USING FREQUENCIES OF TRANSITIONS TO IMPROVE REINFORCEMENT LEARNING WITH HIDDEN STATES

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

BY

HÜSEYİN AYDIN

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

AUGUST 2022

Approval of the thesis:

USING FREQUENCIES OF TRANSITIONS TO IMPROVE REINFORCEMENT LEARNING WITH HIDDEN STATES

submitted by **HÜSEYİN AYDIN** 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. Göktürk Üçoluk
Computer Engineering, METU ______________

Prof. Dr. Faruk Polat
Computer Engineering, METU ______________

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

Prof. Dr. Cem İyigün
Industrial Engineering, METU ______________

Assoc. Prof. Dr. Mehmet Tan
AI Engineering, TOBB Uni. of Econ. and Tech. ______________

Date: 24.08.2022

ABSTRACT

USING FREQUENCIES OF TRANSITIONS TO IMPROVE REINFORCEMENT LEARNING WITH HIDDEN STATES

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

August 2022, 120 pages

Reinforcement learning problems with hidden states suffer from the ambiguity of the environment, since the ambiguity in the agent's perception may prevent the agent from estimating its current state correctly. Therefore, constructing a solution without using an external memory may be extremely difficult or even impossible sometimes.

In an ambiguous environment, frequencies of the transitions can provide more reliable information and hence it may lead us to construct more efficient and effective memory instead of keeping all experiences of the agent like the existing memory-based methods. Inspired by this observation, a selective memory approach based on the frequencies of transitions is proposed in the first part of thesis. The agents with any reinforcement learning method can be equipped with this selective memory, since the memory itself does not have any constraints on the underlying method. Experiments show that a compact and selective memory may improve and speed up the learning on both Q-Learning and Sarsa(λ) methods.

As the second part of the work, sequential association between transitions is used in order to get a solution in more abstract manner for the problems which can be decomposed by using the bottlenecks in the environment. A simple recursive method is proposed for automatic extraction the set of *chains of bottleneck transitions* which are sequences of unambiguous and critical transitions leading to the goal state. At the higher level, an agent trains its sub-agents to extract sub-policies corresponding to the sub-tasks, namely two successive transitions in any chain, and learns the value of each sub-policy at the abstract level. Experimentation shows that this approach learns better and faster in the ambiguous domains where conventional methods fail to construct a solution. Furthermore, it has been shown that our method with its early decomposition approach performs better than a memory-based method in terms of quality of the learning, speed and memory usage. Finally, Diverse Density method is integrated with the proposed method to complete the autonomy of the overall process. Although, identified landmarks are not completely accurate, experimentation shows that the results are promising.

Keywords: Reinforcement Learning, Compact Frequency Memory, Task Decomposition, Chain of Bottleneck Transitions

ÖZ

SAKLI DURUMLU PEKİŞTİRMELİ ÖĞRENMEYİ GELİŞTİRMEK İÇİN GEÇİŞLERİN FREKANSLARININ KULLANIMI

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

Ağustos 2022 , 120 sayfa

Saklı durumlara sahip pekiştirmeli öğrenme problemleri ortamdaki belirsizlikten önemli derecede olumsuz etkilenmektedir. Bunun nedeni, etmenin algılayışındaki belirsizliğin, içinde bulunduğu durumu doğru bir şekilde tespit etmesinin önüne geçmesidir. Bu nedenle, bu problem kümesi için harici bir hafıza kullanmadan bir çözüm üretmek çok zor ya da bazen imkansızdır.

Belirsizliğin yoğun olduğu bir ortamda, geçişlerin frekansları etmenin içinde bulunduğu durumu tespit etmesi açısından daha güvenilir bilgiler sunabilir. Dolayısıyla bu yaklaşım bizi, etmenin tüm deneyimlerini saklamaktan daha verimli ve etkili bir hafıza kullanımı ile beraber daha iyi bir durum tespitine yönlendirebilir. Bu gözlemden yola çıkarak bu tez kapsamında, geçişlerin frekanslarını kullanan seçici bir hafıza yaklaşımı önerilmiştir. Bu hafıza alttaki öğrenme yöntemine yönelik bir kısıtlama barındırmadığından herhangi bir pekiştirmeli öğrenme yöntemini uygulayan etmen bu hafızayı kullanabilecektir. Deneyler kompakt ve seçici bir hafızanın öğrenmeyi geliştirip hızlandırabileceğini Q-Öğrenme ve Sarsa(λ) yöntemleri için göstermiştir.

Çalışmanın ikinci kısmı olarak, etmenin problemi daha soyut bir şekilde çözebilmesi için, darboğaz geçişleri arasındaki sırasal ilinti kullanılmıştır. Etmeni çözüme yönlendirecek olan, belirsiz olmayan ve kritik geçişlerin sıralamalarının, yani *darboğaz geçiş zincirlerinin kümesinin* otomatik tespitini sağlayacak basit yinelemeli bir çözüm önerilmiştir. Üst ve daha soyut bir seviyede, etmen alt-etmenlerini bu zincirdeki herhangi iki geçiş arasında eğiterek, ana hedefe ulaşmak için izlenecek olası alt-politikaları ve bunların değerlerini öğrenebilir. Deney çalışmaları, bu yaklaşımın belirsizliğin yoğun olduğu ve geleneksel yöntemlerin çözüm üretmekte başarısız olduğu ortamlarda daha iyi ve hızlı bir öğrenme gerçekleştirdiğini göstermiştir. Bunun yanı sıra, önerilen yöntemin öğrenme kalitesi, hız ve hafıza kulanımı yönünden, hafıza temelli bir yöntemden daha iyi çalıştığı gözlenmiştir. Son olarak yöntemin kullanıcıdan bağımsız, otomatik bir şekilde problem üzerinde çalışması için Farklı Yoğunluk yöntemiyle entegrasyonu sağlanmıştır. Farklı Yoğunluk yönteminin bulduğu yer işareti durumlar tümüyle doğru olmasa da, deneyler sonuçların potansiyel taşıdığını göstermektedir.

Anahtar Kelimeler: Pekiştirmeli Öğrenme, Kompakt Frekans Belleği, Problem Ayrıştırma, Darboğaz Geçişler Zinciri

To what's coming next, which's already gone...

ACKNOWLEDGMENTS

I firstly want to thank my supervisor, Faruk Polat who made such a study possible by his wise and patient guidance. As a person who dreams about being an academician, I could not find a better role model than him. I have always admired his rational and ethical point of view. During the last seven years, including my MSc studies, it has been a true honor and a pleasure to get the chance to do research with such an exemplary person who has also been like a father to me.

I would also like to thank my co-supervisor, Erkin Çilden for his counseling and the support that he has given through the years whenever I feel stuck. His previous work helped this study to emerge and led us to build a new framework for our methods. As he patiently discussed every detail about our study and he taught me the difference between knowing and expressing something, I cannot appreciate his efforts on this thesis enough.

I want to continue by thanking my former colleague and forever friend, Alper Demir. As we built our new codebase together, his contributions to this study are essential. The enthusiastic discussions we have made about our studies have been a great motivation to me. I really miss the good old days when we got more chance to spend more time together. I am still looking forward to the paper that we will publish together. I also hope that our paths cross again to work closer in the future.

I am proud to be the son of two amazing parents Ali and Melek Aydın. I have always felt my mother's unconditional love and my father's support for every decision I have made. I also would like to thank my sister, Arzu Hazal Aydın who always finds a way to cheer me up. They are the reasons behind everything I do.

I would like to thank Göktürk Üçoluk and Mehmet Tan for patiently monitoring the study for the last five semesters. It has been a great pleasure to assist Göktürk Üçoluk for the several courses in our department as well. I also want to thank Altay Güvenir and Cem İyigün for their valuable comments and revisions on the study.

I would like to thank Melis İlayda Bal for her time as she did the proof-reading of this thesis and checked the clarity of the text. I also thank Saim Sünel and Burak Han Demirbilek, as the fresh members of our research group. It is nice to know there are people to pass the torch.

I want to thank Berat Can Cengiz, for our thought provoking conversations on philosophy and psychology. I will always miss our coffee breaks which turn out to be an open school broadening my horizon.

Finally, I would like to thank all members of our department, especially Fatoş Yarman Vural, Sinan Kalkan, Tolga Can, İsmail Hakkı Toroslu, Onur Tolga Şehitoğlu, Emre Akbaş, Adnan Yazıcı, and Halit Oğuztüzün. It has been a great experience to work with them on several courses, projects and other departmental tasks during my assistantship.

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

AI	Artificial Intelligence
CFM	Compact Frequency Memory
CBT	Chains of Bottleneck Transitions
CR	Congestion Ratio
DD	Diverse Density
DDCF	Diverse Density with Concept Filtering
DP	Dynamic Programming
L-POMDP	Landmark-Partially Observable Markovian Decision Process
LSTM	Long Short-Term Memory
MC	Monte Carlo
MDP	Markovian Decision Process
MIL	Multiple Instance Learning
NSM	Nearest Sequence Memory
POMDP	Partially Observable Markovian Decision Process
RL	Reinforcement Learning
TD	Temporal Difference
UDM	Utile Distinction Memory
USM	Utile Suffix Memory

CHAPTER 1

INTRODUCTION

Over the last decades, Artificial Intelligence (AI) components have been broadly involved in our day to day lives. This being the case, it is natural for the people to want the behaviors of these components to be reasonable and explainable. With its psychological background that is useful in the examination of all living creatures' behavior, Reinforcement Learning (RL) [2] is one of the best alternatives among AI methods that can satisfy people's this need.

Similar to newborns opening their eyes and trying to make sense of their perceptions of the world with their own experiences, an RL agent is encouraged to explore its environment with its own actions evaluated with direct reward signals from the environment itself. However, this is not as easy as it seems when the state space of the given RL problem gets larger, the complexity of finding a solution becomes a challenge for both the agent and the problem designer.

Several studies have been conducted to overcome this difficulty by applying one of the generalization techniques to a problem [3, 4] or breaking the problem into pieces that are easier to solve [5, 6, 7]. Another approach involves the *options framework* [8] supported by a number of subgoal identification methods [9, 10, 11] for the construction of better options. Yet there is another challenge for the learning agent, which is the limited perception of the environment. In most realistic problems, the agent cannot have the complete information of its current state. The ambiguity in the environment (also called *perceptual aliasing* [12, 13, 14]) caused by incomplete information can make the problem very difficult, or even impossible to solve.

There are a number of memory-based method categories trying to cope with the perceptual aliasing problem. *Finite size history* is one of the simplest memory-based approaches [15]. Instead of keeping the current observation, the agent uses a chain of last *n* steps as the current observation. Despite its simplicity of application, it requires prior information about the environment such as the minimum size of the memory needed to construct a solution.

Recurrent-Q is another method that uses *Long Short-Term Memory (LSTM)* [16]. LSTM is a Recurrent Neural Network (RNN) variant used to filter out the irrelevant information about the state space. Yet another memory-based method category is the *variable-length history* family. In this approach, the size of the internal memory can be altered depending on dynamically changing memory requirement for the underlying learning procedure. Nearest Sequence Memory (NSM) and Utile Suffix Memory (USM) algorithms are examples of this category (also called *instance-based* methods) where all raw data of the agent's experience are kept in the form of action-reward-observation tuples [1].

All of the methods above tend to keep the recent or all the experiences of the agent collected throughout the learning process. Unfortunately, especially for problems with large state space, it becomes difficult to determine the size and content of the memory required to provide a distinctive clue about the current state of the agent. Although keeping all the history of the agent overcomes this problem, the estimated state, namely the information unit constructed by the agent to distinguish its current state, can be arbitrarily complex [17].

In this study, a selective memory approach called *Compact Frequency Memory (CFM)*, is build based on the transition frequencies and hence, a more compact and reliable memory becomes available for the learning agent. Therefore, scalability of CFM is much higher than the existing memory-based methods.

As the second part of this thesis, we take a step further in the examination of the transition frequencies by the interpretation of their sequential associations in the completion of RL tasks. We observe that a *Chain of Bottleneck Transitions (CBT)*, as an ordered sequence of critical transitions based on the landmark states [18, 19], can be useful in the decomposition of the main problem.

While there are some divide-and-conquer approaches for RL under full observability, similar work is rare for RL with hidden states. Most research focus on using an external memory or a trace mechanism that imitates a memory for the agent as mentioned above. However, there are few studies making use of decomposition for the problems with hidden states [20, 21]. This is not surprising, since, without any refinement and adaptation for perceptual aliasing, conventional methods cannot guarantee convergence to an optimal solution [22].

CBT method deals with the perceptual aliasing by the sequential partitioning of the problem. Since each sub-problem is separately handled, the number of aliased states is much less compared to the main problem. The main agent of the proposed method can speed up this process by the simultaneous training of its sub-agents for each independent sub-part of the problem. Therefore, this sequential partitioning provides a natural guidance for the learning agent to solve the task in a more abstract manner. In other words, the agent may have different strategies to achieve sub-tasks indicated by given bottleneck transitions and these strategies can be used as intermediate steps for the completion of the whole task.

Although the idea of using the sequential partitioning of the problem has been studied in different fields like *planning* [23, 24], to our best knowledge, this is the first attempt to apply this approach for RL with hidden states. The definition of a landmark state is less strict compared to the notion of landmark in planning domain, in the sense that a landmark here does not have to be included in every optimal solution. The agent may have alternative ways for the solution visiting both a certain set of landmarks and/or regular states and bypassing another set of landmarks. This particular property of learning problems having both hidden and landmark states requires a different technique.

The following subsection provides the contribution and novelties of the technique we proposed to deal with sequential partitioning in the context of reinforcement learning. The same aspects are also covered for our memory-based method that is proposed in the first part of the study.

1.1 Contributions and Novelties

Focusing on partially observable RL problems, two main contributions for are proposed as follows:

- An efficient memory mechanism, namely Compact Frequency Memory (CFM) that uses only significant transitions of the learning agent is constructed [25]. In this way, a compact state estimation becomes available for the agent to deal with the ambiguity in the environment faster. To our best knowledge, this is the first selective memory that is designed for a tabular RL solution.
- A framework that constructs Chains of Bottleneck Transitions (CBT) based on landmark states (states having unique observations) is incrementally built for the RL agent [26]. By the means of these chains, a decomposition of the given RL problem with hidden states is obtained and the learning performance is improved through independent sub-agents dealing with each sub-problem where the ambiguity is lessened compared to overall problem. As far as we know, explicit inclusion of landmark states to the learning phase was achieved first within the scope of this study.

1.2 The Outline of the Thesis

The outline for the rest of this thesis is as below:

- Chapter 2 provides the summary of problem along with its formulation, related works and sample domains used in this study.
- Chapter 3 presents CFM, a memory mechanism which filters out the insignificant transitions of the agent, and hence, produces a compact state estimation.
- In Chapter 4, the validation of CBT about successfully decomposing and solving the given RL problems with hidden states is demonstrated.
- In Chapter 5, the method that achieves the self-construction of CBT is proposed.

- Chapter 6 completes the CBT framework by presenting the automatization of landmark identification and integration of all components.
- Finally, in Chapter 7, the study is concluded and possible research directions for the future of proposed methods are provided.

CHAPTER 2

BACKGROUND AND RELATED WORK

This chapter summarizes the necessary background to provide better understanding of the RL problems with hidden states that are focused throughout this study. The related methods, some of which are used as underlying learning methods in the approaches we work on this thesis, are examined here as well. Finally, along with its theoretical basis, samples of partially observable RL problems that are used in the experimentation of proposed methods are covered in this chapter.

2.1 Reinforcement Learning

Reinforcement Learning (RL) refers to a family of algorithms where the agent is expected to solve a problem by the direct reward as a feedback from the environment reflecting the evaluation of its action taken in the state where the agent reside. Figure 2.1 depicts the essence of this framework. With this evaluative nature of the problem, an RL agent learns from its own experiences how good an action is for a specific situation, instead of learning what is the “correct” action to take. Therefore, there is no complete supervision in RL as in a classification problem. However, since the environment provides a feedback, one cannot claim that RL is a unsupervised method either. Hence, RL must be considered as a third paradigm in the field of machine learning within its distinct characteristics which can be summarized as evaluative yet exploratory.

This brings us to the challenging aspect of an RL problem where the agent has a trade-off between exploitation and exploration. To maximize its cumulative reward, the agent needs to exploit the experience it already has up to that point. However, to

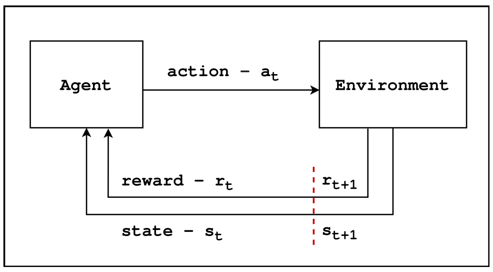
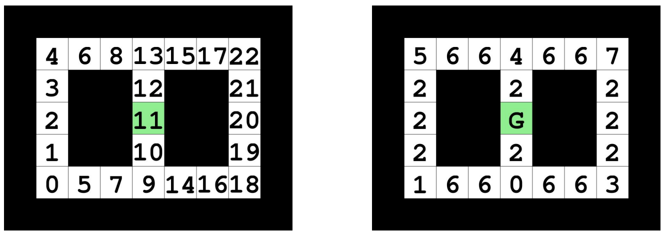


Figure 2.1: The interaction diagram between agent and environment as the essence of the reinforcement learning

exceed possible local maxima and to construct better solution, the agent also needs to explore the environment. Although this dilemma has been studied by many researchers, there is no silver bullet to solve it and the best thing to do for the agent is to find a balance between these strategies considering the nature of the given problem.

Another challenging property of an RL problem is that the agent may not have the complete information about its current state. Consider the sample problem from McCallum's study [1] given in Figure 2.2 where the goal of the agent is to reach the center of the corridor in the middle. Assuming that the agent is not able to know its current location, and its ability to sense the environment is limited to the cell it resides, the agent becomes unable to distinguish the critical states that are just above or just below the goal state. This phenomenon is called *perceptual aliasing* [12, 13, 14] and as can be seen in the example, an RL problem with aliased states might be severely challenging for the agent to construct a stationary and memoryless solution.



(a) states (b) observation

Figure 2.2: McCallum's Hallway Navigation (`mhn`) [1] domain with its state and observation enumerations. The learning suffers from perceptual aliasing especially for the states around the goal state.

Before the examination of the methods which try to overcome these challenges of the RL problems, the next two sections summarize the formulations that are used to properly define a given RL problem.

2.2 Markov Decision Process

An RL problem requires the agent to produce a sequence of decisions for the completion of a task. Within such a sequence, the agent traverses through state space of the problem, and it is convenient to assume that any state in the problem contains the complete information to construct the probability distribution for the possible next states. In this way, problem does not require the agent to memorize its past experience for the decision about what the next action should be. This characteristic is called *Markov Property* and it can be expressed more formally as below:

$$P(s_{t+1}|s_t) = P(s_{t+1}|s_0, s_1, ..., s_{t-1}, s_t). \tag{2.1}$$

Most methods in the field of RL are based on the assumption the given problem, is Markovian, i.e. has the Markov Property. Markov Decision Process (MDP) formulates such a problem, which is a model defined with the tuple $\langle S, A, T, R\rangle$ where,

- S denotes the finite set of states,
- A denotes the finite set of possible actions,
- $T : S \times A \times S \to [0, 1]$ is the transition function that defines the probability of making a transition from the given state to another by taking some action, and
- $R : S \times A \to \mathbb{R}$ is the reward function that gives the immediate reward provided to the agent doing some action in the given state.

A policy for an MDP is defined as, $\pi : S \times A \to [0, 1]$, which defines the probability of taking an action in A from some state in S. An optimal policy is the one that maximizes the expected discounted return (G), namely, the sum of immediate reward and all future rewards in which a discount factor, $\gamma \in [0,1]$, is used to decide the weight of every forthcoming step's reward in the cumulation:

$$G_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + ... + \gamma^{\tau-1} r_\tau = \sum_{k=0}^{\tau} \gamma^k r_{t+k+1}, \tag{2.2}$$

where the upper limit (τ) can be a finite value if the task has a terminal state. Otherwise, the task becomes non-episodic and hence, the limit goes to infinity. By the help of discount factor (γ), the influence of the future reward on agent's behavior can be realistically simulated. In its extreme values, 0 and 1, the agent becomes *myopic* and *farsighted* respectively. In other words if γ is 0, the agent does not care about the future rewards at all, and if γ is 1, the agent takes all of the future rewards into account in the most strongly way that is possible. For most of the tasks, it is preferable to set a value that is closer but not equal to 1 for γ.

MDP provides a well-defined formulation to model RL problems. However, the partially observable case, where the agent fails to grasp its complete state in the problem, is not expressible by using MDP. That is why we need a more comprehensive formulation as Partially Observable MDP which is covered in the next section.

2.3 Partially Observable Markov Decision Process

When the agent has limited sensor capability and fails to capture all of the necessary information about its current state, MDP becomes insufficient to formulate the problem. To deal with such problems, a generalization of MDP, Partially Observable MDP (POMDP) defined by the tuple $\langle S, A, T, R, \Omega, O \rangle$ is used in the formulation of problems with hidden states. POMDP model covers partial observability by extending MDP model with the addition of the finite set of observations as Ω, and the observation function, $O : S \times A \rightarrow \Pi(\Omega)$, which gives a probability distribution over possible observations for each action and resulting state [27].

