals are omitted for better view. . . . . . . . . . . . . . . . . . . . 98

Figure 6.14 Comparison of DDCF + ABG to ABG with the subsets of anchors in terms of average number of steps to goal in `4Rooms4Hallways`$_{\mathrm{P}}$. The percentage of the provided subsets are given in parentheses and shaded areas are the $95\%$ bootstrapped confidence intervals. . . . . . . . 99

# LIST OF ABBREVIATIONS

| | |
|---|---|
| AI | Artificial Intelligence |
| RL | Reinforcement Learning |
| MDP | Markov Decision Process |
| SMDP | Semi-Markov Decision Process |
| POMDP | Partially Observable Markov Decision Process |
| LoBet | Local Betweenness |
| LCut | Local Cuts |
| SegQCut | Segmented Q-Cut |
| ToH | Tower of Hanoi |
| ER | Experience Replay |
| DD | Diverse Density |
| MI | Multiple Instance |
| MDD | McGovern's Diverse Density |
| DDCF | Diverse Density with Concept Filtering |
| ABG | Anchor Based Guiding |
| PBRS | Potential Based Reward Shaping |

# CHAPTER 1

# INTRODUCTION

Learning is an important part of intelligence. Taking lessons from past experiences leads to higher achievements and plays a crucial role in an intelligent creature's prosperity. As a baseline for intelligence, humans are experts on understanding patterns and finding clues in their environment. This skill of ours makes us fast learners and enable us to comprehend abstract notions and solve complex tasks. The speed of learning is especially a key point when each experience may have the potential of danger. Hence, keeping up with the clues of the world is a clever way to overcome such situations.

Reinforcement learning (RL), as many other machine learning methods, focuses not only the final learned concept, but also the efficiency on learning it [1]. A reinforcement learning agent interacts with an environment and learns by feedbacks. Such feedbacks come as rewards or punishments based on the action taken and the aim is to maximize the sum of rewards taken. Thus, the agent forms a policy, a function determining what action to employ in which state. Since the agent is expected to act while learning, the fast convergence to a good policy is the main focus.

## 1.1 Motivation and Problem Definition

A machine learning method faces the scalability problem whenever it operates on real life. As the data grows dramatically, understanding and processing it in a reasonable amount of time becomes the main challenge. To overcome this issue in RL, many offered methods like *partitioning* the problem into smaller and easier chunks, making *abstractions* to shrink the size of the learning space and finding *hierarchies* in the task. These ideas depend on the success of identifying the patterns and clues of the world.

Learning is also challenging when the knowledge about the world is limited. In such a situation, a RL agent has to make the best of its perceptions about the environment. Even though the information from the world is uncertain, there may be some experiences that serve as clues for success.

### 1.1.1 Clues of the Environment

An environment that a RL agent operates on contains natural clues. Such clues may arise from the structure of the problem model or the semantics of the actions. Their presence can be used to increase the speed of learning if they can be identified in the early stages of learning. A clue may point to a partitioning of the problem or show a hierarchy on which states should be visited in which order. Moreover, a clue can be utilized to inform the agent about its progress in the learning task.

#### 1.1.1.1 Subgoals

One type of the clues in an environment is subgoal. A subgoal can be defined as a state that needs to be visited in order to reach the goal [2, 3] or a bottleneck state that enables transition between regions of space [4, 5, 6, 7]. Subgoals help to partition the problem into sub-problems so that these sub-problems can be solved more easily and their solutions are to be used to form the overall solution. They naturally contain a hierarchy in them, thus their presence can improve Reinforcement Learning.

For example, in Figure 1.1, an agent aims to learn the task of driving a taxi. Obviously, this task has two stages; picking up a passenger and carring her/him to the intended location. In this example, picking up the passenger serves as a subgoal to the goal of completing the whole task. By dividing the problem into two, an agent can first learn how to pick up a passenger, then it may make an abstraction over this task and use this abstract action (or macro action) in its learning. Driving the passenger to a destination results in the generation of another abstract action, carrying the passenger.

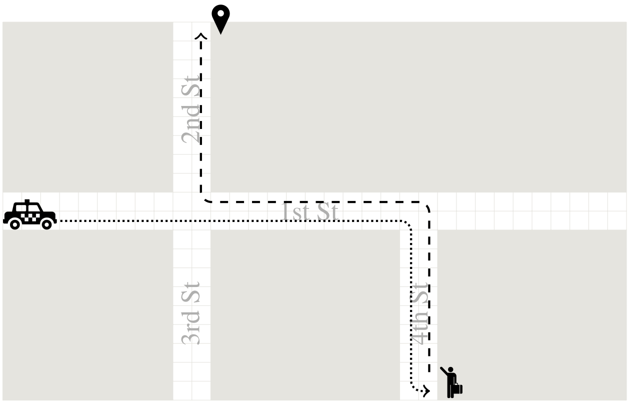


Figure 1.1: An example task of taxi driving where picking up a passenger is a subgoal to reach the goal of carrying a passenger to a destination.

Having these abstract actions improves the learning speed as the agent does not need to consider the primitive actions in each trial anymore.

In Chapter 3, we propose an improvement on the generation of a well known macro action form, called *option*, and experiment on it with the well known subgoal identification methods.

#### 1.1.1.2 Landmarks

Under partial observability, finding clues is both challenging and crucial for learning. When the perceptions are limited and lead to ambiguous observations, an agent has no other way but to hold on to the unique ones such as the ones given by landmark states. A landmark has many definitions over few fields, yet in RL, it is a state that gives a unique observation in partial observable problems [8]. Because states may cause the same observations under limited perception, a landmark acts as an important clue.

Consider the same example of driving a taxi, but now, the streets are not named and they all look like the same as in Figure 1.2. Without any knowledge about the neighbourhood, the agent has to cling on the unique structures. In this example, there is one museum that can be used to navigate towards the passenger to complete the first sub-task. Here, observing a museum on the left is a unique observation in the problem since there is no other location that such an observation can be get. This

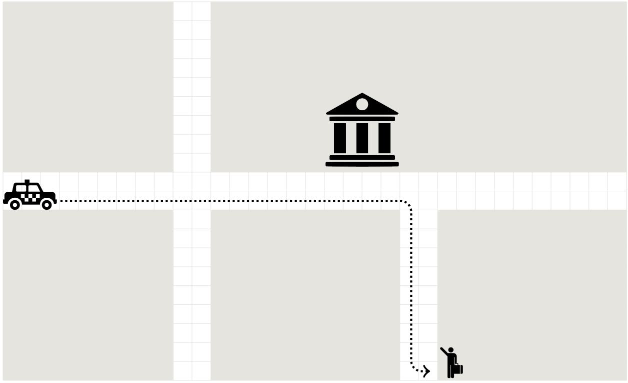


Figure 1.2: An example task of taxi driving in a partially observable environment where the location when the museum is seen on the left is a landmark state.

location, by definition, is a landmark state and it can be used to direct the agent by giving instructions as “take a right turn when you see the museum on the left”.

In Chapter 5, we devised a framework that can identify the landmarks of a partially observable problem and use them in an on-policy algorithm that has been shown to work under such settings [8].

#### 1.1.1.3 Anchors

In the case where uncertainty in the perceptions is high, the agent may have to keep some sort of memory to distinguish the states of the problem from each other. Finding a method to keep a good memory is a challenging task and it may not always end up overcoming ambiguity. In such problems, there may be no landmark states with unique observations but some experiences may be unique. These experiences, that we call *anchors*, can be dependable in an uncertain environment. We define an anchor as a compact information that uniquely maps to a state in a partially observable environment.

In Figure 1.3, the same example of driving a taxi under perceptual aliasing is modified by adding another museum. Assuming that both museums look the same to the agent, now, observing a museum on the left is not unique by itself, hence the problem has no landmark states. However, the sequence of observing one on the left, going straight and observing another on the left is specific to a certain location. When the agent

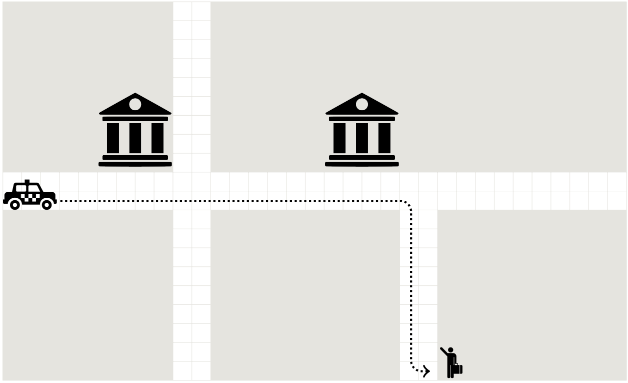


Figure 1.3: An example task of taxi driving in a partially observable environment where observing a museum on the left is not unique but observing a second one after going straight is an anchor.

keeps such a memory, this anchor experience can help it to base its learning on it.

In Chapter 6, we further extend the idea of landmark states to anchors and propose a method to reward the agent when an abstract transition between two anchors occurs in order to help finding a good policy under partial observability.

## 1.2 Proposed Methods

This thesis proposes methods to improve Reinforcement Learning with the help of clues in an environment. It covers both fully and partially observable problems. The methods that we offer are as follows:

- In Chapter 3, a coupling of automatic subgoal identification with an improved option generation mechanism is given. Our heuristic, named *history tree*, focuses on improving the initiation set generation resulting in more effective options,
- In Chapter 4, a concept filtering mechanism for a subgoal identification method, called *Diverse Density* (DD) is introduced. Also, Diverse Density has been shown to work on both fully and partially observable problems for the task of finding clues to success,

- In Chapter 5, we further move our focus to POMDPs with hidden states and more known notion of *landmarks*. We propose an overall framework that can identify the landmarks online and use them in the learning updates of Sarsa-Landmark,
- In Chapter 6, we put a wider definition for clues under partial observability, define *anchors*, and demonstrate a method to inform the agent about its movements by introducing additional rewards. We couple the algorithm with Diverse Density and Concept Filtering (DDCF) to make it a complete framework.

## 1.3 Contributions and Novelties

Throughout this thesis, we have focused on improving Reinforcement Learning by utilizing the clues of the environment. We devise algorithms that helped the agent under both full and limited perceptions.

Our contributions are as follows:

- Our history tree heuristics improves over the greedy approach of including all previously visited states to an option's initiation set in order to generate goal directed options. The method is merged with automatic subgoal identification and forms a complete online algorithm for automatic abstractions in RL,
- We proposed a novel metric, called *congestion ratio*, that is employed in concept filtering in Diverse Density search for subgoals. Eliminating redundant concepts decreases the time and memory consumption of Diverse Density algorithm,
- We experimented with Diverse Density on problems with hidden states and showed that it is useful to identify important observations under partial observability, which is a challenging and unexplored field,
- We employed Diverse Density for automatic landmark identification task which enabled algorithms like SarsaLandmark to work on realistic scenarios since the original algorithm assumes the landmark are known beforehand. Our complete framework is shown to work on several problems,

- We put a wider definition on the unique experiences under uncertainty, called *anchor*, and further utilized such clues to inform the agent about its progress by providing additional rewards. We again used DDCF for online anchor discovery and showed that the proposed algorithm dramatically increases learning speed on different levels of state estimation.

## 1.4 The Outline of the Thesis

The thesis is organized as follows: Chapter 2 establishes a background on the problem that we attacked and states several algorithms that are well accepted in the field. Chapter 3 proposes the history tree heuristics to improve option generation mechanism. Then, Chapter 4 introduces the concept filtering method to a well known subgoal identification algorithm called Diverse Density, in order to improve its computation requirements and experiments on both fully and partially observable problems. Next, the focus of the thesis further moves towards problems with hidden states. In Chapter 5, we further experiment with Diverse Density and our proposed version of it, Diverse Density with Concept Filtering, for the task of finding landmarks online and combine the methods with an on-policy learning algorithm for problems with landmarks, named SarsaLandmark. Finally, Chapter 6 defines anchors and devises a guiding method based on them. It couples DDCF with this guiding algorithm to form a realistic algorithm.

# CHAPTER 2

# BACKGROUND AND RELATED WORK

In this chapter, we provide the necessary background that covers the focus of this thesis. The chapter introduces the environment models, explains well known learning methods on them and summarizes existing related work on the topics of the thesis.

## 2.1 Markov Decision Processes

In RL context, a decision problem is usually modeled by a *Markov decision process* (MDP) which is defined as a tuple $\langle S, A, T, R\rangle$, where

- $S$ is a finite set of *states*,
- $A$ is a finite set of *actions*,
- $T : S \times A \times S \rightarrow [0, 1]$ is a transition function, and
- $R : S \times A \rightarrow \Re$ is the reward function.

$T(s, a, s')$ is a function indicating the probability of being in state $s'$ if action $a$ is performed in state $s$, which has the property $\forall s \in S$, $\forall a \in A$, $\sum_{s' \in S} T(s, a, s') = 1$. It specifies whether a problem is deterministic or stochastic, where a stochastic problem has more than one non-zero values for different $s'$ values unlike a deterministic problem having only one ending state for any state and action pair.

$R(s, a)$ is the immediate reward yielded by the environment after taking action $a$ in state $s$. A problem may have sparse rewards and provide a positive reward only upon reaching a designated goal state. In such settings, it is especially difficult to experience a high reward transition, so that the learning also becomes difficult.

An MDP holds an important property called *Markov property* which is denoted as;

$$P(s_{t+1}|s_1, ..., s_{t-2}, s_{t-1}, s_t) = P(s_{t+1}|s_t)$$

It indicates that the current state of the environment contains enough information about the problem so that the future is independent of the past given the present. In this model, a reinforcement learning agent can base its predictions on the current state without requiring any other information.

### 2.2 Semi Markov Decision Processes

One of the models used in this work is the Semi-MDP (SMDP) which is an abstraction of MDP over time aiming to model transitions with stochastic time duration (i.e. an action can take more than one time step). It is a tuple $\langle S, A, T, R, F\rangle$ where

- the first four terms define an MDP,
- and $F(t|s, a)$ denotes the probability that starting at $s$, action $a$ completes within time $t$ [9].

Obviously, MDP is a special form of SMDP with a step function having a jump at 1 [10]. Importance of SMDP is its ability to model temporal abstractions on an MDP so that improvements can be made both inside the abstracted actions and among the abstractions [10].

### 2.3 Partially Observable Markov Decision Processes

The other model is the Partially Observable MDP (POMDP), which is defined by a tuple $\langle S, A, T, R, \Omega, O\rangle$

- defined by an MDP ($S$, $A$, $T$ and $R$),
- a finite set of observations $\Omega$,
- and an observation function $O : S \times A \times \Omega \rightarrow [0, 1]$.

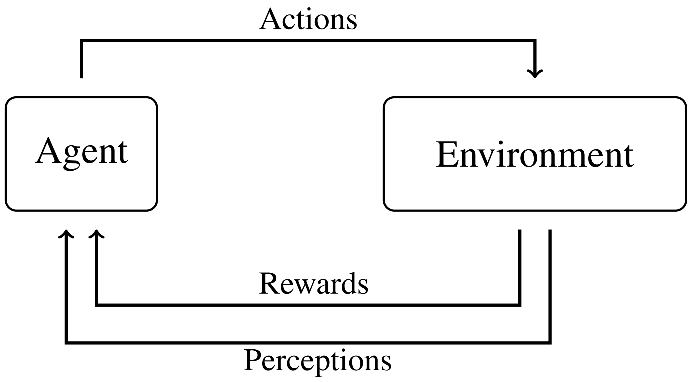


Figure 2.1: Reinforcement learning sketch.

$O(s', a, o)$ represents the probability of getting the observation $o$ after the agent takes the action $a$ in the state $s$ [11]. POMDP is a generalization of MDP that enables to model a partially observable environment.

One of the interpretations of POMDP assumes the set of states are completely hidden, and the model provides a very limited set of observations, violating the Markov property. According to this interpretation, the observation function $O$ can map different states to the same observation, causing a fundamental problem called *perceptual aliasing* [12]. Perceptual aliasing makes it very difficult, sometimes even impossible, to solve the task especially when the optimal actions for these states are different and cannot be identified relying on the same observation.

### 2.4 Reinforcement Learning

Reinforcement learning (Figure 2.1) is a machine learning strategy that aims to learn which action to take on which situation [1]. It forms a model where an agent interacts with an environment, gets perceptions and rewards based on its actions. As the environment dynamics is unknown, the agent needs to discover while learning.

In RL point of view, an environment is modeled by one of the previous forms and the feedbacks provided to the agent is determined by the reward function of the model. The agent aims to map an action to every state so that it learns how to act in the environment.

A specific decision behaviour schema in a RL problem is called a *policy*, defined as $\pi : S \times A \rightarrow [0, 1]$, which is the probability of selecting an action in a state.

The aim of RL is to find the optimal policy $\pi^*$ which maximizes the expected discounted return received by the learning agent, defined as;

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ...$$

where $0 \leq \gamma \leq 1$ is the *discount rate*. Discount rate determines how important the future rewards are. A low discount rate causes agent to put more value on the immediate rewards while a high one leads to more consideration on the future rewards.

The unsupervised characteristic of RL is the fact that the reward and transition functions are initially unknown, otherwise the optimal policy could easily be found using classical dynamic programming techniques. However, $\pi^*$ can still effectively be found by estimating the *value function* (i.e. function giving the value of being in a state on the way to goal) incrementally. *Incremental estimation* approach makes use of the average cumulative rewards over different trajectories obtained by following a policy to calculate the value function and gives rise to the central idea of most RL algorithms, called the *temporal difference* (TD) [13].

### 2.4.1 Q-Learning

A well known TD algorithm using state/action-values (i.e. Q-values) instead of state-values is named Q-Learning [14], and is credited for its simplicity and ease of use. The update rule for Q-Learning is

$$Q(s, a) \leftarrow (1 - \alpha)Q(s, a) + \alpha[r + \gamma \max_{a' \in A} Q(s', a')] \tag{21}$$

where $\alpha \in [0, 1)$ is the *learning rate*, $\gamma \in [0, 1)$ is the *discount factor* and $s$ is the state which the agent employed action $a$ and reached to state $s'$. Q-Learning has been shown to converge to the optimal action-value function under standard stochastic approximation assumptions.

Q-Learning is an off-policy learning algorithm, that is, it uses the action that leads to the highest Q-value in its update rule but not directly the one that the policy chooses.

### 2.4.2 Sarsa

Sarsa is an on-policy learning algorithm that uses the Q-value of the action employed by the current policy [15]. Basically, the update rule is changed to;

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha[r + \gamma Q(s',a')] \tag{22}$$

where $a'$ is the action that is chosen by the current policy. The name of the algorithm originates from the quintuple $\langle s,a,r,s',a' \rangle$.

As the algorithm uses the Q value of the next state and action pair, the learning is focused on the current policy. The difference to the original Q-Learning algorithm resides in the transitions where the agent took a non-greedy action, possibly due to an $\epsilon$-greedy action selection mechanism.

### 2.4.3 Eligibility Traces

As a bridge between the one-step approach of temporal difference methods and Monte Carlo methods, *eligibility traces* were introduced in which the agent leaves decaying traces over the previous transitions and employs the value updates based on these traces.

The traces allow a reward to propagate to the previous transitions much faster, leading a faster convergence. The algorithms that adopted this idea, like Q($\lambda$) and Sarsa($\lambda$), were shown to find good policies [14, 16] where $\lambda$ represents the decay factor of the eligibility traces.

Technically, there are two ways to introduce eligibility traces; traces can be *accumulated* in each time step and decayed after the Q-update or the trace of the current state-action pair can be *replaced* to $1.0$ while all the others decayed and then Q-update can be employed. Due to its better performance, we followed replacing trace technique throughout this thesis [1].

#### 2.4.3.1 Q($\lambda$)

Q($\lambda$) is the adaptation of Q-Learning algorithm that includes eligibility traces [14]. It leaves a decaying trace over the previously visited state-action pairs, representing their eligibility to the current temporal difference update. Although there are different versions of the algorithm, Watkins' Q($\lambda$), which we followed in this study, resets the traces whenever the agent takes a non-greedy action, to keep the algorithm truly off-policy.

At a time step $t$ with transition $\langle s_t, a_t, r_t, s_{t+1}\rangle$, it calculates the temporal difference error $\delta_t = r_t + \gamma \cdot \max_{a' \in A} Q_t(s_{t+1}, a') - Q_t(s_t, a_t)$, and follows the updates;

$$
\begin{gathered}
\eta_t(s_t, a_t) = 1 \\
\forall (s \neq s_t \text{ or } a \neq a_t), \eta_t(s,a) = \begin{cases} 0 & \text{if } Q_{t-1}(s_t, a_t) \neq \max_{a' \in A} Q_{t-1}(s_t, a') \\ \gamma \cdot \eta_{t-1}(s,a) & \text{otherwise} \end{cases} \\
\forall s, a, \quad Q_{t+1}(s,a) = Q_t(s,a) + \alpha \cdot \eta_t(s,a) \cdot \delta_t
\end{gathered}
\tag{23}
$$

where $\eta_t(s,a)$ is the eligibility trace of the state-action pair $\langle s, a\rangle$ at time step $t$, $\alpha$ is the step size and $\lambda$ is the trace decay constant.

Note that this version of the algorithm with replacing traces, resets them if the taken action was non-greedy based on the Q-values in time step $t-1$, i.e. the agent did not take the action with the maximum Q-value.

#### 2.4.3.2 Sarsa($\lambda$)

Sarsa($\lambda$) is a family of on-policy learning algorithms that uses the notion of eligibility traces [16]. It follows the same idea of updating the regular Q values of state-action pairs with an error while leaving a trace over the previously visited state-action pairs. Unlike Q($\lambda$), it uses the next state-action pair's Q-value in the temporal difference error and does not reset the eligibility traces on non-greedy actions, since it is an on-policy learning algorithm.

After the agent experiences the transition $\langle s_t, a_t, r_t, s_{t+1}\rangle$ at time step $t$ in which the agent gathers state $s_{t+1}$ and receives reward $r_t$ by taking action $a_t$ on the state $s_t$, the following update rules are employed in the corresponding order:

$$\begin{aligned} \eta_t(s_t, a_t) &= 1 \\ \forall (s \neq s_t \text{ or } a \neq a_t), \quad \eta_t(s, a) &= \gamma \cdot \lambda \cdot \eta_{t-1}(s, a) \\ \forall s, a, \quad Q_{t+1}(s, a) &= Q_t(s, a) + \alpha \cdot \eta_t(s, a) \cdot \delta_t \end{aligned} \tag{24}$$

where $\eta_t(s, a)$ is the eligibility trace of the state-action pair $\langle s, a \rangle$ at time step $t$, $\alpha$ is the step size, $\lambda$ is the trace decay constant, $\delta_t = r_t + \gamma \cdot Q_t(s_{t+1}, a_{t+1}) - Q_t(s_t, a_t)$ is the TD-error at time step $t$ and $a_{t+1}$ is the action that the agent chooses by its action selection mechanism.

By means of the parameter $0 \leq \lambda \leq 1$, the algorithm maintains a decay over the eligibility traces, together with a reset operation at the end of each episode. Sarsa($\lambda$) is shown to converge to a good policy on both fully and partially observable environments [16].

## 2.5 Abstractions in Reinforcement Learning

Reinforcement learning is a trial-and-error type of learning method and it is important to decrease the number of trials as each trial has the potential of leading a highly negative reward.

One approach to achieve the optimal strategy with the least number of steps is to adopt the divide-and-conquer method. By dividing a given task into sub-tasks, solving each sub-task and merging their solutions, an agent can solve the task much faster.

### 2.5.1 Options

Merging sub-solutions to form the overall solution to a task requires abstractions. One way to form an abstraction over actions in an SMDP model is to use *Options framework* [10]. An *option* is a macro-action, formed by the primitive actions of the underlying MDP, allowing agent to take a multi-step action. Such an option can be used to solve a sub-task and lead to a higher level of learning.

An option $o$ is defined by a tuple $\langle \mathcal{I}, \pi_o, \beta \rangle$ where

- $\mathcal{I}$ is the *initiation set*, $\mathcal{I} \in S$, the set of states that $o$ can be employed at,
- $\pi_o : S \times A \rightarrow [0, 1]$, the local policy of $o$, and
- $\beta : S \rightarrow [0, 1]$, the probability of termination.

An agent employs an option $o$ in a state $s \in \mathcal{I}$, follows $\pi_o$ until $o$ is terminated according to $\beta$. It is common to terminate an option when the agent steps out of the initiation set $\mathcal{I}$.

### 2.5.2 Macro-Q Learning

As the agent forms abstractions and generates options in an SMDP, it needs to use them in the learning process. An adaptation of Q-learning, called Macro-Q Learning, is proposed to serve that purpose [17]. In Macro-Q Learning, we also keep the Q values of state-option pairs, $Q(s, o)$, additional to the Q values of state-action pairs, $Q(s, a)$. The values of state-action pairs are updated according to the regular Update Rule 21 where the value of option $o_t$ which is employed at state $s_t$ at time $t$ is updated according to the following rule;

$$\begin{aligned} Q(s_t, o_t) \leftarrow Q(s_t, o_t) + \alpha(\gamma^n \underset{o'}{max} Q(s_{t+n}, o') - Q(s_t, o_t) \\ + r_{t+1} + \gamma r_{t+2} + ... + \gamma^{n-1} r_{t+n}) \end{aligned} \tag{25}$$

Here, $n$ represents the number of steps that $o_t$ lasted, $s_{t+n}$ represents the state that $o_t$ terminated at, $o'$ is the option from $s_{t+n}$ that has the maximal Q value and $r_{t+i}$ is the reward received at time $t + i$. The update rule uses a discounted reward by the time it is received.

## 2.6 Reinforcement Learning with Hidden States

In a more realistic setting, the agent is not capable of gathering all information regarding the task, yet its perception is somehow limited. Such a model is formed with partially observable MDPs, providing observations rather than states.

There are two interpretations for POMDPs in the literature. One assumes that the MDP structure is either known or estimated, i.e. the agent knows the set of states, the set of observations and the transition function, yet does not know the observation function. In such a setting, the agent is capable of keeping a probability distribution over the set of states, called *belief state* [18], and update it with new observations from the environment by using Bayes rule, leading to the field of Bayesian Reinforcement Learning [19, 20, 21, 22, 23].

The other interpretation, which we followed in this thesis, constitutes a more realistic setting where there is no knowledge about the underlying semantics of the model and the agent only gets observations from the environment. In fact, the agent is clueless on if its sensations are limited to represent the current state of the world.

In a POMDP with *hidden states* [24], the agent has to find a policy based only on the observations from the environment. Since such a setting creates a non-Markovian problem, it is not always feasible to find the optimal action depending only on the observations due to perceptual aliasing. In fact, it has been shown that the regular RL algorithms based on the most recent observation, such as Q-Learning [14], fail to converge to a good policy when the agent's perception is limited [25].

In such a partially observable environment, the agent has to estimate the true state by employing additional approaches. A state estimate is the agent's representation of the current state in the environment and the agent aims to find a policy defined over the set of estimated states. Such an estimate can be formed by a fixed length memory [26], by keeping the previous observation-action pairs. A better way is to extend the memory whenever it is required to form a good policy, leading to variable length memory approaches such as Utile Suffix Memory (USM), Nearest Sequence Memory (NSM) and U-Tree [24]. Also, more complex solutions such as Long Short-Term Memory(LSTM) has been shown to work in these settings [27]. Finally, eligibility traces are also used in POMDPs with hidden states where the state variable $s$ in the update rules of the algorithms like Q($\lambda$) and Sarsa($\lambda$) are replaced with the estimated state variable $x$.

# CHAPTER 3

# GENERATING EFFECTIVE INITIATION SETS

One of the challenges in reinforcement learning [1], as in most of the machine learning frameworks, is the adverse effect of the problem size to time complexity of the solution procedure. A classic approach to cope with this problem is the *divide and conquer* strategy. Essentially based on this strategy, there are a few approaches trying to diminish the adverse effects of dimensionality problem. The idea is that, if one can divide the problem into sub-problems, it may be easier to solve the sub-problems first (also potentially eliminating some repetitions in solution effort by reuse of sub-solutions), and then combine the sub-solutions to achieve the answer to the overall problem. In the solution point of view, there is a close resemblance of this decomposition with the hierarchical algorithmic structures, handling the sub-solutions as the *abstractions* or *macros* invoked whenever it is beneficial to do so. These attempts gave rise to three prominent families in the area: options framework [10], value function decomposition [28], and hierarchical abstract machines [29].

Among the approaches, *options* framework drew attention due to its generality and ease of implementation, where an *option* is nothing but a time extended *abstract action* (or *macro action*). Since the original definition of the framework assumes that the abstractions are provided in some way before learning, it becomes more effective when coupled with a problem partitioning scheme, like *subgoal discovery* [3, 4].

Definition of an option involves an initiation set (states at which an option may start) and a termination condition (how an option terminates). Partitioning mechanisms, like subgoal discovery, usually deal with how to accurately identify the termination condition, since its quality determines effectiveness of the partitioning. However, very few studies deal with the initiation set (actually, almost none of them has its sole focus on the initiation set generation), leaving the whole burden to the related action selection strategy.

In this chapter, we claim that how you select the option starting points is as important as how you determine the termination points, and can have a direct impact on solution quality. Thus, we propose a novel approach to generate effective initiation sets so that higher quality options can be derived, increasing learning speed. Our proposed approach [30] can provide advantage to the agent in the systems that requires reasoning under uncertainty, so that it is universal to the problems that can be modeled as a Markov Decision Process.

## 3.1 Automatic Subgoal Identification

It is known that partitioning the problem into sub-problems boosts learning performance in RL [28, 31]. One of the various approaches is to focus on finding subgoals within the state space, so that they can be used as a partitioning hint. There are number of different automatic subgoal discovery methods which are mainly distinguished based on the types of clues used to seek the bottlenecks. One approach keeps track of the irregularities or peaks of the reward signal, and is not suitable for problems with delayed reinforcement [32]. Another one is based on state observation frequencies, where a state with higher visitation score is a stronger subgoal candidate than others, and is marked so if it passes a statistical filter [2, 33, 4]. These methods usually require problem specific statistical parameter values, and extensive exploration of the problem.

Few researchers tried to make use of the information gathered during RL, like some structural properties or the relative orientation of the reward peak, to identify subgoals [34, 35].

In the graph based methods, a state transition diagram is constructed first, using experiences. Then, different techniques such as clustering are incorporated to separate the strongly connected regions from each other, and the states in between are marked as subgoals [3, 36, 7].

Three graph based methods are in focus of this chapter as they are used in the experiments. Two of them, *Local Betweenness* and *L-Cut*, are proposed by Simsek [4] in her thesis where she models the problem of subgoal identification as a binary classification with target and non-target states. Simsek forms a definition of an *access state* that allows accessing to different regions of the state space. By using metrics working on local information, Simsek's approach keeps observations based on these metrics and feeds them to the following Decision Rule formed with the help of Bayesian decision theory;

$$\frac{n_+}{n} > \frac{\ln\frac{1-q}{1-p}}{\ln\frac{p(1-q)}{q(1-p)}} + \frac{1}{n}\frac{\ln(\frac{\lambda_{fa}}{\lambda_{miss}}\frac{p(N)}{p(T)})}{\ln\frac{p(1-q)}{q(1-p)}} \tag{31}$$

Here, the term "observation" is different then the one in a POMDP model. It can be either a positive or a negative observation determined by a local metric. In the Decision Rule 31, $n$ is the total number of times a state is observed, $n_+$ is the total number of times that a state is observed to be a target candidate, $p$ is the probability of a positive observation given a target state (an access state), $q$ is the probability of a positive observation given a non-target state, $\lambda_{fa}$ is the cost of a false alarm (classifying a non-target state as target), $\lambda_{miss}$ is the cost of a miss (classifying a target state as non-target), $p(N)$ is the prior probability of non-target states and $p(T)$ is the prior probability of target states. Note that $p(T)$ and $p(N)$ represents the prior Bayesian belief observing a target or a non-target state.

By its definition, Decision Rule 31 marks the states which produced positive observations by the used local metric. Since each metric highlights bottleneck states, i.e. states that act as narrow passageways in between regions of state space, a target state in the equation corresponds to a subgoal in this setting. Both of the terms of on the right depend on class conditional probabilities of observations where the second term on the right is also dependent on the prior probabilities and costs of classification errors. Also, the second term prevents initial noisy results and decays as the number of observations increase. Note that, Decision Rule 31 uses parameters that should be guessed or estimated beforehand. For further details, reader is encouraged to check Simsek's thesis [4].

*Local Betweenness* (LoBet) is one of the graph based methods proposed by [4] using local interaction graphs derived through experienced transitions, meaning that the decision is made by using locally collected information. A node in an interaction graph, formed by the experiences of the agent, represents a state where an edge represents a transition between two states. The method is based on the *betweenness* measure of a graph usually credited for its use in social network analysis [37]. Betweenness of a vertex in a graph is defined as the ratio of shortest paths on the graph, between all possible sources and targets, that pass through the vertex of interest to the total number of shortest paths. Local betweenness is the betweenness of a state in a local interaction graph. Formally, the betweenness value of a vertex $v$ is

$$\sum_{s \neq t \neq v} \frac{\sigma_{st}(v)}{\sigma_{st}} w_{st} \tag{32}$$

where $\sigma_{st}$ is the number of shortest paths from vertex $s$ to $t$, $\sigma_{st}(v)$ is the number of such paths that pass through $v$, $w_{st}$ is the weight of the path from $s$ to $t$ as the graph may contain weights (or costs) on edges representing the weight of the transitions. *Local Betweenness* produces positive observations for the states with maximum betweenness value among its neighbours and the observations for each state is then fed to the Decision Rule 31.

*Local Cuts* (LCut) is another local graph based method again proposed by [4], aiming to partition the local interaction graph into two according to a metric called *normalized cut* (*NCut*) introduced by [38]. *NCut* measures the quality of the cut with given two partitions. *NCut* is defined as

$$NCut(A, B) = \frac{cut(A, B)}{vol(A)} + \frac{cut(B, A)}{vol(B)} \tag{33}$$

where $A$ and $B$ are two partitions (two sets of nodes), $cut(A, B)$ is defined as the sum of the weights of the edges that start in $A$ and end in $B$ and $vol(A)$ is defined as the sum of the weights of all edges that start in $A$, and edge weights denote transition frequencies. The algorithm employs spectral clustering for the partitioning of the local interaction graph. Then, it calculates *NCut* value of the partitioning for determining the cut quality. If *NCut* value is lower than a predetermined threshold called *cut threshold* ($t_c$), indicating that the partitions are well separated, then the border states of this cut get a positive observation while the others get a negative one. Again, these observations are fed to the Decision Rule 31 for further elimination.

*Q-Cut* is another automatic subgoal identification approach proposed by [3] which is based on finding border states of the strongly connected areas within the interaction graph. Unlike Simsek's methods, Q-Cut seeks for a *global* criterion that chooses bottlenecks by viewing all state transitions. The algorithm performs a Max-Flow/Min-Cut algorithm on the transition graph built by using the history, paying attention to the arc (i.e. transition) capacities calculated through a relative visit frequency metric. A source state $s$ and a target state $t$ are required for the algorithm to operate, and how to choose these states is problem specific. The quality of the cut is determined by identification of a small number of bottleneck states that can separate balanced chunks of the state space. In *Q-Cut* algorithm, this quality is maintained by the *ratiocut* bipartitioning metric [39]

$$Q[N_s, N_t] = \frac{|N_s||N_t|}{A(N_s, N_t)} \tag{34}$$

where $A(N_s, N_t)$ is the number of arcs connecting both sets ($N_s$, $N_t$), and take into account the cuts whose quality factor is above a domain specific quality threshold ($t_q$) where high $t_q$ means a cut with well separated sets of nodes. Q-Cut works well for problems where one bottleneck state leads to the other, but for many problems, a linear partitioning scheme is not suitable. An extended version of Q-Cut, named *Segmented Q-Cut* (SegQCut) uses bottlenecks discovered so far as a segmentation tool, and iterate over segments by means of a divide-and-conquer approach.

Although they are effective in fully observable domains, these methods require extensive exploration of the domain, and are limited with the capabilities of the underlying graph algorithms. For partially observable problems, on the other hand, there are very few related studies, mostly focusing on temporal abstraction mechanisms without subgoal identification [40].

### 3.2 Automatic Option Generation

By itself, Options framework provides only a formal definition yet does not lead to a method to form an effective option. Most greedy approach may be to focus on the initiation set and the termination condition of an option to shape its effective domain and focus [10].

In RL literature, most studies work on setting the termination condition as options mostly used as a macro action to lead the agent to a particular region in the state space.

Such regions can be the bottleneck parts of the problem, dividing it to sub-problems. Automatic subgoal identification algorithms form options ending in a found subgoal to further make use of the subgoals in the learning process.

On the other hand, forming the initiation set of an option is mostly done greedily. Such an approach includes all the states except the target ones (the states that the option is aimed to) to the initiation sets and expects the agent to learn not to employ an option in a state where the option is not useful. Besides the greedy approach of including all states [41], some studies put restrictions to the states that are going to be included to the initiation set of an option. While Stolle et al. forms a restriction based on reachability to the target state [33], Simsek et al. puts a temporal restriction called *option lag* [42]. Option lag heuristic adds the states seen before a target state (possibly a subgoal) to the initiation set of the option reaching it, with the assumption that there is a path from these states to the target one. The threshold temporal distance from this target state is given with a parameter named *option lag*.

Other studies in the literature concentrate on the repeating sub-sequences in the experiences of the agent instead of finding subgoals [43, 44]. Identifying useful sub-sequences enable the agent to form useful options while learning. Such identification is carried out by keeping past experiences in a memory [43] while some studies improve the storage of experiences by keeping them in a shortcut tree structure, decreasing the memory usage [44].

Options framework is further extended to different types of the problem model. In a problem with continuous state space, skills (or options) can be formed in a way that a skill terminates in the initiation set of the next skill, forming a skill chain [45]. Also, some studies adopt the framework to factored MDPs by focusing on the reachability criteria [46, 47].

The last part of an option, the local policy, is formed with a common method called *Experience Replay* [48]. Experience Replay (ER) stores the previous transitions (state, action, reward, next state tuples) in a memory and replays them to form a policy. In case of option generation, the original rewards taken from the environment are not used. In fact, artificial rewards replace them so that reaching to the target states of the option is further rewarded while leaving the initiation set is punished.

## 3.3 History Tree Heuristic

This section proposes a heuristic method, called *history tree heuristic* [30], to identify useful states to generate initiation sets of options, in the conventional option setting. Our initial findings [49] suggest that a goal oriented initiation set generation based on local history analysis may improve option quality. In majority of the related literature, the initiation set is defined by using a simpler heuristic and the focus is usually on determination of a good termination condition. However, our history tree based heuristic shows that the initiation set of an option is an integral part of the abstraction, and a good heuristic may positively affect the overall learning performance. By extensive experimentation on various problems, we show the effectiveness of our heuristic method, coupled with well known subgoal identification algorithms.

As an integral part of the option generation process, selection of states for the initiation set is important. Guiding the option with some information about the environment characteristics, like the relative orientation of reward peaks, are likely to have positive impact on option performance, especially at the initial stages of learning.

This study aims to improve the widely used greedy approach for initiation sets, which is generating the set using the simple “not in target set” strategy (possibly with some additional restricting criteria), in such a way that it increases the overall option quality. Assuming that the termination conditions are determined in some way beforehand (like subgoal identification, manual design, option transfer etc.) and are potentially useful upon reaching the goal, our intuition is that a target state usually possesses a relative orientation, in terms of short term benefit, with respect to some states in the overall state space. In other words, reaching a state $s$ (designated to a termination condition) from a certain subset of state space can be relatively more beneficial on the way to goal, compared to other portions of the state space from which reaching $s$ is not really meaningful.

One of the popular greedy strategies selects the initiation set states among the ones visited before the target state occurrences within an episode. The number of transitions to check before the target state of the option is an externally supplied parameter called the *option lag* ($o_l$) [42]. In other words, *option lag* limits the number of recently

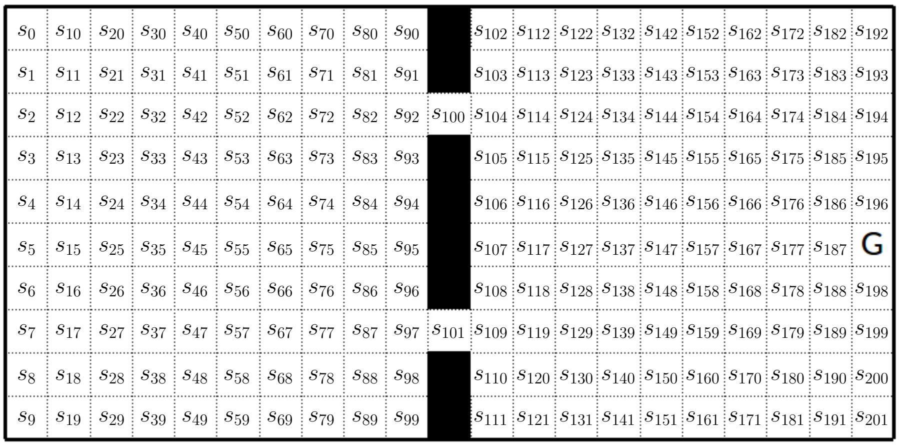
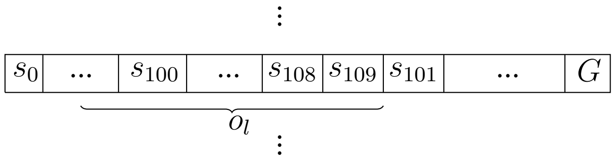

s0 s10 s20 s30 s40 s50 s60 s70 s80 s90 s102 s112 s122 s132 s142 s152 s162 s172 s182 s192
s1 s11 s21 s31 s41 s51 s61 s71 s81 s91 s103 s113 s123 s133 s143 s153 s163 s173 s183 s193
s2 s12 s22 s32 s42 s52 s62 s72 s82 s92 s100 s104 s114 s124 s134 s144 s154 s164 s174 s184 s194
s3 s13 s23 s33 s43 s53 s63 s73 s83 s93 s105 s115 s125 s135 s145 s155 s165 s175 s185 s195
s4 s14 s24 s34 s44 s54 s64 s74 s84 s94 s106 s116 s126 s136 s146 s156 s166 s176 s186 s196
s5 s15 s25 s35 s45 s55 s65 s75 s85 s95 s107 s117 s127 s137 s147 s157 s167 s177 s187 G
s6 s16 s26 s36 s46 s56 s66 s76 s86 s96 s108 s118 s128 s138 s148 s158 s168 s178 s188 s198
s7 s17 s27 s37 s47 s57 s67 s77 s87 s97 s101 s109 s119 s129 s139 s149 s159 s169 s179 s189 s199
s8 s18 s28 s38 s48 s58 s68 s78 s88 s98 s110 s120 s130 s140 s150 s160 s170 s180 s190 s200
s9 s19 s29 s39 s49 s59 s69 s79 s89 s99 s111 s121 s131 s141 s151 s161 s171 s181 s191 s201

Figure 3.1: `2Rooms2Doors`$_{\mathtt{F}}$ domain with state names where the sub-script $F$ represents full observability. The doorways, $s_{100}$ and $s_{101}$, are subgoal states and the goal state is named as G.

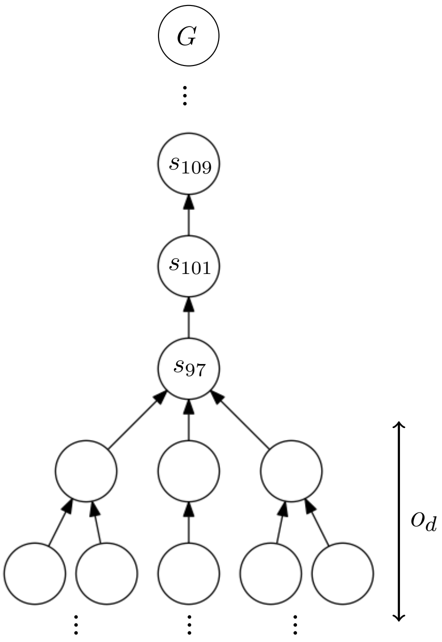
visited states to be included into the initiation set of the option reaching a terminal state. This strategy is based on the idea that there is a path reaching the target state from each state visited before, and a similar ordering is likely to occur again in the future episodes.

Although it seems reasonable, this idea does not always hold. In fact, some states before the target state of an option may not need to visit the target state in order to reach to the goal state. Let us examine such a case in `2Rooms2Doors`$_{\mathtt{F}}$ domain showed in Figure 3.1. Figure 3.2a shows the behaviour of option lag heuristic on an sample episode where the agent travels to the right room by the subgoal $s_{100}$ and goes back to the left room by visiting the subgoal $s_{101}$. In this case, option for reaching $s_{101}$ is not useful from the states on the right room. However, the option lag heuristic ignores such information. It just puts those states into the initiation set, and hopes the action selection mechanism to make the necessary distinction in the future. Unfortunately, this unnecessary option transaction grabs learning time until the utility values saturate, which negatively effects the overall performance of learning. Moreover, the sequence of states in the option lag interval may have loops. In such a case, the set of observed states are much smaller than the interval, causing a smaller initiation set to be generated.

Our proposed approach in Algorithm 1 forms the initiation set of an option by making



(a) Example state history that the agent travels from the right room to the left room through $s_{101}$ in `2Rooms2Doors`$_{\text{F}}$ domain. The states are ordered according to visitation from left to right.



(b) An example tree generated by history tree heuristic on the episode in Figure 3.2a. The circles represent the states and arrows represent the parenthood relationship between the states.

Figure 3.2: Comparison of option lag and history tree heuristics for initiation set generation on an example episode.

a smarter selection of states. Given a target state, our algorithm uses a history tree to include the states that should be directed towards the target state. Such direction is derived by the location of the goal state(s) and only the states that should visit the target one to reach the goal state(s) are included to the initiation set of the option reaching it. Moreover, as the history tree does not contain repetitions, the resulted initiation set is bigger than the one of the option lag heuristic. Figure 3.2b shows how our approach handles the case in the example episode. As the structure of the problem suggests a direction to the right room, our heuristic does not include the states on the

**Algorithm 1** $HISTORY\_TREE\_HEURISTIC$

```
Require: a history $H$
Require: a terminal state $s_t$ {we assume the terminal state is known in advance}
Ensure: initiation set $I$
$s_r \leftarrow$ FIND_ROOT($H$, $s_t$)
$parent(s_r) \leftarrow null$, $\mathcal{V}_{s_r} \leftarrow 0$,
for each episode $e \in H$ ending with $s_r$ do
    $i \leftarrow length(e) - 2$
    while $i \geq 0$ do {calculate state values backwards in history}
        if $\mathcal{V}_{s_i}$ is undefined $\vee$ $(r_{i+1} + \gamma * \mathcal{V}_{s_{i+1}}) > \mathcal{V}_{s_i}$ then
            $\mathcal{V}_{s_i} \leftarrow r_{i+1} + (\gamma * \mathcal{V}_{s_{i+1}})$
            $parent(s_i) \leftarrow s_{i+1}$
        end if{get rid of loops}
        $i \leftarrow i - 1$
    end while
end for
$V \leftarrow \{s_r\}, E \leftarrow \emptyset$
for each state $s \neq s_r$ do
    $V \leftarrow V \cup \{s\}$ {vertices of the tree}
    $E \leftarrow E \cup (s, parent(s))$ {edges of the tree}
end for
$I \leftarrow$ traverse sub-tree of $(V, E)$ rooted by $s_t$ down to depth $o_d$ (excluding $s_t$) and
    collect each visited state
return $I$
```

right room (like $s_{109}$) to the initiation set of the option reaching to $s_{101}$.

Given a target state $s_t$ (possibly a subgoal), Algorithm 1 first establishes a suitable root state $s_r$ by using Algorithm 2 in order to form a tree where each node has a parent node to reach except the root node. Algorithm 1 forms the tree so that a node is directed in a path towards the found root state.

Algorithm 2 finds a possible goal state, visited at the end of an episode, so that it can be used as the root node of the history tree and the new option can be directed at it.

**Algorithm 2** $FIND\_ROOT$

```
Require: a history $H$
Require: a terminal state $s_t$
Ensure: a root state $s_r$
 1: $E_{s_i} \leftarrow 0, \forall s_i \in H$ {number of reward peaked episodes ending with $s_i$ containing $s_t$}
 2: for each episode $e \in H$ do
 3:     Let $s_n$ be the last state of $e$
 4:     Let $r_n$ be the last reward achieved in $e$
 5:     if $s_t \in e$ and $r_j < r_n$ where $1 < j < n$ then
 6:         $E_{s_n} \leftarrow E_{s_n} + 1$
 7:     end if
 8: end for
 9: $s_t \leftarrow \arg\max E_{s_i}$
10: return $s_t$
```

As a problem may have multiple goal states, the algorithm traverses over the episode history of the agent (Algorithm 2, line 2), checks the episodes ending with a reward peak, i.e. episodes where the maximum reward is taken at the end (Algorithm 2, line 5), and returns the last state which the target state $s_t$ is used most to reach (Algorithm 2, lines 9 and 10). Here, the goal is to find an accurate direction for the new option.

With $s_r$ selected as the root node, the algorithm creates a shortest path tree from each visited state to $s_r$ by iterating over the set of all episodes ending with $s_r$ (Algorithm 1, line 3). It keeps a value $\mathcal{V}$ for each state, which is calculated with the discounted rewards taken in the way from a state to the root $s_r$. Then, every sub-history (or episode) is traversed backwards (Algorithm 1, line 5) and a parent state, a state best to go in order to reach the root state, is selected for every visited state (Algorithm 1, lines 6-9). In line 17, a history tree with $V$ vertices and $E$ edges is generated where $V$ consists of the traversed states and $E$ consists of transitions from a state to its parent (Algorithm 1, lines 13-17). Finally, the sub-tree rooted by the target state $s_t$ is traversed down and the states in this sub-tree except $s_t$ are included the initiation set of the option reaching $s_t$ (Algorithm 1, line 18). The traversal depth is controlled by a parameter called *option depth* ($o_d$).

Figure 3.2b illustrates a portion of a tree generated by the algorithm in the example domain of $\texttt{2Rooms2Doors}_\texttt{F}$. The sub-tree traversal starts from $s_{101}$ in the example and ends at the depth $o_d$.

The remaining parts of the option is generated in ways that are identical to the greedy strategy with option lag heuristic. The termination probability of the states in the initiation set are set as $0.0$, and a probability of $1.0$ is assigned for the target states and the policy to reach the terminal state is generated via ER.

## 3.4 Experiments

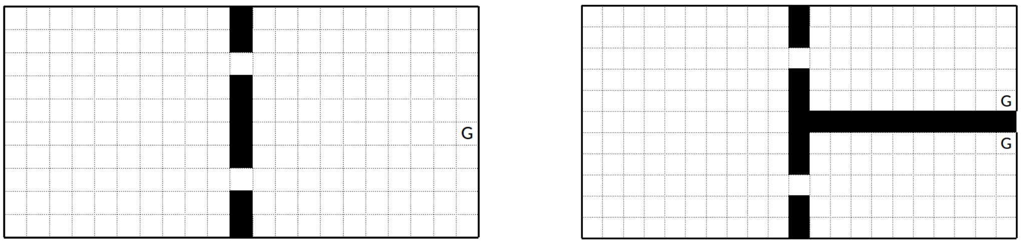
In order to compare the performance of different initiation set generation heuristics, we experimented by coupling option generation with well known automatic subgoal discovery methods, namely Local Betweenness (LoBet), Local Cuts (LCut) and Segmented Q-Cut (SegQCut), so that a completely automatic process is tested. The following sections provide the experiment settings, results and discusses on them.

### 3.4.1 Problem Domains

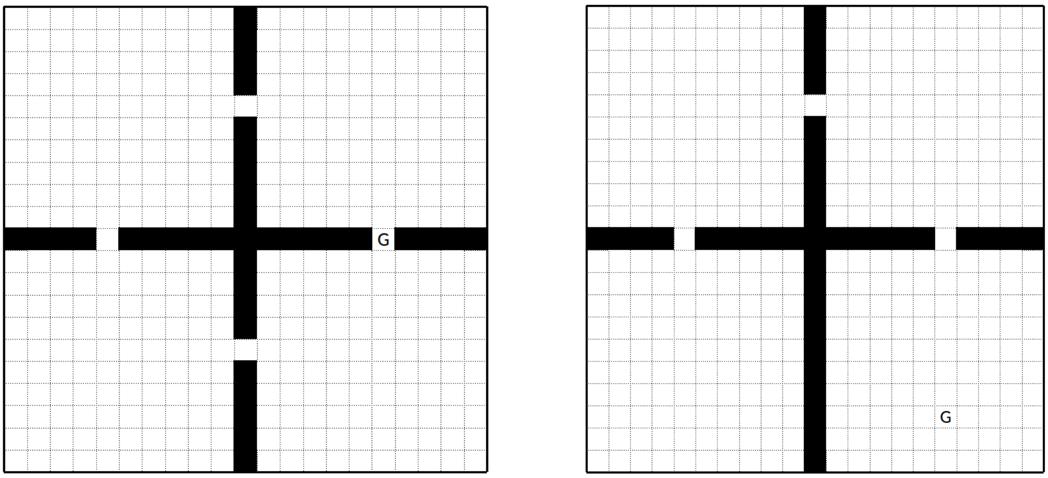
Table 3.1: Sizes and reference publications given with determinism status of the action's outcomes for the environments used in this chapter. The sub-script $F$ represents full observability.

| **Problem** | **\|S\|** | **\|A\|** | **Action Noise** | **Reference** |
|---|---|---|---|---|
| $\texttt{2Rooms2Doors}_\texttt{F}$ | 202 | 4 | Yes | [3] |
| $\texttt{VirtualOffice}_\texttt{F}$ | 212 | 4 | Yes | [50] |
| $\texttt{4Rooms4Doors}_\texttt{F}$ | 404 | 4 | Yes | [2] |
| $\texttt{4Rooms3Doors}_\texttt{F}$ | 403 | 4 | Yes | [2] |
| $\texttt{Taxi}_\texttt{F}$ | 500 | 6 | Yes | [28] |
| $\texttt{ToH5}_\texttt{F}$ | 243 | 6 | No | [51] |

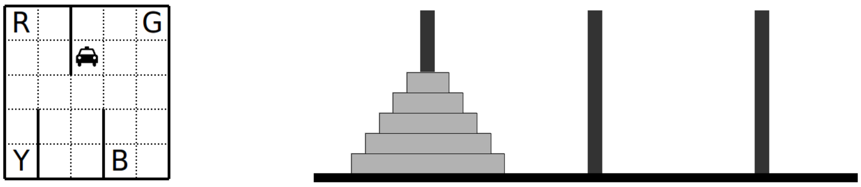
The experiment set of this study (given in Table 3.1) include different versions of grid world domains and deterministic and non-deterministic domains with different number of actions. Sketches of the domains are also given in Figure 3.3.



(a) `2Rooms2Doors`$_{\mathrm{F}}$ (b) `VirtualOffice`$_{\mathrm{F}}$



(c) `4Rooms4Doors`$_{\mathrm{F}}$ (d) `4Rooms3Doors`$_{\mathrm{F}}$



(e) `Taxi`$_{\mathrm{F}}$ (f) `ToH5`$_{\mathrm{F}}$

Figure 3.3: Domains used in the experiments. The goal states are marked as $G$ in the grid world domains (except `Taxi`$_{\mathrm{F}}$).

In grid world domains (`2Rooms2Doors`$_{\mathrm{F}}$, `VirtualOffice`$_{\mathrm{F}}$, `4Rooms4Doors`$_{\mathrm{F}}$ and `4Rooms3Doors`$_{\mathrm{F}}$), the agent is located in the grid cells and can employ four actions as *north*, *east*, *south* and *west*. Actions are non-deterministic, the agent moves to the intended direction with a probability of 0.9 and randomly to any of the movement directions with 0.1 probability. Initially, the agent is located at any cell in the room(s) on west. The agent receives a positive reward of $1.0$ upon reaching the goal state and no reward or punishment for any other movements.

In $\texttt{Taxi}_\texttt{F}$ domain (Figure 3.3e), the agent tries to transfer a passenger from a designated location to his desired location. As in the other grid world problems, the agent can move to the four compass directions, and can perform two additional passenger related actions *pickup* and *putdown*. At any given time, the passenger can be located at one of the cells marked R, G, B, Y, or in the taxi. The *pickup* and *putdown* actions are effective only when the taxi agent is at one of the designated cells. Initially, the taxi agent is in any one of the grid cells, and the passenger is at one of the designated cell locations. Any navigation action succeeds with probability $0.8$, otherwise leading the agent to the left or right with respect to the intended movement direction with probability $0.1$ each. The agent receives $-10.0$ punishment for wrong *pickup*s and *putdown*s and a $+20.0$ reward for a successful transfer of the passenger to its desired location. All other transitions yield a default reward value of $-1.0$.

$\texttt{ToH5}_\texttt{F}$ (Figure 3.3f) problem is the representative of classical puzzle games in our problem set. In its classical definitions, the problem consists of $m$ rods and $n$ disks in different sizes. The problem starts with all of the disks placed on one of the rods in the order of increasing size. The aim is to move the disks one by one to another rod so that at the end, they would be placed on that rod in the same order. The movement of a disk is valid only if it is the uppermost disk in the rod, and is to be placed either on an empty rod or on top of a disk of larger size. In our experiments, we set $m = 3$ and $n = 5$. For this setting, there are $6$ disk movement actions, and the agent can only move one disk per action step. The actions are deterministic, where an invalid action causes no change in the environment. The agent is rewarded by $1.0$ when the goal state is reached, and punished by $-0.1$ if it employs a non-valid action. Any other action yields a default punishment of $-0.01$.