There are two different interpretations of POMDP formulation. The first one assumes that the agent has complete knowledge of the underlying state space but it does not know the current state. Based on this scenario, Kaelbling et. al. proposed the *belief* *state*, which is the probability distribution of the given observation belonging to some set of states by considering the previous experience of the agent [27]. In the second approach, the only information the agent gets is the current observation which it is able to perceive [1]. The problems attacked in this study fall into the second category. Therefore, the learning agents of proposed methods are assumed to be totally oblivious of the underlying MDP model (by following the *model-free* RL approach).

It is clear that MDP is a special form of POMDP where every state is associated with a unique observation. On the other hand, the ambiguity in the problem can be extremely challenging and even impossible to solve without an external memory for any given RL method. One extreme case would be that the same observation is given to the agent for all states (excluding the goal state(s), if we want to keep the problem somehow "solvable"). The main point here is that there is no limitation regarding the observation function. However, as a reasonable example, if the problem is navigational and discrete, it is convenient to assume that the agent has the perception of the surroundings of its current cell at worst or the agent is somehow able to sense the distance to the obstacles around the environment up to some degree. Since navigational problems are favored in this study for easy visualization, the next two subsections explain these semantics in details. This will also provide us the chance to see that the observation function alter the complexity of same problem significantly.

2.3.1 Observations Based on Basic Surroundings

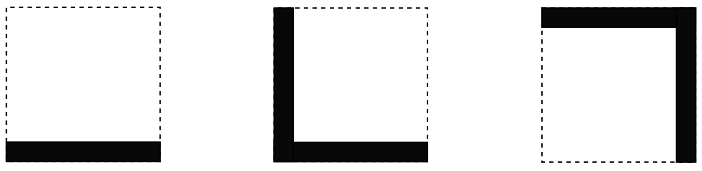
For a navigational RL problem with hidden states, one of the simplest observation semantics is based on the existence of an obstacle around the agent's current cell. There are two possible cases of the agent for each main direction, as being able to move towards a direction or being stuck in the previous location by hitting the obstacle. Hence, there can be 16 distinct observations for a regular state in this semantic.

Table 2.1 shows the 4-bit encoding of all observations, if the agent can only sense the surrounding of its current cell. The encoding is done in clockwise order starting from the `North`. The same table includes the labels of possible observations in the sample domain depicted in Figure 2.2. Obviously, labeling as shown in this table is not obligatory and these labels can be replaced in any way.

Table 2.1: The 4-bit encoding of observations of regular states with basic semantic and their labels for McCallum's Hallway Navigation (`mhn`) [1] problem. The observations labeled with a '-' character do not exist in this domain.

4-bit encoding (`N-E-S-W`)	**label in `mhn`**
0000	-
0001	-
0010	0
0011	1
0100	-
0101	2
0110	3
0111	-
1000	4
1001	5
1010	6
1011	-
1100	7
1101	-
1110	-
1111	-

Figure 2.3 shows example observations from `mhn` domain. If the agent can move from its current cell in any direction other than `South`, then it gets the observation 0 (o_0). When the agent gets the o_1, then it means that there are obstacles in both $West$ and $South$ directions. Finally, the o_7 is like the opposite of o_1 where the agent cannot move towards $North$ and $East$ in the context of this example.



(a) o_0 (0010) (b) o_1 (0011) (c) o_7 (1100)

Figure 2.3: Sample observations from `mhn` domain

2.3.2 Observations Based on Classified Distances

When the agent is assumed to be more perceptive about its whereabouts, the classified distances can be used to construct the observation function in POMDP model of the given navigational RL problem. These observations are still not sufficient for the agent to know its exact location and hence, to distinguish all of the states in the problem, yet they are based on surrounding obstacles exceeding the agent's current cell. Since the environment of a navigational problem is assumed to be a bounded region, it is guaranteed that there exist at least one obstacle for every main direction.

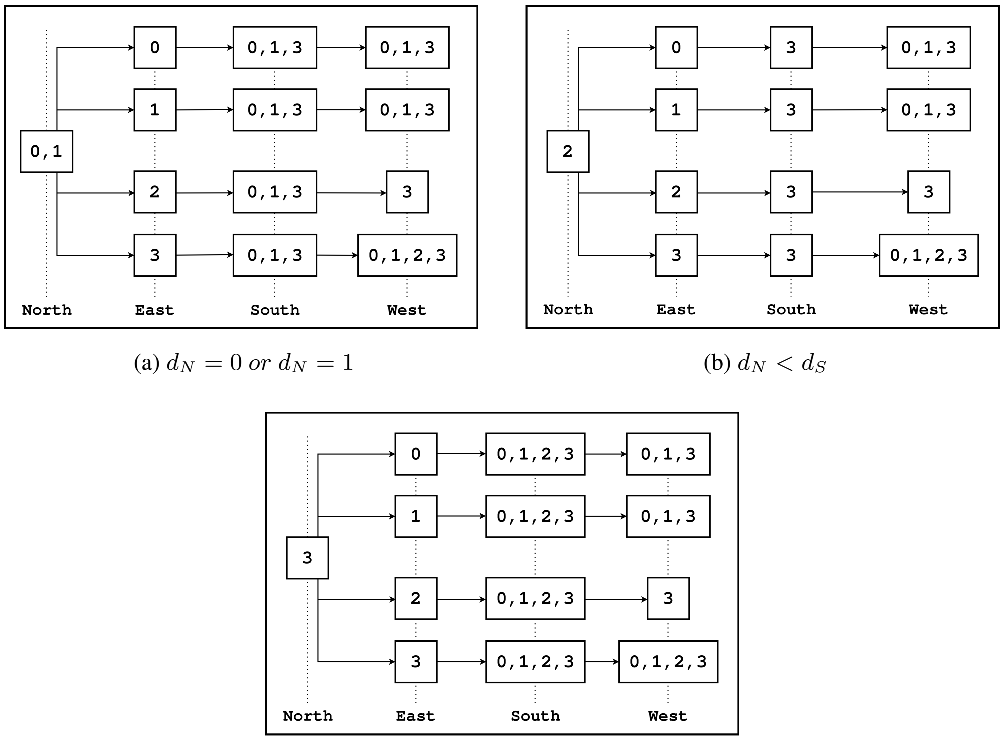
In this semantic, the agent is able to sense the adjacent obstacle and the obstacle that is 1-step away for any of the cardinal directions. Furthermore, it is also capable of comparing the distances to the nearest obstacles in opposite directions. Considering the case of being equally distant case together with being far away from an obstacle, the agent may get 4 distinct perceptions for each main direction.

Let be the agent in the cell next to obstacle, or the cell that is one step away from the obstacle in `North`, then it cannot be any closer to the obstacle in the opposite direction, namely `South`. The same interpretation can be validated for the horizontal orientation as well. Hence, the number of possible combinations for the agent's overall perception of a regular state is much less than 256 (4^4), in fact, is 121.

Before the calculation of this value, let the perception in one direction is encoded as:

- 0 $\rightarrow$ next to the obstacle,
- 1 $\rightarrow$ 1-step away from the obstacle,
- 2 $\rightarrow$ the obstacle in this direction is closer than the one in the opposite direction,
- 3 $\rightarrow$ the obstacles are equidistant or the obstacle in the opposite direction is closer.

With this enumeration at hand, all possible combinations of the perceptions can be seen in the Figure 2.4 as categorized according to the perception on `North` where d_N and d_S represents the distance values to the obstacles in `North` and in `South`



(a) $d_N = 0$ *or* $d_N = 1$

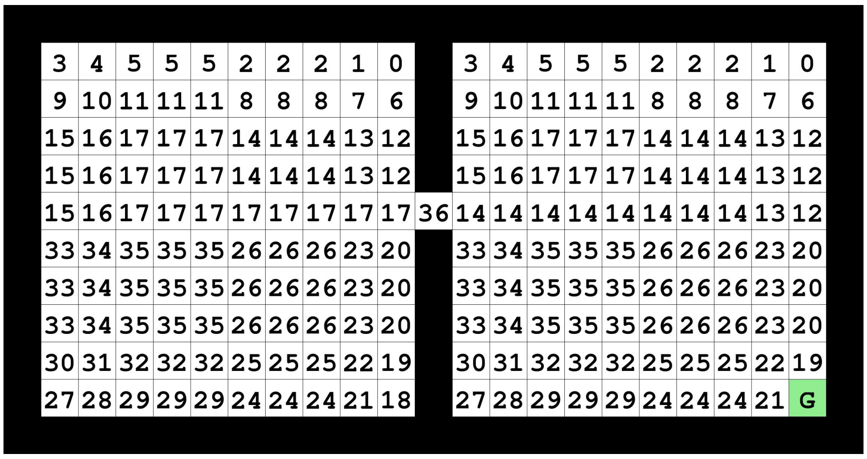
(b) $d_N < d_S$

(c) $d_N \geq d_S$

Figure 2.4: Possible combinations for the observation encoding in the observation semantic with the classified distances.

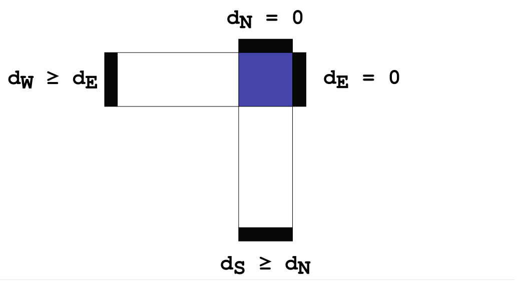
respectively, in terms of the number of available cells in these directions. Possible encoding values of perception in one direction is merged into a single node to avoid repetition of similar entities in the combination tree. Therefore, we get 66 valid combinations when $d_N = 0$ or $d_N = 1$. Similarly, we get 11 and 44 possible combinations for the cases that $d_N < d_S$ and $d_N \geq d_S$ respectively and get the total number of perceivable observations as 121.

Now that the enumeration in this observation semantic is known, let us look at the types of observations with classified distances in a sample domain, namely `2 rooms` problem [11] shown in Figure 2.5. In this problem, the agent is placed in a cell in the left room and it is expected to find its path to the south-western corner of the right room, denoted by letter 'G'. Including the goal observation, there are 37 different observations in this domain when this semantic is chosen to construct the observation function.

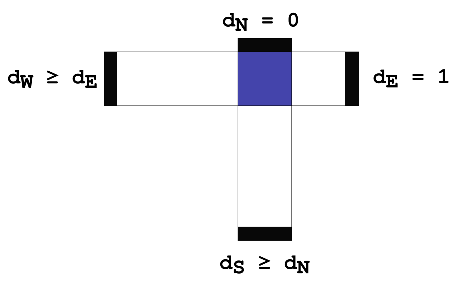


3	4	5	5	5	2	2	2	1	0		3	4	5	5	5	2	2	2	1	0
9	10	11	11	11	8	8	8	7	6		9	10	11	11	11	8	8	8	7	6
15	16	17	17	17	14	14	14	13	12		15	16	17	17	17	14	14	14	13	12
15	16	17	17	17	14	14	14	13	12		15	16	17	17	17	14	14	14	13	12
15	16	17	17	17	17	17	17	17	17	36	14	14	14	14	14	14	14	14	13	12
33	34	35	35	35	26	26	26	23	20		33	34	35	35	35	26	26	26	23	20
33	34	35	35	35	26	26	26	23	20		33	34	35	35	35	26	26	26	23	20
33	34	35	35	35	26	26	26	23	20		33	34	35	35	35	26	26	26	23	20
30	31	32	32	32	25	25	25	22	19		30	31	32	32	32	25	25	25	22	19
27	28	29	29	29	24	24	24	21	18		27	28	29	29	29	24	24	24	21	G

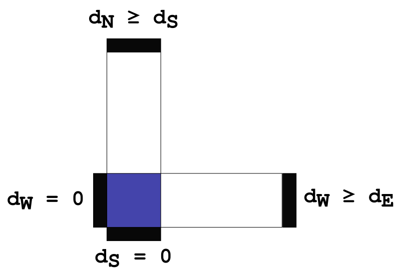
Figure 2.5: The observation enumeration of `2 rooms` domain with classified distances semantic



(a) o_0 (0033)



(b) o_1 (0133)



(c) o_{28} (3300)

Figure 2.6: Sample observations from `2 rooms` domain with the classified distances semantic.

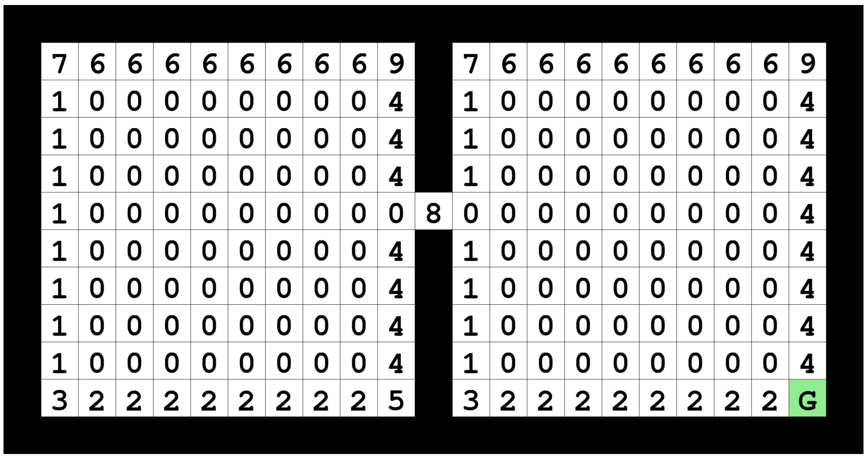


Figure 2.7: The observation enumeration of `2 rooms` domain with basic semantic

Figure 2.6 depicts a closer look for 3 types of observations that can be seen in `2 rooms` domain. The first one in 2.6a, namely o_0 is corresponding to the north-eastern corner of the rooms. The second one in 2.6b is the o_1 is the cell where the agent gets slightly different observation since the obstacle in `East` is one step away now. Finally, Figure 2.6c shows the observation for south-western corner of the rooms.

To be able to compare this observation semantic with the one explained in the previous section, let us see the difference from the point of agent's view. Figure 2.7 shows the observation enumeration of 2 rooms is given with the basic semantic. Having the extended observations in Figure 2.5, the agent is able to get more distinct observations for the problem whereas there are only 11 distinct observations available for the same problem including the goal observation. Although aliasing does not always harm the learning process, since it may lead a useful abstraction for its policy, this is not the case in `2 rooms` domain with the basic semantic. As it can be seen from the figure, o_0 is perceived by the agent for 128 states and the optimal actions for these states can be in opposite directions. The agent may still overcome this problem if the `East` action converges to its correct value, yet, that is not possible for o_4 where the agent cannot avoid the conflict of optimal actions. It is most likely that the agent will get stuck in the south-eastern corner of the left room where the agent repeats `North` and `South` over and over again if there is a stationary and completely greedy policy. However, as careful readers may notice, the same issue arises in the same domain with the classified distances semantic as well. Hence, once again it can be inferred that despite of being more improved version, the classified distances semantic does not guarantee for the agent to reach an optimal policy that is stationary and reactive like in the fully observable case.

2.3.3 Observation Semantics with Landmark States and Landmark-POMDPs

There are some places in the real world having a unique feature that makes them the only instance of their kind, for instance, the seven wonders of the ancient world. Similarly, in a partially observable RL setting there may exist a type of state, that can be uniquely perceived, and hence, fully observable to the agent. In other words, for such a case, the ambiguity still takes it place in the environment due to aliased states, yet, there might be some set of states, that are completely distinguishable by the agent. Such states are called landmark states and formally defined as:

Definition 2.1 *A state $s \in S$, is a landmark state if $\exists o \in O$, such that $P(o \mid s) = 1$ and $\forall s' \neq s$, $P(o \mid s') = 0$ where $P(o \mid s)$ is the probability of getting $observation\ o$ for given current $state\ s$.*

For example, the state corresponding to the location of doorway between the rooms in `2 rooms` domain, can be considered as a landmark state, with both observation semantics explained in preceding subsections (see Figure 2.5 and Figure 2.7.). If a navigational domain has more than one door, then it is generally assumed that each door has a unique feature (like a number in a door of a building) that makes it landmark. This assumption allows us to model the problem with Landmark-POMDP setting and benefit from the concept of distinguishable state for the learning agent. As the name implies, *Landmark-POMDP (L-POMDP)* model is a special type of POMDP formulation, where it is assumed that there is at least one landmark state in the problem [18, 19]. Although it is easy to imagine the concept of landmarks in navigational domains, the concept itself, cannot be restricted to this type of problems and landmark states can be naturally seen in any type RL problem with hidden states. An example to confirm this deduction is provided in Section 2.5 with the domain of well-known `Tower of Hanoi` problem [28].

2.4 Reinforcement Learning Methods

Now that the basic properties and widely-used formulations of RL problems have been covered, existing methods that attempt to solve these problems can be examined. RL methods follow one of the these two main approaches: *tabular* or *function approximation*. Tabular methods uses a lookup table to represent and update the value of a state or a state-action pair from the experiences of the agent whereas the function approximation methods uses an approximator to estimate the value of given state or state-action pair. Although function approximation provides more comprehensive solutions for especially continuous case of RL problems, tabular methods produce more explainable solutions for the given problem. Tabular methods can be *policy-based* or *value based*. In policy-based approach, the policies are iterated and evaluated consecutively until the policy converges to optimal one. In value-based approach, values are iterated until the optimal value function is obtained. In this way, the optimal policy can be directly extracted from the values. The methods we proposed in this study fall into the second category of tabular methods.

The value-based tabular methods can be further classified according to two dimensions of the value updates: the depth and width of the updates on values. The depth of the updates corresponds to when the updates are done, for example within each step of the agent, or when the the agent reached its goal state (assuming that the agent always achieves the task soon or later). The second dimension, namely the width of the update, represents the number of states whose values will be recalculated by the update within a step. Extreme points of these dimensions are depicted in Figure 2.8.

All extreme points other than *Temporal Difference (TD)* learning require that either all the way to the goal should be given for each update and/or all states must be included in the recalculation of the values. For example, as *Dynamic Programming (DP)* methods, *Policy Iteration* or *Value Iteration* methods do their calculations over all state space by **bootstrapping** one-step estimated values of them. *Monte Carlo (MC)*, on the other hand, does its calculation by **sampling** all the states down to the goal one and the updates are done with the actual value of discounted return for one state at a time.

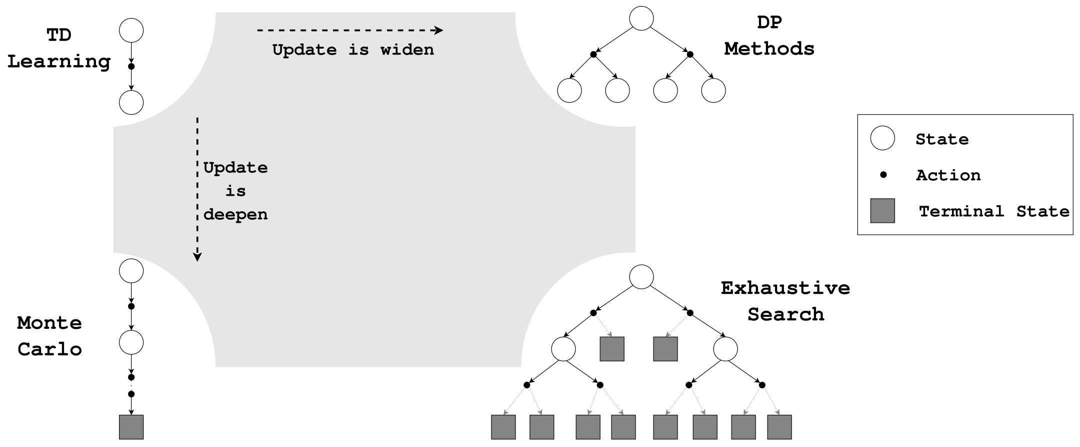


Figure 2.8: Tabular RL methods projected to the solution space where the dimensions are the depth and width of the updates on values (adapted from [2]).

TD, uses the bootstrapping one-step estimated value as DP methods, but it does the sampling of one state among all available states as in the case of MC methods. In this way, with TD learning, the agent does not have to wait until it reaches the goal to update the values. Furthermore, the does not need to traverse through the whole state space in one step of the update. Hence, it is like an efficient compromise among these extreme points.

2.4.1 Q-Learning

Q-Learning [29] is a widely used RL algorithm that facilitates the learning process using *Temporal Difference* over state-action pairs instead of using only states. Given the experience $\langle s, a, r, s' \rangle$ at time t, the update rule for the Q-value of a state-action pair at time $t+1$, $Q_{t+1}(s, a)$, is defined as:

$$Q_{t+1}(s,a) \leftarrow Q_t(s,a) + \alpha[r + \gamma max_{a' \in A} Q_t(s', a') - Q_t(s,a)], \tag{2.3}$$

where $0 \leq \alpha \leq 1$ is the learning rate and $0 < \gamma < 1$ is the discount factor. Most RL approaches inherit the Q-Learning's action-value update idea. Q-Learning is an *off-policy* method as it separates the policy that is being evaluated and the policy that produce the experience of the agent unlike the *on-policy* approach where the exact policy that determines the behavior of the agent is evaluated.

Q-Learning can easily be adapted to the problems with POMDP formulation over observations, by simply replacing s with o in the update rule (2.3). However, since the states are hidden, the Markov property does not hold due to the notion called *perceptual aliasing*, which is the situation that occurs when the same observation can be obtained by the agent in at least two distinct states in the environment as discussed in Section 2.1. Unfortunately, convergence is not guaranteed in this circumstance [22]. There are RL algorithms to cope with this problem and are successful to a certain extent [2]. One powerful representative of these algorithms, Sarsa(λ), will be summarized in the next subsection.