### 3.4.2 Settings

While the general learning parameters are given in Table 3.2, the parameters used for the subgoal discovery methods are as in Table 3.3. Parameters in Table 3.3 are identified by a number of trial-and-error runs, so that each method finds enough number of useful subgoals to effectively improve learning.

Table 3.2: Learning parameters.

| **Algorithm** | **Parameter** | **Value** |
|---|---|---|
| Q/Macro-Q Learning | $\alpha$ | $0.05$ |
| | $\gamma$ | $0.9$ |
| | $\epsilon$ | $0.1$ |
| Experience Replay | $\alpha$ | $0.125$ |
| | $\gamma$ | $0.9$ |
| | $r_\beta$ | $+1000.0$ |
| | $r_{\tilde{g}}$ | $-100.0$ |
| | $r_{default}$ | $-1.0$ |

After the terminal states are set and the formation of the initiation sets is complete, ER is employed by the agent to generate the policy through the states in the initiation set to reach the terminal state. A transition is rewarded $+1000.0$ upon reaching the terminal state, and punished with $-100.0$ for leaving the initiation set, and given a $-1.0$ punishment for any other transition. The learning parameters of ER are $\alpha = 0.125$ and $\gamma = 0.9$. An option is terminated upon reaching the terminal state with probability $1.0$, while it is terminated at any state in the initiation set with probability of $0.0$. The ER parameters are given in Table 3.2. The ER procedure is repeated $10$ times to achieve a fast convergence of the Q function.

Macro-Q Learning is used with learning parameters as given in Table 3.2. All of the methods are compared against the regular Q-Learning using same learning parameters without options. Test results are averaged over 200 experiments.

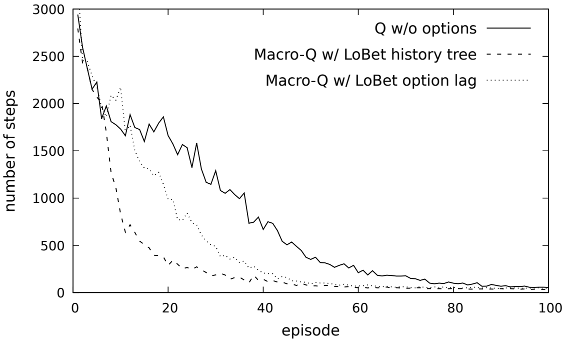
The parameters *option lag* and *option depth* are set in such a way that the resulting options have initiation sets with approximately the same number of states on the average for both approaches with a maximum difference of $3$ states. On the other hand, the initiation set generation heuristic of Segmented Q-Cut algorithm (which we will call *segmentation* from now on) does not possess any parameters to control the initiation set size. Thus, the related experiments are not restricted with the “almost equal initiation set size” condition, that is, the resulted options does not have the same amount of states in their initiation sets for each heuristics.

Table 3.3: Parameter values used for automatic subgoal identification methods.

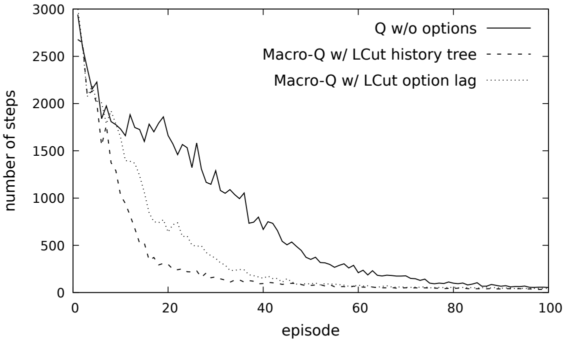
| **Problem** | **Parameters used** | | | | | |
|---|---|---|---|---|---|---|
| | **Method** | $p$ | $q$ | $t_c$ | $t_q$ | $t_e$ |
| $\texttt{2rooms 2doors}_\texttt{F}$ | LoBet | 0.7 | 0.07 | - | - | - |
| | LCut | 0.2 | 0.01 | 0.05 | - | - |
| | SegQCut | - | - | - | 1000 | 10 |
| $\texttt{Virtual Office}_\texttt{F}$ | LoBet | 0.8 | 0.08 | - | - | - |
| | LCut | 0.1 | 0.01 | 0.05 | - | - |
| | SegQCut | - | - | - | 1000 | 15 |
| $\texttt{4Rooms 3Doors}_\texttt{F}$ | LoBet | 0.5 | 0.05 | - | - | - |
| | LCut | 0.2 | 0.01 | 0.015 | - | - |
| | SegQCut | - | - | - | 1000 | 10 |
| $\texttt{4Rooms 4Doors}_\texttt{F}$ | LoBet | 0.5 | 0.05 | - | - | - |
| | LCut | 0.2 | 0.002 | 0.05 | - | - |
| | SegQCut | - | - | - | 2000 | 10 |
| $\texttt{Taxi}_\texttt{F}$ | LoBet | 0.3 | 0.03 | - | - | - |
| | LCut | 0.04 | 0.002 | 0.05 | - | - |
| | SegQCut | - | - | - | 1000 | 200 |
| $\texttt{ToH5}_\texttt{F}$ | LoBet | 0.9 | 0.09 | - | - | - |
| | LCut | 0.25 | 0.025 | 0.05 | - | - |
| | SegQCut | - | - | - | 200 | 10 |

Segmented Q-Cut is run upon reaching an episode threshold $t_e$, in order to guarantee that the agent has explored the problem enough to derive an almost stable (i.e. unchanging) state transition graph. The initial source and target state pairs used in Segmented Q-Cut is given manually (specific for each problem) so that the algorithm can make a reasonable partitioning. The heuristic for including the states in the segments partitioned by Segmented Q-Cut is also compared with the other heuristics. Each subgoal discovery method is separately tested with different initiation set formation heuristics.

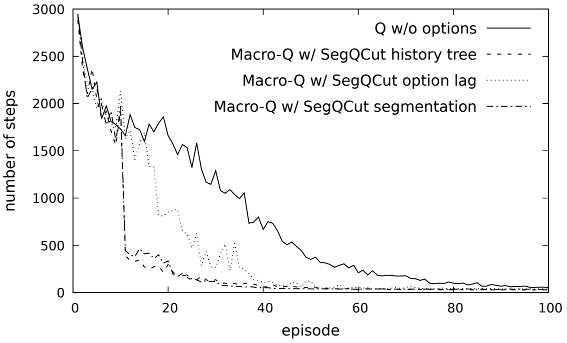
### 3.4.3 Results and Discussion



(a) LoBet



(b) LCut



(c) SegQCut

Figure 3.4: Number of steps to reach the goal state for each subgoal identification method with different heuristics in `4Rooms4Doors`$_{\mathrm{F}}$ domain.

One representative result for `4Rooms4Doors`$_{\mathrm{F}}$ domain is given in Figure 3.4 and all of the results are given in Table 3.4 due to sake of space. As it can be seen in Table 3.4, the learning performance of the options generated by history tree heuristic

Table 3.4: Average number of steps to goal per episode for automatic subgoal identification experiments with 95% confidence interval. Standard deviation of the values are given in the parentheses.

| **Problem** | | **Macro-Q w/ LoBet** | | **Macro-Q w/ LCut** | | **Macro-Q w/ SegQCut** | | |
|---|---|---|---|---|---|---|---|---|
| | **Q** | **option lag** | **history tree** | **option lag** | **history tree** | **option lag** | **segmentation** | **history tree** |
| `2Rooms`<br>`2Doors`$_F$ | $664.14 \pm 115.35$<br>$(401.80)$ | $448.54 \pm 125.03$<br>$(434.53)$ | $\mathbf{303.59 \pm 117.31}$<br>$(\mathbf{408.64})$ | $437.78 \pm 114.97$<br>$(400.47)$ | $\mathbf{319.09 \pm 114.94}$<br>$(\mathbf{400.38})$ | $524.05 \pm 137.06$<br>$(477.43)$ | $\mathbf{311.50 \pm 137.58}$<br>$(\mathbf{479.22})$ | $312.64 \pm 136.62$<br>$(475.90)$ |
| `Virtual`<br>`Office`$_F$ | $524.73 \pm 107.13$<br>$(478.39)$ | $365.21 \pm 101.22$<br>$(451.98)$ | $\mathbf{265.88 \pm 86.63}$<br>$(\mathbf{386.82})$ | $351.97 \pm 99.41$<br>$(443.92)$ | $\mathbf{282.23 \pm 92.75}$<br>$(\mathbf{414.18})$ | $474.49 \pm 130.99$<br>$(584.93)$ | $\mathbf{268.43 \pm 107.01}$<br>$(\mathbf{477.85})$ | $332.73 \pm 106.69$<br>$(476.42)$ |
| `4Rooms`<br>`4Doors`$_F$ | $735.04 \pm 139.73$<br>$(700.66)$ | $510.22 \pm 145.21$<br>$(728.17)$ | $\mathbf{310.78 \pm 114.23}$<br>$(\mathbf{572.78})$ | $429.62 \pm 129.93$<br>$(651.54)$ | $\mathbf{303.52 \pm 112.71}$<br>$(\mathbf{565.19})$ | $457.91 \pm 138.97$<br>$(696.88)$ | $292.90 \pm 124.20$<br>$(622.82)$ | $\mathbf{284.42 \pm 122.21}$<br>$(\mathbf{612.81})$ |
| `4Rooms`<br>`3Doors`$_F$ | $2996.29 \pm 612.96$<br>$(3073.69)$ | $1462.88 \pm 477.44$<br>$(2394.13)$ | $\mathbf{589.65 \pm 379.91}$<br>$(\mathbf{1905.04})$ | $3055.58 \pm 825.86$<br>$(4141.30)$ | $\mathbf{1137.27 \pm 456.27}$<br>$(\mathbf{2287.95})$ | $1379.49 \pm 530.97$<br>$(2662.55)$ | $\mathbf{917.44 \pm 510.30}$<br>$(\mathbf{2558.89})$ | $961.35 \pm 513.35$<br>$(2574.20)$ |
| `Taxi`$_F$ | $157.83 \pm 3.65$<br>$(58.81)$ | $114.67 \pm 4.00$<br>$(64.48)$ | $\mathbf{77.92 \pm 3.97}$<br>$(\mathbf{63.99})$ | $112.64 \pm 3.65$<br>$(58.73)$ | $\mathbf{92.89 \pm 3.72}$<br>$(\mathbf{59.97})$ | $151.22 \pm 3.65$<br>$(73.82)$ | $150.01 \pm 4.62$<br>$(74.37)$ | $\mathbf{148.14 \pm 4.76}$<br>$(\mathbf{76.63})$ |
| `ToH5`$_F$ | $857.21 \pm 108.91$<br>$(546.15)$ | $827.67 \pm 128.77$<br>$(645.71)$ | $\mathbf{304.95 \pm 104.01}$<br>$(\mathbf{521.57})$ | $921.17 \pm 113.18$<br>$(567.56)$ | $\mathbf{530.32 \pm 120.94}$<br>$(\mathbf{606.44})$ | $1103.91 \pm 201.50$<br>$(1010.43)$ | $687.55 \pm 118.93$<br>$(596.37)$ | $\mathbf{662.32 \pm 114.60}$<br>$(\mathbf{574.67})$ |

outperforms the one with the option lag heuristic, for all of the subgoal discovery methods and for all of the problem domains. History tree heuristic includes only the states that the option is proven to be useful on the way to goal, and by this way, the agent gets rid of a significant amount of unnecessary learning time. On the other hand, the performance of the options with segmentation heuristic is very close to the ones with history tree heuristic. Since segmentation heuristic includes all the non-terminal states in the source segment partitioned by the cut to the initiation set of the terminal state, the generated options tend to have larger initiation sets (Table 3.5). For almost every domain, although history tree heuristic shows a similar performance with segmentation heuristic in terms of number of steps to goal, it achieves this performance via significantly smaller initiation sets.

The memory consumption of heuristic methods, in terms of additional memory required for subgoal identification and keeping histories, are given in Table 3.6. Even though the history tree heuristic uses an additional tree structure, its memory consumption is better than the option lag, since the options with history tree heuristic lead the agent to a goal state much sooner, causing shorter episodes. For segmentation heuristics, on the other hand, the only additional memory usage is for the segments generated in the partitioning to create an initiation set. Similarly, the increase in learning performance seems to decrease the overall memory usage. Mostly because they share this advantage, the memory consumptions of both history tree and segmentation heuristics are very close to each other.

Table 3.5: Average initiation set sizes with Segmented Q-Cut algorithm given with 95% confidence interval. Standard deviations are given in the parentheses.

| **Problem** | **option lag** | **segmentation** | **history tree** |
|---|---|---|---|
| 2Rooms2Doors$_F$ | 28.51 ± 0.54<br>(3.88) | 98.53 ± 1.38<br>(9.90) | 44.65 ± 0.46<br>(3.30) |
| VirtualOffice$_F$ | 52.18 ± 0.68<br>(4.89) | 159.29 ± 0.97<br>(6.94) | 122.98 ± 3.05<br>(21.81) |
| 4Rooms4Doors$_F$ | 56.09 ± 0.98<br>(7.04) | 122.73 ± 4.97<br>(35.55) | 49.51 ± 0.71<br>(5.11) |
| 4Rooms3Doors$_F$ | 76.01 ± 0.74<br>(5.31) | 114.06 ± 2.28<br>(16.31) | 78.44 ± 0.37<br>(2.63) |
| Taxi$_F$ | 16.81 ± 0.24<br>(1.28) | 22.00 ± 0.49<br>(2.35) | 24.36 ± 1.13<br>(5.23) |
| ToH5$_F$ | 16.58 ± 1.39<br>(9.96) | 32.03 ± 4.83<br>(34.54) | 27.63 ± 3.71<br>(26.57) |

Table 3.6: Average memory usage per episode (KB) for automatic subgoal identification experiments with 95% confidence interval. Standard deviation of the values are given in the parentheses.

| **Problem** | **Macro-Q w/ LoBet** | | **Macro-Q w/ LCut** | | **Macro-Q w/ SegQCut** | | |
|---|---|---|---|---|---|---|---|
| | **option lag** | **history tree** | **option lag** | **history tree** | **option lag** | **segmentation** | **history tree** |
| 2Rooms<br>2Doors$_F$ | 1287.35 ± 123.84<br>(431.36) | **917.13 ± 68.29**<br>**(237.86)** | 1235.84 ± 121.25<br>(422.37) | **950.61 ± 74.19**<br>**(258.43)** | 1499.47 ± 154.95<br>(539.74) | 997.17 ± 72.08<br>(251.10) | **995.96 ± 73.09**<br>**(254.58)** |
| Virtual<br>Office$_F$ | 1769.54 ± 118.03<br>(527.05) | **1311.18 ± 75.16**<br>**(335.61)** | 1706.88 ± 111.46<br>(497.72) | **1394.66 ± 80.33**<br>**(358.70)** | 2294.12 ± 173.72<br>(775.75) | **1408.03 ± 75.49**<br>**(337.10)** | 1690.17 ± 105.33<br>(470.33) |
| 4Rooms<br>4Doors$_F$ | 3174.31 ± 170.85<br>(856.72) | **1975.30 ± 85.08**<br>**(426.64)** | 2685.31 ± 135.87<br>(681.32) | **1931.33 ± 81.93**<br>**(410.85)** | 2926.20 ± 151.59<br>(760.15) | 1955.96 ± 76.07<br>(381.47) | **1900.95 ± 73.49**<br>**(368.52)** |
| 4Rooms<br>3Doors$_F$ | 9419.42 ± 465.74<br>(2335.44) | **4024.82 ± 108.28**<br>**(542.98)** | 18961.61 ± 1211.39<br>(6074.54) | **7499.97 ± 308.89**<br>**(1548.92)** | 9198.69 ± 415.65<br>(2084.29) | **6384.41 ± 217.62**<br>**(1091.27)** | 6644.05 ± 235.08<br>(1178.81) |
| Taxi$_F$ | 5411.34 ± 137.82<br>(2219.91) | **3915.84 ± 86.47**<br>**(1392.75)** | 5200.94 ± 135.16<br>(2177.07) | **4468.30 ± 108.35**<br>**(1745.13)** | 7185.40 ± 194.34<br>(3130.20) | 7143.63 ± 192.38<br>(3098.63) | **7109.40 ± 189.26**<br>**(3048.29)** |
| ToH5$_F$ | 4009.47 ± 303.75<br>(1523.17) | **1653.14 ± 84.30**<br>**(422.71)** | 4397.43 ± 348.25<br>(1746.32) | **2927.78 ± 175.70**<br>**(881.06)** | 5296.54 ± 446.88<br>(2240.90) | 3565.73 ± 246.80<br>(1237.59) | **3393.49 ± 235.57**<br>**(1181.25)** |

Overall CPU time consumption results averaged per episode are given in Table 3.7. Please note that, since the subgoal identification methods may produce diverse results affecting the overall learning procedure, in order to be fair among the methods, this table shows the total learning time. The results show that options with history tree saves CPU time compared to both regular Q-Learning and learning with option lag heuristic. We can deduce that time gained by Macro-Q Learning by incorporating the history tree based initiation set generation is more than the additional computation required. Again, like the similar performances in learning speed, the time consumptions for segmentation and history tree heuristics are very close to each other. In terms of CPU time, history tree heuristic is better than option lag, and competitive with segmentation heuristic.

Table 3.7: Average CPU time consumption (msec) per episode for automated experiments with 95% confidence interval. Standard deviation of the values are given in the parentheses.

| **Problem** | **Q** | **Macro-Q w/ LoBet** | | **Macro-Q w/ LCut** | | **Macro-Q w/ SegQCut** | | |
|---|---|---|---|---|---|---|---|---|
| | | **option lag** | **history tree** | **option lag** | **history tree** | **option lag** | **segmentation** | **history tree** |
| `2Rooms`<br>`2Doors`$_F$ | *70.18 ± 12.18*<br>*(42.42)* | 67.12 ± 18.98<br>(66.12) | **48.29 ± 16.57**<br>(**57.72**) | 79.09 ± 19.95<br>(69.49) | **59.69 ± 19.35**<br>(**67.39**) | 116.20 ± 65.92<br>(229.62) | **66.49 ± 55.67**<br>(**193.92**) | 67.73 ± 56.09<br>(195.38) |
| `Virtual`<br>`Office`$_F$ | *72.38 ± 14.72*<br>*(65.73)* | 63.89 ± 17.76<br>(79.31) | **48.17 ± 15.01**<br>(**67.04**) | 76.39 ± 20.26<br>(90.49) | **61.09 ± 18.70**<br>(**83.50**) | 90.47 ± 36.60<br>(163.44) | **51.46 ± 29.19**<br>(**130.32**) | 64.42 ± 29.74<br>(132.82) |
| `4Rooms`<br>`4Doors`$_F$ | *295.91 ± 56.69*<br>*(284.28)* | 298.12 ± 84.53<br>(423.89) | **210.12 ± 67.52**<br>(**338.58**) | 166.28 ± 44.24<br>(221.84) | **122.14 ± 37.93**<br>(**190.20**) | 197.93 ± 88.28<br>(442.68) | 124.65 ± 71.57<br>(358.88) | **121.59 ± 62.32**<br>(**312.49**) |
| `4Rooms`<br>`3Doors`$_F$ | *645.78 ± 132.35*<br>*(663.69)* | 434.56 ± 152.45<br>(764.48) | **189.64 ± 116.67**<br>(**585.03**) | 874.78 ± 239.04<br>(1198.69) | **368.36 ± 137.60**<br>(**690.01**) | 627.73 ± 335.97<br>(1684.74) | **396.21 ± 298.85**<br>(**1498.56**) | 415.85 ± 308.60<br>(1547.46) |
| `Taxi`$_F$ | *41.25 ± 0.95*<br>*(15.24)* | 107.14 ± 3.77<br>(60.64) | **80.63 ± 3.32**<br>(**53.42**) | 55.42 ± 2.57<br>(41.44) | **43.45 ± 2.53**<br>(**40.68**) | 87.80 ± 33.48<br>(539.29) | **86.49 ± 33.55**<br>(**540.35**) | 88.70 ± 33.76<br>(543.69) |
| `ToH5`$_F$ | *47.02 ± 5.96*<br>*(29.87)* | 143.35 ± 40.35<br>(202.32) | **66.81 ± 22.47**<br>(**112.70**) | 181.05 ± 20.24<br>(101.47) | **148.64 ± 17.04**<br>(**85.46**) | 381.71 ± 198.28<br>(994.27) | 198.54 ± 130.49<br>(654.37) | **179.54 ± 57.65**<br>(**289.08**) |

Even though segmentation heuristic and history tree heuristic seem to perform similarly, it is worth reminding here that Segmented Q-Cut method requires the whole state space to be explored before a meaningful subgoal identification can be made, employing a global approach. On the other hand, history tree heuristic can be used whenever a subgoal is identified. Also, the initial source and target states must be provided beforehand for the Segmented Q-Cut to operate, so that it can find a reasonable segmentation. The selection of these states are problem specific and may not always be straightforward to manually identify. However, history tree heuristic can work with no additional information specific to the problem domain.

### 3.5 Summary and Discussion

In this chapter, we proposed a goal oriented option initiation set generation method utilizing a history tree heuristic. It restricts the initiation set of an option to the states from which employing the option would be useful on the way to the previously experienced goals.

By achieving higher quality options, the new approach increases the learning performance of the agent without requiring additional computation time and memory, compared to the other methods. Moreover, a history tree does not contain any information regarding the actions taken, so it is not affected by the stochasticity of the problems. Experimentation is done with automatically generated subgoals, using well known subgoal identification algorithms. The final algorithm can find the subgoals online and form options that the agent can effectively employ for reaching goal states.

Our experiments show that careful creation of the initiation set plays an important role in the efficacy of an option as much as the terminal state of it. Better approaches like history tree can save learning time by creating goal directed options.

# CHAPTER 4

# A CONCEPT FILTERING APPROACH FOR DIVERSE DENSITY

A prominent way to decompose a problem is to search for bottlenecks in the state space which are presumed to cluster the problem, then learn how to reach the identified bottlenecks separately, while preserving an abstract learning strategy among the clusters using the sub-policies learned so far. There are various attempts that aim to automatically discover bottlenecks. An effective one is the Diverse Density (DD) method [2], however, its computational complexity limits its use for realistic problems.

In this chapter, we propose an extension for McGovern's DD approach, called *Concept Filtering* [52], which significantly improves its computation time. The method makes use of a novel local graph metric, called the *Congestion Ratio*, which uses and enhances bridgeness and clustering coefficients. The effectiveness of the modified algorithm is empirically shown by experimentation, for both fully observable and partially observable domains.

## 4.1 Diverse Density

Diverse Density (DD) is originally an effective method to attack multiple-instance (MI) problems. DD treats an MI problem so that it consists of positive and negative bags of instances. Each positive bag contains at least one positive instance, and each negative bag contains only negative instances. The aim is to find the target concept by using only the sign tags (positive or negative) of bags, since the instances are not labeled individually.

Maron *et al.* [53] propose a metric called *Diverse Density* (DD) and define the DD of a target concept $c_t$ as;

$$DD(c_t) = Pr(c_t|B_1^+, ..., B_n^+, B_1^-, ..., B_m^-) \tag{41}$$

where $Pr(c_t)$ is the probability that $t^{th}$ concept is the correct concept, $B_i^+$ is the $i^{th}$ positive bag, $B_i^-$ is the $i^{th}$ negative bag. The target concept is the one with the highest DD value and the search space is

$$DD(c_t) = \prod_{1 \leq i \leq n} Pr(c_t|B_i^+) \prod_{1 \leq i \leq m} Pr(c_t|B_i^-). \tag{42}$$

The terms in this equation are defined with a noisy-or model:

$$\begin{aligned} Pr(c_t|B_i^+) &= 1 - \prod_j (1 - Pr(B_{ij}^+ \in c_t)) \\ Pr(c_t|B_i^-) &= \prod_j (1 - Pr(B_{ij}^- \in c_t)) \end{aligned} \tag{43}$$

where $B_{ij}$ is the $j^{th}$ instance of the $i^{th}$ bag, and $Pr(B_{ij} \in c_t)$ is defined to be a Gaussian probability inversely proportional with the distance from the particular instance to the target concept.

The original algorithm uses the term concept as an abstract notion that is learned by using the bags. Under RL setting, a concept can be a state in fully observable problems, or an observation or an experience of observations and actions in partially observable domains.

### 4.2 Diverse Density for Automatic Subgoal Identification

McGovern *et al.* [2] model the task of automatic subgoal identification in an MDP as a multiple instance problem and adopt Diverse Density for the solution. Their approach considers each state gathered from the environment as a concept, presuming every successful episode as a positive bag and all other episodes as negative bags. On this matter, besides being defined as whether the agent reached the goal state or not, the success of an episode may also be dependent on a step threshold, i.e. an episode may be required to reach the goal state under a given number of steps in order to be considered as successful.

McGovern *et al.* propose that a bottleneck state, possibly a subgoal, must be observed in positive bags but not in negative ones. In that case, a bottleneck state is expected to have a higher DD value as the result of an exhaustive DD search. Their study shows that DD can be effectively used for finding subgoals while learning on an MDP [2].

### 4.3 Diverse Density with Concept Filtering

The original version of McGovern's DD approach (MDD) was proposed in order to discover subgoals in fully observable RL tasks. It is shown that the extensive DD search idea is able to find bottleneck regions of the problem [2]. However, some of its drawbacks prevent its applicability to real life problems.

First of all, in order to calculate the DD values of each concept, the method uses a similarity measure between concepts. McGovern et al. suggests a Gaussian distance measure (for the term $Pr(B_{ij} \in c_t)$) which depends on the graph distances between the observations [2]. The distances are precalculated and are provided to the agent before the learning process starts. However, this means that the agent has *a priori* knowledge about the structure (or transition dynamics, to be more accurate) of the problem. It is arguable that providing this knowledge in advance conflicts with the "learn-from-scratch" philosophy of RL.

Another drawback of the DD approach is that it takes each and every concept into account for evaluation as a bottleneck candidate. Since an extensive DD search is employed among the bags upon finishing every episode, this computation is very time consuming. However, apparently not all of the observations are eligible for candidacy and here is room for improvement by means of eliminating some of the concepts through evaluation of certain graph features.

Besides the drawbacks, DD's utility is not limited to fully observable problems. The method does not depend on any of the features of an instance, but focuses the problem on a bag-level. This approach is also useful in partially observable environments. As an observation that plays role in reaching high rewards should also be seen in positive bags, DD is a promising candidate for identifying bottleneck concepts, corresponding to an observation, in partially observable problems where the agent cannot discover them by analyzing their features.

### 4.3.1 Bridging and Clustering Coefficients

In this study, we utilized two graph metrics to form a new one suitable for concept filtering.

*Bridging Coefficient* $BC(v)$ of a vertex $v$ in a graph is a local measure showing how much connection $v$ brings to its neighbours:

$$BC(v) = \frac{d^{-1}(v)}{\sum_{i \in N(v)} d^{-1}(i)} \tag{44}$$

where $d(v)$ is the degree of $v$, $N(v)$ is the set of direct neighbours of $v$.

*Clustering Coefficient* $CC(v)$ of a vertex $v$ is also a local measure determining how well $v$ is clustered [54]. A generalization of $CC(v)$ that extends the neighbourhood of $v$ to the neighbours at depth $k$, called the *k-Clustering Coefficient*, denoted $CC^{(k)}(v)$ [55], and is formulated as;

$$CC^{(k)}(v) = \frac{2e^{(k)}}{n^{(k)}(n^{(k)} - 1)} \tag{45}$$

where $e^{(k)}$ is the number of edges among $k$ neighbours of $v$, and $n^{(k)}$ is the number of nodes within $k$ neighbourhood of $v$. Obviously, $CC(v) = CC^{(1)}(v)$.

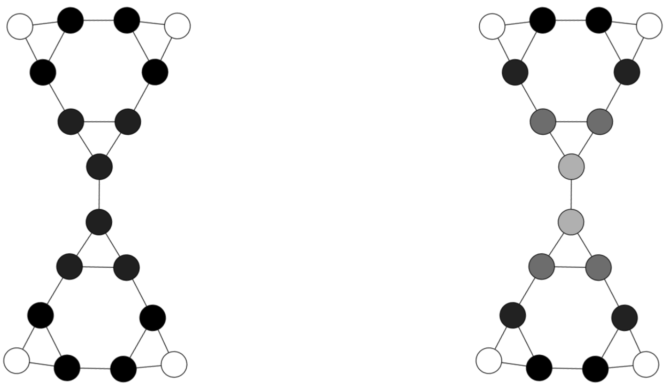
Both $BC(v)$ and $CC^{(k)}(v)$ require less time compared to their global counterparts [56, 57] which span the entire graph of concern. On the other hand, they also have certain disadvantages in terms of solution quality, depending on the structure of the target graph [52].