2.4.2 Sarsa(λ)

Sarsa(λ) is an *on-policy* RL algorithm that applies Q update on *all* state-action pairs proportional to their eligibility for the given update [17]. The eligibility of a state-action pair is controlled by the decaying traces that the agent leaves behind for previous transitions. When a pair is visited, its trace is set to 1 and the trace decays over time exponentially with the parameter $\lambda \in [0, 1]$. The value of the state-action pair is updated in proportion to its trace. Therefore, most recent pairs become more eligible to update.

Like in Q-Learning, Sarsa(λ) can be extended to problems with hidden state by keeping traces for observation-action pairs. The update rule of Sarsa(λ) for experience $\langle x_t, a_t, r_t, x_{t+1} \rangle$ is formulated as follows under partial observability where x_t represents the current perception of the agent:

$$\begin{gathered} n_t(x_t, a_t) = 1, \\ \forall (x \neq x_t \; or \; a \neq a_t); \; n_t(x, a) = \gamma \lambda n_{t-1}(x, a), \\ \forall \; x \; and \; a; \; Q_{t+1}(x, a) = Q_t(x, a) + \alpha \times \delta_t \times n_t(x, a), \end{gathered} \tag{2.4}$$

where γ and α are discount factor and learning rate respectively and,

$$\delta_t = r_t + \gamma Q_t(x_{t+1}, a_{t+1}) - Q_t(x_t, a_t). \tag{2.5}$$

As a Sarsa based approach, Sarsa(λ) update the values at time t after sampling the action at time $t + 1$, hence, it differs from TD learning and becomes closer to MC. Also, by the means of updating the set of states based on their eligibility traces, it resembles a DP method. Therefore, Sarsa(λ) falls into a point among these three approaches in the space of tabular solutions (see Figure 2.8). Since only the states with non-zero λ values need to be updated, Sarsa(λ) can still be an efficient tabular method. Furthermore, learning with Sarsa(λ) can be faster, as the estimated values are propagated to all eligible states sooner.

Although Sarsa(λ) imitates a short-term memory for the agent, it still suffers from perceptual aliasing, especially when there is no reactive policy solution for the problem. That is, the agent cannot choose an optimal action for at least one observation in the problem without the usage of an external memory.

2.4.3 Memory-based RL Methods

One other aspect to categorize RL methods is the usage of an external memory for the past experience of the agent and how that memory is built. Since this aspect and some of the memory-based methods are more related to the topics of this thesis, they will be examined in details in the following subsections.

2.4.3.1 Finite Size History

Finite Size History [15] is the basic approach for building a memory for the learning agent. In this method, the past N observations are kept and used to construct a state estimation in the structure of a sliding window as the current perception of the agent. Although this model allows the agent to deal with perceptual aliasing up to a certain degree that is subject to N, the decision of what the value of N should be is not always easy for the problem designer. If the value of N is not sufficiently great, then the ambiguity in the problem is not resolved and hence, the agent may not be able to distinguish a critical state to complete its task.

For instance, setting N as 2 is sufficient to solve the `mhn` problem depicted in Figure 2.2, since remembering the observation just before entering the corridor is enough to distinguish the critical and aliased states. However, if the length of the corridor were greater, then keeping only 2 past experience of the agent would not solve this ambiguity.

On the other hand, if N is set to a greater value than it should be, then the new state space of the agent, built up by its state estimations can be extremely and unnecessarily complex. In such a case, there might be so many combinations that correspond to same state and the agent may try to learn the values of each estimation separately. This redundancy may prevent the agent completing its task in a reasonable time for a problem with a larger state space. For instance, in the `2 rooms` domain with basic observation semantic, the o_5 is unique to the south-western corner of the left room (see Figure 2.7.). However, with Finite Size History approach, this state will correspond to multiple state estimations, and the number of these state estimations grows exponentially with respect to the value of N.

2.4.3.2 Long Short-Term Memory

Long Short-Term Memory is proposed as a variant of Recurrent Neural Network which aims to catch long term dependencies on the given data [16] to achieve supervised learning. In the context of RL, it is used to filter out the irrelevant part of the agent's experience and tries to memorize the useful information coming from the agent's interaction to solve given non-Markovian RL tasks.

This approach is usually applied on factored (high dimensional) RL environments especially assuming that RL agent is capable of high visual perception. In this way, it is able to solve complex non-Markovian RL tasks. However, such capability may not be always available to the agent. Furthermore, application of LSTM generally requires a different structure of neural network model with different number of layers and nodes for each problem. Considering the construction of model and parameter fine-tuning as additional costs to the learning process of the agent, LSTM may not be preferable over other simple RL approaches for most of RL tasks [30].

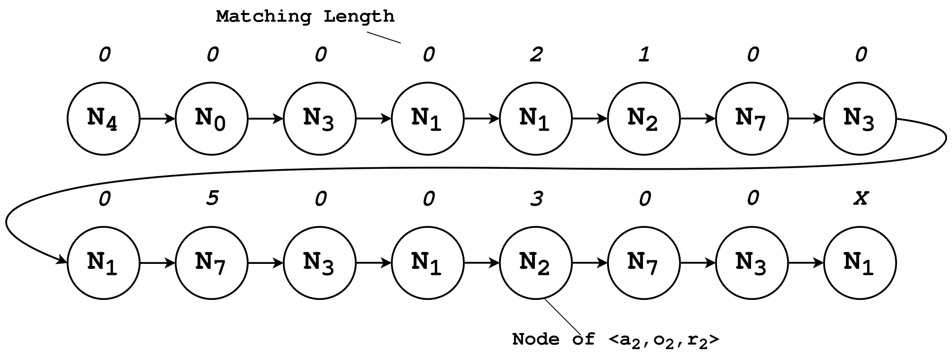


Figure 2.9: Neighborhood calculation in NSM

2.4.3.3 Variable-Length History

As the last category, The methods that use variable-length history will be covered in this section. These methods (except the first one) are also known as *instance based* methods and they are proposed in an incremental manner by McCallum [1].

- **Utile Distinction Memory (UDM)** uses a statistical test to split or join the current state estimation of the agent throughout the learning. Since the experience of the agent is not kept in a structured manner, the computation that UDM needs can be significantly expensive. Therefore, the learning of a UDM agent can be extremely slow. Also, this algorithm fails to catch the utility of a sequence, namely more than one step of the agent. Although UDM itself is not an instance-based methods, it leads the way for its succeeding algorithms.

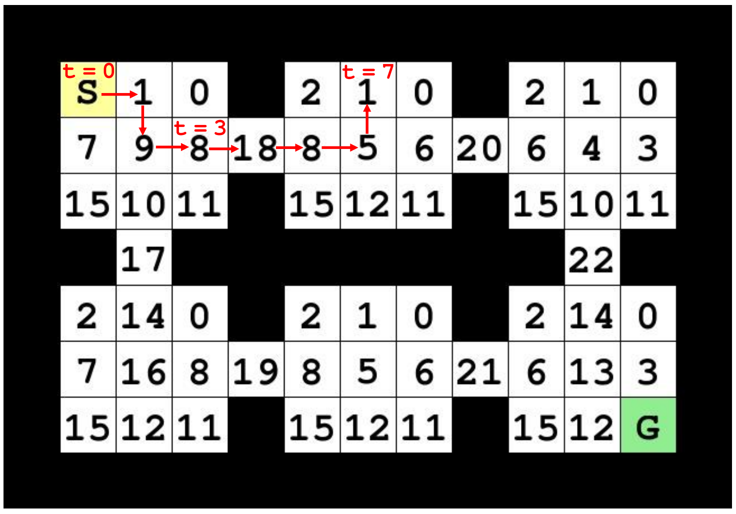
- **Nearest Sequence Memory (NSM)** is proposed to overcome the drawbacks of UDM about being inefficient and incapable of catching benefits coming from more long sequences of the agent's experience. NSM keeps the agent's interaction history in a linear structure of nodes each of which is a tuple constructed in the form of $< action,\ observation,\ reward >$. Then it recursively looks for the k-nearest neighbors of current experience within this linear structure where the distance between two experiences is expressed by the matching length, i. e. the number of common nodes while searching back in the sequence (see Figure 2.9.). In this manner, the utility of an action in the current state is calculated by averaging the values of the actions from these nearest neighbors.

Despite being better than UDM in complex tasks, NSM still fails even simple tasks when noisy calculations arise while sampling from the agent's own history. To be able to handle such noises, a more sophisticated structure to process the experience is needed as in the case of Utile Suffix Memory.

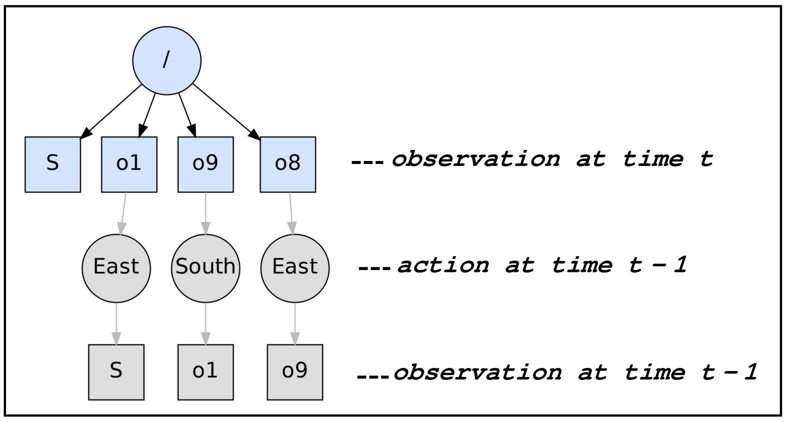
- **Utile Suffix Memory (USM)** combines the raw experience tracking with the utile distinction mechanism. It keeps track of a history of agent experiences throughout learning, in the form of a linear structure of nodes as NSM. Moreover, instead of directly processing the linear structure, it maintains the instances in a tree form, imitating a finite state machine inspired by Prediction Suffix Tree [31]. In this way, it can predict how much memory length is sufficient to use for that instance to estimate a distinctive state of the agent. Since the similar experiences have been kept in a tree that reflects their utile distinction, USM does not suffer from the noisy calculations as NSM do.

  Immediate children of the root node of the suffix tree represent the most recent observation. Lower levels are constructed by the sequence of previous actions and observations derived from the history of the agent. Leaf nodes of the tree are containers where the instances are kept. The agent adds a new instance to the tree following the path from the node to the matching leaf node. Every leaf node is created as a *fringe* node and ignored in the calculation until it is statistically proven to provide additional distinction. If the instances belonging to a fringe node can be shown to be from a different distribution (for example, by using the Kolmogorov-Smirnov test), then it is promoted as an official node of the tree.

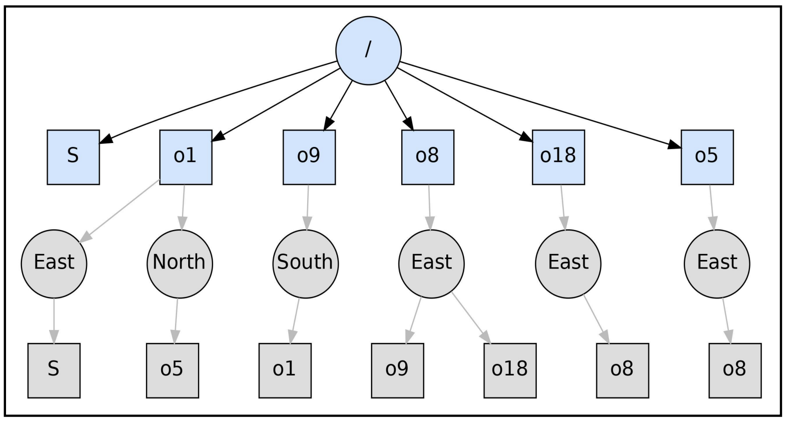
  Figure 2.10 depicts an example history of transitions in the downscaled version of `6 rooms` domain [11] along with the corresponding trees at time $t = 3$ and $t = 7$. The sensor capability of the agent to observe its current state is assumed to be based on classified distances to the surrounding obstacles and distinct doorways as landmarks. The maximum depth for fringe (grey) nodes in the trees is set as 2. In other words, at most two levels of fringe nodes may exist under one of the official (blue) nodes. The node labeled with "/" is the root node.



(a)



(b)



(c)

Figure 2.10: Sample transition history in a downscaled version of `6 rooms` domain (a), and corresponding suffix trees at time (b) t = 3, and (c) t = 7.

In a sense, USM tries to decompose the problem online, within the branches of its tree structure. If the state space of the problem is sufficiently small, USM effectively deals with the aliased states. However, it redundantly checks for further distinctions for the aliased states where a common optimal action, therefore, a reasonable abstraction for these states is available. Along with this property, keeping histories in both linear and tree structures results in extensive memory usage and longer processing time for the construction of a solution. Hence, most of the time, USM is far from being practical in larger problems.

- **U-Tree** applies the idea of USM for the learning agent having multi-dimensional perception. In this sense, the perception of the agent consists of features from the environmental state and they are subject to the utile distinction test as the agent's actions do. However, being an extension of USM with further branching based on the features of perception, U-Tree becomes an inefficient solution for the partially observable RL problems with larger state space as well.

2.5 Sample Problem Domains

In this section, the sample problems that are used in experimentation of proposed methods will be introduced. As mentioned before, navigational problems have been favored in our study as they are easy to visualize and trace. However, as a non-grid case, a well-known mathematical puzzle called `Tower of Hanoi` [28] will be covered as well.

All of the problems are assumed to be partially observable, and hence, the agents that try to solve them are not capable of perceiving their state in the environments completely. For all of the navigational problems, the observation semantic is based on either basic surroundings or classified distances as explained in Section 2.3.1 and Section 2.3.2 respectively. For the latter one, it is assumed that the problem is formulated with L-POMDP, in other words, there exists at least one landmark state in the problem. As the non-navigational example problem, `Tower of Hanoi` domain has its own type of observation semantic as explained in Section 2.5.3. Yet, this problem is formulated with L-POMDP as well.

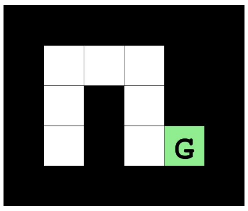
Along with the Tower of Hanoi problem, there are three types of sample domains that are covered in the following subsections including their environments, observation enumerations. Although they have different sizes and different observation semantics, sample problems essentially have similar transition graphs which is built according to the limited perceptive capability of the learning agent. For better understanding of the change in the problems by shifting to the case where the states are hidden, examples of these graphs are depicted for both partially and fully observable cases in these subsections as well.

2.5.1 Relatively Small Domains

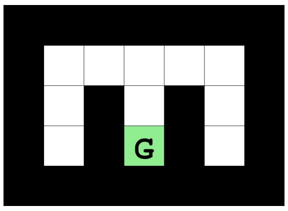
Relatively small domains as the first category of the sample domains include McCallum's maze and Hallway Navigation (mhn) domain along with the downscaled versions of other benchmark domains in the RL literature (see Figure 2.11-2.13.). The problems in this category has a smaller state space, but they may still be difficult to solve under partially observability due to perceptual aliasing. The cardinalities of the set of states and set of observations are given in Table 2.2 along the with number of possible actions. If a problem has the extended observations semantic by using the classified distances, then it is formulated with L-POMDP. Hence, the number of landmarks for such problems is included in the same table as well.

Table 2.2: The number of states, observations, landmarks and actions in relatively small sample domains

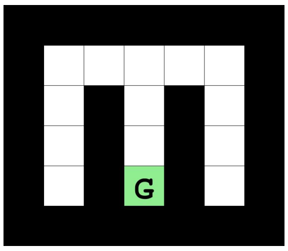
	$\|S\|$	$\|O\|$	$\|L\|$	$\|A\|$
maze1	8	7	-	4
maze2	11	7	-	4
maze3	14	7	-	4
maze4	15	7	-	4
mhn	23	9	-	4
downscaled 2 rooms	51	11	-	4
downscaled zigzag rooms	39	21	9	4
downscaled 6 rooms	60	25	12	4



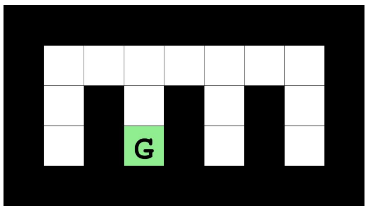
(a) `maze1`



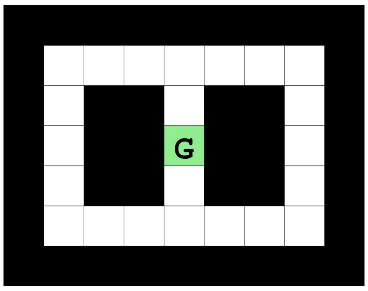
(b) `maze2`



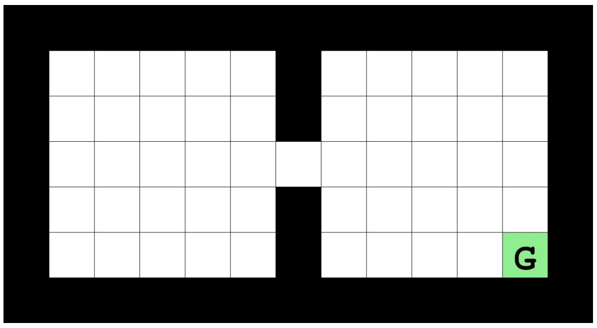
(c) `maze3`



(d) `maze4`



(e) `mhn`



(f) `downscaled 2 rooms`

Figure 2.11: *Relatively* small domains with basic observation semantic used in experimentation.

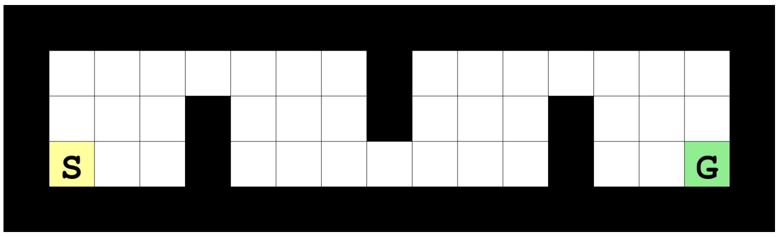


Figure 2.12: `downscaled` version of `zigzag rooms` as one of relatively small domains

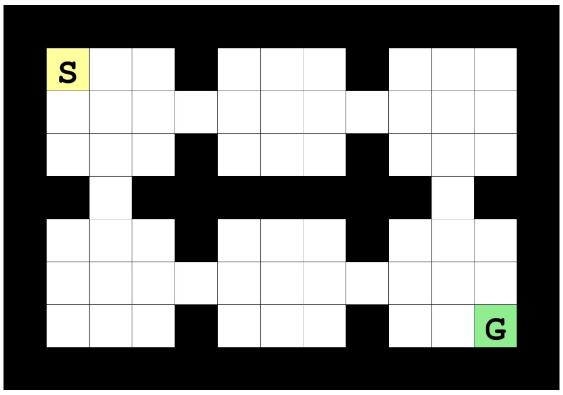


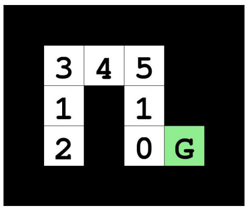
Figure 2.13: `downscaled` version of `6 rooms` as one of relatively small domains

Figure 2.11 shows the relatively small domains used in the experimentation each of which has basic observation semantic explained in Section 2.3.1. First five of them are McCallum's navigational problems [1] and the last one is a variant of `2 rooms` domain [11] where the state space is smaller. For these domains, the agent is assumed to be placed a random location (in left room for `downscaled 2 rooms` domain) other than the goal state indicated by both the letter 'G' and light-green colored cell at the beginning of each episode.

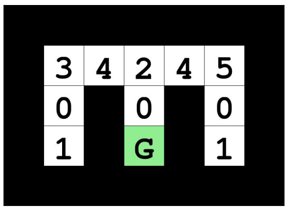
Figure 2.12 and Figure 2.13 show remaining relatively small domains in which the observation semantic based on the classified distances is applied. For these domains the initial states of the agent is assumed to be fixed in the corners indicated by both the letter 'S' and light-yellow colored cell at the beginning of each episode.

For all of the domains mentioned above, the task for the agent is finding the path to the goal. There is no other terminal states for these domains other than the goal ones. All the tasks are episodic and an episode is terminated within a limit for number of steps taken by the agent if the task has not been completed yet. If the agent hits an obstacle, then it is assumed to remain its current cell. The possible actions are moving in 4 main cardinal directions, namely, `North`, `East`, `South`, and `West`.

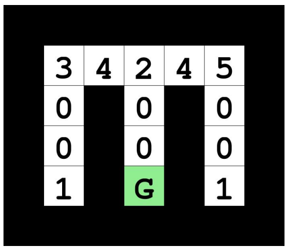
Observation enumerations for all of the domains are given in the Figure 2.14-2.16. Although using the extended semantic in `downscaled 6 rooms` makes the problem easier to solve, `downscaled zigzag rooms` severely suffers from the perceptual aliasing despite this semantic.



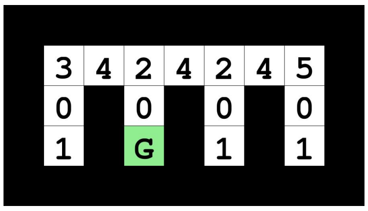
(a) maze1



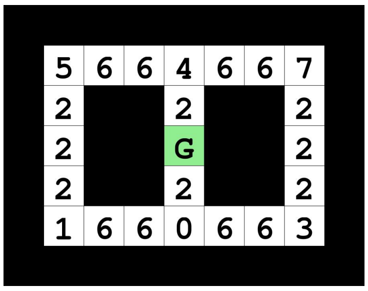
(b) maze2



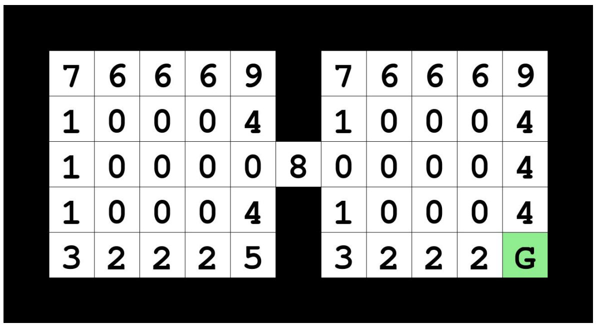
(c) maze3



(d) maze4



(e) mhn



(f) downscaled 2 rooms

Figure 2.14: Observation enumerations in the domains where the agent is assumed to sense the surroundings of its current cell only.