### 4.3.2 Congestion Ratio

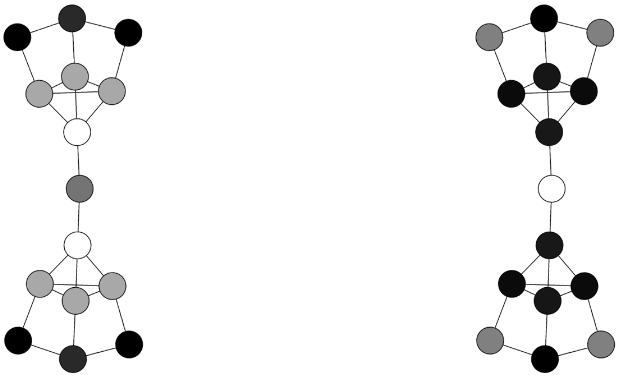
In order to use it in the concept filtering task, we define *Congestion Ratio* ($CR^{(k)}$) for a graph node $v$ as;

$$CR^{(k)}(v) = \frac{BC(v)}{CC^{(k)}(v)} \tag{46}$$

where $BC(v)$ is the Bridging Coefficient of $v$, $CC^{(k)}(v)$ is the k-Clustering Coefficient of $v$ and $k$ is the neighborhood distance for k-Clustering Coefficient. This metric highlights the nodes that connect different regions of the graph, but do not apparently



(a) $BC$ (b) $CR^{(3)}$



(c) $1/CC^{(2)}$ (d) $CR^{(2)}$

Figure 4.1: Sample graphs colored by (a) $BC$ (b) $CR^{(3)}$ (c) $1/CC^{(2)}$ (d) $CR^{(2)}$ values. Brighter is better.

belong to a cluster. Thus, $CR^{(k)}$ can be said to pinpoint the *most bridging and least clustering* nodes.

The need for $CR^{(k)}$ emerged on the inadequacy of $BC$ and $CC^{(k)}$ on some certain circumstances. For example, for a graph with an almost uniform degree distribution among nodes, we expect the $BC$ value to be more or less the same for each node. Thus, it is highly probable that we can miss an apparent bottleneck node between clusters because of this special formation (Figure 4.1a). The corresponding $CR^{(3)}$ calculation clearly identifies the bridging nodes more accurately (Figure 4.1b). Similarly, suppose $1/CC^{(k)}$ is calculated for every node for a graph like the one given in the Figure 4.1c so that the least clustering node gets a higher value. Note that, the nodes around the apparent bottleneck node (i.e. the central node of this graph) have higher values of $1/CC^{(2)}$, which misleadingly means that they are less clustered compared to the bottleneck state. $CR^{(2)}$, however, explicitly pinpoints the node which does not belong to any cluster (Figure 4.1d). Note that brighter color is better in Figure 4.1.

$CR^{(k)}$ can be thought of as a metric that not only unifies $BC$ and $CC^{(k)}$, but also overcomes their weaknesses.

### 4.3.3 Concept Filtering with Congestion Ratio

MDD method for subgoal discovery [2] classifies each episode either as a positive or a negative bag. This decision can be made depending on whether the episode ends with a goal state or not. Also, one may introduce a step threshold for a bag to be a positive bag. Afterwards, the algorithm calculates DD values for each concept, and updates the running averages $\rho$ of the ones with the peak value. Finally, the concepts passing the threshold $\theta$ are marked as targets (concept of interest). Possible noise in the results are eliminated via a decay mechanism over the running averages of all the concepts throughout the process.

The first improvement that our approach offers is that it removes the necessity to have the shortest path distances between concepts to calculate DD values, by adding a routine to calculate them incrementally. It keeps track of the transitions of the agent via the interaction graph $G$ [4], concurrently with the learning procedure.

**Definition 4.4** *An interaction graph $G$ is a weighted directed graph where each vertex $v \in V$ corresponds to a state/observation obtained from the model and each edge $e = (v, v') \in E$ corresponds to a transition from vertex $v$ to vertex $v'$. A directed edge $e = (v, v')$ is given with a weight of 1 if and only if the agent experiences a transition from $v$ to $v'$.*

Whenever there is an update on the graph, the shortest path distance matrix $D_G$ is also updated (line 8 of Algorithm 3), for which, Dijkstra's shortest path algorithm [58] is employed.

**Algorithm 3** $DD\_WITH\_CONCEPT\_FILTERING$

```
Require: $\lambda_{DD}, \theta, k$
Initialize full trajectory database to $\emptyset$
Initialize running averages $\rho_c$ to 0
$G \leftarrow \emptyset$, $D_G \leftarrow \emptyset$
$C \leftarrow \emptyset$
for each episode do
    Interact with environment / Learn using RL
    Add observed full trajectory to database
    Update $G$ and $D_G$ if necessary
    Create positive or negative bag from filtered trajectory
    Update the concept set $C$ with new observations
    $C_f \leftarrow FILTER\_CONCEPTS(C, G, D_G, k)$
    Search for diverse density peaks in $C_f$
    for each peak concept $c$ found do
        Update the running average by $\rho_c \leftarrow \rho_c + 1$
        if $\rho_c$ is above threshold $\theta$ then
            if $c$ passes static filter and $c \in C_f$ then
                Mark $c$ as a target
            end if
        end if
    end for
    Decay all running averages by $\rho_c \leftarrow \lambda_{DD} \cdot \rho_c$
end for
```

**Definition 4.5** *A graph distance matrix $D_G$ is a $|V| \times |V|$ matrix that contains the shortest path distance $d(v_i, v_j)$ between any two vertices $v_i$ and $v_j$ in the interaction graph $G$.*

Therefore, the method becomes free of the transition characteristics of the problem domain. Even if the initial interaction graphs are incomplete and the initial shortest path distances are noisy or inadequate, $D_G$ usually rapidly evolves to its actual value in the early stages of learning as the graph gets completed throughout the ex-

**Algorithm 4** $FILTER\_CONCEPTS$

```
Require: $C$, $G$, $D_G$, $k$
1:  $C_f \leftarrow \emptyset$
2:  for each concept $c \in C$ do
3:      Calculate $BC(c)$ by using $D_G$ {Eqn. 44}
4:      Calculate $CC^{(k)}(c)$ by using $G$, $D_G$ and $k$ {Eqn. 45}
5:      $CR^{(k)}(c) \leftarrow \frac{BC(c)}{CC^{(k)}(c)}$ {Eqn. 46}
6:  end for
7:  for each concept $c$ do
8:      $N(c) \leftarrow$ immediate neighbors of $c$
9:      if $c$ has a peak value in $N(c)$ then
10:         $C_f \leftarrow C_f \cup \{c\}$
11:     end if
12: end for
13: return $C_f$
```

plorations, depending on the size of the problem.

The other contribution ensures that the search for DD peaks is employed only among concepts that survive the *concept filtering* procedure, simply ignoring other concepts that fail to pass the filter. The filtering procedure is given by Algorithm 4. It first calculates $CR^{(k)}$ for each concept (lines 2-6). Afterwards, each concept is added to the filtered concepts set ($C_f$) provided that it has the peak $CR^{(k)}$ value among its immediate neighbors. This way, neighboring concepts having the same $CR^{(k)}$ are also added to $C_f$, as long as they are the peaks among the concepts in the neighborhood. Algorithm 4 ensures that DD method deals with significantly less amount of concepts on the average, since only a small subset of concepts undergo the DD peak search (line 12 of Algorithm 3). The overall procedure is called *Diverse Density with Concept Filtering* [52], and abbreviated as DDCF.

In this chapter, we exclude the option generation phase of the original algorithm because of two reasons: (1) We neither attempt to discover higher quality subgoals, nor generate better options. Our contribution addresses discovery speed. (2) Our method can effectively be used for problems with hidden state, but no profound approaches exist that can generate options for non-Markovian environments.

For partially observable case, we make two assumptions for DDCF to work. First, since DD itself requires good quality on positive bags in order to calculate accurate DD values, the agent must be able to learn the problem at some level. Second, for obvious reasons, we assume a bottleneck state yields a distinct observation so that DDCF can identify it (as is usually the case in practical situations, like a "door" is distinguished via visual sensory information).

## 4.6 Experiments

We have tested MDD and DDCF on both fully and partially observable problems (Table 4.1) by comparing their time consumptions and identification performances. We also tested another well known subgoal discovery method, Local Betweenness (LoBet) [4], on POMDPs with hidden states. LoBet incorporates a graph metric called *betwenness* [59] to find bottleneck states in the agent's interaction graph, and is shown to perform well under full observability.

### 4.6.1 Problem Domains

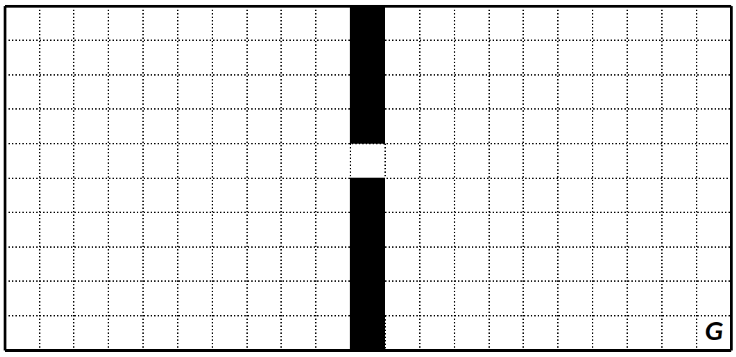
The experiments were carried out on two versions of the grid world domain with 2 rooms and 1 door. In the first version ($\texttt{2Rooms1Door}_\texttt{F}$) shown in Figure 4.2a, the agent can fully observe the underlying states, and aims to identify the subgoal state. In the second one ($\texttt{2Rooms1Door}_\texttt{P}$) given in Figure 4.2b, the agent tries to find the bottleneck state through the limited observation semantics.

Table 4.1: Sizes and reference publications given with determinism status of the action's outcomes for the environments used in this chapter. The sub-scripts $F$ and $P$ represents full and partial observability.

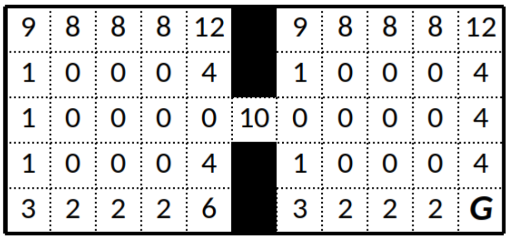
| **Problem** | **\|S\|** | **\|A\|** | **\|Ω\|** | **Action Noise** | **Reference** |
|---|---|---|---|---|---|
| $\texttt{2Rooms1Door}_\texttt{F}$ | 202 | 4 | - | Yes | [36] |
| $\texttt{2Rooms1Door}_\texttt{P}$ | 212 | 4 | 11 | No | [36] |

In `2Rooms1Door`$_{\mathrm{F}}$ (Figure 4.2a), the agent is located in the grid cells and can employ four actions as *north*, *east*, *south* and *west*. Actions are non-deterministic, the agent moves to the intended direction with a probability of 0.9 and randomly to any of the movement directions with 0.1 probability. Initially, the agent is located at any cell in the room on west. The agent receives a positive reward of $1.0$ upon reaching the goal state and no reward or punishment for any other movements.

Like in `2Rooms1Door`$_{\mathrm{F}}$, in `2Rooms1Door`$_{\mathrm{P}}$ (Figure 4.2b), the agent can take four actions as *north*, *east*, *south* and *west*. However, here, the actions are deterministic. The agent is initially located randomly at any state of the left room. In this partially observable problem, there is a punishment of $-0.01$ for each action, a higher punishment of $-0.1$ for bumping into a wall and a positive reward of $1.0$ for reaching the goal state. The agent's observation semantics is formed upon its noise-free sensing of surrounding walls (whether its immediate neighboring cells are walls or not).



(a) `2Rooms1Door`$_{\mathrm{F}}$



| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 9 | 8 | 8 | 8 | 12 | | 9 | 8 | 8 | 8 | 12 |
| 1 | 0 | 0 | 0 | 4 | | 1 | 0 | 0 | 0 | 4 |
| 1 | 0 | 0 | 0 | 0 | 10 | 0 | 0 | 0 | 0 | 4 |
| 1 | 0 | 0 | 0 | 4 | | 1 | 0 | 0 | 0 | 4 |
| 3 | 2 | 2 | 2 | 6 | | 3 | 2 | 2 | 2 | **G** |

(b) `2Rooms1Door`$_{\mathrm{P}}$

Figure 4.2: Domains used in the experiments where the goal states are marked as G. Numbers in `2Rooms1Door`$_{\mathrm{P}}$ indicate the identifiers of observations gathered by the agent when it resides in the corresponding grid cell.

### 4.6.2 Settings

Throughout the experiments, MDD is provided with the shortest path distances between observations. For both versions of DD, a concept is presumed to be a single observation under partial observability while it corresponds to a state in fully observable environment.

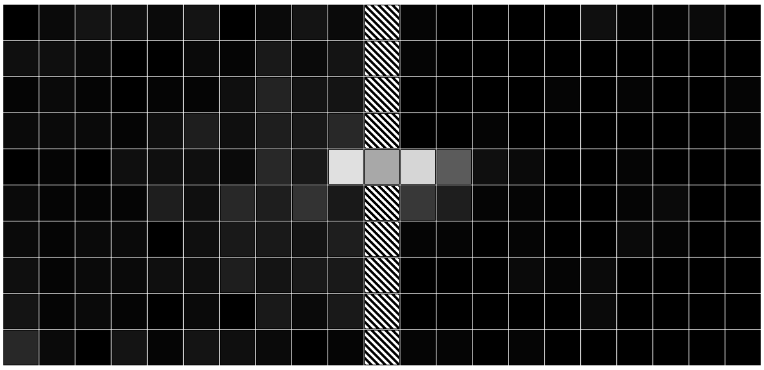
The DD parameters are set as $\theta = 0.8$ and $\lambda_{DD} = 0.9$ for both MDD and DDCF. For DDCF, the clustering coefficient depth is set as $k = 3$ for the fully observable case, and $k = 2$ for the partially observable case. For full observability, all of the successful episodes were considered as positive, as suggested by [2]. On the other hand, a step limit on episodes is used in the partially observable problem to guarantee that there are equal number of positive and negative bags. In this setting, a bag is considered positive if the episode lasts for number of steps less than the step limit, and negative otherwise. Whenever a concept's running average $\rho$ passes $\theta$, both algorithms mark the concept as a subgoal. During the subgoal identification process, the agent employs Q-Learning with $\epsilon$-greedy action selection mechanism, using parameter values $\epsilon = 0.1$, $\alpha = 0.05$ and $\gamma = 0.9$. Each experiment lasted for 100 episodes and the results are averaged over 50 runs.

### 4.6.3 Results and Discussion

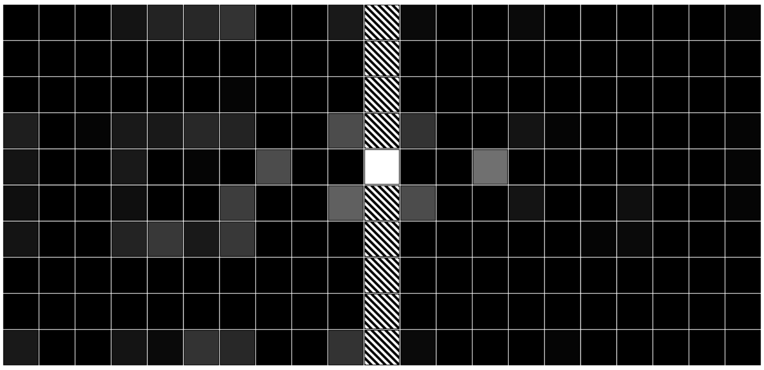
Table 4.2: Reward per step results averaged over 50 experiments.

| **Problem** | **MDD** | **DDCF** |
|---|---|---|
| `2Rooms1Door`$_\texttt{F}$ | 0.0018 | 0.0019 |
| `2Rooms1Door`$_\texttt{P}$ | -0.0046 | -0.0049 |

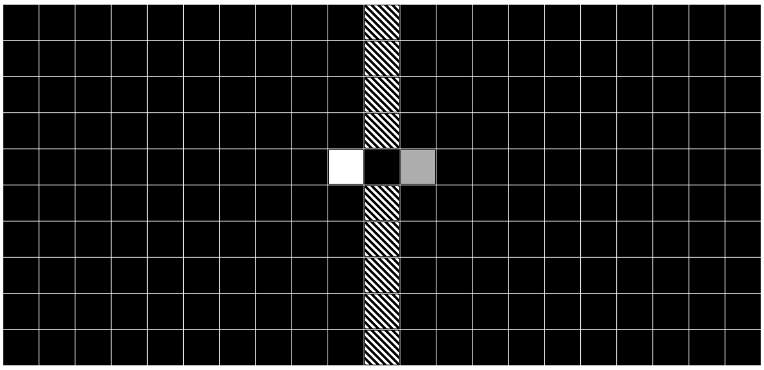
The experiment results are interpreted upon two aspects: (1) quality of discovered subgoals, (2) computation performance. Since the problems have very obvious subgoal states, their quality can be visually verified by using gray-scale color coded sketches, drawn according to how many times the state is identified as a subgoal throughout the episodes, averaged over 50 experiments. For partially observable problems, the states revealing the same observation are colored according to the corresponding observation's relative value. Brighter color represents higher frequency of identification. For the computation performance results, Table 4.2 is given, where "reward-per-step" values are provided as an evidence that the modifications does not have any effect on learning performance.



(a) MDD



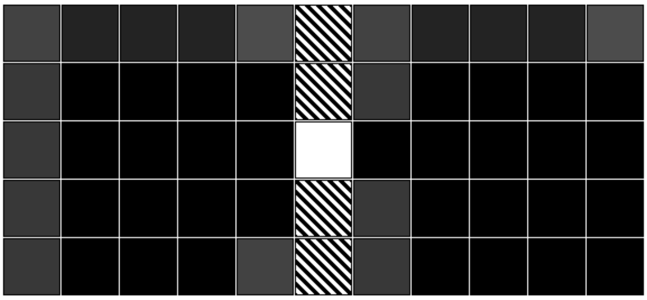
(b) DDCF



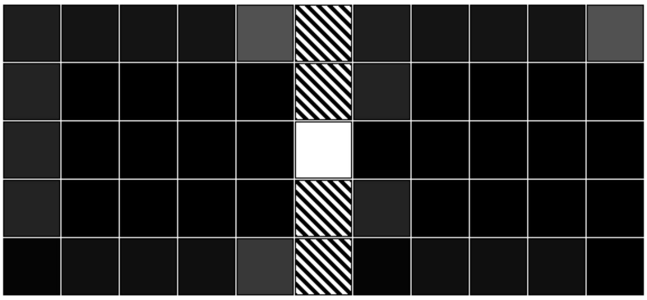
(c) LoBet

Figure 4.3: Subgoals found in `2Rooms1Door`$_F$ domain. Brighter color means higher number of identification as a subgoal.

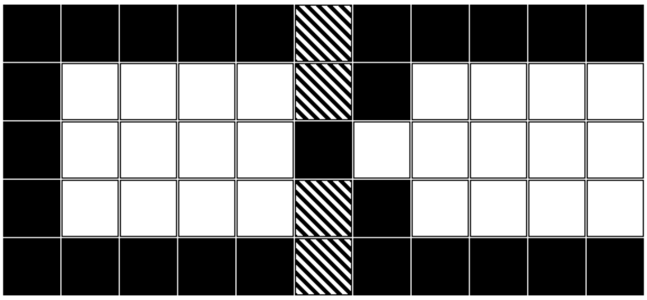
Figure 4.3 shows the subgoal discovery performances in `2Rooms1Door`$_F$ domain. All of the algorithms successfully find the states near doorway. While LoBet seems to give the least noisy results, DDCF has also low noise compared to MDD algorithm. As the concepts with lower $CR^{(k)}$ (like the ones near the doorway) are eliminated, DDCF searches for a DD peak in a narrow and more meaningful concept set, leading to less number of noisy results. Moreover, this improvement significantly reduces the time required to find peak DD values through less number of candidate concepts (Table 4.3).



(a) MDD



(b) DDCF



(c) LoBet

Figure 4.4: Subgoals found in `2Rooms1Door`$_P$ domain. Brighter color means higher number of identification as a subgoal.

For partially observable case, we experienced that DD algorithm requires at least one negative bag to distinguish the highly ambiguous observations from the observations corresponding to the actual subgoal states. Otherwise, ambiguous observations (like observation `0` in `2Rooms1Door`$_P$) may yield high DD values since they are to be more frequently observed by the agent.

Discovered subgoals in `2Rooms1Door`$_P$ are presented in Figure 4.4. Note that Lo-Bet fails to find the actual subgoal states, since the agent's interaction graph is entirely different than the state transition graph due to perceptual aliasing. Betweenness metric highlights the ambiguous observations, which misleadingly act like central nodes for the interaction graph. In that sense, algorithms using graph based features seem to have a certain disadvantage on the problems with hidden states.

On the other hand, using a more robust approach, DD algorithm performs well in general under partially observability. Since ambiguous observations are also frequently observed in negative bags, they achieve a lower DD value compared to the observations corresponding to the subgoal states. This distinction helps DD to identify the actual subgoals of a partially observable environment.

Table 4.3: Number of used concepts and time results averaged over 50 experiments.

| **Problem** | **# concepts used** | | **time (msec)** | |
|---|---|---|---|---|
| | **MDD** | **DDCF** | **MDD** | **DDCF** |
| `2Rooms1Door`$_F$ | 198.96 | **55.90** | 252885.18 | **80170.94** |
| `2Rooms1Door`$_P$ | 9.94 | **4.93** | 1436.82 | **939.98** |

In the both fully and partially observable cases, Tables 4.2 and 4.3 indicate that DDCF produces fewer number of concepts and consumes less time to discover almost the same subgoals as MDD without affecting the overall learning performance.

## 4.7 Summary and Discussion

In this chapter, a *concept filtering* method is proposed as a novel idea to improve McGovern's Diverse Density algorithm in terms of computation time for subgoal discovery. The performance of the proposed method is empirically shown to outperform its antecedent. To our knowledge, this is also the first study to explicitly handle an automatic subgoal discovery technique to cover both fully observable and partially observable problems.

The resulting algorithm, Diverse Density with Concept Filtering, makes use of a new graph based metric called *congestion ratio* to reduce the number of concepts used for subgoal identification. Moreover, it incrementally calculates the shortest path distances between concepts during learning, removing the implicit need to feed them to the algorithm in advance. Empirical results indicate that MDD can be a step forward in subgoal discovery challenge to cover a broader category of domains.

# CHAPTER 5

# AUTOMATIC LANDMARK DISCOVERY

Partial observability is a challenging characteristic of the environments that Reinforcement Learning algorithms operate on [1]. As the states of the environment are not directly available, the learning agent has to cope with partial and possibly ambiguous observations in order to learn the dynamics of the underlying environment. Such a task is especially difficult when multiple states map to the same observation, causing a fundamental problem called *perceptual aliasing* [60, 12]. Depending on the degree of ambiguity on the observations, finding the optimal action for each state may not be possible.

Landmarks, on the other hand, provide sufficient information to distinguish a problem state, so that they can help the agent in the learning phase. Providing a reliable knowledge, landmarks can guide an agent in the learning task and hence speed up the learning [8]. Although their presence are used in the literature, automatic identification of landmarks is left unexplored.

In robotics, a landmark may correspond to a location or an object that is unique in that environment. It may provide some insight to the robot about its progress and guide it to complete the given task. Under partial observability, the robot may benefit from the landmarks in order to locate itself in the environment. Thus, discovery of the landmarks of an environment may be useful for planning in robotics domain, especially for navigational tasks.

Making use of landmarks has been shown to improve learning performance for memoryless RL agents in partially observable environments. We refer to this class of tasks as *Landmark-POMDPs*. SarsaLandmark [8] enhances the classical Sarsa($\lambda$) algorithm [16] by skipping the non-landmark states in the eligibility trace mechanism, so that a higher-level tracking of reliable observations (provided by landmarks) is promoted. However, SarsaLandmark assumes that the agent knows in advance all of the landmark states, which is not a realistic scenario both in real life, and in terms of "learn-from-scratch" philosophy of RL.

In this chapter, we propose a RL framework to extend SarsaLandmark with automatic landmark identification capability [61]. For this purpose, we utilize a statistical method that we introduced in Chapter 4, named Diverse Density (DD) [2]. DD classifies instances based on raw experiences and ignores the direct features of the states/observations, which makes it ideal for landmark identification since the state space is hidden from the agent. The resulting framework succeeds to automatically identify landmarks during learning, feed the identified landmarks to SarsaLandmark, and achieves much the same learning performance with SarsaLandmark with manually provided landmarks.

A side contribution of the chapter is that it provides a deeper analysis on Diverse Density with Concept Filtering (DDCF), which we proposed in Chapter 4, on the task of finding landmarks. With DDCF, McGovern's original DD algorithm (MDD) [2] is significantly improved, in terms of both time and quality. We also show that, DDCF can effectively be incorporated within the proposed framework.

In the following subsections, a novel framework that automatically identifies landmarks and uses them in the learning process is incrementally presented. It starts by explaining the notion of landmarks, the Landmark-POMDP model and SarsaLandmark. Then, a motivation is set on why Diverse Density can be used effectively for discovering landmarks online. Finally, a novel framework which fuses SarsaLandmark with automatic landmark discovery is presented.

## 5.1 Landmarks and Landmark-POMDPs

A *landmark* has its obvious meaning in the real life as "a recognizable natural or artificial feature used for navigation, a feature that stands out from its near environment and is often visible from long distances" [62]. In Artificial Intelligence (AI) literature, landmark has different formal definitions and meanings.

Planning researchers define landmark states as those that lie on every optimal path to the goal state [63, 64, 65, 66]. This is, in a sense, a strong definition of landmark, restricting any solution to contain the landmark instance. In this setting, landmarks are commonly assumed stationary.

In robotic navigation research, a landmark is usually defined over perceptions as "locally distinctive features," such as walls and doorways. In that sense, since the identification is local, there is no theoretical need for the landmarks to be stationary, although sensor aliasing due to noise is a major problem [67, 68, 69, 70].

In RL context, the use of landmarks dates back to mid-2000s, with James and Singh's studies. They prefer to make a weaker definition of landmark, in terms of state-observation mapping, not restricting its existence in any policy, as in the planning domain [8]. Our study follows their definition where a *landmark* is a state that has a unique observation. Note that, this definition assumes that a landmark state is an integral part of the problem structure, existing in the model without requiring any other information.

Following the formal definition of a landmark state, *Landmark-POMDP* is a special form of POMDP which contains at least one landmark state [8].

## 5.2 SarsaLandmark

SarsaLandmark [8] is an adaptation of Sarsa($\lambda$) to handle Landmark-POMDPs. The motivation behind SarsaLandmark is that in the presence of landmark states, skipping over non-landmark states during the Q-value updates may help Sarsa($\lambda$) to converge faster. The method incorporates $\lambda = 1$ throughout learning, with the exception that whenever the agent resides in a landmark state, $\lambda$ is set to zero. With this adaptation, upon experiencing a transition $\langle x_t, a_t, r_t, x_{t+1}\rangle$ at time $t$ between estimated states of $x_t$ and $x_{t+1}$, the agent executes the following update rules;

$$\begin{aligned} \eta_t(x_t, a_t) &= 1 \\ \forall(x \neq x_t \text{ or } a \neq a_t), \eta_t(x,a) &= \begin{cases} 0 & \text{if } x_t \text{ is given by} \\ & \text{a landmark state} \\ \gamma \cdot \eta_{t-1}(x,a) & \text{otherwise} \end{cases} \\ \forall x, a, \quad Q_{t+1}(x,a) &= Q_t(x,a) + \alpha \cdot \eta_t(x,a) \cdot \delta_t \end{aligned} \tag{51}$$

where the terms are defined as in Sarsa($\lambda$). Note the differences between rules (24) and (51) of the corresponding update sequences, and that $\lambda$ is ignored in rule (51) since it is ineffective to the result.

SarsaLandmark reduces to Sarsa(0) if every state is a landmark (full observability), while it is equivalent to Sarsa(1) in a partially observable setting with no landmark states. It is shown that SarsaLandmark is an improvement over Sarsa($\lambda$), in terms of fast policy convergence [8].

### 5.3 Diverse Density with Concept Filtering

In Chapter 4, we stated that McGovern's Diverse Density (MDD) had some drawbacks that makes it impractical for real life cases. First, it required the distances between concepts as a priori knowledge in order to calculate the DD values. Second, it considered every concept as a candidate while some of them are indeed redundant in the exhaustive search of DD peaks. Our proposed method, Diverse Density with Concept Filtering (DDCF), improved its original by calculating the distances between concepts online and filtering concepts that are not real candidates. It showed a better identification performance compared to its original version.