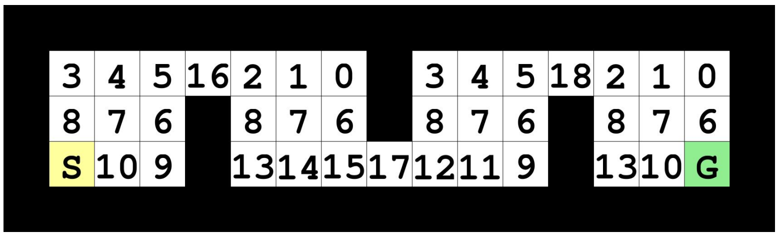


Figure 2.15: Observation enumeration based on classified distances in `downscaled zigzag rooms`

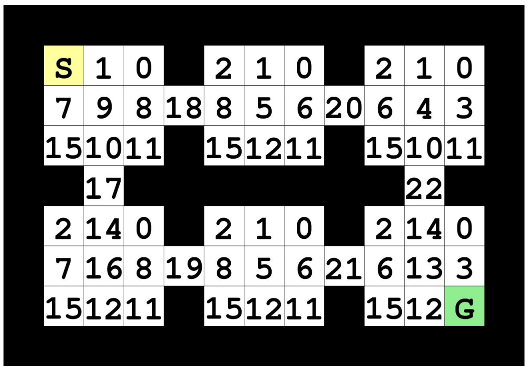


Figure 2.16: Observation enumeration based on classified distances in `downscaled 6 rooms`

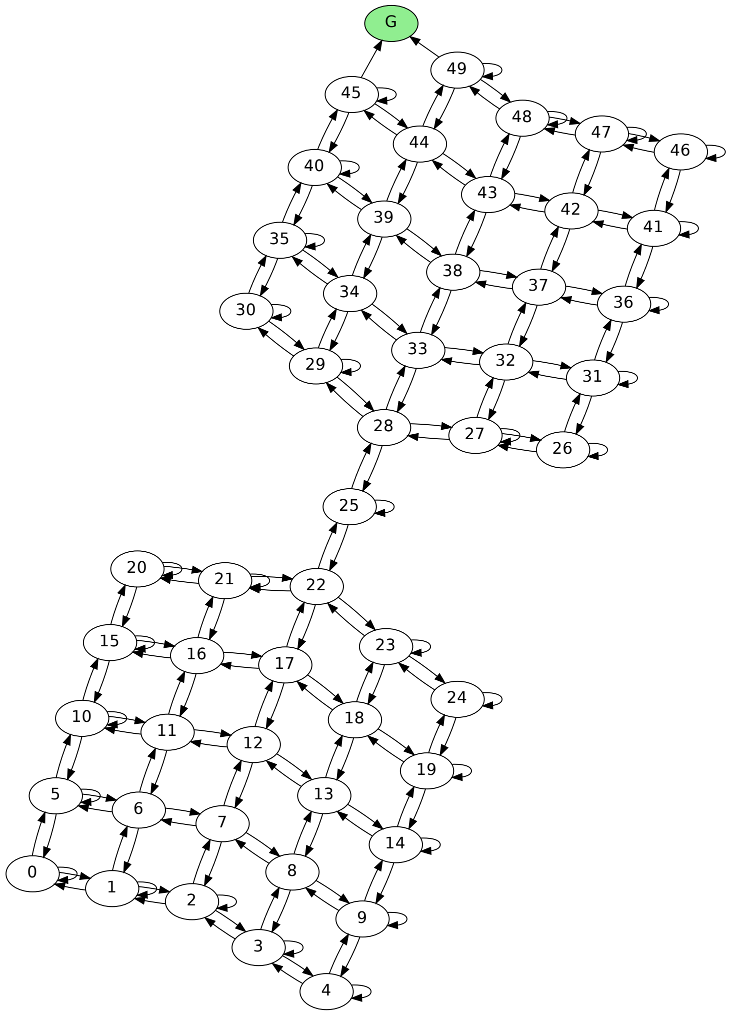


Figure 2.17: State transition graph of `downscaled 2 rooms` domain

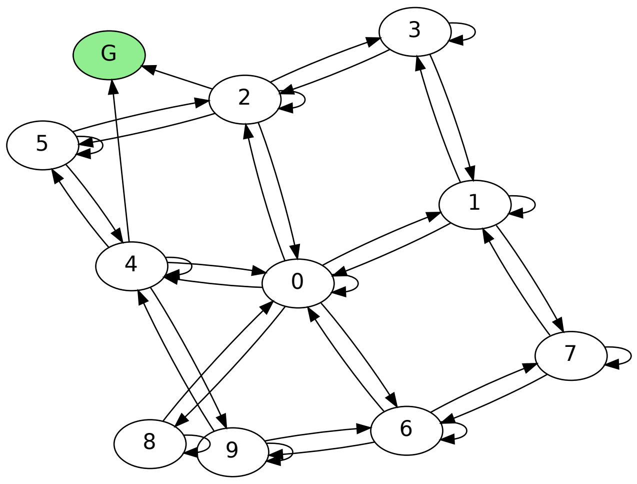


Figure 2.18: Observation transition graph of `downscaled 2 rooms` domain

In order to show that how the agent's point of view is changed with its capability of perceiving its current state completely or partially, the transition graphs of `downscaled 2 rooms` domain are given for both cases in Figure 2.17 and Figure 2.18 respectively. These figures clearly indicate that even the simplest RL problem can get really complicated under partially observability.

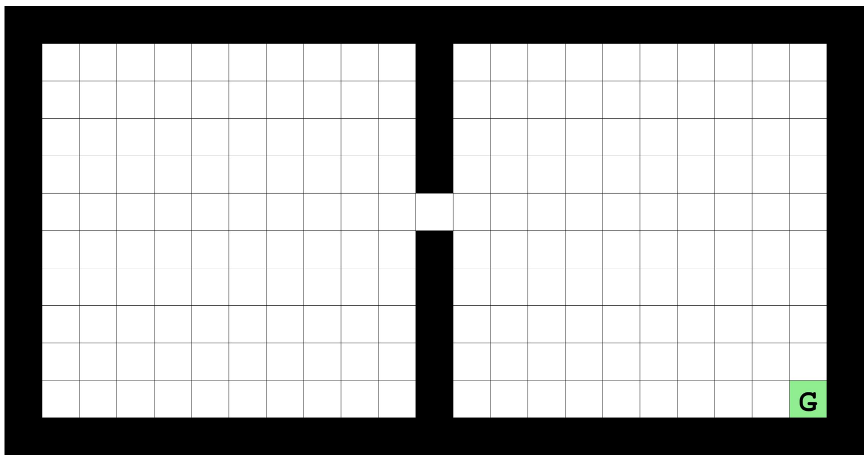


Figure 2.19: `2 rooms` problem

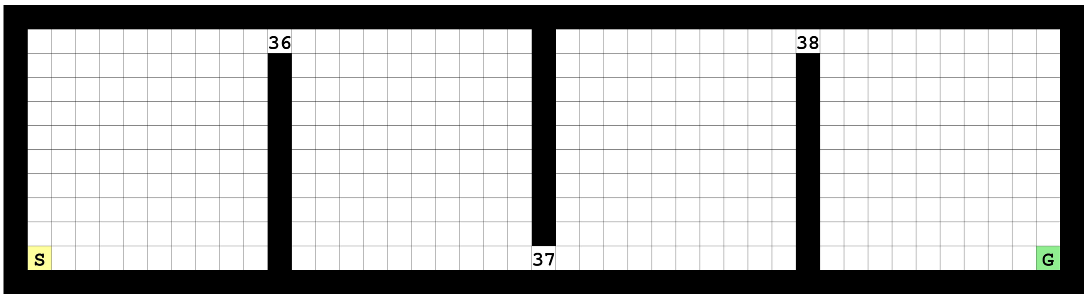


Figure 2.20: `zigzag rooms` problem

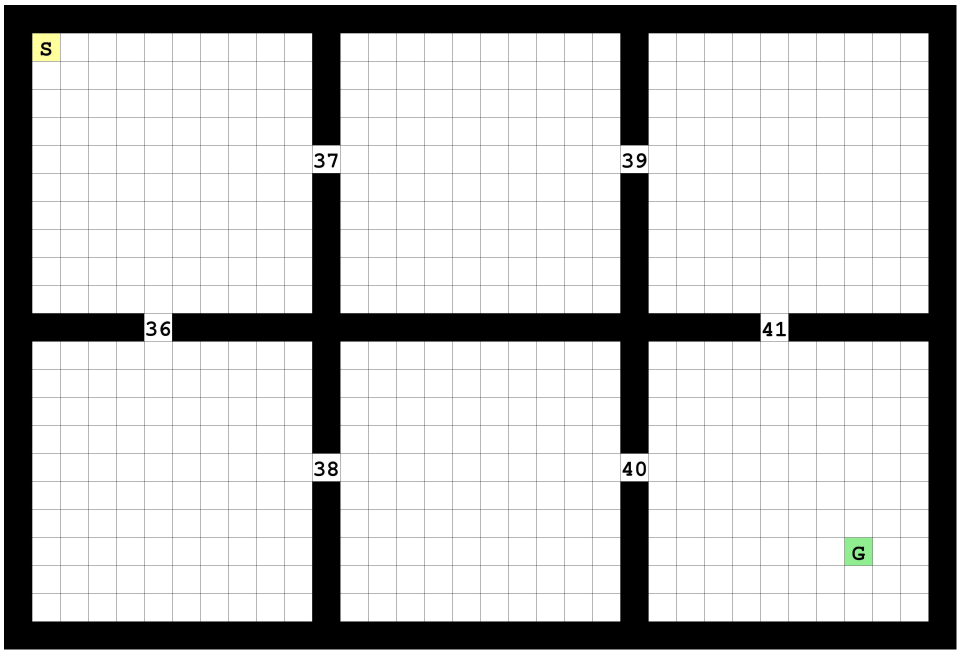


Figure 2.21: `6 rooms` problem with a fixed initial state and enumerated landmarks

2.5.2 Larger Domains

Second category contains the benchmark problems with larger state space including `2 rooms` and `6 rooms` from Menache et.al's study [11], `zigzag rooms` [19] and `6 rooms with locked shortcut` as a version of `6 rooms` domain where one of the doors are locked until the state with key is visited (see through Figure 2.19-2.22.). All of the domains comply with the extended observation semantic based on classified distances as explained in Section 2.3.2.

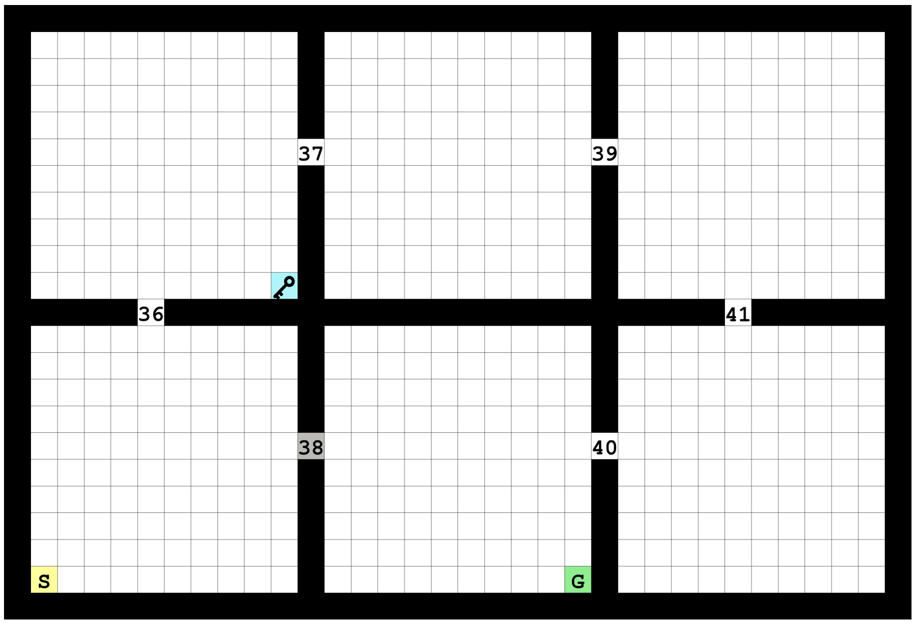


Figure 2.22: `6 rooms with locked shortcut` problem

Table 2.3: The number of states, observations, landmarks and actions in larger domains

	$\|S\|$	$\|O\|$	$\|L\|$	$\|A\|$
`2 rooms`	201	38	3	4
`6 rooms`	606	43	7	4
`6 rooms`$_{FS}$	606	44	8	4
`zigzag rooms`	403	41	5	4
`6 rooms with locked shortcut`	606	85	15	4

The domains called `zigzag rooms` and `6 rooms with locked shortcut` are assumed to have an initial state (denoted by both the letter 'S' and light-yellow colored cell in the figures) which yields a unique observation. For the experimentation of the second proposed method, the same assumption is applied for `6 rooms` domain as well. To ease the following of the configurations, the domain is called `6 rooms`$_{FS}$, if it has an only one initial state. Figure 2.21 shows this version of `6 rooms` to avoid redundant repetition. For the original domain, the agent is assumed to be randomly placed in the upper-left room at the beginning of each episode. The assumption of fixed initial location is not applied for `2 rooms` domain, and hence such a notation is not need for this problem. Table 2.3 shows the cardinalities of the sets of states, observations, landmarks, and the possible actions for larger domains.

Observation enumerations for all of the larger domains are given in the Figure 2.23-2.26. If the fixed initial state is not assumed in `6 rooms` problem, then the upper-left corner of the first room yields the o_3 exactly like corresponding locations in other rooms. In the problem named `6 rooms with locked shortcut`, beside the given enumeration, an additional 1-bit information is provided for the agent to indicate whether the location denoted with letter ‘K’ is visited and hence, the agent has the key or not. Although these domains have the advantage of extended observation semantic, perceptual aliasing still prevents the learning agent from an easy solution.

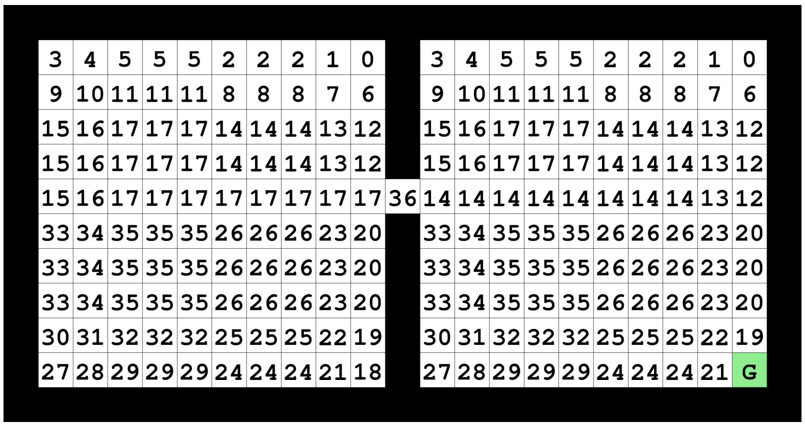


Figure 2.23: Observation enumeration in `2 rooms` problem

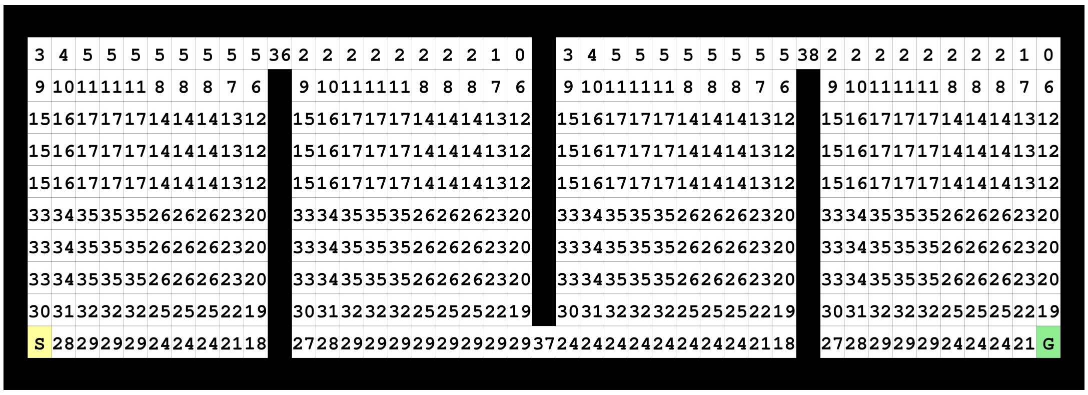


Figure 2.24: Observation enumeration in `zigzag rooms` problem

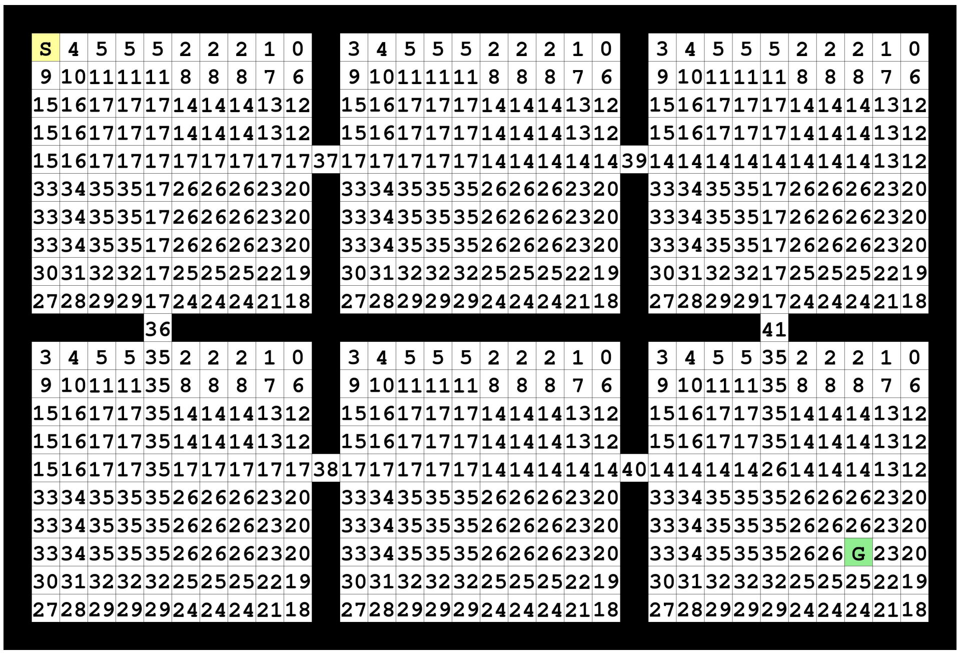


Figure 2.25: Observation enumeration in `6 rooms` problem with a fixed initial state

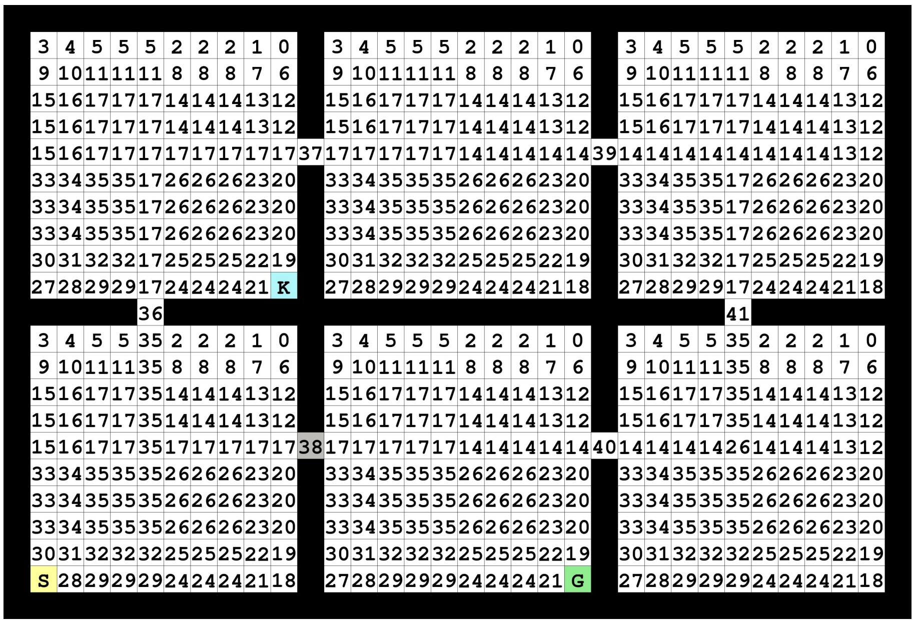


Figure 2.26: Observation enumeration `6 rooms with locked shortcut` problem. Another 1-bit information is provided for this domain indicating whether the location denoted with letter 'K' is visited or not

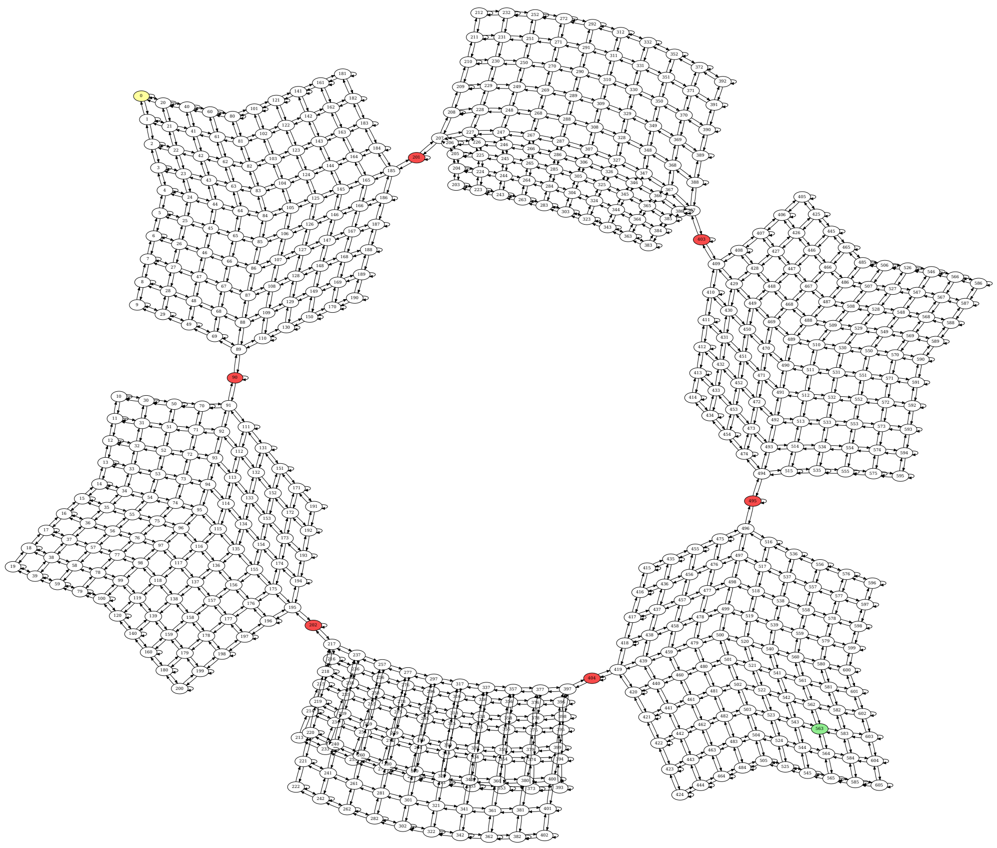


Figure 2.27: State transition graph of `6 rooms`$_{FS}$ domain

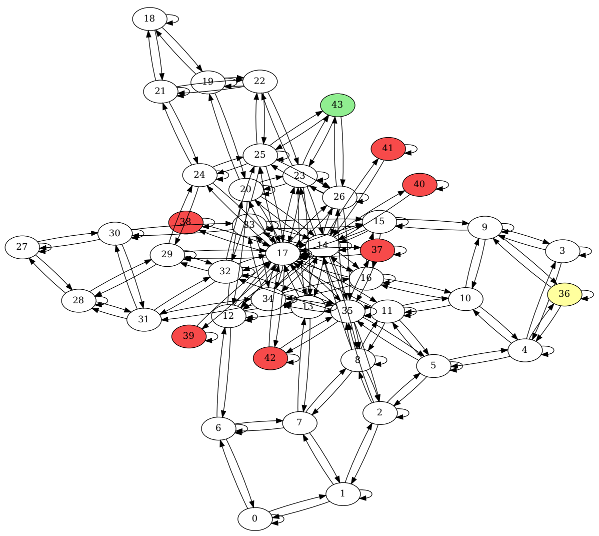


Figure 2.28: Observation transition graph of `6 rooms`$_{FS}$ domain

Figure 2.27 and Figure 2.28 show the transition graphs of the agent in `6 rooms`$_{FS}$ problem based on states and observations respectively. Initial and final states (or observations) are indicated with light-yellow and light-green color respectively where the landmarks are red-colored. These figures can be useful for better understanding of the aliasing problem with a larger state space.

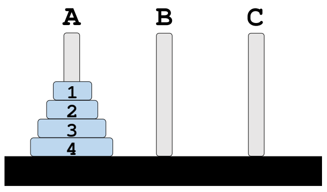
2.5.3 A Non-Grid Case: Tower of Hanoi

The Tower of Hanoi problem was invented in 19^{th} century and became a well-known puzzle which consists of 3 rods and a number of disks with varying sizes [28]. Initially, all disks are placed onto a rod in the ascending order of their sizes, so it looks like a neat, canonical shape (see Figure 2.29a). The task is moving the whole stack of disks to another rod (see Figure 2.29b) without violating the following rules:

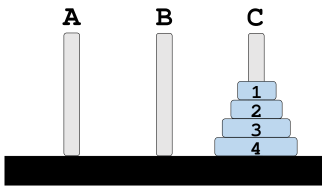
- Only one disk can be moved at a time.
- A disk can be moved only if it is on the top of its rod.
- No larger disk can be placed onto the top of a smaller one at any time. In other words, the ascending order regarding the size of disks should not be disturbed on any rod during the entire solution.

In our experiments, we assumed that the number of disks is 4 and the agent is able to perceive only the disks on the top of the rods with their labels. In other words, the agent cannot differ the disks with their sizes or cannot know how many disks are placed on a rod. Therefore, it has a point of view as exampled in Figure 2.30a. If the agent gets the sample observation shown in this figure, there are 2 possible cases for the actual state of the agent, since the 3^{rd} disk can be under 1^{st} or 2^{nd} disk (see Figure 2.30b and Figure 2.30c).

With this configuration of the problem, there are 99 different states in which the agent can reside and 39 possible observations including 15 unique observations for the landmark states. The number of actions is 6 assuming that the agent can directly move a disk from the top of a rod to another (from rod A to rod B, A to C, and so on).

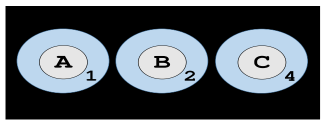


(a) Initial state

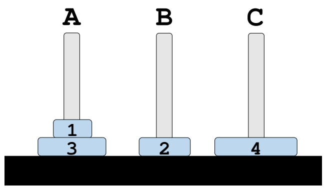


(b) Goal state

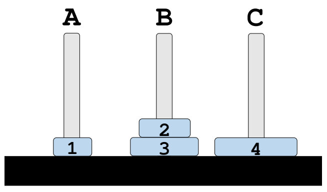
Figure 2.29: Initial and goal states of `Tower of Hanoi` problem with 4 disks



(a) A sample observation of agent



(b) A possible corresponding state



(c) Another possible corresponding state

Figure 2.30: A sample observation of the agent while it can only sense the labels of disks on the top of rods in the `Tower of Hanoi` problem and possible corresponding states for the same observation

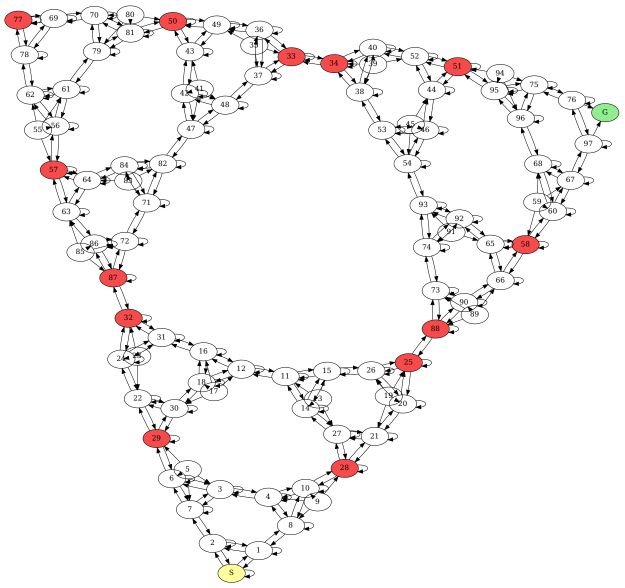


Figure 2.31: State transition graph of `Tower of Hanoi` domain

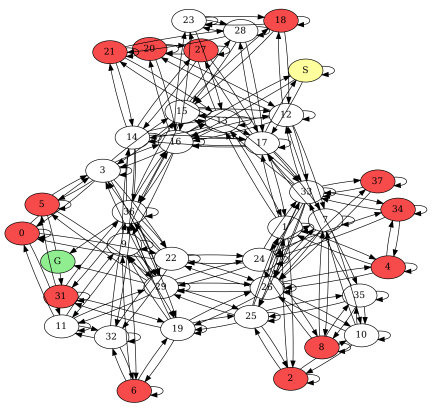


Figure 2.32: Observation transition graph of `Tower of Hanoi` domain

As the last of the sample domains, this chapter is concluded with transition graphs for `Tower of Hanoi` problems based on both states and transitions (see Figure 2.31 and Figure 2.32.). Same color-mapping in the case of `6 rooms`$_{FS}$ domain's transition graphs is applied here as well. These graphs show the complexity of the puzzle for both fully and partially observable cases. Now that all necessary background information is provided for better understanding of both methods and experimentation, we can start to examine the proposed methods beginning from the next chapter.

CHAPTER 3

COMPACT FREQUENCY MEMORY FOR REINFORCEMENT LEARNING WITH HIDDEN STATES

Memory-based reinforcement learning approaches tend to keep track of only recent or whole experiences of the agent in partially observable environments. This may require extensive use of memory that limits the practice of these methods in a real-life problem. Therefore, using a selective memory can enhance the scalability of the methods by leading a form of compendious state estimation.

In this chapter a selective and compact form of memory is proposed, considering the fact that less frequent transitions provide more reliable information about the current state of the agent in ambiguous environments [25]. Our approach avoids the redundancy in the memory by eliminating the frequent transitions which are more likely the ones introducing the ambiguity in the environment and misleading the agent. Experiments show that the usage of a selective and compact memory may improve and speed up the learning process in a more scalable way.

3.1 Motivation behind the Usage of a Selective Memory

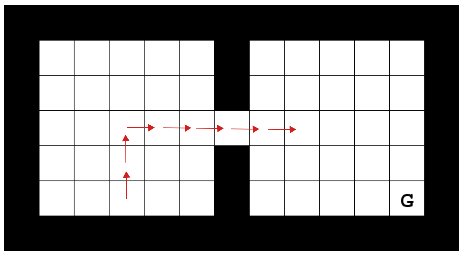
There are a number of memory-based method categories trying to cope with the perceptual aliasing in a partially observable RL problem. *Finite size history* is one of the simplest memory-based approaches [15]. Instead of keeping the current observation, the agent uses a chain of last *n* steps as the current observation. Despite its simplicity of application, it requires prior information about the environment such as the minimum size of the memory needed to construct a solution. Recurrent-Q is another method that uses *Long Short-Term Memory (LSTM)* [16]. LSTM is a Recurrent

Neural Network (RNN) variant used to filter out the irrelevant information about the state space from the all experiences of the agent. Although the method is proven to be useful over visual domains, the generalization of the method is not practical for the reinforcement learning tasks since the procedures before learning like separate parameter tuning for each problem take too much time [30]. Yet another memory-based method category is the *variable-length history* family. In this approach, the size of the internal memory can be altered depending on dynamically changing memory requirement for the underlying learning procedure. Nearest Sequence Memory (NSM) and Utile Suffix Memory (USM) algorithms are examples of this category (also called *instance-based* methods) where all raw data of the agent's experience are kept in the form of action-reward-observation tuples [1].

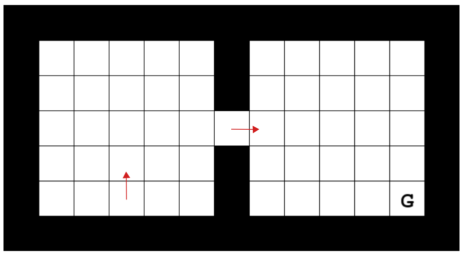
All of the methods above tend to use the recent or all the experiences of the agent collected throughout the learning process. Unfortunately, especially for problems with large state space, it becomes difficult to determine the size and content of the memory required to provide a distinctive clue about the current state of the agent. Keeping the whole history of the agent can solve this problem, yet the estimated state, namely the information unit constructed by the agent to be able to separate the aliased states, can be severely complex [17]. Hence, this approach is not scalable.

In a typical RL problem with hidden states, the observation function leads the perceptual aliasing which means that the agent gets the same observation for more than one of the states, based on its own capability of sensing the environment. As the ambiguity in the problem increases, some observations and the transitions over these observations are more frequently seen than the rest. Although keeping these common experiences in the memory may be useful for the construction of unambiguous state estimations in the long term, it mostly results in a larger state space and prevents the agent from finding a practical solution for the most of the problems where a faster solution is available by the selection of correct set of experiences providing the information that the agent needs.

Consider the domain given in Figure 3.1 where the agent gets an observation for the current location based on the surrounding obstacles. The agent should know whether it passed through the door to the neighboring room where the goal state (the cell represented with the letter “G”) resides. In order to do so, methods like NSM and USM keep track of all of the transitions between the starting state and the current one. However, by the elimination of the frequent transitions in the environment and keeping significant transitions as in Figure 3.1b, the agent can obtain more reliable and efficient information about its current path. Keeping the recent one or two transitions also fails to provide information about the agent's current path if the agent keeps moving to east in the same environment. The fixed size of memory can be increased for the domain, but larger domains will require a greater size for the memory. Hence, the time that method needs for the learning process in such cases will be beyond a reasonable amount.



(a)



(b)

Figure 3.1: (a) The transitions form the memory in NSM and USM (b) The transitions needed in a selective memory

3.2 Compact Frequency Memory

Being in an unknown city, a tourist needs some reference points to find a path to their destination. These reference points should be distinguishable among others. This case is similar to that of a learning agent in an unknown environment. If a solution exists for the given problem with hidden states, the agent may find its path to the goal more easily by using *relatively* reliable reference points. Usually, the most reliable information comes from the unique transitions which occurs in only one state in the environment. However, by using not only unique transitions but also less frequent ones, the agent may still distinguish the aliased states which lead to the same observation.

By equipping the agent with a selective memory, the state space of the problem can be projected into a space including estimated states each of which carries the information of the agent's own whereabouts in smaller units. In this way, the agent avoids processing an extensive memory and dealing with much larger estimated state space, and therefore, the time needed for the construction of a solution becomes less than other existing memory-based methods. Therefore, it is wise to use a selective memory for the RL problems having a larger state space, since the scalability of the method becomes one of the key criteria in such cases.

Algorithm 1 explains how CFM works during the learning process. The agent keeps track of the frequency of every transition in the domain, based on observations and actions. Then it compares the frequencies of all transitions with a given threshold to select the infrequent ones. This selection is repeated periodically, because the agent may experience a useful transition for the first time with the discovery of a new path in a very large domain. In this way, the information gathered from a new transition can be preserved. After generating a set of infrequent transitions, the agent uses this set to form a state estimation about its current state. The state estimation is composed of last n significant transitions and the most recent observation. If the current transition is also significant and is not the same as the last one, then it is updated by removing the oldest transition.

Algorithm 1 CFM Algorithm

```
1: procedure CFM
2:     require: $\alpha$, $\gamma$
3:     require: $P$, $FT$                                    ▷ Period and Frequency Threshold
4:     $IP \leftarrow \emptyset$                                         ▷ set of infrequent pairs
5:     while learning continues do
6:         $t \leftarrow 0$
7:         $CS \leftarrow \emptyset$                        ▷ current selection from infrequent pairs
8:         while episode continues do
9:             take action $a_t$, observe $o_t$ and $r_t$
10:            increase $f(p_t)$, namely the frequency count of the pair ($o_t$, $a_t$)
11:            if ($o_t$, $a_t$) $\in IP$ then
12:                if ($o_t$, $a_t$) $\neq$ ($o_{t-1}$, $a_{t-1}$) then
13:                    remove $argmin_t(o_t, a_t)$ from $CS$
14:                    add ($o_t$, $a_t$) to $CS$
15:                end if
16:            end if
17:            apply learning update on the estimation ($CS$, $o_t$)
18:            $t \leftarrow t + 1$
19:        end while
20:        if episode number matches with $P$ then
21:            for each pair $p_i$ in transition pairs do
22:                $nf_{p_i} = \frac{(f_{p_i} - min_j f_{p_j})}{(max_j f_{p_j} - min_j f_{p_j})}$          ▷ normalized $f$
23:                if $nf_{p_i} < FT$ then
24:                    $IP = IP \cup \{p_i\}$
25:                end if
26:            end for
27:        end if
28:    end while
29: end procedure
```

After the construction of the state estimation with current selections from infrequent transitions and the current observation, the underlying learning update can be applied on this estimation. At first, CFM was implemented over Q-Learning and published in this way [25], then it was generalized to work with any underlying learning method as it should do.

The idea of building up a memory using infrequent transitions is based on the assumption that for most of the problems with hidden states, there are less frequent unaliased states than aliased ones. If this assumption does not hold for a domain where the transitions are observed almost uniformly, the current selection from the infrequent transitions used in the algorithm will be updated for almost every step of the agent. Hence, CFM reduces to the fixed sized memory-based Q-learning. However, one should know that a uniform distribution of the transitions rarely occurs in the RL problems with hidden states where the observation capability of the agent can be variously limited.

3.3 Experiments

Experimentation of CFM was initially conducted over Q-learning for two different types of domains. In the first one, the selected domains are relatively small but aliased states make the problems difficult to solve without a memory. In these domains, CFM is compared with both memory-based (NSM and USM) and memoryless (Q-Learning and Sarsa(λ)) approaches. In the second one, larger domains are used in experimentation where NSM and USM approaches cannot be scaled to perform learning in a reasonable time and Q-Learning is not able to reach a convergence during learning. Therefore, in these domains only the results of CFM and Sarsa(λ) are given.

During the shift of programming language for our codebase from `C++` to `Python`, the implementation of CFM was extended so that a Sarsa(λ) agent can be equipped with it. Hence, the learning performance of this version is examined by sampling one relative small domain and one of the larger domain as well. Details of the experimentation settings are covered in the next subsection.

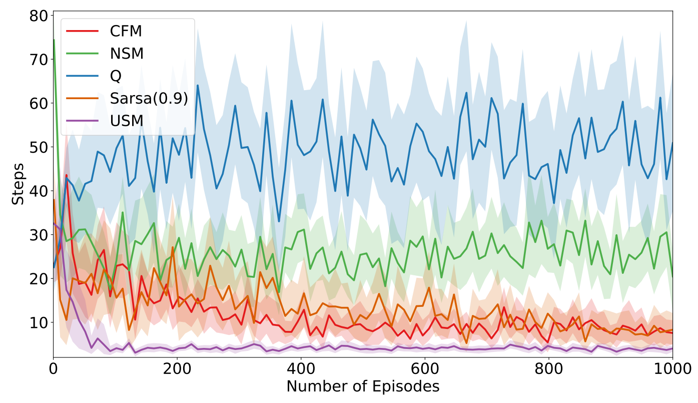
3.3.1 Settings and Results

3.3.1.1 CFM over Q in Relatively Small Domains

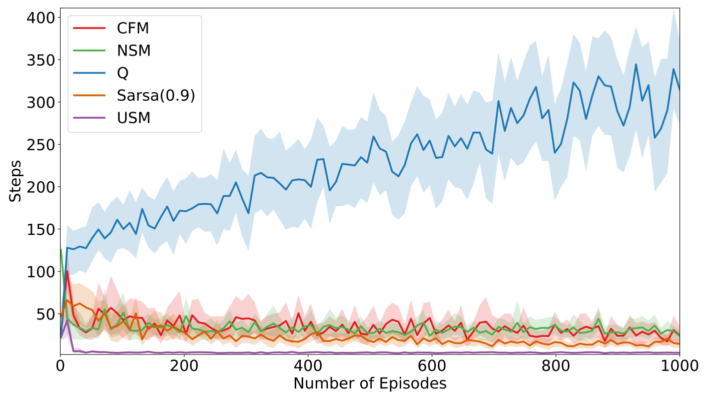
In the first setting of the experimentation, six relatively domains are used. First five of them are McCallum's maze domains and Hallway Navigation (mhn) domain [1] (see Section 2.5). In these domains, the methods with an internal memory usage have advantages due to the ambiguity. Only Sarsa (λ) algorithm achieves to get similar results with memory-based approaches by the help of its eligibility trace mechanism. The last domain is downscaled version (depicted in Figure 2.11f) of 2 rooms [11] which is less complex than McCallum's domains. However, in this domain, there are more aliased states and hence, the memory-based approaches suffer from redundantly growing state space during the learning. The environment yields different reward values for hitting a wall, passing to a neighboring state, and reaching the goal state, which are -0.1, -0.01, 1 for McCallum's maze domains and the downscaled 2 rooms domain, and -1, -0.1, 5 for the mhn, respectively.

In each domain, 50 experiments are executed for 1000 episodes. For every experiment, the agent is placed some random starting state (in the left room for the downscaled 2 rooms domain) other than the goal state and expected to find its path to goal state. As each method use ϵ -greedy learning strategy, ϵ value which makes the agent enable to explore the environment better is set as 0.1. The discount factor γ is chosen as 0.9 for all methods.