Furthermore, in Chapter 4, we showed that DD can work on both fully and partially observable environments for finding important observations of the environment that led the agent to success. Although Chapter 4 did not put a strict definition as a *land-*

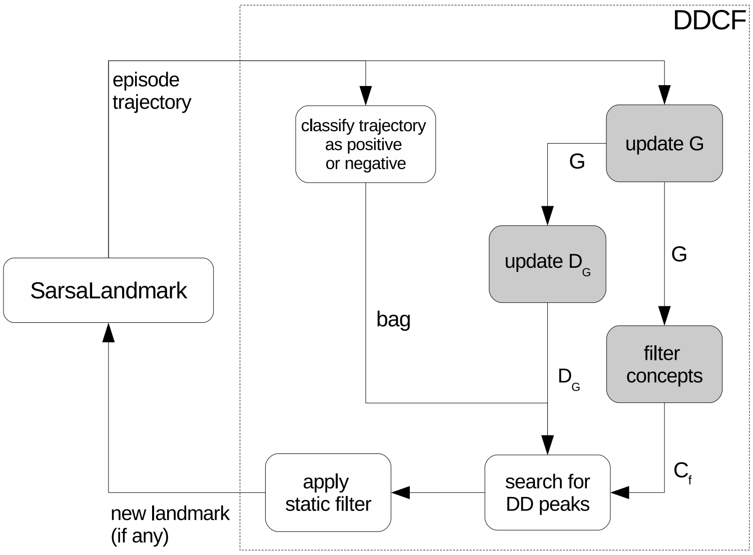


Figure 5.1: SarsaLandmark with MDD/DDCF workflow. Shaded parts are the steps that DDCF introduces to MDD.

*mark* to its target, DD can also be utilized for discovering landmarks during learning. In a POMDP with hidden states, the agent operates on state estimations which may be formed by the observations and actions of the agent. Due to perceptual aliasing, many of the observations are gathered in different parts of the state space. Yet, an observation provided by a landmark state, by definition, is unique to that state. Thus, such an observation is expected to be seen less in an episode, compared to the ambiguous observations. Besides, an observation by a useful landmark is more likely to be seen on the successful episodes, causing a high Diverse Density value.

## 5.4 The Extended SarsaLandmark Framework

SarsaLandmark algorithm has a strong assumption stating that the agent will always identify a landmark state upon reaching it. That is why the agent is assumed to know in advance what observations are unique in the problem prior to learning.

A more realistic problem setting, however, will require the agent to also identify by itself the “distinctive observational characteristics” of the overall problem, namely landmarks. Automatic identification of the landmark states is a challenging task due to perceptual aliasing for problems with hidden state. The observation transition semantics of those problems are much different than state transition semantics. Therefore, methods that make use of the features of observations for identification are likely to fail. DD, on the other hand, is shown to work on partially observable environments, thus, it is a suitable candidate for automatic landmark discovery [52].

In this section, we propose a novel framework that manages to automatically identify the landmarks states of a problem with hidden states, which may later be used to enhance learning performance.

Our proposal synthesizes two approaches, *SarsaLandmark* and *Diverse Density*, building up a complete framework. The main workflow of the overall framework has the following steps (Figure 5.1):

- initially, SarsaLandmark starts with an empty set of landmarks,
- at the end of each episode, the episodic trajectory is fed to DD,
- DD classifies the given trajectory as either positive or negative according to a success criteria,
- with the given set of bags, the method calculates the DD value for each concept in the concept set,
- if a concept is a consistent peak in DD values, i.e. it has the highest DD value among the other concepts for several number of episodes, it is marked as a target, i.e. landmark for this setting,
- then, the landmark set of SarsaLandmark is updated and the algorithm uses this set in its update rules.

The shaded regions in Figure 5.1 are extensions of DDCF on McGovern's DD method (MDD). Although SarsaLandmark can safely be coupled with MDD alone, due to MDD requirements, the designer should also provide the distances between concepts. DDCF not only incrementally calculates these distances, but also applies an effective concept filtering mechanism, completing the framework by making all computations free of designer's intervention, without sacrificing quality and performance.

Table 5.1: Problem sizes, the number of landmarks in the problems (including the goal states) and their reference publications. The problems marked with * are modified versions of their originals and the sub-script $P$ represents partial observability.

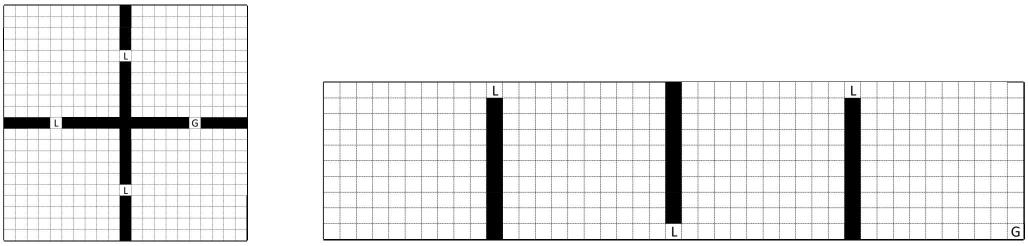
| **Problem** | \|S\| | \|A\| | \|Ω\| | **Number of landmarks** | **Reference** |
|---|---|---|---|---|---|
| `4Rooms4Doors`$_P$* | 404 | 4 | 40 | 4 | [2] |
| `Zigzag`$_P$ | 403 | 4 | 40 | 4 | - |
| `VirtualOffice`$_P$* | 262 | 4 | 40 | 4 | [71] |
| `LargeLandmarkGrid`$_P$ | 1148 | 4 | 107 | 50 | [8] |
| `ElevatorEscape`$_P$ | 674 | 5 | 245 | 12 | - |

## 5.5 Experiments

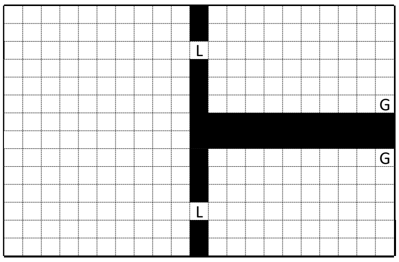
Since the contribution here is two-fold, the experiments were designed accordingly. In one aspect, the experimentation needs to show that automatic landmark identification framework (i.e SarsaLandmark with MDD or DDCF) not only *just works*, but also performs no worse than its manual counterparts (i.e. Sarsa($\lambda$) and SarsaLandmark). On the other hand, we also need to provide empirical evidence that DDCF outperforms McGovern's original DD heuristic (MDD) for non-Markovian problem settings.

### 5.5.1 Problem Domains

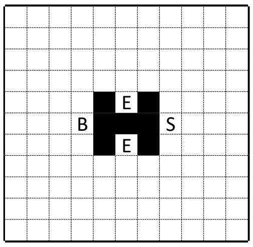
SarsaLandmark with Automatic Landmark Identification using MDD/DDCF is experimented on five problem domains. Three of them (`4Rooms4Doors`$_P$, `VirtualOffice`$_P$ and `LargeLandmarkGrid`$_P$) are known benchmark problems from the literature suitable to experiment for landmarks, while two of them (`ElevatorEscape`$_P$ and `Zigzag`$_P$) are newly introduced to further analyze special characteristics of the method. `4Rooms4Doors`$_P$ and `VirtualOffice`$_P$ problems are modified in such a way that they contain landmark states only on the doorways and `LargeLandmarkGrid`$_P$ is one of the large problems that was introduced in [8].



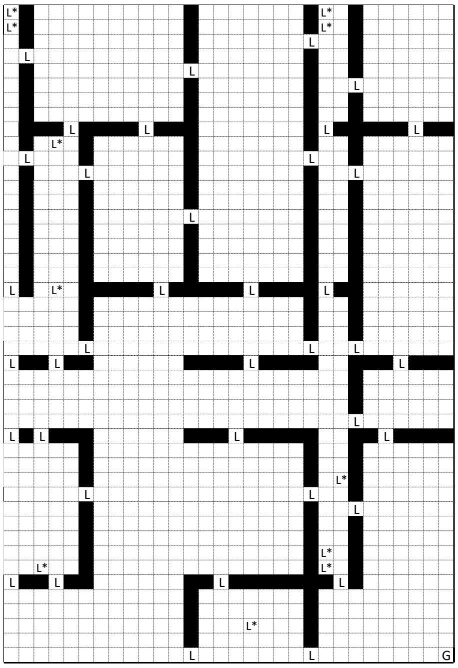
(a) 4Rooms4Doors$_P$ (b) Zigzag$_P$



(c) VirtualOffice$_P$



(d) ElevatorEscape$_P$



(e) LargeLandmarkGrid$_P$

Figure 5.2: Domain sketches. In ElevatorEscape$_P$, the light switch, the button and the elevator positions are marked with S, B, E and the goal states and the landmark states are marked with G and L in the other domains. The states marked as L* in LargeLandmarkGrid$_P$ are the additional landmark states, emerging due to the observation semantics of the problem.

The problem sizes with the number of landmarks (including the goal states) and their corresponding references are given in Table 5.1. To remind or give an idea of the domain characteristics, the domain illustrations and their observation transition graphs are provided in Figures 5.2 and 5.3 5.4, respectively.

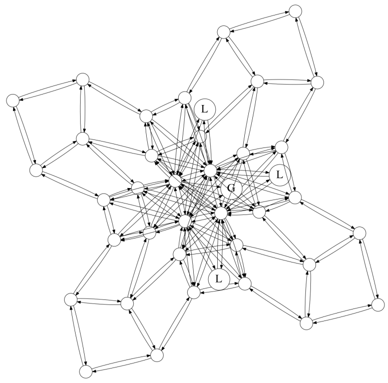
In the problems with multiple connected rooms (all problems except Elevator-

`Escape`$_P$), the doorways are the natural landmark states, yielding unique observations. Moreover, in `LargeLandmarkGrid`$_P$, there are additional landmark states which were not addressed in the original paper [8]. Whenever the agent is at equal distances to the surrounding walls of the room, or if the current observation distinguishes that part of the room from any other rooms in the domain, due to the observation semantics of the environment, the corresponding state is a landmark (see Figure 5.2e).

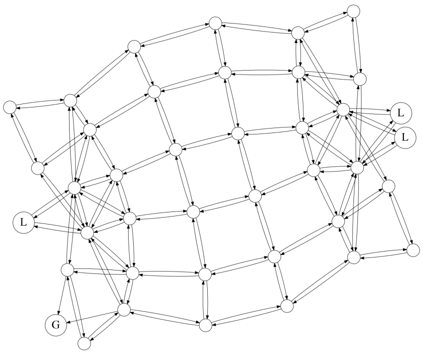
In `4Rooms4Doors`$_P$, `Zigzag`$_P$ and `VirtualOffice`$_P$ problems, the agent is initially at a random point in the westernmost room(s), and it can execute four deterministic move actions to four neighboring directions; *north*, *east*, *south* and *west*. Arriving at a goal state is rewarded with $1.0$ while every other transition is punished by $-0.01$. The agent receives an observation depending on the distance of it to the surrounding walls in the four compass directions [8]: one step from the wall, two steps from the wall, closer to the wall in this direction than the other, further from the wall in this direction than the other.

`LargeLandmarkGrid`$_P$ possesses the same action and observation semantics, but the initial states and the rewarding mechanism is different. As in the experiments of the original paper, initially, the agent is located in any of the landmark states (the ones in the original study, marked with L in Figure 5.2e). Upon reaching the goal state, it gathers a reward of $20$, and other movements does not have a reward or a punishment.

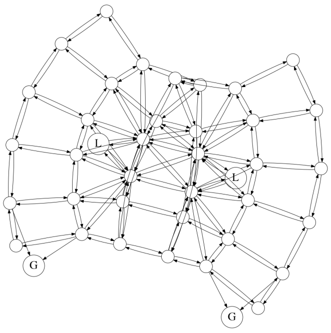
`ElevatorEscape`$_P$ is a domain that we propose, which contains an inherent partitioning in terms of its observation space, as three sub-regions separated by landmarks. The goal of the agent is to escape from a $11 \times 11$ grid (Figure 5.2d) by using an elevator. In order to call an elevator, it needs to press the button, and in order to see the button, it needs to turn on the lights by the light switch. The locations of the elevators, the button and the light switch are marked by *E*, *B* and *L* respectively. When the agent presses the button, randomly only one of the elevator doors open. Button press and light switch actions are irreversible, i.e. the agent can not turn off the lights once they are on and pressing the button when the elevator door is open has no effect. With this structure, a state in this problem is formed by 4 features: agent's position, the light state (on or off), the button state (pressed or not pressed) and the position of the



(a) `4Rooms4Doors`$_P$



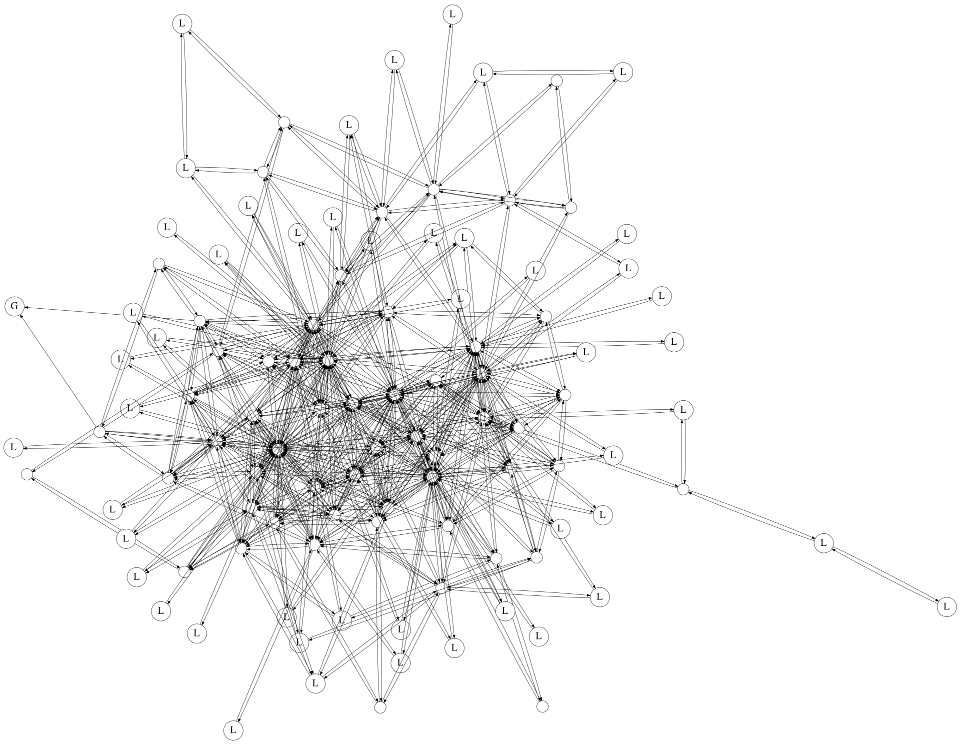
(b) `Zigzag`$_P$



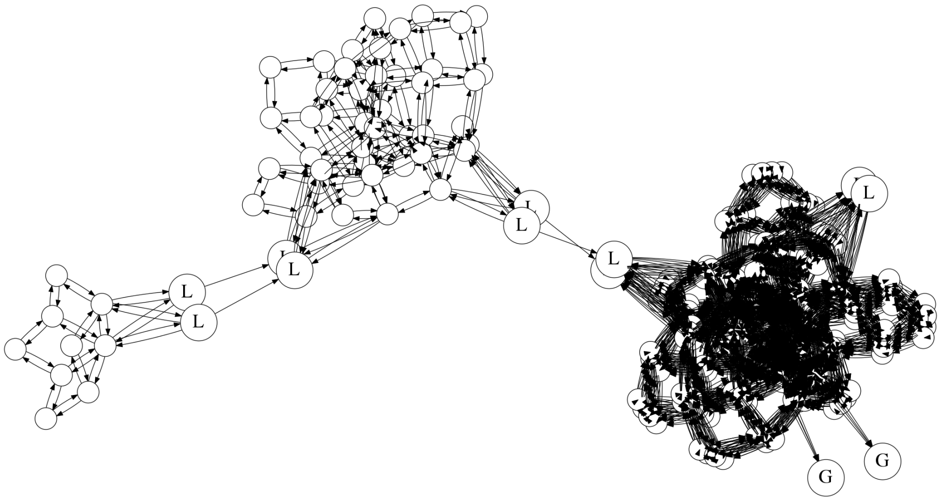
(c) `VirtualOffice`$_P$

Figure 5.3: Observation transition graphs of the smaller domains. The goal states and landmark states are labeled as G and L, respectively.

elevator whose door is to open. The agent starts from any position in the grid (except the elevator positions) with the lights off, and the button unpressed. The goal state is defined as: the lights are on, the button is pressed and the agent is in the elevator position with the door open.



(a) `LargeLandmarkGrid`$_\mathrm{P}$



(b) `ElevatorEscape`$_\mathrm{P}$

Figure 5.4: Observation transition graphs of the larger domains. The goal states and landmark states are labeled as G and L, respectively.

The actions are deterministic. In addition to the movement actions *north*, *east*, *south* and *west*, the agent can take an *interact* action for both turning the lights on and pressing the elevator button. Reaching to the goal state is rewarded by $100.0$ while the agent gets a $-1.0$ punishment for any other action.

The observation semantics of the problem is as follows: in the first observational sub-region, the lights are off and the agent only possesses a basic vision of observing whether there is a wall in any of the four directions or not. After the lights are turned on, which is the second sub-region, the agent acquires an improved vision, allowing it to get an observation according to the distance to the walls in each of the four compass directions (just like in the other grid world domains). Finally, in the third sub-region, the pressing of the elevator button opens one of the elevator doors, and the agent starts to hear the music playing in the elevator, in addition to its improved vision. However, the agent's hearing sensor is noisy. It gives the correct direction of the elevator with probability $0.8$ and a random direction with probability $0.2$. In addition to this observation semantics, the agent achieves a unique observation when it is in the cells *E*, *B* and *L*, which are, by definition, the landmarks of the problem. Consequently, there are $12$ landmarks including the goal states. The observation transition graph of the problem is given in Figure 5.4b.

Each problem domain has its own characteristics providing a different aspect to show the impact of landmarks on learning performance. `4Rooms4Doors`$_{\mathrm{P}}$ problem is, compared to the others, an easy task since the goal state resides in a doorway, leading to an early convergence of a policy. `Zigzag`$_{\mathrm{P}}$ is of a similar size in terms of state and observation sets, however, an ordered visitation to the landmarks is necessary to reach the goal state. Additionally, the agent should devise a different policy for every room, by overcoming the challenge that each room yields almost the same observation semantics. `LargeLandmarkGrid`$_{\mathrm{P}}$ is the largest problem in our problem set, with plenty of landmarks. `VirtualOffice`$_{\mathrm{P}}$ contains multiple goal states which is another challenge for the agent. Unlike a bottleneck state in a state transition graph, the observation transition graphs clearly illustrate how a landmark can be located as if it is redundant on the way to goal, which makes its identification much more challenging (Figures 5.3a, 5.3b, 5.4a and 5.3c). On the other hand, observation transition graph of `ElevatorEscape`$_{\mathrm{P}}$ domain has a structure where the observations of the landmark states act as bottlenecks, diving the problem into sub-problems (Figure 5.4b), although this scenario is usually unlikely in real world situations. Note that, none of the problems tested here has a memoryless policy solution.

### 5.5.2 Settings

SarsaLandmark with DDCF was tested against SarsaLandmark with MDD, while maintaining Sarsa($\lambda$) and the original SarsaLandmark results as baselines. The results are averaged over 50 experiments where each episode ended with either the agent's arrival to a goal state, or upon reaching 5000 action steps. The experiments for `4Rooms4Doors`$_{\text{P}}$ and `VirtualOffice`$_{\text{P}}$ were run for 2000 episodes, and experiments for the other domains were run for 10000 episodes.

For both Diverse Density heuristics (i.e. MDD and DDCF), a single observation was considered as a concept. MDD was provided the shortest path distances between the concepts before learning, while DDCF calculated them throughout learning. We employed a step threshold for a bag to be positive so that the methods managed to collect enough number of positive and negative bags. Also, we used $\theta = 8.8$ and $\lambda_{DD} = 0.9$. In the DD search, we preferred to pick the concepts acting as peak values in the resulting DD distribution, instead of simply selecting the maximum value. In all experiments, both DD versions were executed in the initial 200 episodes.

In order to test the landmark identification performance of both MDD and DDCF, we have compared *precision* and *recall* values by using the set of landmarks in the problems as ground truth. Here, precision is defined as the ratio of the number of true landmarks in the identified set to the size of the identified set where recall is defined as the ratio of the number of true landmarks in the identified set to the total number of landmarks in the problem. That is, precision and recall represent *accuracy* and *coverage* of the identification, respectively.

For Sarsa($\lambda$), $\lambda = 0.9$ is used as in [16]. To ensure enough exploration, an $\epsilon$-greedy action selection mechanism was used, where the agent takes a random action with probability $\epsilon$ and an action according to the current policy with probability $(1 - \epsilon)$. All of the Sarsa algorithms incorporated $\alpha = 0.01$, $\gamma = 0.9$ and employed a linear $\epsilon$ decay, starting from $0.5$ down to $0$. While SarsaLandmark extended with MDD and DDCF identified the landmarks online, SarsaLandmark was provided with the landmarks in advance. In `LargeLandmarkGrid`$_{\text{P}}$, however, only the landmarks marked with L in Figure 5.2e were used, even though there are additional ones in the problem, as described in previous sections.

### 5.5.3 Results and Discussion

Table 5.2: Average number of steps taken in the problem domains. The values are given with their %95 confidence interval.

| **Problem** | **Without landmarks** | **With landmarks provided** | **With landmarks automatically identified** | |
|---|---|---|---|---|
| | **Sarsa($\lambda$)** | **SarsaLandmark** | **SarsaLandmark with MDD** | **SarsaLandmark with DDCF** |
| $\texttt{4Rooms4Doors}_\texttt{P}$ | $84.54 \pm 6.69$ | $71.25 \pm 4.92$ | $72.22 \pm 5.11$ | $\mathbf{70.55 \pm 5.30}$ |
| $\texttt{Zigzag}_\texttt{P}$ | $3006.68 \pm 30.34$ | $\mathbf{335.79 \pm 8.10}$ | $593.43 \pm 10.63$ | $455.25 \pm 10.45$ |
| $\texttt{VirtualOffice}_\texttt{P}$ | $150.08 \pm 8.29$ | $\mathbf{48.76 \pm 2.45}$ | $85.21 \pm 7.73$ | $61.83 \pm 5.27$ |
| $\texttt{LargeLandmarkGrid}_\texttt{P}$ | $1543.29 \pm 12.89$ | $621.79 \pm 4.43$ | $\mathbf{416.45 \pm 3.70}$ | $419.88 \pm 3.66$ |
| $\texttt{ElevatorEscape}_\texttt{P}$ | $963.70 \pm 14.64$ | $412.35 \pm 7.20$ | $\mathbf{376.78 \pm 5.97}$ | $379.10 \pm 6.00$ |

Table 5.2 shows the average number of steps during learning for each algorithm of discourse. Sarsa($\lambda$) acts as a baseline and it is clear that SarsaLandmark significantly improves learning performance (by a factor ranging from $1.2$ to $8.9$) by means of the additional landmark information. The landmark information which is provided by the designer before learning improves the policy update mechanism and accelerates learning, verifying the effectiveness of SarsaLandmark.

However, when SarsaLandmark is coupled with a DD mechanism, not only the landmarks are identified automatically throughout learning, but also a similar learning performance is achieved compared to SarsaLandmark. This result indicates that, DD can effectively be incorporated in identification of landmarks for problems with hidden states, without sacrificing solution quality. Moreover, we can argue that DD is more focused on useful landmarks on the way to a goal state, so that it reaches to the goal states much faster than SarsaLandmark with provided landmarks in the large problems. Overall, in all of the domains, SarsaLandmark with automatic landmark identification outperforms Sarsa($\lambda$), in terms of policy convergence.

Moreover, there is no significant difference between learning performances of SarsaLandmark, SarsaLandmark with MDD and SarsaLandmark with DDCF. So, we can comfortably conclude that automatic landmark identification framework removes the necessity to identify the landmarks by analyzing the problem beforehand. Sarsa-

Landmark with automatic landmark identification is clearly a forward step to adopt the landmark idea to more realistic problems.

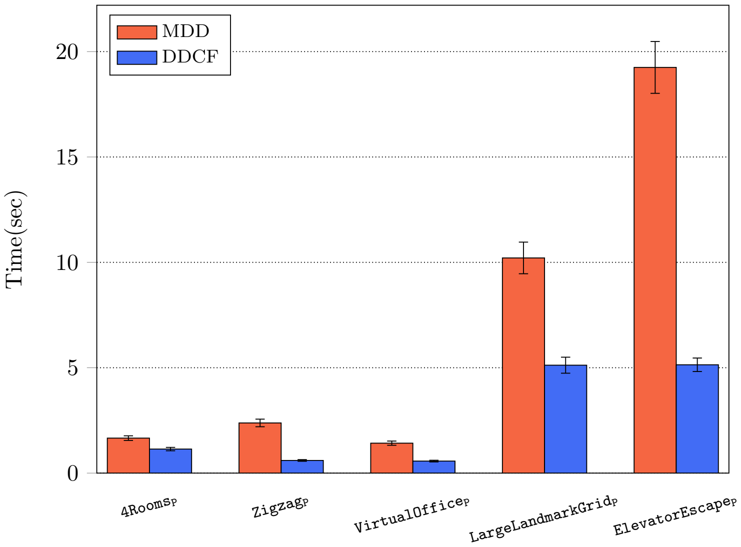


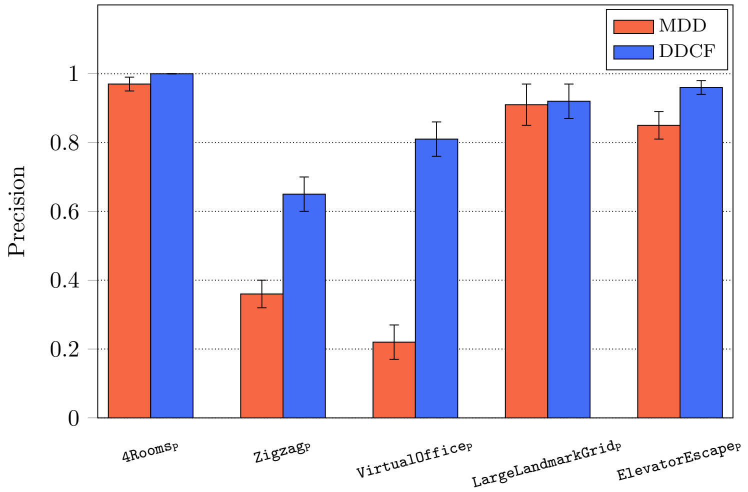
Figure 5.5: Average CPU time usage for DD approaches.

Table 5.3: Number of concepts used for automatic landmark identification.

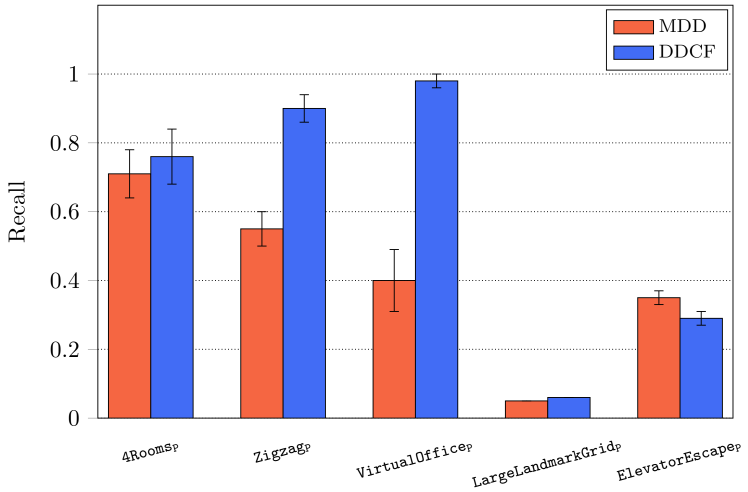
| **Problem** | **MDD** | **DDCF** |
|---|---|---|
| `4Rooms4Doors`$_P$ | 37 | **15** |
| `Zigzag`$_P$ | 37 | **11** |
| `VirtualOffice`$_P$ | 30 | **11** |
| `LargeLandmarkGrid`$_P$ | 104 | **53** |
| `ElevatorEscape`$_P$ | 59 | **31** |

Focusing on automatic landmark identification approaches, it can be clearly seen in Figure 5.5 that DDCF significantly outperforms MDD in terms of computation time. For all problem domains, DDCF requires much less time to identify landmarks, simply because the size of the DD search concept set is reduced, thanks to the concept filtering heuristic (Table 5.3). Additionally, as the observation transition graph matures by the early phases of learning, calculation of shortest path distances takes relatively less time compared to the DD calculations of the remaining concepts.

DDCF not also operates faster than MDD, but also seems to produce higher quality solutions. Figure 5.6a shows that DDCF is more precise in finding landmarks



(a) Precision



(b) Recall

Figure 5.6: Precision and recall comparisons of DDCF against MDD.

for all problem domains, that is, DDCF's resulting set of identified states contains more of the real landmarks of the problems, compared to MDD. This means that the concept filtering heuristic eliminates some of the concepts that create noise. Moreover, the recall performance of DDCF is also generally higher than MDD, showing a bigger coverage of the true landmark set and meaning that DDCF has a tendency to identify more of the landmarks that the problems contain. The recall value for `LargeLandmarkGrid`$_P$ is low for both of the DD methods. What we observe here is that the agent is more directed to the goal state as the learning proceeds, causing less exploration. In order to identify a bigger set of landmarks, DD requires more data covering these landmarks. Also DD tends to favor the concepts that are seen in the path to a goal state and some of the landmarks in `LargeLandmarkGrid`$_{\mathtt{P}}$ do not play a key role in such paths. We argue that the performance of DD is dependent on the way the agent traverses the environment, i.e. the agent has to visit a landmark frequently in order for DD to identify it.