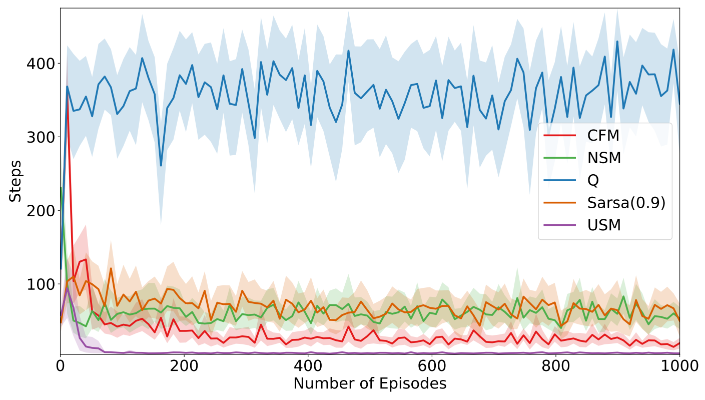
In each domain λ value for Sarsa(λ) is set as 0.9. NSM uses the learning rate α of 0.9 while all other methods use 0.05. Since NSM uses a different semantic for the value update, this does not harm the fair comparison between this method and the others. The other NSM specific parameter which represents the number of neighbors, k, is chosen as 4. The maximum fringe depth for USM method is set as 4. The threshold and period parameters of the CFM are configured as 0.2 and 20 for the given domains. The memory length of CFM is set as 1, namely only one transition is used by CFM for the state estimation.



(a) `maze 1`

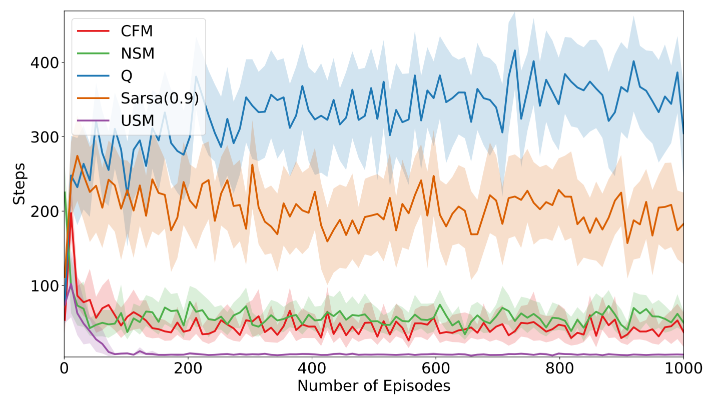


(b) `maze 2`

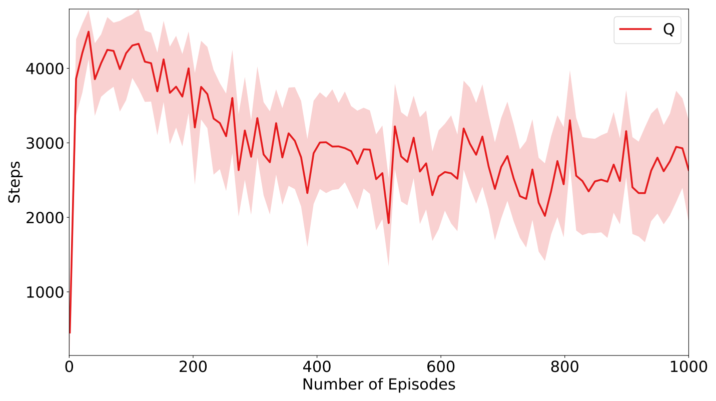


(c) `maze 3`

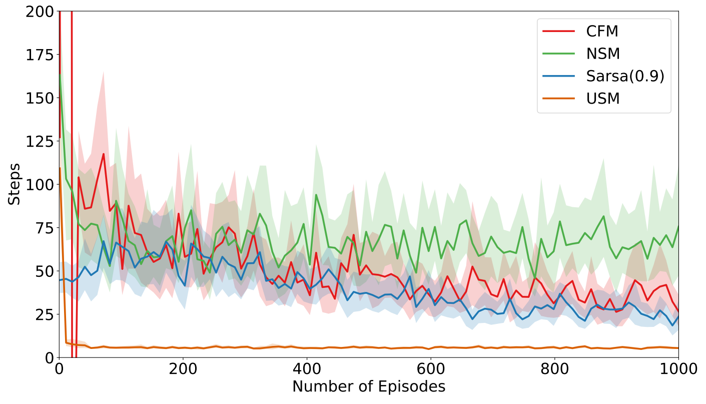
Figure 3.2: Learning performances of CFM over Q and other methods in `maze1-3` domains in terms of number of steps to the goal



(a) `maze 4`



(b) `Q-Learning in mhn`



(c) `others in mhn`

Figure 3.3: Learning performances of CFM over Q and and other methods in the rest of McCallum's domains in terms of number of steps to the goal

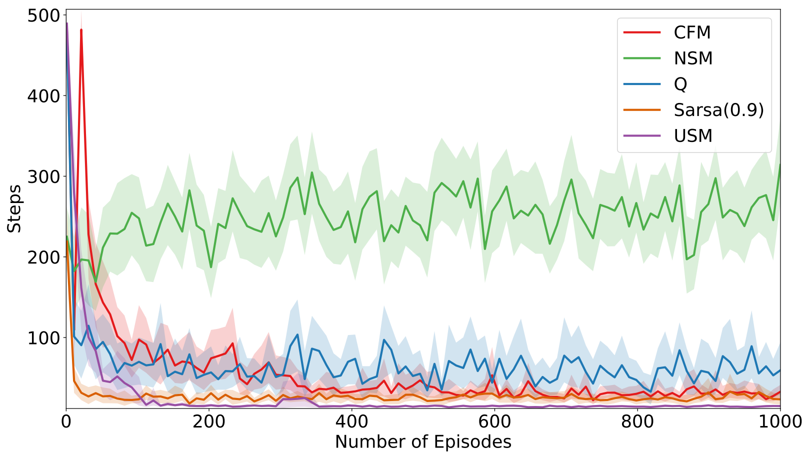


Figure 3.4: Learning performances of methods in `downscaled 2 rooms` in terms of number of steps to the goal

Figure 3.2-3.4 show the learning performances of the methods in terms of number of steps to reach the goal for the methods in McCallum maze domains, `mhn` domain and `downscaled 2 rooms` domain. For all of the domain, CFM shows a similar performance to the other methods which use a complex memory like USM and NSM or Sarsa (λ) which imitates a memory by trace mechanism. Since Q method does not have either of these properties, it suffers from poor learning performance except the last domain.

The time spent in the learning process is as important as finding an optimal and near-optimal solution for the given problem. Table 3.1 shows the average elapsed time (msec) for the methods in an episode. CFM clearly takes significantly less time than the other memory-based methods.

Table 3.1: Average Elapsed Time (msec) for The Methods in An Episode

	`maze 1`	`maze 2`	`maze 3`	`maze 4`	`mhn`	`downscaled 2 rooms`
NSM	10.14	15.25	70.56	50.74	1.22	1237.00
USM	10.72	17.69	72.57	163.13	21.21	4256.43
CFM	**0.09**	**0.25**	**0.29**	**0.39**	**0.89**	**1.60**
Sarsa(λ)	**0.08**	**0.07**	**0.42**	**1.22**	**0.30**	**0.78**
Q	0.32	1.53	2.39	2.32	19.91	1.75

Table 3.2: Average Number of Estimated States for Memory-Based Methods in An Episode

	maze 1	maze 2	maze 3	maze 4	mhn	downscaled 2 rooms
NSM	14241.56	17816.17	31834.79	30615.69	35289.19	123290.16
USM	97.68	124.47	203.70	277.02	**117.46**	588.19
CFM	**88.48**	**85.38**	**97.20**	**98.28**	120.88	**190.61**

Table 3.3: Average Memory Units for Memory-Based Methods in An Episode (Roughly Given)

	maze 1	maze 2	maze 3	maze 4	mhn	downscaled 2 rooms
NSM	166.89	208.78	373.06	358.78	413.55	1444.81
USM	56.01	61.99	100.70	126.22	64.97	294.63
CFM	**0.44**	**0.45**	**0.44**	**0.44**	**0.54**	**0.70**

When we compare the number of estimated states of memory-based methods, we get the result in Table 3.2. The number of estimated states is the number of nodes in the history chain for the NSM method, and is the number of official nodes in the suffix tree for the USM method. CFM has the advantage of creating less estimated states in almost every domain.

The last measurement examined for this experimentation setup is the memory usage of the NSM, USM and CFM methods. For the measurement of the NSM method, the number of nodes in the history chain is counted. The memory usage of USM consists of a similar history chain but also the official and fringe nodes in the tree. Each of the nodes in NSM and USM requires 12 Bytes (the total size of 2 `int` and a `float` in a standard C++ implementation). The memory required by CFM is the number of all transitions in the domain to keep track of the frequencies (12 Bytes for each transition) and the number of the selected infrequent transitions (8 Bytes for each transition). Obviously, this setup underestimates the memory usage of NSM and USM since it ignores the memory required for linking the nodes. However, even with this measurement, the memory usage of CFM is significantly less than NSM and USM. Table 3.3 shows that memory requirement of CFM is less than the other methods, which indicates that CFM is more scalable for the larger domains.

3.3.1.2 CFM over Q in Larger Domains

In order to show the scalability of CFM method, two larger domains, `2 rooms` and `6 rooms` [11] respectively given in Figure 2.19 and Figure 2.21 (without a fixed initial state) are used. As other memory-based methods cannot handle the growing space of estimated states in the problems, and Q-Learning fails to achieve the tasks, this setting only includes the comparison of CFM over Q with Sarsa(λ).

In each domain, 50 experiments are executed for 10000 episodes. ϵ is set as 0.1. The agent is placed in a random state in left and left-upper room for the first and second domain respectively and expected to find its path to goal state represented by the letter *G*. Both Sarsa(λ) and CFM methods use the learning rate α as 0.01. λ is chosen as 0.9 for the Sarsa(λ), like in the small domains. The memory length, threshold and period parameters of CFM are configured as 1, 0.1 and 100 respectively.

Figure 3.5 and Figure 3.6 show the learning performances of both methods in terms of number of steps to goal in larger domains. Despite being slower in `2 rooms` domain, CFM reaches a similar near-optimal solution. On the other hand, CFM outperforms Sarsa(λ) in `6 rooms` domain as can be seen in Figure 3.6. This indicates that even Sarsa(λ) which uses a powerful tracing mechanism may suffer from the aliased states in a very large domain, where CFM is effected less via its compact memory.

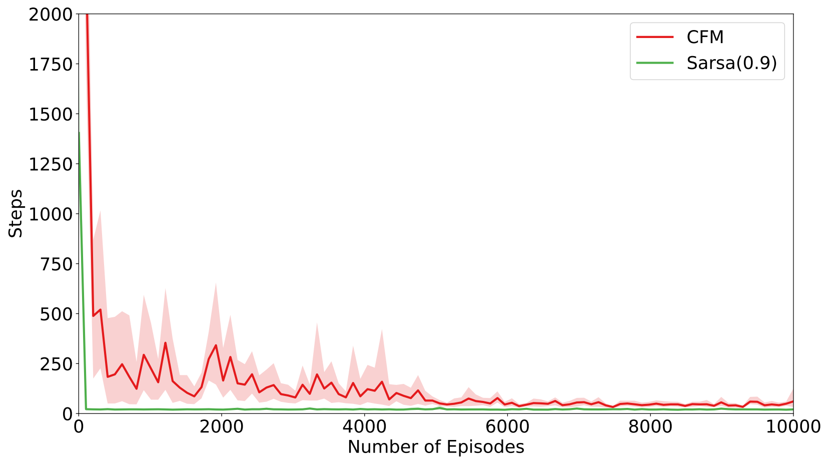


Figure 3.5: Learning performances of methods in `2 rooms` in terms of number of steps to the goal

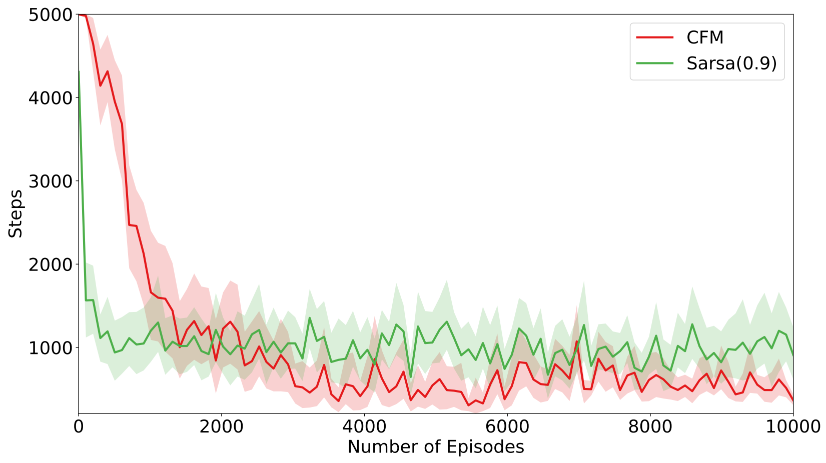


Figure 3.6: Learning performances of methods in `6 rooms` in terms of number of steps to the goal

3.3.1.3 CFM over Sarsa(λ)

As CFM becomes independent from underlying learning method, with the necessary adjustments, a Sarsa(λ) agent can be equipped with the proposed memory as well. Therefore, the setting including the comparison of CFM over Sarsa(λ) and Sarsa(λ) method itself is available in the experimentation.

To avoid redundant repetition, `mhn` domain is selected among the relatively small domains within this setting. Similarly, the methods are compared in `2 rooms` problem as one of the larger domains.

In each domain, 50 experiments are executed for 1000 episodes. For every experiment, the agent is placed some random starting state (in the left room for the `2 rooms` domain) other than the goal state and expected to find its path to goal state. For each method, decaying ϵ -greedy learning strategy is applied, with initial value of 0.2 down to 0. The discount factor γ and λ are chosen as 0.9 for all methods in both domains. The threshold and period as the CFM specific parameters are set to 0.1 and 20 respectively. Finally the memory length of CFM is chosen as 1 for each domain.

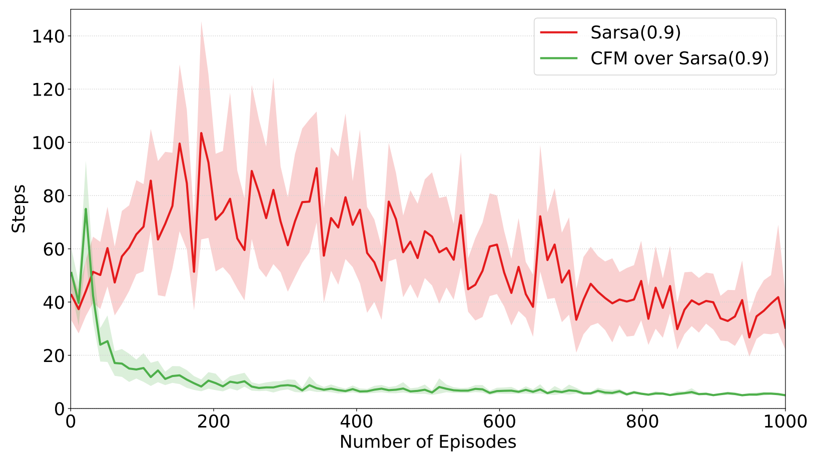


Figure 3.7: Learning performances of Sarsa(λ) agent with and without CFM in `mhn` in terms of number of steps to the goal

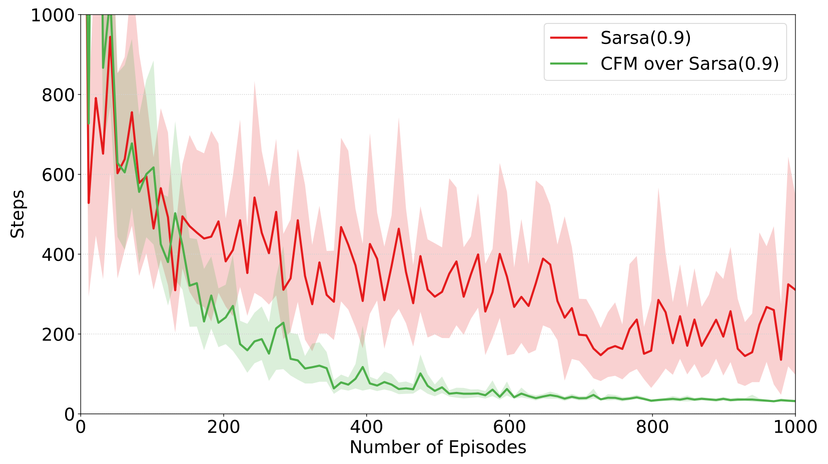


Figure 3.8: Learning performances of Sarsa(λ) agent with and without CFM in `2 rooms` in terms of number of steps to the goal

Figure 3.7 and Figure 3.8 depict the learning performances of Sarsa(λ) and CFM over Sarsa(λ) methods in terms of the number of steps to goal in `mhn` and `2 rooms` domains respectively. Sarsa(λ) clearly suffers from the perceptual aliasing in `mhn` domain without an external memory. Although it performs better in `2 rooms` domain, CFM still outperforms the method itself.

3.4 Summary and Discussion

In this chapter, a selective memory based on transition frequencies is proposed. With this memory, a smaller estimated state space is obtained and hence, faster solution for the learning agent becomes available. Although it may not always construct the optimal solution, by building a near optimal solution with much less memory in a much faster time it outperforms existing memory-based methods.

Moreover, since it projects the perception of the agent an estimated space where the number of estimated states is much less than the other memory-based methods and each of these estimated states is more compact, CFM is more scalable than them as shown in the experiments.

Finally, CFM does not assume any underlying learning methods and therefore, any RL agent can be equipped with this type of memory. With the necessary adjustment it is extended to work on Sarsa(λ) method as well. Thus, its independence from the base RL method is empirically proven.

CHAPTER 4

USING CHAINS OF BOTTLENECK TRANSITIONS TO DECOMPOSE AND SOLVE PARTIALLY OBSERVABLE REINFORCEMENT LEARNING PROBLEMS

In the previous chapter, we focused on the frequencies of the transitions by themselves, and building a memory for the learning agent upon them. As the second part of the study, beginning from this chapter, we will get benefit from the associations among the unique transitions which can provide a more abstract view of the given partially observable RL problems for the agent. Hence, these transitions can be useful in the decomposition of the related problems.

Instead of diving into the complete picture of the fully automated approach, we will follow an incremental examination of the overall method. Doing so, we now focus on how a RL problem with hidden states can be decomposed via the bottleneck transitions on unaliased states, *a.k.a, landmark states*, and how a further enhancement can be obtained by knowing the correct order of these transitions, as *chains* to follow in the construction of a practical solution like connecting the dots.

4.1 A Motivating Example for a Decomposition Based on Bottleneck Transitions

A complex task is usually composed of a series of actions, a subset of which plays a significant role to achieve the goal if executed in a certain sequence. For instance, the instructions in the user manual of an electrical device are given in an ordered manner. Similarly, while giving directions to a person who is unfamiliar with the neighborhood, each critical action in the route is given in a sequence.

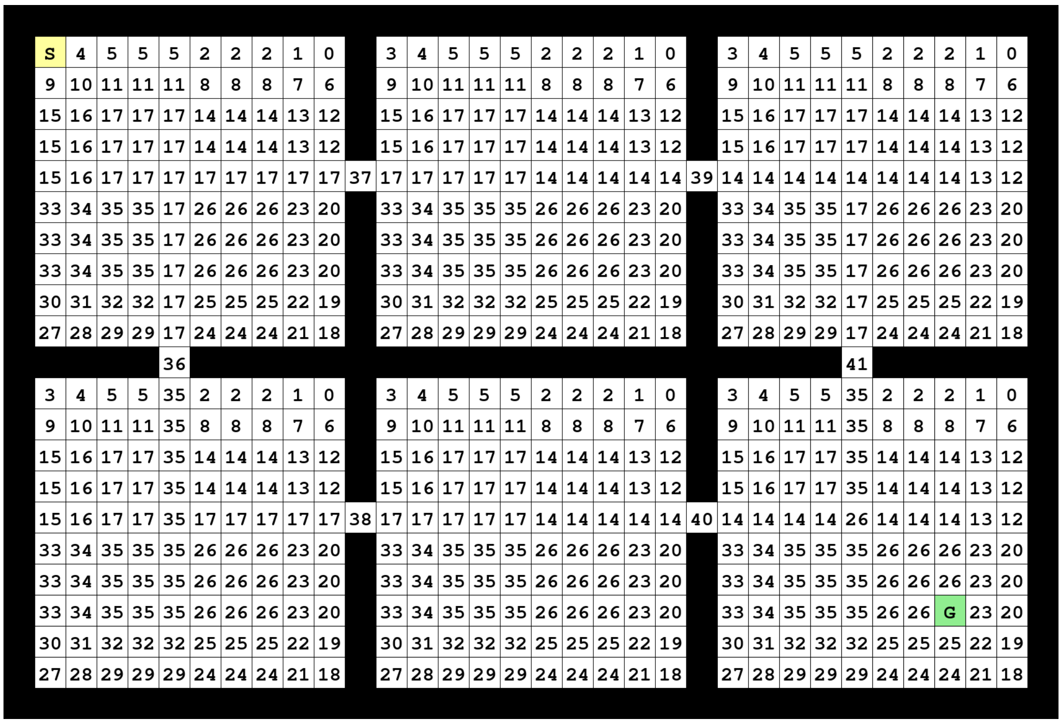


Figure 4.1: Observation enumeration in `6 rooms`$_{FS}$ domain where the doors between rooms are landmarks. Observation enumerations are provided for each state. Note that doors (landmarks) yield unique observations.

With this in mind, consider the partially observable version of `6 rooms`$_{FS}$ domain [11] with the observation enumeration given in Figure 4.1. The agent is located at the northwest corner (`S`) at the beginning of each episode and trained to reach the state in southeast room denoted with the letter `G`, which is the main task of the agent for this problem. The doors are landmarks of this domain. In other words, they are assumed to yield unique observations. The observation of any other state depends on the four categorized distances to the walls in four directions as explained in Section 2.3.2.

Landmark states can be useful for the abstraction of the main task in the same problem. Figure 4.2 shows the domain where these landmark states are represented with famous monuments (The Statue of Liberty, London Eye, Tower of Pisa, Acropolis of Athens, Colosseum, and Eiffel Tower). The example will focus on this second figure, in order to ease the understanding of this abstraction.

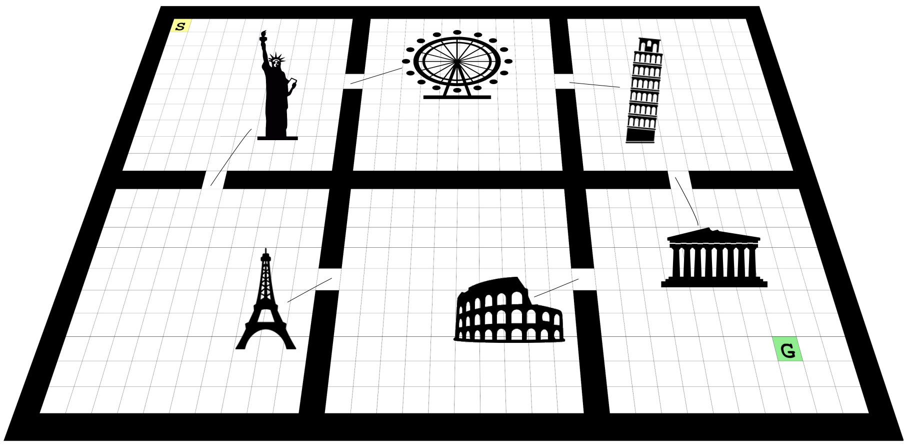


Figure 4.2: 6 rooms$_{FS}$ domain of Figure 4.1 where the landmarks in the domain are depicted as famous monuments

If a person is asked to reach the goal state here, the solution would be based on the chain of actions that are taken on the monuments. For instance, *"go south from The Statue of Liberty"*, then *"go east from Eiffel Tower"* and finally *"go east from Colosseum"*. Alternatively, a chain could be first *"go east from London Eye"*, second *"go east from Tower of Pisa"* and finally *"go south from Acropolis of Athens"*.

Figure 4.3 shows the further abstraction on 6 rooms$_{FS}$ using the landmarks. After the problem is reduced to the bottlenecks in the domain, then it is much easier for the agent to draw a path to the solution in the interaction graph of the problem.

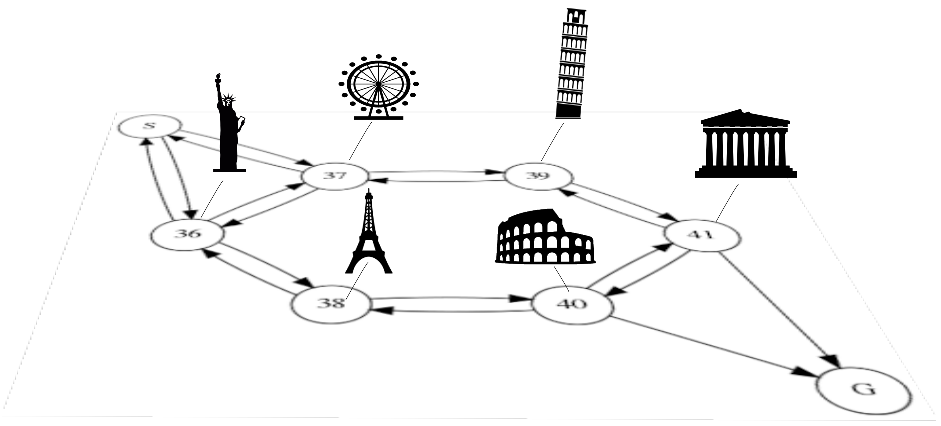


Figure 4.3: Abstract view of 6 rooms$_{FS}$ domain using landmarks

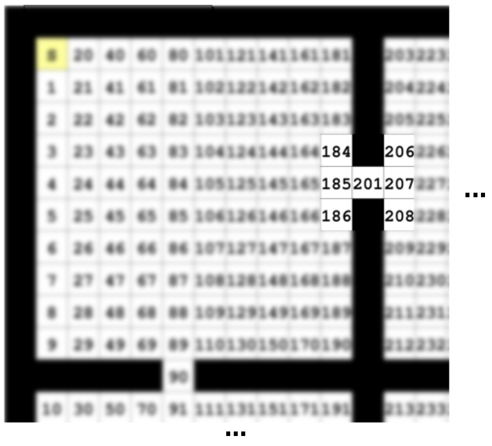
The motivation behind this study is to train the learning agent with a similar capability of abstract thinking for similar ambiguous problems. In this way, the agent becomes capable of decomposing the problem so that the ambiguity of similar states among sub-problems could be reduced.

4.2 Landmark-POMDPs and the Chain of Bottleneck Transitions

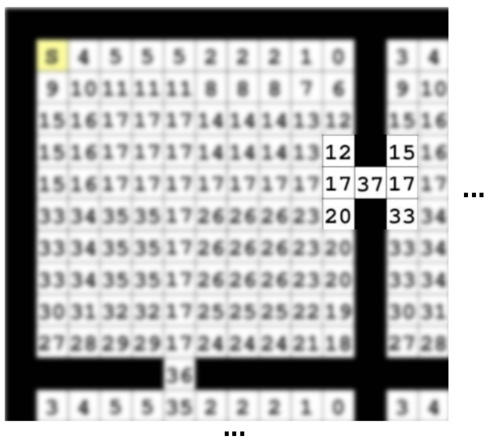
Landmark-POMDP (L-POMDP) is a POMDP model with one or more landmark states, where a *landmark* is defined as a state yielding a unique observation in the environment [18, 19]. That is, the probability of having the same observation mapping with a landmark state is 0 for any other state in the state set as explained in Section 2.3.3. All of the sample domains used in this chapter comply with this formulation.

Although the information provided by a landmark state is reliable and distinctive for the agent while it resides in it, a decomposition of the task, based solely on landmarks, may not be as reliable within an ambiguous environment. The justification of this claim is depicted in Figure 4.4 focusing on the critical states near the door between the northwest and north rooms for `6 rooms`$_{FS}$ domain introduced in Section 4.1.

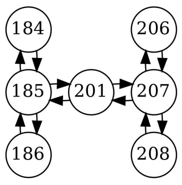
As can be identified by the enumerated states nearby the door in Figure 4.4a and the corresponding interaction graph of Figure 4.4c, the landmark state `201`, representing the door between the two rooms, is clearly a bottleneck which acts as a bridge between different regions of the state space. However, this is not the case if the state space is hidden from the agent. As shown in Figure 4.4b, the agent receives the same observation `17` before and after passing through the door. Furthermore, when other states yield the same observation as the states around the door (notice the observations at the states in the west and east border of the domain in Figure 4.1), interactions based on observations may become more complicated as shown in Figure 4.4d. Nevertheless, the agent in this ambiguous environment fails to perceive the significance of the landmark state for passing through between these rooms.



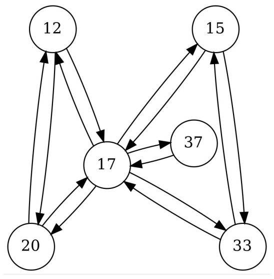
(a)



(b)



(c)



(d)

Figure 4.4: States (a) and observations (b) of the `6 rooms`$_{FS}$ domain with their corresponding interaction graphs (c and d, respectively) focused on the region near the northwest door

Therefore, while a bottleneck state can be useful in partitioning the state transition graph, the corresponding observation is usually insufficient to split the observation transition graph of the same problem. Within the learning agent's limited sensor capability, the easiest way to decompose a partially observable environment is by including the action executed by the agent upon a bottleneck observation.

Following the above informal introduction on why the actions are included to define bottlenecks between different state regions in a partially observable environment, they can now be defined in a more formal way:

Definition 4.1 *A state region $\mathcal{R}$ is a strongly connected sub-graph of the global state-interaction graph excluding the landmark states in the common borders with another state region where no direct transition is possible.*

Each room in the `6 rooms`$_{FS}$ example can be seen as a region. By means of the regions in the domain, the bottleneck transitions can be defined as below:

Definition 4.2 *The set of bottleneck transitions (BTs) is defined as*

$$\mathcal{B} = \{\ b = (l_t, a_t) \mid l_t \in L,\ \mathcal{R}_{t-1} \neq \mathcal{R}_{t+1}\ \},$$

where l_t is a landmark state that the agent resides at time t, L is the set of landmarks, a_t is the action taken by the agent at time t, and $\mathcal{R}_{t-1}$ and $\mathcal{R}_{t+1}$ are the state regions in which the agent is located at times $t-1$ and $t+1$ respectively, namely right before and immediately after visiting the landmark state l_t.

Note that the changing over the state region via a bottleneck transition is intentionally emphasized in this definition. In other words, a random action in a random landmark state resulting in no change on the current state region should not be regarded as a bottleneck transition.

Another important feature of such transitions is that since the landmark states remain distinguishable by the agent under partial observability, the transitions initiated on them could still be useful for the construction of a solution. In the scope of this study, such transitions are used to create a chain to guide the agent with indirect information about its own whereabouts on the way to the goal.

Definition 4.3 *Let a chain of bottleneck transitions,*

$$c = ((l_s, a_s), b_1, ..., b_n, g)$$

be the sequence of bottleneck transitions following a starting landmark state and leading the agent to a goal state. Then the set of all possible chains in the problem domain can be represented by $\mathcal{C}$.

The agent starts interacting with its first state region by taking the action a_s from its starting landmark state l_s, and it leaves its final region by reaching the goal state g. Therefore, the leading transition from the initial state and reaching the goal state are also bottleneck transitions by definition. Moreover, the chains of bottleneck transitions can be classified by their resulting goal states. In this way, the agent may use them more wisely in an environment with multiple goal states.

Knowing the bottleneck transitions implies the state regions in the problem for the agent. Therefore, it becomes trivial to build a decomposition based on these transitions as we focus in the next section. Furthermore, since each state region can be considered as a sub-problem with less aliased states for the main task, the construction of a solution also becomes trivial where traditional methods suffer or fail completely.

4.3 Decomposing a Partially Observable RL Problem using Chains of Bottleneck Transitions

Like every decomposable task in any context, there are some significant transitions for a learning agent in an ambiguous domain which lead to the main solution if they are performed in the correct order. These transitions can be passing through a physical bottleneck like a door in a navigational problem or visiting a certain state to have the key in an escape room problem. The agent can be provided an artificial reward to reach these transitions. However, the selection of the correct values for such transitions is a difficult problem of its own. If the artificial rewards are not chosen properly, this approach can be misleading for the agent by creating an overvalued state and result in an unsuccessful policy.

Considering the properties of the problem mentioned above, a solution that neither requires much effort for the problem designer nor harms the main policy of the learning agent is needed. Based on this necessity, separate training of the sub-agents between bottleneck transitions is proposed to solve each problem part *independently*. Then, the learning agent is only required to employ these *sub-policies* in an abstract layer to compose the main strategy to solve the whole problem without wasting any effort on primitive actions.

Using sub-policies for the solution of the problem can be related to the options framework. However, when the state space information is hidden from the learning agent, using the options extracted during learning ought to fail as shown in [20], if any adjustment in initiation or termination conditions is not applied.

In such an attempt, perceptual aliasing would prevent the agent to learn how to construct and make use of the options in the correct order. In this study, we propose the *Chains of Bottleneck Transitions (CBT)* method having very strict initial and termination conditions, that are based on the early-extracted chains, are applied to decide which sub-policy should be next in a bottleneck state.

CBT approach differs from the options framework in that options are formally applicable only to fully observable RL problems. Although a tree based option derivation mechanism exists for POMDP with hidden states, it still lacks the convergence guarantee of options framework based on semi-MDP formalism [32]. Moreover, by using the chains leading to a solution, the agent does not have to deal with filtering useful sub-policies among the all sub-policies available. That's why the CBT method proposed in this paper focuses on the direct decomposition of state space by the ordered bottleneck transitions, instead of applying the options framework in a restricted manner. In addition to that, since the state space is divided into smaller and independent sub-problems, CBT are able to speed up the learning process by training the sub-agents simultaneously.

4.3.1 Training the Sub-Agents

Once a chain of bottleneck transitions is provided for the learning agent, this information can be used to train the sub-agents within the borders indicated by the bottleneck transitions. For instance, consider the two chains constructed by transitions given in Figure 4.5. The first chain is:

$$(S, south) \Rightarrow (o_{36}, south) \Rightarrow (o_{38}, east) \Rightarrow (o_{40}, east) \Rightarrow G,$$

and the second one is:

$$(S, east) \Rightarrow (o_{37}, east) \Rightarrow (o_{39}, east) \Rightarrow (o_{41}, south) \Rightarrow G.$$

There are four sub-tasks to follow in each chain. For example, for the first chain, learn to move from S to $(o_{36}, south)$, then from $(o_{36}, south)$ to $(o_{38}, east)$, so on. Therefore, eight sub-agents in total can be trained to extract sub-policies to achieve each sub-task. Assuming that no collision occurs in the environment, multi-agent simulations can be done for the training of the sub-agents for the sake of time efficiency.

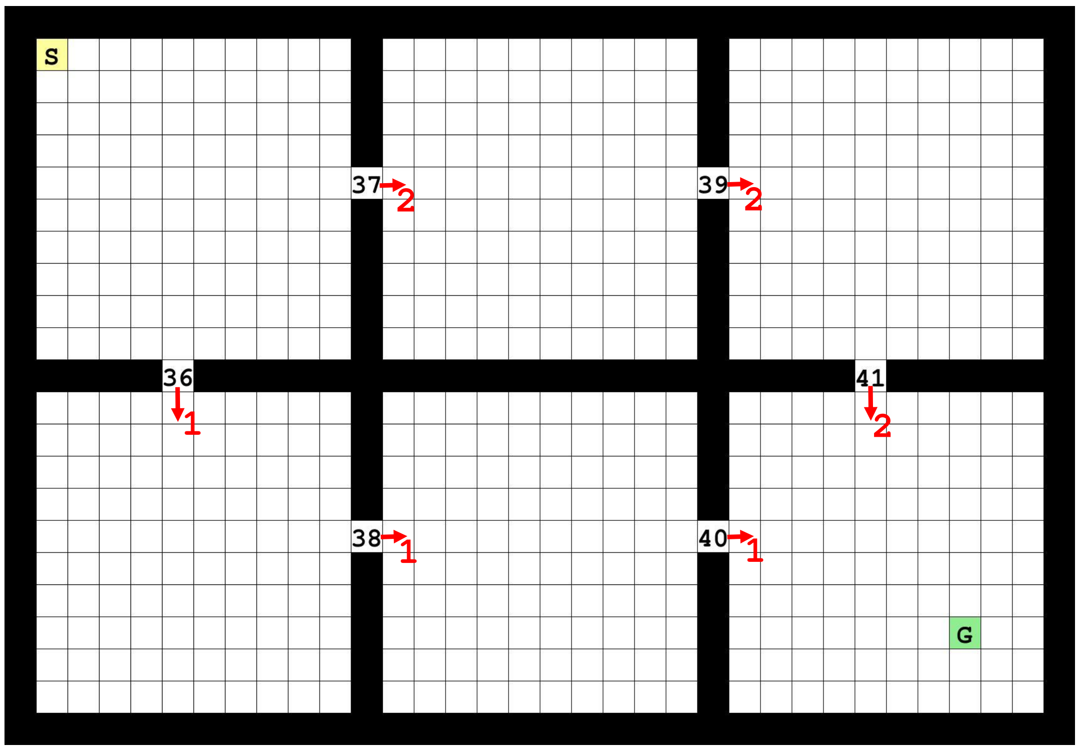


Figure 4.5: Useful bottleneck transitions in `6 rooms`$_{FS}$ domain

In the first sample chain, a sub-agent starts each episode in the starting state of the main learning agent and its goal is to reach the landmark enumerated as observation `36`. The second sub-agent starts each episode by going *south* from the same landmark state and tries to reach the landmark observation `38`. The rest of the chain can be completed with the other two agents which are simultaneously trained in a similar fashion. Each sub-agent is forced to execute its bottleneck transition when it is in its initial state. Additionally, if it encounters a state which is another sub-agent's initial state, regarding this state as a pitfall, the episode is terminated. In this way, the training of each sub-agent can be restricted to its own region.

This setting can be achieved by parameterizing the original domain to have different start, goal, and terminating pitfall states. Other than this parameterization, nothing is changed in the original environment, and the sub-agents are capable of sensing only observations during the interaction with the domain as it is also the case for the main learning agent. Namely, sub-problems are still partially observable. Hence, this can be thought of as creating multiple copies of the original problem where each sub-agent can be simultaneously trained within its region.

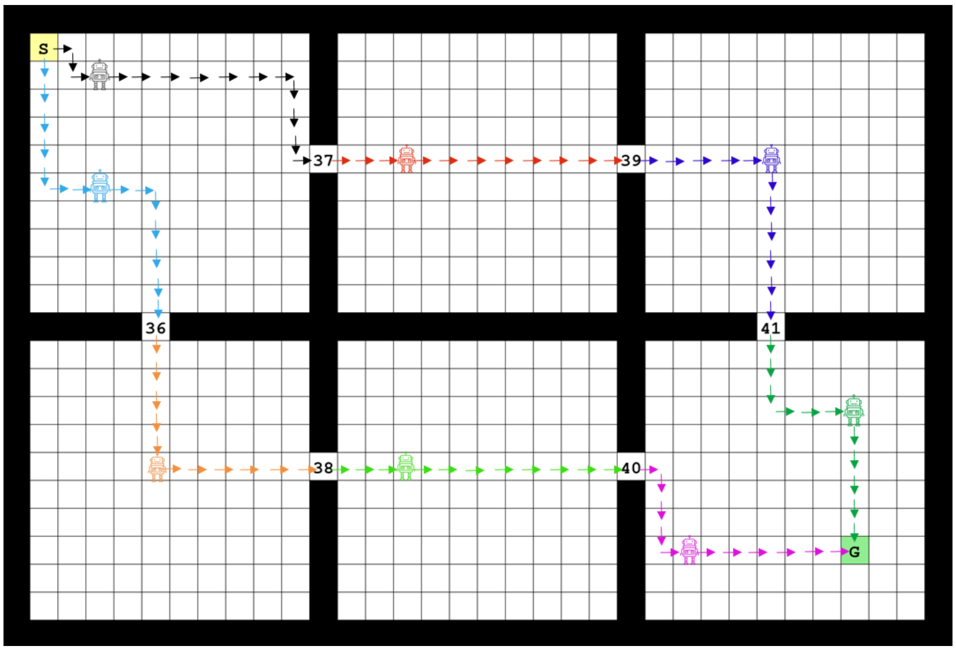


Figure 4.6: Sample sub-policies in `6 rooms`$_{FS}$ domain

Although it is the same environment that the sub-agents are dealing with, each sub-problem is much less ambiguous compared to overall problem, since each sub-agent is forced to remain within its region. This is why it becomes trivial to build sub-policies and main solution from these sub-policies as shown in 4.6.

As shown in the example above, there can be multiple ways to build a solution from sub-policies. At this point, the main agent must be capable of evaluating these sub-policies to choose optimal ones.

4.3.2 Learning the Values of Sub-Policies

After training the sub-agents and passing the sub-policies to the main learning agent, the next step is the evaluation of the sub-policies. Since the problem may include multiple chains of bottleneck transitions, this step ensures that the agent learns the optimal sub-policies indicated by these chains. Moreover, the problem can have different rewarding states each of which requires a different chain to be followed. Hence, a mechanism for the calculation of each sub-policy's value is crucial for the generalization of the method as well.

To calculate the value of the sub-policy π_s, the following update rule is applied when the episode ends or the agent switches to the next sub-policy:

$$\mathcal{V}(\pi_s) = \mathcal{V}(\pi_s) + \alpha \times (\frac{\Re_s + \max_{\pi_{s'}} \mathcal{V}(\pi_{s'})}{\tau_s} - \mathcal{V}(\pi_s)), \tag{4.1}$$

where $\mathcal{V}(\pi_s)$ is the value of the sub-policy, α is the learning rate, $\Re_s$ and τ_s are the total reward gathered by the agent, and the total number of timesteps required for the execution of this sub-policy, respectively. Finally, each $\pi_{s'}$ is a sub-policy that the agent may follow after the termination of sub-policy π_s. As might be noticed, this formula is quite similar to the standard update rule of Q-Learning, except that it requires a cumulative reward the agent gets and a normalization based on the timestep the agent needs while a sub-policy is carried out.

With the cumulative reward and normalization, the formula can calibrate the evaluation mechanism by using the information coming from primitive actions in the environment. For instance, the total reward gathered in an episode can differ by following two different sub-policies which lead to the same terminating state. Obviously, the one with greater total reward should be promoted.

As another example, there can be fewer bottleneck transitions in a chain than another one. However, if the agent has to take more steps by following the sub-policies indicated by this chain, it would not be proper to encourage the agent to choose them. Therefore, both cumulative reward and normalization according to the number of steps in a sub-policy are essential to fairly evaluate all sub-policies.

4.4 Experiments

Experiments are carried out in three different domains to evaluate the learning performance of CBT with the provided chains. These domains are selected as the larger domains, namely `6 rooms`$_{FS}$, `zigzag rooms`, and `6 rooms with locked shortcut` where natural sequences with different characteristics exist (see Section 2.5.).