For `Zigzag`$_{\mathtt{P}}$ and `VirtualOffice`$_{\mathtt{P}}$ domains, for example, the observation transition structure seems to cause static filter of the DD mechanism to fail, since all landmark observations are in the same *observational distance* to the goal observation, including both the distant landmark states and the landmark states that are close to the goal state (although they are all in different locations in terms of the underlying state transition graph). This is why, using a relatively high static filter eliminates useful landmarks, while a low static filter introduces noise to the results. DDCF seems to get rid of this problem via its concept filtering heuristic, since it eliminates those observations yielded by the states closer to the goal state(s).

Another interesting aspect of DDCF worth mentioning is that its identification covers more landmarks than MDD in all of the domains except `ElevatorEscape`$_{\mathtt{P}}$. In `ElevatorEscape`$_{\mathtt{P}}$ domain (Figure 5.4b), the local comparison of our concept filtering guides the method to pick the observation of only one of the landmarks in between the observational sub-regions since only one of them has the observation with the maximum congestion ratio value within its neighborhood.

### 5.6 Summary and Discussion

The contribution of this chapter is two-fold: As the primary contribution, we propose a Reinforcement Learning framework, called SarsaLandmark with Automatic Landmark Identification, for problems with hidden states. The proposed framework enhances its predecessor [8] (which is an adaptation of a well known on-policy learning algorithm Sarsa($\lambda$) to Landmark-POMDPs), so that it removes the necessity to provide the landmarks of the domain prior to learning, since they are automatically identified online. For this purpose, we incorporate Diverse Density (DD) approach

[2], which has been known to effectively identify bottleneck regions in solution space. To our knowledge, this is the first study in Reinforcement Learning literature that manages to automatically identify landmarks online, and use them in the learning procedure for problems with hidden states.

As a secondary contribution, we elaborate on the Chapter 4 which introduces concept filtering by using a novel graph based metric, congestion ratio. We not only show DD with Concept Filtering (DDCF) to be applicable in Landmark-POMDPs, but also provide empirical evidence that it significantly improves landmark identification performance and quality, with much less computation effort compared to the original DD method, without any prior information about the problem.

The resulting algorithm, SarsaLandmark with Automatic Landmark Discovery removes the necessity for the agent to know in advance when it is in a landmark state. It also possesses a learning performance very close to SarsaLandmark, for which the designer ought to provide all landmarks prior to learning. Experiment results indicate that the combination of SarsaLandmark with Diverse Density is a step forward to apply SarsaLandmark to real life problems.

# CHAPTER 6

# ANCHOR BASED GUIDANCE

As stated in Chapter 5, the learning capabilities of an agent are diminished under partial observability. Since the true states of the task are hidden from the agent, there is no guarantee of forming an optimal policy that can ensure highest returns. Under such circumstances, an agent can only be expected to be *rational* given these limited sensations from the world.

In order to overcome the uncertainty of the environment, the agent can devise mechanisms; such as leaving eligibility traces to improve the convergence speed of a good policy or keeping a state estimate to distinguish the true state that it is in. Although the idea of eligibility traces seems simple, it is shown to be practical and useful. Faster propagation of the temporal difference error can improve convergence to a policy. However, there may be some domains that have no good policy that can be formed upon the pure observations. In such a case, the agent has to move the learning problem to a higher dimension by introducing a state estimation to overcome perceptual aliasing. By keeping a memory, it may distinguish one experience from another, leading to a more effective policy. Yet, finding a good estimation is the challenge in a partially observable environment and mostly problem specific.

In this chapter, we further dive into the definition of state estimation. But we do not suggest a way to find a good one. Our focus is on the case where the state estimation is true. We put a formal definition for the state estimates that can distinguish a true state clearly and propose a method on using these dependable estimations to improve learning speed.

## 6.1 State Estimation

An agent can keep a state estimate $x$ by using its previous experiences in the environment. A straightforward example can be using the past $k$ observations and actions, creating an estimated state of $x_t = o_{t-k}a_{t-k}...o_{t-1}a_{t-1}o_t$ for the time step $t$. Regardless of the state estimate's structure, the set of estimated states $X$, then, becomes the plane that the agent forms its policy upon, i.e. the policy is now formed as $\pi : X \times A \rightarrow [0, 1]$. As a natural result, the relation $M : X \rightarrow S$ emerges and determines the mapping between the two sets (Figure 6.1). Note that, $M$ is not a function because a state estimate can map to multiple true states.

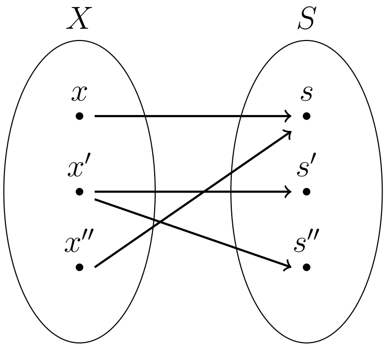


Figure 6.1: An example mapping from the set of estimated states $X$ to the set of true states $S$ in a POMDP with hidden states.

If there is a one-to-one mapping between $X$ and $S$, there is no ambiguity and all the state estimates are clear so that the agent can separate all the states from each other. But this is hardly the case since finding a clear state estimation is a challenging task. On the other hand, one may not need to separate all the states clearly, it is enough to separates the ones having different optimal actions. In fact, the perfect scenario would be to have a smaller set of state estimates, making abstraction over the states that have the same optimal action. Methods like USM [24], somewhat aims such abstract representation by determining the distinctions based on the return distributions where a branch in a USM tree can be considered as an estimated state.

## 6.2 Anchors

In Chapter 5, we experimented with eligibility traces in POMDPs with hidden states and showed that some states, called *landmark states* (Section 5.1), can contribute to learning as they provide unique observations. It is natural to assume that not all of the sensations of the environment have ambiguity, yet there are few that can be dependent upon. Although proved useful, the definition of a landmark state, being a state that provides a unique observation, has some drawbacks [8].

First, the term “landmark” is misleading. A landmark has a basic meaning of a unique structure that one can use to locate itself in an environment. Therefore, observing a landmark from distance can still be useful to get position. However, in RL setting, an agent can only be *in* a landmark state for a time step, yet, in the next step it may be lost due to the ambiguity of perceptions coming from the environment.

Second, the definition of a landmark state [8] is strict to a state with a unique observation. The agent may keep a state estimate with an extended memory, formed by more than one observation and in this configuration, there may be clear estimates that can completely distinguish a true state but not formed by a unique observation. That is, the model may not have a landmark state, yet the set of estimated states which the agent keeps can contain some estimated states causing no aliasing.

For the reasons before, we propose the term “anchor” having the basic meaning of “a thing that provides stability or confidence in an otherwise uncertain situation” and put a formal definition for Reinforcement Learning setting as in Definition 6.3.

**Definition 6.3** *The state estimate $x \in X$ is an **anchor** if*

$$P(s_t = s | x_t = x) = 1,$$

*and*

$$\forall s' \neq s, P(s_t = s' | x_t = x) = 0.$$

Definition 6.3 states that an *anchor* is an estimated state mapping to only one state, so that there is no ambiguity of the current true state of the agent when it resides in an anchor. By Bayes rule;

$$\forall s' \neq s, P(s_t = s'|x_t = x) = \frac{P(x_t = x|s_t = s') \cdot P(s_t = s')}{P(x_t = x)} = 0$$

Since this definition is meaningful when there exists a $x \in X$, $P(x_t = x) = 0$ is unlikely. Then, either $\forall s' \neq s, P(s_t = s') = 0$ or $P(x_t = x|s_t = s') = 0$ should be true. Therefore, we conclude that there is no other state that can be represented with this estimate $x$, that is $x$ can only map to one true state.

Considering the mapping $M : X \to S$, a state estimate $x$ is an anchor if $|M(x)| = 1$. In the example of Figure 6.1, the estimated states $x$ and $x''$ are anchors since $M(x) = \{s\}$ and $M(x'') = \{s\}$ but $x'$ is not since $M(x') = \{s', s''\}$. Note that the definition does not require a one-to-one mapping, the agent may have more than one representation of the same true state as in the case of $x$ and $x''$. Although they may be redundant, they still act as anchors. Moreover, Definition 6.3 does not put any restrictions on the structure of the estimated states as long as the representations map to only one true state. Also, the definition is for a state estimate, not the true state that it maps to. That is, we call an estimated state as an anchor, unlike *landmark states*.

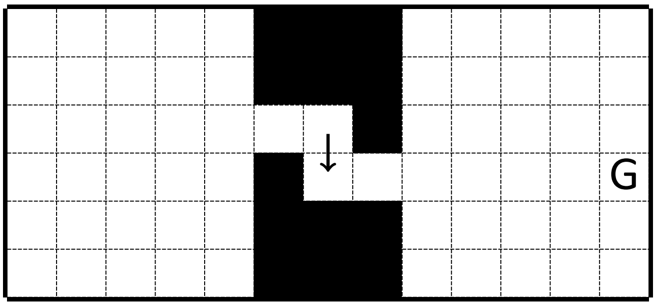


Figure 6.2: Example 2D grid world domain where the agent can take four navigational actions and gets observations based on the presence of a wall in those directions. The goal state is marked as G.

To make it more clear, let us put an example on the difference between a landmark state and an anchor. In the example problem given in Figure 6.2, the perceptions of the agent is limited to whether there is a wall in the next cell in four directions and only the goal state provides a unique observation. For the rest of the observations, there are at least two states giving them. Therefore, there is only one landmark state, which is the goal state. Now, lets say our agent aims to learn a policy on the estimations formed as $x = o_{t-1}o_t$, that is, it keeps the previous observation in memory. In such a setting, observing a top-right corner and a bottom-left corner in this order, can only happen when the agent makes a transition as shown in the figure. Although these observations by themselves do not clear ambiguity (as they can be seen on three states), such a combination can determine the location of the agent clearly, therefore, creating an anchor.

It is natural to think that the agent will have a set of anchors $\mathscr{A}$ in its representation of the world, the set of estimated states $X$. These anchors carry the same features of a true state, making them dependable parts of an uncertain environment. Therefore, their presence can be further utilized to inform the agent while learning. In this chapter, we argue that we can apply additional rewards based on anchors to guide the agent towards reward peaks.

On the other hand, the agent has to know which of the state estimates are anchors, in order to use them. Therefore, a method to discover them while learning is necessary. Here, the core idea of Diverse Density seems to hold. Since an anchor can be visited when the agent is located in a singular true state, we expect it to be seen less, compared to an aliased estimated state mapping to multiple states. Also, a useful anchor should be uniquely seen in successful episodes of the agent. Thus, we believe that DD is a still good candidate for automatic anchor discovery task. In this chapter, we couple DD with applying guiding rewards to lead to a better learning under partial observability.

## 6.4 Reward Shaping

One way to introduce additional rewards to the reinforcement learning agent is called Reward Shaping. It has been used so that the learning process is further improved. RL with reward shaping operates on the new reward function $R'$ where $R' = R + F$ and $F$ represents the shaping reward. It was noticed earlier that greedy reward shaping may lead to poor results as in the case of learning how to ride a bicycle [72]. Ng et al. showed that policy invariance with reward shaping can be guaranteed in an MDP by proposing a potential based reward shaping (PBRS) approach where the arbitrary potential function $\Phi$ is defined for each state and the shaping reward function is formed as $F(s, a, s') = \gamma\Phi(s') - \Phi(s)$ [73]. PBRS is further extended with potential based advice by forming the potential function with the actions as $\Phi(s, a)$ [74].

Most of the time, knowledge about a domain is more comprehensive and abstract compared to specific potentials of the individual states. Studies on how to inject the abstract domain knowledge to the underling model via reward shaping exist. Plan based reward shaping uses a STRIPS plan where each state maps to an abstract state and the current step in the plan is used for the potential of a state [75, 76]. Such an approach is further used on a real time strategy game and shown to outperform its competitors without plan based reward shaping [77]. Efthymiadis et al. introduced knowledge revision to plan based reward shaping when the plan is inconsistent or wrong [78]. Another form of reward shaping is used in a real time strategy game by forming an abstract model and using a time based punishment for each action taken [79]. When a hierarchy exists in the task and it is known beforehand, it is shown that reward shaping approach can be formulated into MAX-Q, a well known hierarchical RL (HRL) algorithm, and can outperform its predecessor [80].

PBRS, on the other hand, can be applied for both model-free and model-based RL [81], and its effects are further analyzed. Grzes et al. employed parameter analysis and argued that PBRS should conform several conditions in order to form a consistent advice to the agent [82, 76]. Wiewiora proved that a RL agent with reward shaping shows equivalent performance with a RL agent with Q-values initialized using the same potential function [74]. Devlin et al. also showed that dynamic reward shaping, where the potential function is not fixed, can maintain the guarantees of policy invariance [83]. More recently, Grzes stated that the potential value of any terminal state must be zero in order to keep the policy guarantees of PBRS [84] and Marom et al. proposed an algorithm that decays the effect of shaping rewards by experience so that any possible convergence problem can be avoided [85]. Reward shaping idea also found itself a place in multi-agent RL and it has been shown that potential based reward shaping does not alter the Nash equilibria [86, 87, 88, 89, 90].

Automatic learning of the potential function turns out to be an interesting problem and gained attention. Marthi proposed an algorithm that completely solves an abstract model with macro actions formed by sampling from the task and using the value function of this abstract model as the potential function [91]. Grzes et al. adopted a similar abstraction idea which learns the values of the abstract states by value iteration while acting in the environment [92].

Although most related work focused on MDP models, there are very few that take partial observability into account. In [93], PBRS is applied to online POMDP planning so that the shaping reward function is defined via the belief states of the agent.

## 6.5 Intrinsic Motivation

Another way to introduce additional rewards in RL is called intrinsic motivation. Intrinsic motivation comes internally and an intrinsically motivated action is employed for its own sake, not to active a certain goal. Unlike extrinsic motivation, intrinsic motivation does not require to have a climax. An example for intrinsic motivation would be the rats choosing to explore rather than eat under certain situations. This kind of behaviour is explained by the intrinsic motivation towards novelty. In a evolutionary perspective, notions like discovery and playing are important for survival, thus related actions are intrinsically motivated [94].

RL framework started to adopt the psychological perspective by making a distinction between the sensations from the environment and the reward signal [95, 94, 96, 97]. In this new framework, all the reward signals are internal, yet they are combinations of intrinsic and extrinsic motivations given by an internal critic. This way, the framework aligns with the reinforcement learning in real life more clearly and enables an agent to not only learn a specific task but also to adopt to different situations since its intrinsic motivation rewards behaviours such as exploration.

Since this new framework, many employed intrinsic motivation to find more effective ways of exploration in RL by providing additional internal rewards for novel states [98, 99, 100]. Moreover, there are studies which aim to learn skills to solve tasks with hierarchy and use intrinsic motivations within the hierarchies [101, 102, 103].

### 6.6 Anchor Based Guidance

Anchors, by definition, are dependable parts of an uncertain situation. Under partial observability, they carry reliable information. They pinpoint to only one location in the state space and they carry the *potential* of a singular state, unlike uncertain estimations mapping to multiple states. If the agent makes a set of primitive transitions from a low-potential anchor to a high-potential one, it can be rewarded for this abstract move, utilizing the anchors to inform the agent about its progress. Such rewards can be very helpful in partially observable problems with delayed or sparse rewards [104].

One may simply propose to give a bonus for reaching an anchor. Yet, such a bonus under partial observability may cause the agent to get stuck on an anchor. As visiting an anchor would be rewarded, the agent may prefer such actions rather than stepping up towards uncertainty. That is why, the agent must be rewarded based on whether its actions leading it to high rewards or not, suggesting a check on the differences between the potentials of anchors.

We argue that applying additional rewards for problems with hidden states can be meaningful if only the transitions among the anchors are taken into account. Since a potential value assigned to an ambiguous state estimate is unreliable, providing an additional reward to a transition including an ambiguous state estimate will not be beneficial in a partially observable environment. The agent might be encouraged to take an action at a state, based on the potential value of its current ambiguous state estimate, and the action might thus mislead the agent away from a goal state. On the other hand, assuming that the state transitions between two anchors form a temporal abstraction, one can define a meta-level transition among two anchors in an abstract level. This meta-level transition may correspond to a series of primitive actions in the underlying model but the additional reward is provided only when these actions complete a transition between two anchors.

Note that, this approach does not align with idea of the potential based reward shaping and benefit from the guarantees of it since it does not shape the reward in each transition and changes the return of the sequence of states that the agent follows. On the other hand, policy invariance is obviously not a concern in a non-Markovian environment since an optimal or even a “good enough” deterministic memoryless policy is not guaranteed at all [105]. We avoid using the term “reward shaping” due to these differences. Moreover, this idea cannot be considered as an intrinsic motivation. Although we provide additional and internal rewards to the learning algorithm, our rewards are not intrinsically motivated. They are given to increase the learning speed on a specific task, that is to maximize the external rewards.

To apply additional rewards, one needs to calculate the potentials of the anchors. This study is influenced by [92] (the value iteration approach on the abstract model) for the abstraction and value iteration ideas. In [92], an abstraction over the set of states of an MDP is created and an online method for learning the potentials of these abstract states is proposed. Similarly, the set of anchors form an abstract model where the agent takes abstract actions lasting more than one time step between these anchors. However, our study differs from theirs because we assume the problem model is a POMDP with hidden states and we do not make further abstractions over the set of observations. We follow their value iteration approach on the abstract model of anchors since it catches the reward mechanism of the problem unlike other heuristic. The abstract model is in fact an SMDP [10] and we will call it *Anchor-SMDP*. Figure 6.3 depicts how the Anchor-SMDP of the `6Rooms`$_P$ domain looks like, also indicating which true problem state each anchor corresponds to.

**Definition 6.7** *An Anchor-SMDP is a SMDP whose set of states $S$ is formed by the anchors in the formed state estimation set.*

The Anchor-SMDP is inherently composed of less number of states compared to the set of estimated states of the POMDP since most states are perceptually aliased in a partially observable environment. Therefore, the potentials of these anchors can be easily calculated by means of value iteration and then used for applying internal rewards in the underlying POMDP while acting in the environment.

The overall algorithm for Anchor Based Guidance (ABG) is given in Algorithm 5. The algorithm assumes that the anchors of the estimated state set are known. It requires as input the learning rate $\alpha_v$ and the discount rate $\gamma_v$ for the Anchor-SMDP value iteration. The algorithm starts by initializing the value function $V$ of the Anchor-SMDP, the time index $t$, the previous and the current anchor variables $a$ and

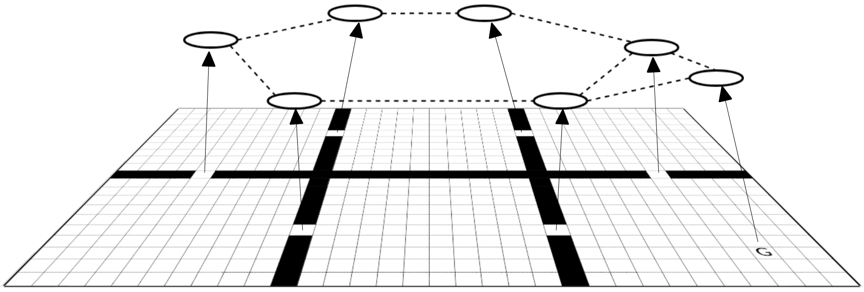


Figure 6.3: An illustration of an abstract model for a sample grid world domain (`6-Rooms`$_{\texttt{P}}$, see Figure 6.5a for the original sketch). The anchors correspond to the doorways and the goal state is marked with $G$ where circles and dashed lines represent the anchors and transitions between anchors in the abstract model.

$a'$, the previous and the current time variables $\tau$ and $\tau'$ that keep the time that $a$ and $a'$ are seen and the short history $H$ (Lines 1-3). It checks if the initial estimated state is an anchor and if so, sets $a$ and $\tau$ (Lines 4-5). Then, it goes into the regular loop of interacting with its environment and observing transitions between the estimated states $x_t$ and $x_{t+1}$. It keeps track of a transition in $H$ to check if there is an anchor previously observed, in order to be able to calculate the discounted sum of rewards that is used in the Line 18.

As the precondition to provide anchor based reward through to the anchor abstraction, the algorithm checks if the agent lands in an anchor. If this is the case, it sets the current anchor $a'$ and the current anchor time $\tau'$ (Lines 13-14). If there is a anchor previously observed, then it is possible to provide the additional reward. The algorithm calculates the reward to be provided to the learning algorithm in Line 16.

Following the internal reward calculation, the algorithm calculates the sum of discounted rewards, gathered between the previous anchor $a$ and the current anchor $a'$ by using $H$ and applies value update (Line 18), where $n$ represents the number of steps taken between the two (Line 17).

The algorithm continues by shifting the previous anchor variables with the current anchor variables (Line 20) and resetting the current anchor variables and the history $H$ (Lines 21-22). Finally, it provides the updated reward to the underlying learning

**Algorithm 5** Reinforcement Learning with Anchor Based Guidance

```
Require: $\gamma, \mathcal{A}, \alpha_v, \gamma_v$
$\forall a \in \mathcal{A}, V(a) \leftarrow 0, t \leftarrow 0$
$a \leftarrow \emptyset, \tau \leftarrow \emptyset, a' \leftarrow \emptyset, \tau' \leftarrow \emptyset$
$H \leftarrow \emptyset$
if $x_t \in \mathcal{A}$ then
    $a \leftarrow x_t, \tau \leftarrow t$
end if
repeat
    Observe transition $\langle x_t, a_t, r_t, x_{t+1} \rangle$
    if $a \neq \emptyset$ then
        $H \leftarrow H \cup \langle x_t, a_t, r_t, x_{t+1} \rangle$ {Save the transition if necessary}
    end if
    if $x_{t+1} \in \mathcal{A}$ then {Check if the reached estimated state is an anchor}
        $a' \leftarrow x_{t+1}$
        $\tau' \leftarrow t + 1$
        if $a \neq \emptyset$ then {Check if there is a previous anchor}
            $f_t = V(a') - V(a)$
            $n = \tau' - \tau$
            $V(a) = (1 - \alpha_v) \cdot V(a) + \alpha_v \cdot (r_\tau + \gamma_v \cdot r_{\tau+1} + ... + \gamma_v^{n-1} \cdot r_{\tau'-1} + \gamma_v^n \cdot V(a'))$
        end if
        $a \leftarrow a', \tau \leftarrow t + 1$
        $a' \leftarrow \emptyset, \tau' \leftarrow \emptyset$
        $H \leftarrow \emptyset$
    end if
    Learn by using $R'(x_t, a_t, x_{t+1}) = r_t + \mathbb{1}_{f_t > 0} \cdot f_t$
until episode ends
```

algorithm and continues to interact with the environment. Here, the internal reward is used only if it is positive. As the agent is aiming to learn a policy over a somewhat uncertain state estimation set, using a punishment over them may not be beneficial. On the other hand, rewarding the right way is enough to guide the agent towards high rewards.

## 6.8 Automatic Anchor Discovery

Algorithm 5 requires the set of anchors beforehand. Yet, in a realistic setting, the agent has no prior knowledge about which state estimates are true and clear. In fact, the anchor set must be discovered as the learning continues.

In Chapter 5, we showed that Diverse Density is a good candidate to work under partial observability. Its performance on automatic landmark discovery makes it a promising method for online anchor identification too.

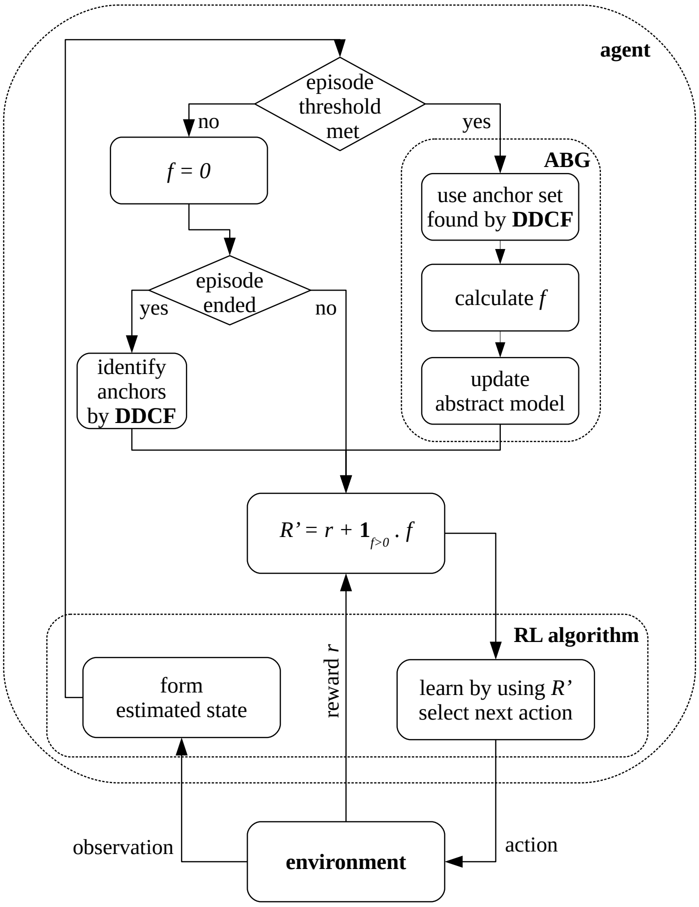


Figure 6.4: The workflow of Reinforcement Learning when the agent is equipped with the coupling of DDCF and ABG.

The idea of Diverse Density is to check the unique points in history that led to success. It requires a bag level classification to have successful and unsuccessful episodes. Then, it checks the instances that are uniquely seen on positive bags but not on negative ones. By definition, an anchor maps to only one state, unlike other unclear state estimations that can be visited on multiple states. Hence, in a successful episode, visiting an ambiguous state estimation is more likely than an anchor, causing a useful anchor to be seen more *diversely* in an episode yet more *densely* on successful ones.

With this motivation, we can employ DD, especially our version DDCF, to identify the anchors during learning to form an overall framework that can run in a realistic Reinforcement Learning setting. Then, the identified ones can be used to provide guiding rewards to the agent. However, these two processes should not be run at the same time, since a newly identified anchor may disrupt the value iteration and cause false guiding rewards. That is why, we first employ DDCF to find a set of anchors to be used in guiding the agent. After a certain episode threshold is met, the Anchor-SMDP is formed by the found anchors and used by ABG to introduce guiding rewards. The overall framework is given in Figure 6.4.

## 6.9 Experiments

In the experiments, we employed two well known algorithms with eligibility traces, Q($\lambda$) and Sarsa($\lambda$), due to their effectiveness in POMDPs with hidden states. We implemented Watkins' Q($\lambda$) to keep it truly off-policy.

The aforementioned algorithms are tested against two proposed methods; Anchor Based Guidance with the anchors given beforehand and its variant coupled with DDCF for automatic anchor identification task. We experimented with different forms of estimated states because higher level of representations are required for some domains in order to contain a good policy. Note that, our aim is not to find the best state estimation, yet to show that ABG can be helpful on different levels of estimated states.

In the results, we showed the discovery performance of DDCF, learning performances of all the methods and further analyzed the learning performances of ABG with subsets of anchors. The results are presented and discussed extensively on several problems.

### 6.9.1 Problem Domains

Experiments are made on three domains two of which ($\texttt{6Rooms}_{\texttt{P}}$ and $\texttt{4Rooms4Hallways}_{\texttt{P}}$) are grid world domains and one ($\texttt{ToH3}_{\texttt{P}}$) is a puzzle. The sketches and the size characteristics of the problems are shown in Figure 6.5 and Table 6.1.

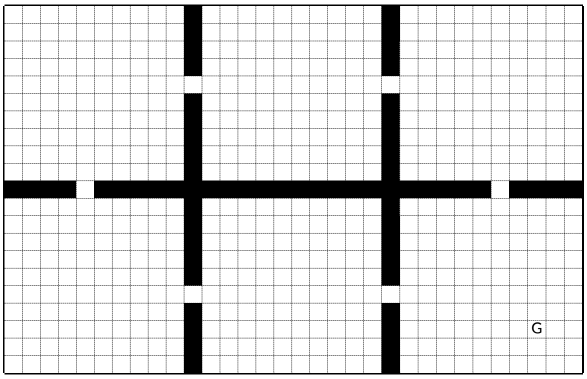
$\texttt{6Rooms}_{\texttt{P}}$ and $\texttt{4Rooms4Hallways}_{\texttt{P}}$ are navigational tasks where the agent can take four actions as *north*, *east*, *south*, *west*. Actions are stochastic, resulting in the intended direction with 0.95 probability and either left or right of the intended direction with 0.025 probability.

Table 6.1: Details of the domains used in the experiments for Anchor Based Guidance. The sub-script $P$ represents partial observability.

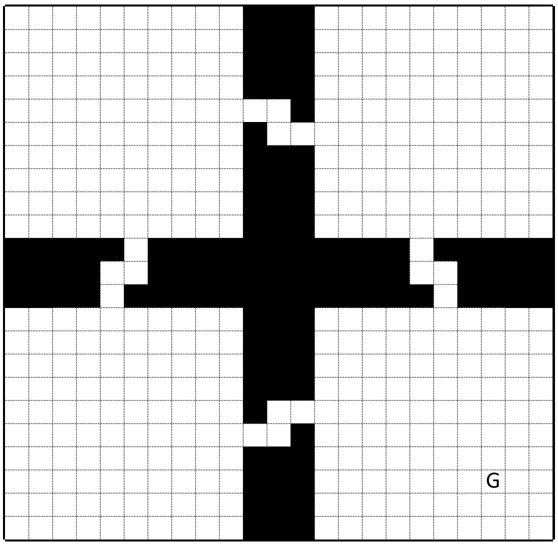
| **Problem** | **\|S\|** | **\|A\|** | **\|Ω\|** | **Action Noise** | **Reference** |
|---|---|---|---|---|---|
| $\texttt{6Rooms}_{\texttt{P}}$ | 564 | 4 | 43 | Yes | [3] |
| $\texttt{4Rooms4Hallways}_{\texttt{P}}$ | 374 | 4 | 12 | Yes | [2] |
| $\texttt{ToH3}_{\texttt{P}}$ | 161 | 4 | 31 | No | [51] |

In $\texttt{6Rooms}_{\texttt{P}}$, the agent's observations are formed according to its distance to the walls in four compass directions (one step from the wall, two steps from the wall, closer to the wall in this direction than the other, further from the wall in this direction than the other). Additionally, the doorways of $\texttt{6Rooms}_{\texttt{P}}$ provide unique observations. On the other hand, $\texttt{4Rooms4Hallways}_{\texttt{P}}$ limits the perceptions to the presence of a wall in the next cell in four directions. In both of the navigational domains, the agent starts from the upper left room and aims to reach to uniquely observable goal state (marked as G in Figures 6.5a and 6.5b) where each regular action is punished by $-0.01$ and reaching the goal state is rewarded by $1$.

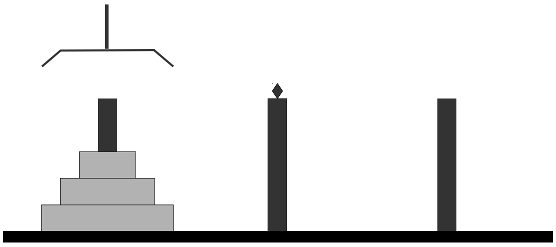
$\texttt{ToH3}_{\texttt{P}}$ is a classical puzzle that contains $m$ rods and $n$ disks where initially, the disks are placed either on the left-most or the middle rod in the increasing size order. The goal of the agent is to move the disks one at a time so that eventually they are placed on the right-most rod with the same order. In this version of the problem, the agent has an arm that can employ four actions as *left*, *right*, *pick up* and *put down*. The actions are deterministic and an action yields a reward of $10$ upon reaching to the



(a) `6Rooms`$_P$



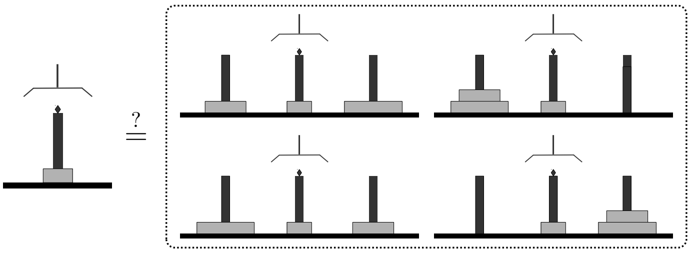
(b) `4Rooms4Hallways`$_P$



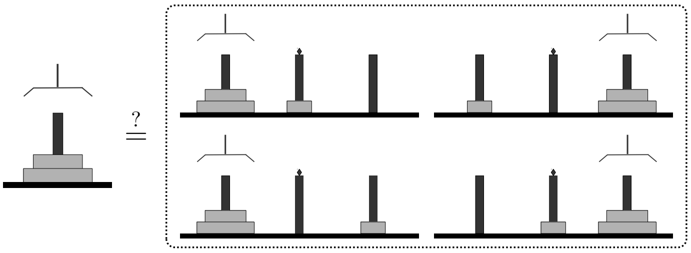
(c) `ToH3`$_P$

Figure 6.5: Sketches of the domains used in the experiments for Anchor Based Guidance. The goal states are marked with G in the grid world domains.

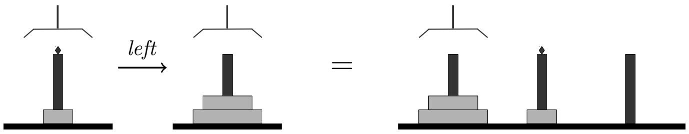
uniquely visible goal state, no reward or punishment otherwise. `ToH3`$_P$ contains 3 disks and 3 rods where the middle rod has an additional shape that is visible by the



(a) An observation on the middle rod with the smallest disk, corresponding to four different states.



(b) An observation on a rod with the disks except the smallest one, corresponding to four different states.

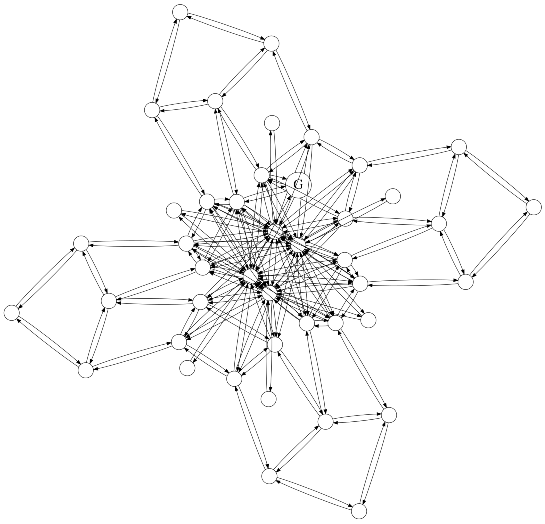


(c) The anchor generated by the transition between the observations via *left* action, mapping to only one state.

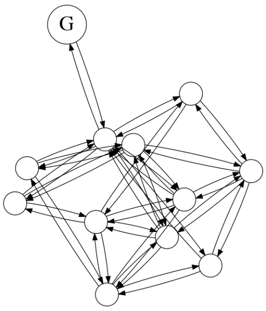
Figure 6.6: A simple example in `ToH3`$_{\mathrm{P}}$ problem where seeing only one rod may correspond to multiple configurations but a transition from one to another with *left* action is specific to a single state, creating an anchor formed as $x = o_{t-1}a_{t-1}o_t$.

agent. The agent's perception is limited to the contents of the rod that the arm is on, whether the current rod is the middle one or not and whether the arm is holding a disk or not. Due to this partial observability, the agent has to keep some sort of memory to form a good policy on the task.

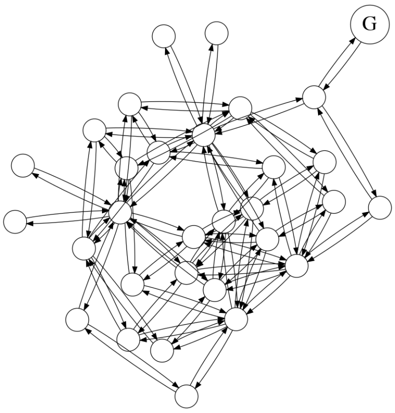
This study focuses on the domains that may lead to anchors in different forms of state estimation. `6Rooms`$_{\mathrm{P}}$ contains anchors in pure observational level since the doorways provide unique observations. On the other hand, the observation in `4-Rooms4Hallways`$_{\mathrm{P}}$ are highly ambiguous but the keeping a memory can create unique estimations. As in the example in Figure 6.2, observing top-right and bottom-



(a) `6Rooms`$_{\mathrm{P}}$



(b) `4Rooms4Hallways`$_{\mathrm{P}}$



(c) `ToH3`$_{\mathrm{P}}$

Figure 6.7: Observation transition graphs of the domains. The goal states are labeled as G.

left corners in an order can happen in only one state while these observations map to three states individually. For the case of `ToH3`$_\mathtt{P}$, the agent may further improve its memory to contain the action between the previous observation and the current one, creating anchors as shown in Figure 6.6.

Table 6.2: Details of the estimated state space with different state estimation forms where $\kappa$ is the average aliasing ratio.

| **State Estimate** | `6Rooms`$_\mathtt{P}$ | | | `4Rooms4Hallways`$_\mathtt{P}$ | | | `ToH3`$_\mathtt{P}$ | | |
|---|---|---|---|---|---|---|---|---|---|
| | $\lvert X\rvert$ | $\lvert\mathcal{A}\rvert$ | $\kappa$ | $\lvert X\rvert$ | $\lvert\mathcal{A}\rvert$ | $\kappa$ | $\lvert X\rvert$ | $\lvert\mathcal{A}\rvert$ | $\kappa$ |
| $x = o_t$ | 43 | 7 | 14.09 | 12 | 1 | 34.66 | 31 | 6 | 5.16 |
| $x = o_{t-1}o_t$ | 305 | 125 | 6.28 | 69 | 21 | 12.58 | 153 | 20 | 3.15 |
| $x = o_{t-1}a_{t-1}o_t$ | 1010 | 418 | 6.23 | 244 | 76 | 12.85 | 286 | 179 | 2.25 |

As it can be seen from Figure 6.7, the transition semantics of observations do not represent the structure of the problems. Moreover, Table 6.2 shows how the size of the set of estimated states $X$ and the number of anchors, change with different forms of state estimates. To further understand the how ambiguous the estimated states are, we propose a metric called *aliasing ratio*, which is defined as;

$$\kappa = \frac{\sum_{x\in X} |M(x)|}{|X|} \tag{61}$$

where $|M(x)|$ represents the number of true states that the state estimate $x$ maps to. Note that, this metric does not directly shows how difficult a problem is, yet it gives an idea about the uncertainty. It can be seen in Table 6.2, having memory can decrease the aliasing ratio on all of the problems, yet including the action between observations is only useful for `ToH3`$_\mathtt{P}$ due to nondeterminism on the outcomes of actions in other problems.

### 6.9.2 Settings

Q($\lambda$) and Sarsa($\lambda$) algorithms are compared to their versions with ABG and the coupling of DDCF and ABG, in terms of the number of steps to reach the goal state. The main experiments are performed with different forms of state estimation for `6Rooms`$_\mathtt{P}$, `4Rooms4Hallways`$_\mathtt{P}$ and `ToH3`$_\mathtt{P}$ as $x = o_t$, $x = o_{t-1}o_t$ and

$x = o_{t-1}a_{t-1}o_t$, respectively. Optimal MDP policy performance for the problems are also plotted as a baseline.

In the experiments, an episode starts at a randomly selected initial state of the problem and ends either when the agent reaches a goal state or after 5000 steps are taken. The results are averaged over 50 experiments, each of which took 25000 episodes.

For the learning parameters, we used the values that are previously used for these algorithms. $\lambda = 0.9$ is used for the main experimentation. For both RL algorithms and all of the domains, $\alpha = 0.01, \gamma = 0.9$ are used, and an $\epsilon$-greedy action selection method is employed with $\epsilon$ starting at $0.2$ and linearly decaying down to $0.0001$ throughout an experiment (until the last episode).

The value iteration of the Anchor-SMDP used $\gamma_v = 0.99$ and $\alpha_v = 0.05$ in all of the domains which give the best results for ABG. For ABG, we assumed that the anchor set $\mathscr{A}$ is provided beforehand, that is the agent is capable of perfectly sensing whether it is in an anchor or not. On the other hand, for the coupling of DDCF and ABG, we let DDCF to run for 2000 episodes, then employed ABG with the identified set of anchors. For DDCF, we used $\theta = 10.0$ and $\lambda_{DD} = 0.95$ where an episode is considered successful if it ends with a peak reward and lasts shorter than a determined step threshold. We used a static filter of 2 steps in `6Rooms`$_{\text{P}}$ and `4Rooms4Hallways`$_{\text{P}}$ where it is set to 3 for `ToH3`$_{\text{P}}$ problem. Moreover, we employed concept filtering by setting $k = 2$ in the congestion ratio metric.

### 6.9.3 Learning Performances

For the main experiments, we reported the average number of steps taken to reach the goal state with the $95\%$ bootstrapped confidence intervals. The agent used different forms of state estimation so that we can show the performance of our proposed method with different forms of anchors.

In Table 6.3, the identification performance of DDCF + ABG is given when the agent used state estimates as $x = o_t$, $x = o_{t-1}o_t$ and $x = o_{t-1}a_{t-1}o_t$ for `6Rooms`$_{\text{P}}$, `4Rooms4Hallways`$_{\text{P}}$ and `ToH3`$_{\text{P}}$ respectively. As stated earlier, DDCF was run for the first 2000 episodes to find the anchors in the task. In case of accuracy, DDCF is

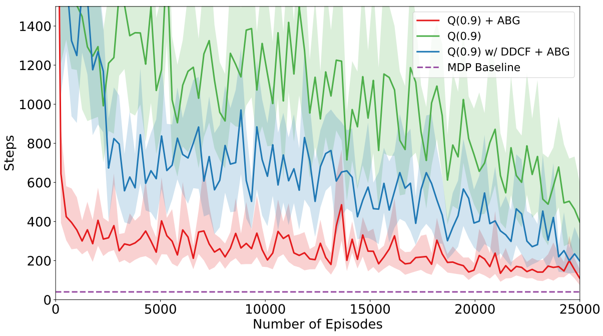
Table 6.3: Anchor identification performance of DDCF under different learning algorithms. Values are given with their lower and upper bound of confidence intervals.

| **Problem** | Q($\lambda$) | | Sarsa($\lambda$) | |
|---|---|---|---|---|
| | Precision | Recall | Precision | Recall |
| `6Rooms`$_\text{P}$ | 1.000 (1.000, 1.000) | 0.631 (0.586, 0.674) | 1.000 (1.000, 1.000) | 0.703 (0.657, 0.743) |
| `4Rooms4Hallways`$_\text{P}$ | 1.000 (1.000, 1.000) | 0.402 (0.383, 0.427) | 1.000 (1.000, 1.000) | 0.412 (0.388, 0.442) |
| `ToH3`$_\text{P}$ | 0.979 (0.965, 0.990) | 0.058 (0.054, 0.062) | 0.974 (0.961, 0.985) | 0.054 (0.050, 0.058) |

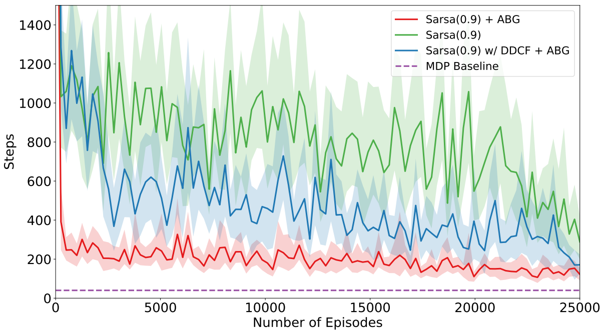
shown to perform well on different levels of state estimation. It resulted in noise free sets of anchors in almost all of the problems. On the other hand, the table shows that DDCF does not cover all the anchors. This behaviour originates from the focus of DDCF on the anchors that are most useful for success. Especially in `ToH3`$_\text{P}$, there are 179 anchors in the form $x = o_{t-1}a_{t-1}o_t$ (Table 6.2), yet only a few of them causes the highest DD values and are enough for guiding the agent.

Figure 6.8 shows the learning performances in `6Rooms`$_\text{P}$. Learning a policy based on the pure observations ($x = o_t$) seems to work in this domain. Although it is not optimal, all of the algorithms can find a good policy that can lead the agent towards high rewards. It is clear from the figure that ABG dramatically improves the learning performance since it guides the agent to the goal state from the beginning of the learning with the anchor set provided beforehand. Likewise, the agent's performance improves when the set of anchors found by DDCF is fed to ABG and guidance starts. Around the episode 2000, the number of steps to the goal state start to decrease significantly. This shows that the prior knowledge of the anchor set is not necessary, yet it can be found online. Although DDCF does not discover all the anchors, the found ones are sufficient for improving the learning speed.

For `4Rooms4Hallways`$_\text{P}$ domain, keeping a memory may be required in order to observe some sort of learning. In Figure 6.9, it is shown that keeping the previous observation in the estimated state form can improve the performance of the baseline algorithms, Q($\lambda$) and Sarsa($\lambda$). In this level of state estimation, Figure 6.10 shows a similar result. Both baseline algorithms gives a slow learning sign where Sarsa($\lambda$) performs better compared to Q($\lambda$). On the other hand, ABG improves both of the underlying algorithms. Moreover, DDCF + ABG has a similar sudden effect on the



(a) Q($\lambda$)



(b) Sarsa($\lambda$)

Figure 6.8: Average number of steps taken to reach the goal state in `6Rooms`$_P$ domain where the state estimation has the form of $x = o_t$. The dashed line represents the best value from the MDP version of the problem and shaded areas are the $95\%$ bootstrapped confidence intervals.

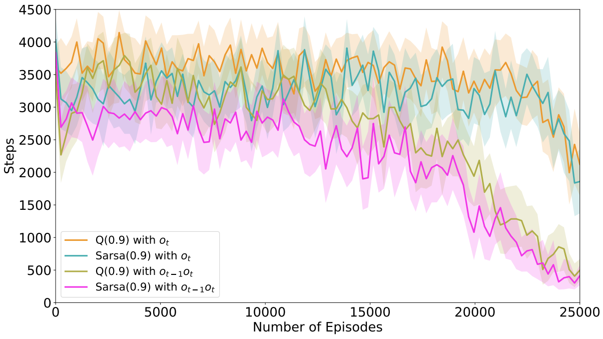
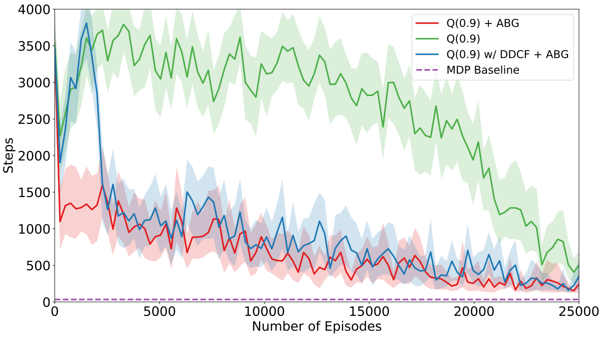
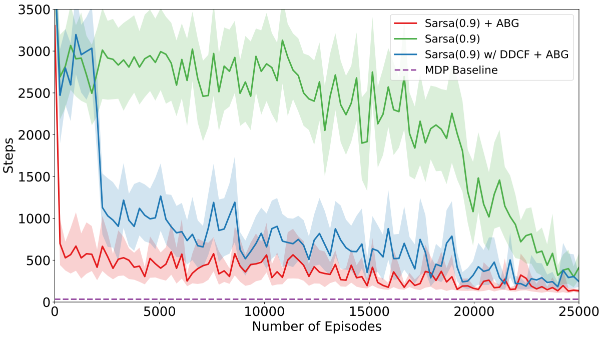


Figure 6.9: Average number of steps taken to reach the goal state in `4Rooms4-Hallways`$_P$ domain with different state estimation forms. Shaded areas are the $95\%$ bootstrapped confidence intervals.

learning performance after the guidance begins. This shows that the overall algorithm is also helpful when more complex anchors are present.



(a) Q($\lambda$)



(b) Sarsa($\lambda$)

Figure 6.10: Average number of steps taken to reach the goal state in `4Rooms-4Hallways`$_{\text{P}}$ domain where the state estimation has the form of $x = o_{t-1}o_t$. The dashed line represents the best value from the MDP version of the problem and shaded areas are the $95\%$ bootstrapped confidence intervals.

Figure 6.11 shows that the baseline algorithms cannot learn by using the single observations from the environment, yet including the actions between the previous and the current observations seems to make the agent solve the task in `ToH3`$_{\text{P}}$. It can be seen in Figure 6.12, all of the algorithms almost converge to the best value taken from the MDP version of the problem when the state estimation is formed as $x = o_{t-1}a_{t-1}o_t$. Again, anchor based guidance led the agent towards the best policy much sooner. Although DDCF's coverage on the anchors is low, its improvement over the baseline algorithms are significant. This proves that not all of the anchors are to be used to achieve good guidance.

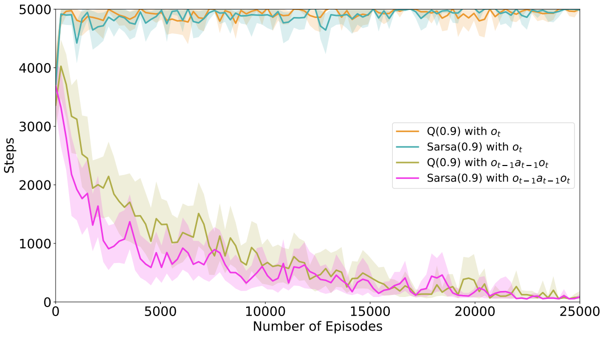
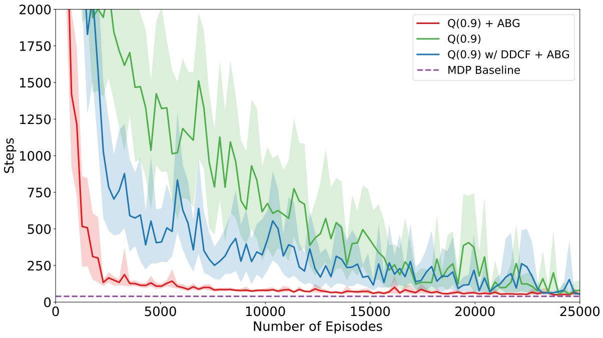
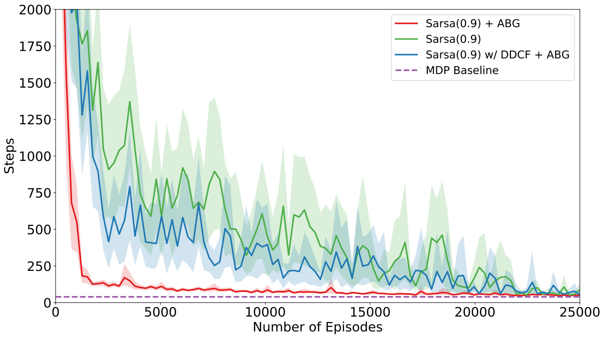


Figure 6.11: Average number of steps taken to reach the goal state in `ToH3P` domain with different state estimation forms. Shaded areas are the $95\%$ bootstrapped confidence intervals.



(a) Q($\lambda$)



(b) Sarsa($\lambda$)

Figure 6.12: Average number of steps taken to reach the goal state in `ToH3P` domain where the state estimation has the form of $x = o_{t-1}a_{t-1}o_t$. The dashed line represents the best value from the MDP version of the problem and shaded areas are the $95\%$ bootstrapped confidence intervals.

### 6.9.4 Analysis on Guidance with Subsets of Anchors

We further analyzed the learning performance of Anchor Based Guidance when only a subset of the true anchor set is provided to the agent. This way, we can demonstrate the effect of the anchor set on the guidance method.

At the beginning of each experiment, we take a random subset of anchors and run ABG with this subset. We experimented with four percentages as $20\%$, $40\%$, $60\%$ and $80\%$ and compared the results to ABG with all the anchors given ($100\%$) and the baseline algorithm without any guidance ($0\%$), which is Sarsa($\lambda$) for this setting. In the experiments, we used `4Rooms4Hallways`$_\texttt{P}$ domain with the state estimation $x = o_{t-1}o_t$ where there are 21 anchors in this form (Table 6.2).

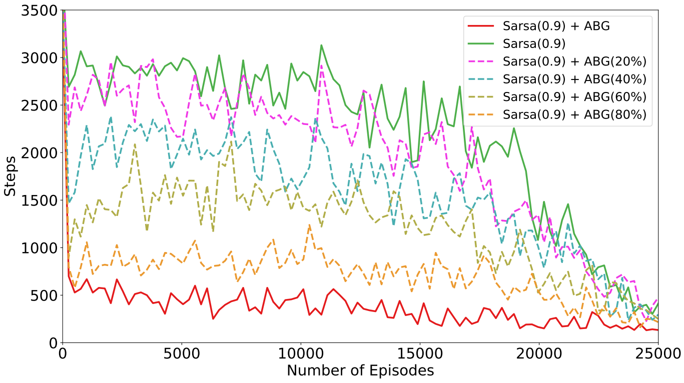


Figure 6.13: Average number of steps to goal in `4Rooms4Hallways`$_\texttt{P}$ when ABG is fed with the different subsets of the anchor set. The percentage of the provided subsets are given in parentheses and the confidence intervals are omitted for better view.

Figure 6.13 shows the number of steps taken to the goal averaged over 50 experiments. As expected, knowing a bigger subset of the anchors leads to a better learning performance since the agent can depend on more anchors and rewarded more frequently. Additionally, we compared these results to the performance of DDCF + ABG where the method scored $0.41$ recall, which is nearly $40\%$ of the anchors. It can be seen in Figure 6.14 that the performance of DDCF + ABG matches to having a random subset of $80\%$, even though it had a late start on guidance due to the episode threshold. This supports our claim that DDCF is more focused on anchors that are useful for learning the task at hand.

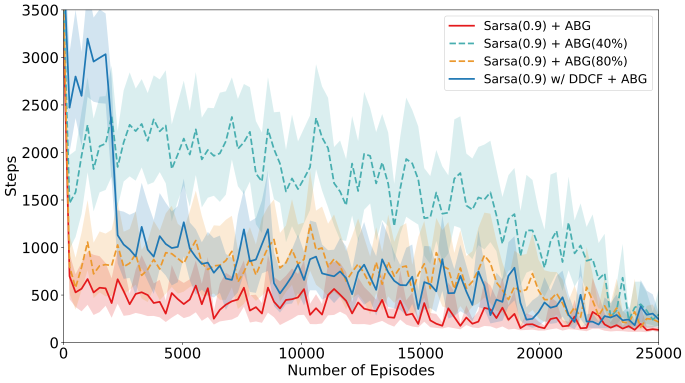


Figure 6.14: Comparison of DDCF + ABG to ABG with the subsets of anchors in terms of average number of steps to goal in `4Rooms4Hallways`$_{\text{P}}$. The percentage of the provided subsets are given in parentheses and shaded areas are the $95\%$ bootstrapped confidence intervals.

## 6.10 Summary and Discussion

In this chapter, an anchor based guidance approach is applied to the partially observable problem setting, where the states are hidden from the agent. The proposed method, ABG, makes use of anchors to baseline a potential function for introducing additional rewards. ABG approach argues that the RL agent can achieve a better solution by applying internal rewards whenever a transition is completed among two anchors at the abstract level. Experiments on several problems show that ABG can significantly improve the learning performance of well known off-policy and on-policy learning algorithms, like Q($\lambda$) and Sarsa($\lambda$). ABG can be further coupled with DDCF to remove the necessity of having anchors beforehand by discovering them online. DDCF + ABG performed significantly better than the baselines yet close to the case where all the anchors are given a priori.

The study shows that anchors can be found on different levels of state estimation forms. Their natural presence can be utilized by guiding with additional rewards. Both the anchors and their potentials can be found online and DDCF + ABG is a complete algorithm that combines discovery and usage of anchors.

# CHAPTER 7

# CONCLUSION

This chapter summarizes the attacked problem, the work in the thesis and discusses possible future ways to continue.

## 7.1 Summary

Reinforcement Learning aims to maximize the rewards taken from an environment while interacting with it. Each experience in the world may be costly. Hence, the agent needs to learn efficiently, that is, solve the task in a small amount of time while avoiding harmful consequences. One way to accelerate Reinforcement Learning is to look for the clues of the environment. Such clues may correspond to the bottleneck regions of the state space, defined as *subgoals*, or unique experiences in uncertainty which may be defined as *landmarks* or *anchors* to be more general. A Reinforcement Learning agent with the capability of identifying these clues while learning can indeed learn the problem much faster by taking advantage of their presence.

This thesis focuses on the clues of a world in terms of their automatic discovery and usage in Reinforcement Learning. It covers a wide range of models such as MDP, SMDP and POMDP.

Firstly, Chapter 3 attacks the problem of automatic option generation and aims to develop the common method to create initiation sets for subgoal driven options in MDPs. This greedy heuristic includes all the states visited before the subgoal state to the initiation set of the option reaching it. However, this idea ends up adding redundant states to the initiation set, creating less effective options. The agent may have a shorter path to high rewards on some states and employing an option to reach a subgoal from these may send the agent further away from a goal state, causing a waste of time. Our proposed heuristic, *history tree*, forms a tree where a parent state is the best state to go from a child according to their values. Then, our algorithm employs a search in the tree and only includes the states that are below the target subgoal. This way, the created option is goal oriented and redundant states are avoided to be added to the initiation set. In the study, we coupled our heuristic with well known online subgoal discovery methods to form a complete online framework. The experiments showed that the options created with history tree heuristic are more effective to lead the agent towards high rewards and our heuristic provide a better set of options.

Secondly, in Chapter 4, we studied on a multiple instance learning algorithm adopted to subgoal identification, called Diverse Density, since there is a room for improvement and it is a good candidate to work under partial observability. We proposed a novel metric named *congestion ratio* to eliminate redundant candidates from the exhaustive search for diverse density values. Our version of the algorithm including concept filtering mechanism, *Diverse Density with Concept Filtering*, requires less time to identify the subgoals since it uses a local metric to filter out unnecessary concepts. Moreover, the original Diverse Density algorithm requires the distances between concepts beforehand in order to calculate a similarity value. Our modified algorithm removes this necessity by keeping an interaction graph formed during learning. Hence, this improvement enables Diverse Density to be more suitable for realistic tasks. In this chapter, we experimented on both fully and partially observable problems. We showed that Diverse Density, unlike other online subgoal discovery methods, is able to identify bottleneck regions in POMDPs with hidden states, which is a promising step towards an unexplored field.

Thirdly, we further dived into problems with hidden states in Chapter 5 and focused on the definition of a landmark which found some ground in the field. A landmark is a state that provides a unique observation and can be used to accelerate the speed of well known algorithms such as Sarsa($\lambda$). It has been shown that the modification of Sarsa($\lambda$) to problems with landmarks, called SarsaLandmark, can converge to a good policy faster. However, the algorithm assumes that the landmarks are known in advance. We employ Diverse Density for online landmark discovery task as its idea still holds in this model. Our detailed experiments shows that Diverse Density is able to find the landmarks in a POMDP with hidden states and the overall framework of *SarsaLandmark with Automatic Landmark Discovery* outperforms its baseline Sarsa($\lambda$) while performing similarly to SarsaLandmark with landmark provided beforehand. In case of identification, DDCF performs more accurately and has a wider coverage compared to the McGovern's original Diverse Density.

Finally, Chapter 6 examines the cases where the agent requires additional memory, forming a state estimate, to solve a given partially observable task. In such a case, there may be no landmark states, yet some of the agent's experiences can clear ambiguity. We put a new definition, *anchor*, for the experiences that constitutes no uncertainty about the current state of the agent. Then, we devised a new method to inform the agent about its progress by providing additional rewards. These rewards are applied when the agent completes an abstract transition between two anchors, that is, two dependable estimated states. To calculate the potentials of the anchors, we followed a similar study in the reward shaping field and formed the potential function during learning. Furthermore, we again used Diverse Density with Concept Filtering to identify the anchors while learning. Our results demonstrate that applying guiding rewards based on anchors dramatically improves the performance of two well known algorithms, Q($\lambda$) and Sarsa($\lambda$). Although not optimal, the algorithms coupled with Anchor Based Guiding converges to a good policy much faster. Also, the complete algorithm of DDCF and ABG again outperforms the baselines and gives a close performance to ABG with anchors provided beforehand. The experiments with DDCF + ABG showed that DDCF is definitely more focused on useful anchors due to its performance almost matching with the best scenario of knowing all the anchors.

## 7.2 Future Work

The work proposed in this thesis has several follow up directions.

History tree heuristic given in Chapter 3, like possibly any other option initiation set generation heuristic, is expected to suffer from dynamically changing environments. In other words, one of the assumptions these heuristics make is that the environment is non-stationary. Although the learning agent will adapt to the change after a while, the previously constructed history tree structure will remain, without being re-used. In that sense, to handle non-stationary problems, a mechanism is needed to forget options that are not used for a long time. Since many problems contain more than one goal state, another future work can be exploring the improvement opportunities for the selection of the root state (among many goal states) which determines the direction of the generated options. It can be reasonable to choose a root state that an identified subgoal is most useful to reach, or choose multiple roots causing multiple options aiming to go to the same terminal state through different directions. Moreover, the generated tree can also directly be used for construction of the option policy. The only missing parts are the actions that lead the agent through the paths in the tree. The artificial rewards used in experience replay mechanism can be modified so that best target for each state corresponds to the parent of that state in the tree.

We believe that the congestion ratio metric, introduced in Chapter 4, can be useful in other areas like social networks. A possible follow up to DDCF can focus on improving the DD's identification performance further. Another apparent future work for Chapter 5 involves making use of temporal ordering of identified landmarks to improve learning performance. Moreover, DDCF can be adapted for continuous spaces with a proper discretization method, which seems to be an immediate extension.

As a follow-up work to Anchor Based Guiding in Chapter 6, one can experiment with algorithms that devise their state estimations while learning, rather than having a fixed form at the beginning. Methods like USM, extend the memory whenever necessary, causing a set of estimated states with different sizes. The proposed framework of DDCF + ABG is still a good candidate to work under such circumstances. DDCF can pick the anchors among the set of estimated states and ABG can fuse them to provide guiding rewards.

## 7.3 Final Remarks

Identifying clues while learning is a challenging task but it benefits the agent a great deal. Additionally, operating under partial observability comes with its own difficulties. In fact, it is always possible to create problems with hidden states, that allows no learning. When the information about the problem is so limited, it may seem as if the future is completely unpredictable. It resembles a situation of having more unknowns than the number of equations. One may need to have more information to solve the system. Finding that information plays a crucial role under such circumstances.

# REFERENCES

[1] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. MIT Press, second ed., 2018.

[2] A. McGovern and A. G. Barto, “Automatic discovery of subgoals in reinforcement learning using diverse density,” in *Proceedings of the Eighteenth International Conference on Machine Learning*, ICML ’01, pp. 361–368, Morgan Kaufmann Publishers Inc., 2001.

[3] I. Menache, S. Mannor, and N. Shimkin, “Q-Cut—Dynamic Discovery of Subgoals in Reinforcement Learning,” in *Machine Learning: ECML 2002: 13th European Conference on Machine Learning Proceedings*, vol. 2430 of *LNCS*, pp. 295–306, Springer-Verlag, 2002.

[4] Ö. Şimşek, *Behavioral Building Blocks for Autonomous Agents: Description, Identification, and Learning*. Ph.D. thesis, University of Massachusetts Amherst, 2008.

[5] S. J. Kazemitabar and H. Beigy, “Automatic discovery of subgoals in reinforcement learning using strongly connected components,” in *Advances in Neuro-Information Processing: ICONIP ’08 Revised Selected Papers, Part I*, vol. 5506 of *LNCS*, pp. 829–834, Springer-Verlag, 2008.

[6] F. Chen, S. Chen, Y. Gao, and Z. Ma, “Connect-based subgoal discovery for options in hierarchical reinforcement learning,” in *Proceedings of the Third International Conference on Natural Computation*, vol. 4 of *ICNC ’07*, pp. 698–702, IEEE, 2007.

[7] S. Mannor, Ishai Menache, Amit Hoze, and Uri Klein, “Dynamic abstraction in reinforcement learning via clustering,” in *Proceedings of the Twenty-first International Conference on Machine Learning*, ICML’04, pp. 71–78, ACM, 2004.

[8] M. R. James and S. P. Singh, “Sarsalandmark: an algorithm for learning in pomdps with landmarks,” in *8th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2009)*, pp. 585–591, 2009.

[9] S. J. Bradtke and M. O. Duff, “Reinforcement learning methods for continuous-time markov decision problems,” in *Advances In Neural Information Processing Systems*, vol. 7 of *NIPS 1994*, (Cambridge, MA), pp. 393–400, MIT Press, 1994.

[10] R. S. Sutton, D. Precup, and S. Singh, “Between MDPs and semi-MDPs: a framework for temporal abstraction in reinforcement learning,” *Artificial Intelligence*, vol. 112, no. 1-2, pp. 181–211, 1999.

[11] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra, “Planning and acting in partially observable stochastic domains,” *Artificial Intelligence*, vol. 101, no. 1-2, pp. 99–134, 1998.

[12] S. D. Whitehead and D. H. Ballard, “Learning to perceive and act by trial and error,” *Machine Learning*, vol. 7, no. 1, pp. 45–83, 1991.

[13] R. S. Sutton, “Learning to predict by the methods of temporal differences,” *Machine Learning*, vol. 3, pp. 9–44, 1988.

[14] C. Watkins, *Learning from Delayed Rewards*. Ph.D. thesis, Cambridge University, 1989.

[15] G. A. Rummery and M. Niranjan, *On-line Q-learning using connectionist systems*, vol. 37. University of Cambridge, Department of Engineering Cambridge, England, 1994.

[16] J. Loch and S. P. Singh, “Using eligibility traces to find the best memoryless policy in partially observable markov decision processes,” in *Proceedings of the Fifteenth International Conference on Machine Learning*, ICML’98, pp. 323–331, Morgan Kaufmann Publishers Inc., 1998.

[17] A. McGovern, R. S. Sutton, and A. H. Fagg, “Roles of macro-actions in accelerating reinforcement learning,” *Proceedings of the 1997 Grace Hopper Celebration of Women in Computing*, pp. 13–18, 1997.

[18] K. J. Astrom, “Optimal control of markov processes with incomplete state information,” *Journal of mathematical analysis and applications*, vol. 10, no. 1, pp. 174–205, 1965.

[19] M. O. Duff and A. Barto, *Optimal Learning: Computational procedures for Bayes-adaptive Markov decision processes*. PhD thesis, University of Massachusetts at Amherst, 2002.

[20] R. Jaulmes, J. Pineau, and D. Precup, “Active learning in partially observable markov decision processes,” in *European Conference on Machine Learning*, pp. 601–608, Springer, 2005.

[21] S. Ross, B. Chaib-draa, and J. Pineau, “Bayes-adaptive pomdps,” in *Advances in neural information processing systems*, pp. 1225–1232, 2008.

[22] S. Ross, J. Pineau, B. Chaib-draa, and P. Kreitmann, “A bayesian approach for learning and planning in partially observable markov decision processes,” *Journal of Machine Learning Research*, vol. 12, pp. 1729–1770, 2011.

[23] N. Vlassis, M. Ghavamzadeh, S. Mannor, and P. Poupart, “Bayesian reinforcement learning,” in *Reinforcement learning*, pp. 359–386, Springer, 2012.

[24] A. McCallum, *Reinforcement Learning with Selective Perception and Hidden State*. Ph.D. thesis, University of Rochester, 1996.

[25] S. P. Singh, T. Jaakkola, and M. I. Jordan, “Learning without state-estimation in partially observable markovian decision processes,” in *Machine Learning Proceedings 1994*, pp. 284–292, Elsevier, 1994.

[26] L. J. Lin and T. M. Mitchell, *Memory approaches to reinforcement learning in non-Markovian domains*. Citeseer, 1992.

[27] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[28] T. G. Dietterich, “Hierarchical reinforcement learning with the MAXQ value function decomposition,” *Journal of artificial intelligence research*, vol. 13, pp. 227–303, 2000.

[29] R. Parr and S. Russell, "Reinforcement learning with hierarchies of machines," in *Advances In Neural Information Processing Systems*, vol. 10 of *NIPS '97*, pp. 1043–1049, MIT Press, 1998.

[30] A. Demir, E. Çilden, and F. Polat, "Generating effective initiation sets for subgoal-driven options," *Advances in Complex Systems*, vol. 22, 2019.

[31] B. Hengst, "Hierarchical approaches," in *Reinforcement Learning: State-of-the-Art*, vol. 12 of *Adaptation, Learning, and Optimization*, pp. 293–323, Springer Berlin Heidelberg, 2012.

[32] B. L. Digney, "Learning hierarchical control structures for multiple tasks and changing environments," in *Proceedings of the fifth international conference on simulation of adaptive behavior on From animals to animats*, pp. 321–330, MIT Press, 1998.

[33] M. Stolle and D. Precup, "Learning options in reinforcement learning," in *Proceedings of the 5th International Symposium on Abstraction, Reformulation, and Approximation*, vol. 2371 of *LNCS*, pp. 212–223, Springer Berlin / Heidelberg, 2002.

[34] S. Goel and M. Huber, "Subgoal discovery for hierarchical reinforcement learning using learned policies," in *In Proceedings of the 16th International FLAIRS Conference* (I. Russell and S. M. Haller, eds.), FLAIRS'03, pp. 346–350, AAAI Press, 2003.

[35] D. Xiao, Y.-t. Li, and C. Shi, "Autonomic discovery of subgoals in hierarchical reinforcement learning," *The Journal of China Universities of Posts and Telecommunications*, vol. 21, pp. 94–104, Oct. 2014.

[36] Ö. Şimşek, A. P. Wolfe, and A. G. Barto, "Identifying useful subgoals in reinforcement learning by local graph partitioning," in *Proceedings of the 22nd international conference on Machine Learning*, ICML'05, pp. 816–823, ACM, 2005.

[37] U. Brandes, "A faster algorithm for betweenness centrality," *The Journal of Mathematical Sociology*, vol. 25, pp. 163–177, June 2001.

[38] J. Shi and J. Malik, “Normalized cuts and image segmentation,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 8, pp. 888–905, 2000.

[39] Y. A. Wei and C. Cheng, “Ratio cut partitioning for hierarchical designs,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 10, no. 7, pp. 911–921, 1991.

[40] T. Yoshikawa and M. Kurihara, “An acquiring method of macro-actions in reinforcement learning,” in *IEEE International Conference on Systems, Man, and Cybernetics*, vol. 6 of *SMC’06*, pp. 4813–4817, 2006.

[41] N. K. Jong, T. Hester, and P. Stone, “The utility of temporal abstraction in reinforcement learning,” in *Proceedings of the 7th International Conference on Autonomous Agents and Multiagent Systems*, vol. 1 of *AAMAS ’08*, pp. 299–306, International Foundation for Autonomous Agents and Multiagent Systems, May 2008.

[42] Ö. Şimşek and A. G. Barto, “Using relative novelty to identify useful temporal abstractions in reinforcement learning,” in *Proceedings of the Twenty-first International Conference on Machine Learning*, ICML ’04, pp. 95–102, ACM, 2004.

[43] E. A. McGovern, *Autonomous Discovery of Temporal Abstractions from Interaction with an Environment.* Ph.D. thesis, University of Massachusetts Amherst, 2002.

[44] S. Girgin, F. Polat, and R. Alhajj, “Improving reinforcement learning by using sequence trees,” *Machine Learning*, vol. 81, no. 3, pp. 283–331, 2010.

[45] G. Konidaris and A. S. Barreto, “Skill discovery in continuous reinforcement learning domains using skill chaining,” in *Advances in Neural Information Processing Systems*, pp. 1015–1023, 2009.

[46] A. Jonsson and A. Barto, “Causal graph based decomposition of factored MDPs,” *Journal of Machine Learning Research*, vol. 7, pp. 2259–2301, Nov. 2006.

[47] O. Kozlova, O. Sigaud, and C. Meyer, “Automated discovery of options in factored reinforcement learning,” in *Proceedings of the ICML/UAI/COLT workshop on abstraction in reinforcement learning*, pp. 24–29, 2009.

[48] L. J. Lin, “Self-improving reactive agents based on reinforcement learning, planning and teaching,” *Machine Learning*, vol. 8, pp. 293–321, 1992.

[49] A. Demir, E. Çilden, and F. Polat, “A history tree heuristic to generate better initiation sets for options in reinforcement learning (extended abstract),” in *ECAI 2016 - 22nd European Conference on Artificial Intelligence*, pp. 1644–1645, 2016.

[50] L. T. Dung, T. Komeda, and M. Takagi, “Solving POMDPs with Automatic Discovery of Subgoals,” in *Theory and Novel Applications of Machine Learning*, pp. 229–238, InTech, 2009.

[51] D. R. Hofstadter, *Metamagical Themas: Questing for the Essence of Mind and Pattern.* Basic Books, Inc., 1985.

[52] A. Demir, E. Çilden, and F. Polat, “A concept filtering approach for diverse density to discover subgoals in reinforcement learning,” in *Proceedings of the 29th IEEE International Conference on Tools with Artificial Intelligence*, ICTAI ’17, pp. pp. 1–5, (Short Paper), Nov. 2017.

[53] O. Maron and T. Lozano-Pérez, “A framework for multiple-instance learning,” in *Proceedings of the 1997 conference on Advances in Neural Information Processing Systems 10*, NIPS’97, pp. 570–576, MIT Press, 1998.

[54] D. J. Watts and S. H. Strogatz, “Collective dynamics of ’small-world’ networks,” *Nature*, vol. 393, no. 6684, pp. 440–442, 1998.

[55] B. Jiang and C. Claramunt, “Topological Analysis of Urban Street Networks,” *Environment and Planning B: Planning and Design*, vol. 31, no. 1, pp. 151–162, 2004.

[56] W. Hwang, T. Kim, M. Ramanathan, and A. Zhang, “Bridging centrality: graph mining from element level to group level,” in *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 336–344, ACM, 2008.

[57] B. Yang and J. Liu, “Discovering global network communities based on local centralities,” *ACM Transactions on the Web*, vol. 2, no. 1, pp. 1–32, 2008.

[58] E. W. Dijkstra, “A note on two problems in connexion with graphs,” *Numerische mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[59] L. C. Freeman, “A set of measures of centrality based on betweenness,” *Sociometry*, vol. 40, pp. 35–41, Mar. 1977.

[60] L. Chrisman, “Reinforcement learning with perceptual aliasing: the perceptual distinctions approach,” in *Proc. of the Tenth National Conf. on AI*, AAAI’92, pp. 183–188, AAAI Press, 1992.

[61] A. Demir, E. Çilden, and F. Polat, “Automatic landmark discovery for learning agents under partial observability,” *The Knowledge Engineering Review*, vol. 34, 2019.

[62] Wikipedia, “Landmark,” 2018.

[63] M. Elkawkagy, P. Bercher, B. Schattenberg, and S. Biundo, “Improving Hierarchical Planning Performance by the Use of Landmarks,” in *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence*, pp. 1763–1769, 2012.

[64] J. Hoffmann, J. Porteous, and L. Sebastia, “Ordered landmarks in planning,” *Journal of Artificial Intelligence Research*, vol. 22, pp. 215–278, 2004.

[65] E. Karpas, D. Wang, B. C. Williams, and P. Haslum, “Temporal Landmarks: What Must Happen, and When,” in *Proceedings of the Twenty-Fifth International Conference on Automated Planning and Scheduling*, pp. 138–146, 2015.

[66] A. Lazanas and J. C. Latombe, “Motion planning with uncertainty: a landmark approach,” *Artificial Intelligence*, vol. 76, no. 1-2, pp. 287–317, 1995.

[67] L. Frommberger, “Representing and selecting landmarks in autonomous learning of robot navigation,” in *Intelligent Robotics and Applications, First International Conference, ICIRA 2008*, pp. 488–497, 2008.

[68] A. Howard and L. Kitchen, “Navigation using natural landmarks,” *Robotics and Autonomous Systems*, vol. 26, no. 2-3, pp. 99–115, 1999.

[69] S. Koenig and R. G. Simmons, "Xavier: A robot navigation architecture based on partially observable markov decision process models," in *Artificial Intelligence and Mobile Robots*, pp. 91–122, MIT Press, 1998.

[70] T. Välimäki and R. Ritala, "Optimizing gaze direction in a visual navigation task," in *IEEE International Conference on Robotics and Automation*, ICRA, pp. 1427–1432, IEEE, 2016.

[71] L. T. Dung, T. Komeda, and M. Takagi, "Reinforcement learning in non-markovian environments using automatic discovery of subgoals," in *SICE, 2007 Annual Conference*, pp. 2601–2605, 2007.

[72] J. Randløv and P. Alstrøm, "Learning to Drive a Bicycle Using Reinforcement Learning and Shaping," in *ICML 1998, Madison, Wisconsin, USA, July 24-27, 1998*, pp. 463–471, 1998.

[73] A. Y. Ng, D. Harada, and S. J. Russell, "Policy invariance under reward transformations: Theory and application to reward shaping," in *Proceedings of the Sixteenth International Conference on Machine Learning (ICML 1999)*, pp. 278–287, 1999.

[74] E. Wiewiora, "Potential-based shaping and q-value initialization are equivalent," *Journal of Artificial Intelligence Research*, vol. 19, pp. 205–208, 2003.

[75] M. Grzes and D. Kudenko, "Plan-based reward shaping for reinforcement learning," in *2008 4th International IEEE Conference Intelligent Systems*, vol. 2, pp. 10–22, IEEE, 2008.

[76] M. Grzes, *Improving exploration in reinforcement learning through domain knowledge and parameter analysis*. PhD Thesis, University of York, 2010.

[77] K. Efthymiadis and D. Kudenko, "Using plan-based reward shaping to learn strategies in starcraft: Broodwar," in *2013 IEEE Conference on Computational Inteligence in Games (CIG)*, pp. 1–8, IEEE, 2013.

[78] K. Efthymiadis, S. Devlin, and D. Kudenko, "Overcoming incorrect knowledge in plan-based reward shaping," *The Knowledge Engineering Review*, vol. 31, no. 1, pp. 31–43, 2016.

[79] M. Midtgaard, L. Vinther, J. R. Christiansen, A. M. Christensen, and Y. Zeng, "Time-based reward shaping in real-time strategy games," in *International Workshop on Agents and Data Mining Interaction*, pp. 115–125, Springer, 2010.

[80] Y. Gao and F. Toni, "Potential Based Reward Shaping for Hierarchical Reinforcement Learning," in *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015*, pp. 3504–3510, 2015.

[81] J. Asmuth, M. L. Littman, and R. Zinkov, "Potential-based Shaping in Model-based Reinforcement Learning," in *AAAI 2008, Chicago, Illinois, USA, July 13-17, 2008*, pp. 604–609, 2008.

[82] M. Grzes and D. Kudenko, "Theoretical and empirical analysis of reward shaping in reinforcement learning," in *2009 International Conference on Machine Learning and Applications*, pp. 337–344, IEEE, 2009.

[83] S. M. Devlin and D. Kudenko, "Dynamic potential-based reward shaping," in *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems*, pp. 433–440, IFAAMAS, 2012.

[84] M. Grzes, "Reward shaping in episodic reinforcement learning," in *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems, AAMAS 2017*, pp. 565–573, 2017.

[85] O. Marom and B. Rosman, "Belief reward shaping in reinforcement learning," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[86] M. Babes, E. M. De Cote, and M. L. Littman, "Social reward shaping in the prisoner's dilemma," in *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 3*, pp. 1389–1392, International Foundation for Autonomous Agents and Multiagent Systems, 2008.

[87] X. Lu, H. M. Schwartz, and S. N. Givigi, "Policy invariance under reward transformations for general-sum stochastic games," *Journal of Artificial Intelligence Research*, vol. 41, pp. 397–406, 2011.

[88] S. Devlin, D. Kudenko, and M. Grześ, “An empirical study of potential-based reward shaping and advice in complex, multi-agent systems,” *Advances in Complex Systems*, vol. 14, no. 02, pp. 251–278, 2011.

[89] S. Devlin and D. Kudenko, “Theoretical considerations of potential-based reward shaping for multi-agent systems,” in *The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 1*, pp. 225–232, International Foundation for Autonomous Agents and Multiagent Systems, 2011.

[90] S. Devlin and D. Kudenko, “Plan-based reward shaping for multi-agent reinforcement learning,” *The Knowledge Engineering Review*, vol. 31, no. 1, pp. 44–58, 2016.

[91] B. Marthi, “Automatic shaping and decomposition of reward functions,” in *Proceedings of the 24th International Conference on Machine learning*, pp. 601–608, ACM, 2007.

[92] M. Grzes and D. Kudenko, “Online learning of shaping rewards in reinforcement learning,” *Neural Networks*, vol. 23, no. 4, pp. 541–550, 2010.

[93] A. Eck, L.-K. Soh, S. Devlin, and D. Kudenko, “Potential-based reward shaping for finite horizon online POMDP planning,” *Auton. Agent. Multi Agent Syst.*, vol. 30, no. 3, pp. 403–445, 2016.

[94] S. Singh, R. L. Lewis, A. G. Barto, and J. Sorg, “Intrinsically Motivated Reinforcement Learning: An Evolutionary Perspective,” *IEEE Transactions on Autonomous Mental Development*, vol. 2, no. 2, pp. 70–82, 2010.

[95] P. Y. Oudeyer and F. Kaplan, “What is intrinsic motivation? A typology of computational approaches,” *Frontiers in Neurorobotics*, vol. 3, no. NOV, p. 6, 2009.

[96] A. G. Barto and Ö. Şimşek, “Intrinsic motivation for reinforcement learning systems,” in *Proceedings of the Thirteenth Yale Workshop on Adaptive and Learning Systems*, pp. 113–118, 2005.

[97] A. G. Barto, “Intrinsic motivation and reinforcement learning,” in *Intrinsically Motivated Learning in Natural and Artificial Systems*, pp. 17–47, Springer, 2013.

[98] M. G. Bellemare, S. Srinivasan, G. Ostrovski, T. Schaul, D. Saxton, and R. Munos, “Unifying count-based exploration and intrinsic motivation,” in *Advances in Neural Information Processing Systems*, pp. 1479–1487, Curran Associates, Inc., 2016.

[99] H. Tang, R. Houthooft, D. Foote, A. Stooke, X. Chen, Y. Duan, J. Schulman, F. De Turck, and P. Abbeel, “Exploration: A study of count-based exploration for deep reinforcement learning,” in *Advances in Neural Information Processing Systems*, pp. 2754–2763, Curran Associates, Inc., 2017.

[100] G. Ostrovski, M. G. Bellemare, A. Van Den Oord, and R. Munos, “Count-based exploration with neural density models,” in *34th International Conference on Machine Learning, ICML 2017*, vol. 6, pp. 4161–4175, JMLR. org, 2017.

[101] C. M. Vigorito and A. G. Barto, “Intrinsically Motivated Hierarchical Skill Learning in Structured Environments,” *IEEE Transactions on Autonomous Mental Development*, vol. 2, no. 2, pp. 132–143, 2010.

[102] G. Baldassarre and M. Mirolli, “Deciding which skill to learn when: Temporal-difference competence-based intrinsic motivation (TD-CB-IM),” in *Intrinsically Motivated Learning in Natural and Artificial Systems*, pp. 257–278, Springer, 2013.

[103] T. D. Kulkarni, “Deep Reinforcement Learning with Temporal Abstraction and Intrinsic Motivation,” in *Advances in neural information processing systems*, vol. 48, pp. 3675–3683, 2016.

[104] A. Demir, E. Çilden, and F. Polat, “Landmark based reward shaping in reinforcement learning with hidden states,” in *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems, AAMAS ’19, Montreal, QC, Canada, May 13-17, 2019*, pp. 1922–1924, 2019.

[105] M. L. Littman, “Memoryless policies: theoretical limitations and practical results,” in *From Animals to Animats 3: Proceedings of the third International Conference on Simulation of Adaptive Behavior*, pp. 238–245, MIT Press, 1994.

# CURRICULUM VITAE

## Personal Information

Surname, Name : Demir, Alper
Nationality : Turkish
Date and Place of Birth : 22 March 1991, Kocaeli
Phone : +90 505 395 39 37
Email : alper91demir@gmail.com

## Education

| Degree | Institution | Year of Graduation |
|---|---|---|
| M.Sc. | METU Computer Engineering | 2016 |
| B.Sc. | METU Computer Engineering | 2014 |

## Work Experience

| Year | Place | Enrollment |
|---|---|---|
| 2013-2014 | BTT Inc. | Software Developer |
| 2012 September | Turkish Aerospace Industries (TAI) | Intern |

## Foreign Languages

Advanced English, Beginner Russian

## Publications

1. Demir, Alper, Erkin Çilden, and Faruk Polat. "A history tree heuristic to generate better initiation sets for options in reinforcement learning." In Proceedings of the Twenty-second European Conference on Artificial Intelligence, pp. 1644-1645. IOS Press, 2016.
2. Demir, Alper, Erkin Çilden, and Faruk Polat. "Local roots: A tree-based subgoal discovery method to accelerate reinforcement learning." In Joint European Conference on Machine Learning and Knowledge Discovery in Databases, pp. 361-376. Springer, Cham, 2016.
3. Demir, Alper, Erkin Çilden, and Faruk Polat. "A Concept Filtering Approach for Diverse Density to Discover Subgoals in Reinforcement Learning." In 2017 IEEE 29th International Conference on Tools with Artificial Intelligence (ICTAI), pp. 1-5. IEEE, 2017.
4. Demir, Alper, Erkin Çilden, and Faruk Polat. "Generating Effective Initiation Sets For Subgoal-Driven Options." Advances in Complex Systems 22, no. 02 (2019): 1950001.
5. Demir, Alper, Erkin Çilden, and Faruk Polat. "Landmark Based Reward Shaping in Reinforcement Learning with Hidden States." In Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems, pp. 1922-1924. International Foundation for Autonomous Agents and Multiagent Systems, 2019.
6. Demir, Alper, Erkin Çilden, and Faruk Polat. "Automatic landmark discovery for learning agents under partial observability." The Knowledge Engineering Review 34 (2019).

## Interests and Hobbies

Basketball, Photography, Movies