Table 4.1: The methods used in the experimentation of the decomposition of problems based on bottleneck transitions

Name in legend	**Explanation**
Chains Provided over Q	The method learns from the sub-policies extracted from given chains over sub-agents trained with Q
Chains Provided over Sarsa(λ)	The method learns from the sub-policies extracted from given chains over sub-agents trained with Sarsa(λ)
All BTs over Q	The method learns from all possible sub-policies between bottlenecks over sub-agents trained with Q
All BTs over Sarsa(λ)	The method learns from all possible sub-policies between bottlenecks over sub-agents trained with Sarsa(λ)

In order to assess the contribution of the bottleneck transition chains, beside the decomposition based on the transitions by themselves, another main agent is trained according to all possible bottleneck transitions. This brute-force approach uses the same sub-policy evaluation mechanism as the CBT method. However, it is fed with all possible transitions between state regions. For example, the sub-policy which brings the agent back to observation `36` from observation `37` in `zigzag rooms` domain is also produced for the brute-force version. This section of the experimentation aims to assess the improvement gained by introducing the chains of bottleneck transitions compared to decomposing the task by all possible bottleneck transitions. Also, for a fair comparison, the Q-Learning agents are trained to achieve the main tasks in the fully observable versions of the domains. As CBT is built to be independent from the underlying learning method, sub-agents are trained with both Q-learning and Sarsa(λ). Names of the configurations used in the experimentation are given in Table 4.1.

4.4.1 Settings and Results

The environment feedback yields two reward values for the domains: 1 upon reaching the goal and 0 for any other steps. The same setting is used in the training of the sub-agents by adjusting their rewarding state according to their sub-task. The initial landmarks of other sub-agents are regarded as pitfalls without introducing any extra punishment.

In each domain, 20 experiments were executed with 5000 episodes for each. Each episode is limited to 5000 steps for all domains. Discount factor (γ) is set to 0.9. In both training of the sub-agents and sub-policy evaluations, the learning rate (α) is chosen as 0.05. In both phases, $\epsilon - greedy$ approach is applied with the ϵ value decaying linearly after each episode from 0.9 down to 0.1. Finally, in the training of the Sarsa(λ) sub-agents, λ is set to 0.9.

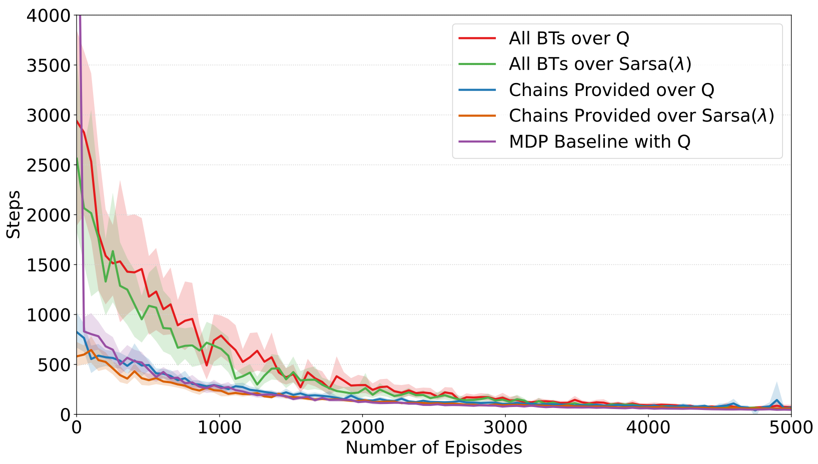


Figure 4.7: The learning performances of the methods in `6 rooms`$_{FS}$ problem in terms of the steps needed to reach the goal

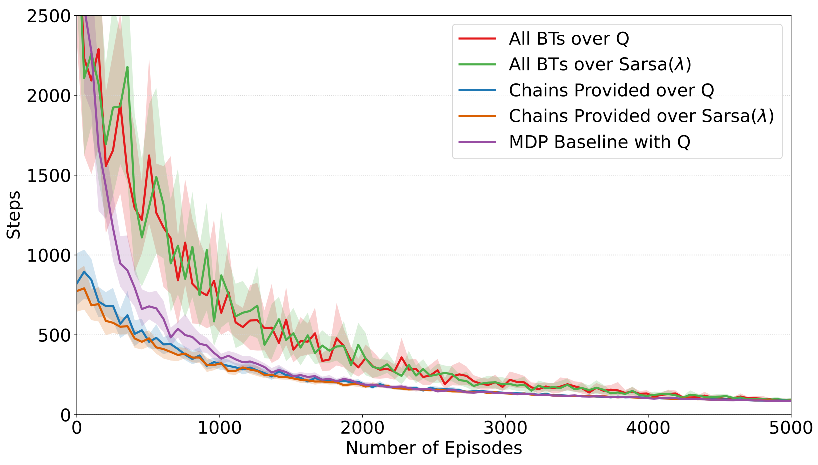


Figure 4.8: The learning performances of the methods in `zigzag rooms` problem in terms of the steps needed to reach the goal

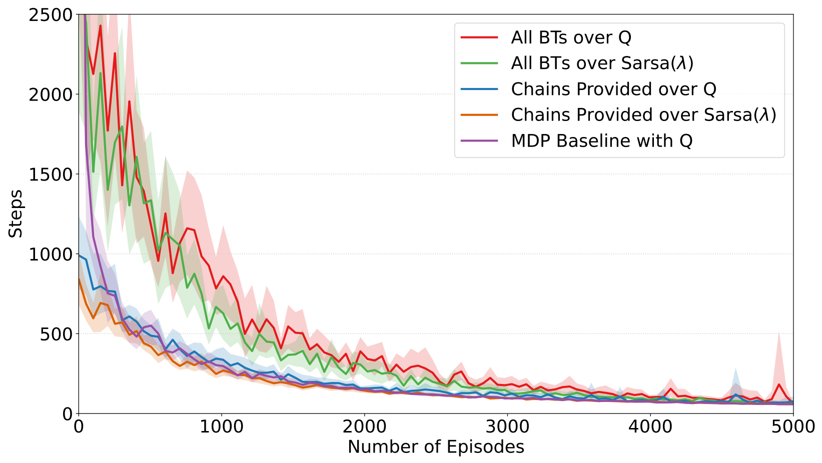


Figure 4.9: The learning performances of the methods in `6 rooms with locked shortcut` problem in terms of the steps needed to reach the goal

Figure 4.7-4.9 depict the learning performances of the methods in terms of the number of steps to reach the goal in the domains. It is clear that the decomposition based on bottleneck transitions make the problem is easily solvable. As can be seen in Figure 4.8, a proper decomposition of a POMDP problem may lead to faster construction of a solution, even compared to the fully observable version of the problem. Moreover, for all of the domains, knowing the correct order of bottleneck transitions with the constructed chains clearly outperforms an exhaustive search among all possible sub-policies. It is obvious that the number of possible sub-policies which the agent has in a landmark state has a direct effect on these performances and the number of distinct sub-policies extracted by these methods are given in Table 4.2.

Table 4.2: The number of distinct sub-policies extracted by the methods in larger domains

	Chains Provided	All BTs
`6 rooms`$_{FS}$	8	18
`zigzag rooms`	4	7
`6 rooms with locked shortcut`	16	31

4.5 Summary and Discussion

This chapter covers the origin of the CBT method as the decomposition of the problem based on bottleneck transitions. Before achieving the automatic construction of the bottleneck transitions, it is empirically proven that with such a decomposition, the problem is divided into smaller sub-problems that have less ambiguous states than the original. In this way, challenging partially observable problems turn out to be easier to solve. Even the tasks, for which traditional methods fail to succeed, can be accomplished by this divide-and-conquer based approach.

Beside the benefit coming from the decomposition of the problem, the knowledge about the correct order clearly gives the main agent an advantage in the problems as shown in the experimentation. The results ensure that if the agent is equipped with the correct order of the bottleneck transitions, the solution can be constructed in the earlier stages of the learning process without any need for additional external memory. Hence, we are ready for the next phase of the CBT method which is the self construction of the bottleneck transition chains as covered in the following chapter.

CHAPTER 5

SELF-CONSTRUCTION OF BOTTLENECK TRANSITION CHAINS WITH GIVEN LANDMARKS

We observed that the decomposition based on bottleneck transitions are quite useful to bring the solution for the learning agent and the chains are like the shortcuts in this decomposition. Therefore, it is time to take the next step in the automation of the overall process.

In this chapter, we propose a method that leads the learning agent to construct the chains of bottleneck transitions automatically. The method requires that the landmarks in the problem are provided beforehand, filters the history of the agent based on the given landmarks and then recursively looks for the patterns that can be identified as chains.

5.1 Motivation behind the Proposed Method

Looking for the sequences of bottleneck transitions can be examined under the field of sequential association rule mining. However, our problem differs from a regular sequential association. We look for specific patterns that leads the agent to the solution in the problem. Yet, for most of the partially observable problem, the agent construct the solution too late, even if it can. Hence, considering the associations among all transitions with a rule mining method would be like looking for a needle in a haystack. What we need at this point is the classification of the agent's experience based on the nature of a partially RL problem.

Considering that we apply episodic learning for the agent, we have an initial and a terminal state for each episode. Therefore, we can use this property for the further categorization of the agent's filtered history, instead of performing a blind search. In this way, we also do not have to deal with the patterns with smaller lengths that are already included in the main sequences. There are also longer patterns which include detours instead of directly following the main sequences. It is not possible to eliminate them completely without making an exhaustive search, however, we observed that the usage of last seen landmarks before termination can provide a noise elimination in the classification of the agent's experience. Thus, for all these practical reasons, we come up with the idea of recursive and greedy search starting from the landmarks that are last seen before the termination of an episode until reaching an initial landmark.

5.2 Construction of Bottleneck Transition Chains

Algorithm 2 is given as the pseudocode for the automatic construction of bottleneck transition in detail. Firstly, we need our agent to experience all possible bottleneck transitions for sufficient amount of times. Therefore, the self-construction of bottleneck transition chains is achieved with random traverse in the domain. While the learning agent acts randomly in the given problem, it records each of the transitions that occurs from a landmark state. For each episodic history, the landmarks that are firstly and lastly seen are saved to be used in the classification of the sequences.

If the episode does not terminate in a goal state for the agent, then the bottleneck transition history is dumped because the associations in a unsuccessful episode would not be useful for the agent. For the successful case, starting and ending landmarks in the sequences are added to sets of firstly and lastly seen landmarks respectively. Then the consecutive occurrences of the landmark observations and their transitions are included in the calculation of the associations.

Algorithm 2 Construction of Chains

```
 1: procedure CONSTRUCT_CHAINS
 2:     require: $L$                                              ▷ set of landmarks
 3:     $FSL \leftarrow \emptyset$                                ▷ first seen landmarks in success
 4:     $LSL \leftarrow \emptyset$                                ▷ last seen landmarks in success
 5:     Initialize $BOA$ as 0 for all pairs                       ▷ bottleneck observation associations
 6:     Initialize $BTA$ as 0 for all pairs                       ▷ bottleneck transition associations
 7:     $\mathcal{C} \leftarrow \emptyset$                        ▷ set of chains
 8:     while in random walk do
 9:         $BTH \leftarrow \emptyset$                            ▷ episodic BT history
10:         while episode continues do
11:             take a random action $a_t$, observe $o_t$ and $r_t$
12:             if $o_t \in L$ then
13:                 add $(o_t, a_t)$ to $BTH$
14:             end if
15:         end while
16:         $l_f \leftarrow$ first landmark in $BTH$
17:         $l_l \leftarrow$ last landmark in $BTH$
18:         if episode is successful then
19:             $FSL \leftarrow FSL \cup \{l_f\}$
20:             $LSL \leftarrow LSL \cup \{l_l\}$
21:             for each transition $b_i$ in $BTH$ do
22:                 increment $BOA\ of\ (o_i, o_{i+1})$
23:                 increment $BTA\ of\ (b_i, b_{i+1})$
24:             end for
25:         end if
26:     end while
27:     for each $l_i$ in $LSL$ do
28:         $chain \leftarrow \emptyset$
29:         $chain \leftarrow \{l_i\} \cup chain$
30:         $current \leftarrow l_i$
31:         while $current$ not in $FSL$ do
32:             $current \leftarrow \arg\max_{(l,current)} BOA$
33:             $chain \leftarrow \{current\} \cup chain$
34:         end while
35:         $\mathcal{C} \leftarrow \mathcal{C} \cup \{chain\}$
36:     end for
37:     for each $chain$ in $\mathcal{C}$ do                    ▷ include actions
38:         for each $l_i$ in $chain$ do
39:             $a_i \leftarrow action\ a\ of\ \arg\max_{((l_i,a),(l_{i+1},a'))} BTA$
40:             $replace\ l_i\ with\ (l_i, a_i)$
41:         end for
42:     end for
43:     return $\mathcal{C}$
44: end procedure
```

When the agent have sufficient experience in the problem, chains are constructed based on the recursive search from lastly seen landmarks towards the first ones. Then for the most associated ones, the most selected actions are included to complete the chains. It may be argued that including the actions in the computation of associations from the beginning can provide an advantage for the agent to avoid an unnecessary traversal of the history and to save more time. However, if a bottleneck landmark state has two or more critical actions which may lead to the next landmark state, then capturing the actual association between these landmark states can be difficult. Furthermore, a traversal after the detection of the sequence based only on landmark states does not cause a significant increase in complexity, since it requires only one more operation on the recorded associations.

5.2.1 Complexity Analysis

The complexity of the construction method depends on the global landmark interaction graph of the given problem. Since the bottleneck transitions are saved immediately after each action (see lines 10-15 of Algorithm 2.), the time spent there becomes negligible. Also, saving the associations is done separately for each successful episode (see lines 16-23 in Algorithm 2) where each episode is assumed to terminate within a fixed number of steps. Therefore, this phase of the method takes a linear time with respect to the average episode length.

The main part of the construction of the chains is based on the number of landmarks that can be seen at the end of a successful episode. The same number also determines the number of chains to be constructed. Therefore, the worst-case complexity of the method for the next phase of the chain construction can be expressed by $|LSL| \times |L| \times |E|$, where $|LSL|$ is the cardinality of the set including last seen landmarks in the successful episodes, $|L|$ is the total number of the landmarks and $|E|$ is the number of edges in the landmark interaction graph. $|E|$ determines the time spent on processing the associations. In the final phase of the method, looking for the most frequently selected actions can be traced in linear time concerning the number of possible actions in the problem, by keeping the bottleneck transition associations in an indexed data structure.

In most of the problems with hidden states, the number of landmarks is very few compared to the number of all states. Furthermore, since most landmarks are bottlenecks for non-neighboring regions in the problems, the number of edges is far less than the $|L|^2 - 1$. For example, there are 44 different observations for 606 states in the `6 rooms`$_{FS}$ problem examined throughout the paper. 8 of these states are landmarks. Only 2 landmarks are included in the set of last seen landmarks and there are 18 edges in the global landmark interaction graph of this problem as shown Figure 5.1. The ideal chains have 8 transitions between landmarks as mentioned in Section 4.3.1. However, in the worst scenario, almost half of the 18 edges are eliminated since the association for the useful transitions is favored by looking for successful episodes only. Therefore, we obtain a smaller number of sub-policies extracted from the chains constructed. Hence, the number of sub-agents to be trained and the number of sub-policies to be evaluated are limited with this smaller number.

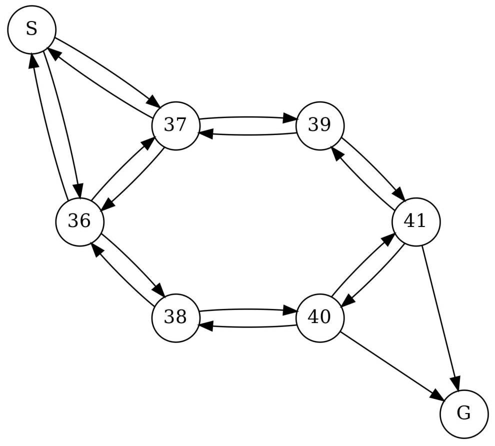


Figure 5.1: The global landmark interaction graph of `6 rooms`$_{FS}$ problem

5.3 Experiments

As the methods attack to the same or similar problem of our study are limited, the experiments were executed in different categories regarding the size and type of the domains. All the source code of the proposed method and the experimental settings for the sample domains are publicly available online.[1]

[1] `https://github.com/huseyinaydinmetu/cbt_codebase`

The first category includes the examination for the performance by comparison with a memory-based method, namely Utile Suffix Memory (USM) algorithm. This comparison is done by considering not only the quality of learning in terms of the number of steps to goal but also the time and memory required during the learning process.

In the second category of experiments, scalability of the CBT method is shown in the problems with larger state spaces. These problems cannot be solved with USM in a reasonable time. Therefore, it is intentionally excluded from this setting.

The final category includes the experiments on a well-known puzzle, namely the Tower of Hanoi problem. Although the agent still has the same perspective for the problem, which is provided by the observation transition graph based on its limited perception, this case enables us to observe the behavior of a CBT agent in a non-grid domain.

Despite all our efforts, the option-observation-initiation method proposed by Steckelmacher et al. [20] could not achieve the tasks without providing whole sub-policies by hand (for both cases with and without using LSTM [16]). Since it is not practical to do so for our problems, we excluded their method in our experiments on purpose.

The methods used in experimentation are given in Table 5.1. To ease reading, some details are not provided in the following subsections, but can be found in Appendix A instead.

Table 5.1: The methods used in the experimentation of CBT with self construction of the chains

Name in legend	**Explanation**
CBT over Q	Proposed method over sub-agents trained with Q-Learning
CBT over Sarsa(λ)	Proposed method over sub-agents trained with Sarsa(λ)
CBT over Q (All Land.)	Proposed method including non-bottleneck landmarks over sub-agents trained with Q-Learning
CBT over Sarsa(λ) (All Land.)	Proposed method including non-bottleneck landmarks over sub-agents trained with Sarsa(λ)
USM	Utile Suffix Memory by McCallum [1]
Chains Provided over Q	The method learns from the sub-policies extracted from given chains over sub-agents trained with Q
Chains Provided over Sarsa(λ)	The method learns from the sub-policies extracted from given chains over sub-agents trained with Sarsa(λ)

5.3.1 Settings and Results

5.3.1.1 Comparison with USM

The experimentation regarding the comparison of CBT with USM is held in two downscaled domains. First one is a version of `6 rooms`$_{FS}$ domain and the second domain is the `zigzag rooms` with the reduced size (see Section 2.5). In the `downscaled 6 rooms` domain, the agent is placed in the northwest corner of the domain and is trained to reach the southeast corner of the domain by switching its current room using the doors. The task is similar in the `downscaled zigzag rooms` except that the initial state is the southwest corner of the domain in this case. 20 experiments were executed with 200 episodes for each. An episode is terminated when the agent reaches the goal or 2000 steps are taken by the agent without reaching the goal.

In a regular state of these domains, the agent gets an observation based on the categorized distances in four directions as explained in Section 2.3.2. Finding a solution for `downscaled 6 rooms` domain might be trivial with this setting since there are no conflicting actions needed for the aliased states. However, if we consider the `downscaled zigzag rooms`, the agent is still required to move in opposite directions for the states in which the agent gets the same observation, although they reside in different rooms.

In both domains, reaching to the goal state is rewarded by 1. To speed up learning, the agent receives -0.01 upon moving to regular state, and -0.1 when it hits an obstacle. In the training of the sub-agents this setting is applied in the same manner, except their goals are different from the main goals in the domain. The initial landmarks of other sub-agents are regarded as pitfalls for a sub-agent, however, being in those states is not punished with a larger negative reward.

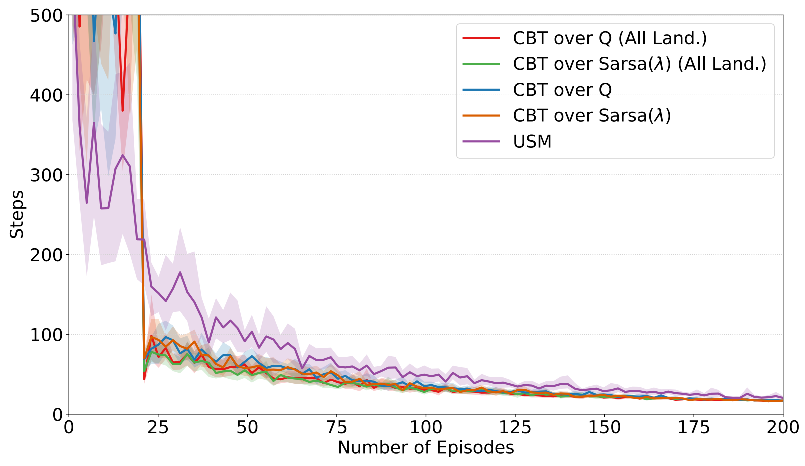
USM method is implemented by using the Kolmogorov-Smirnoff test to decide on the promotion of the fringe nodes. Its threshold is set as 0.01 for both of the domains. This test is applied after every 10 steps of the agent. The maximum depth for fringe nodes is set to 2 in both domains. In the original algorithm, USM checks every pair of leaf nodes to update the values after insertion of an instance. To reduce the overhead, the update mechanism is triggered with a probability of 0.1. This approach does not affect the convergence of the values, but it makes the algorithm much faster so that the experimentation can be completed within a reasonable time.

In the experimentation of both methods, ϵ-greedy approach is applied where the ϵ is linearly decayed with respect to the number of episodes that have been executed. The initial value for ϵ is 0.9 to encourage the exploration of the agents while the final value is 0.1.

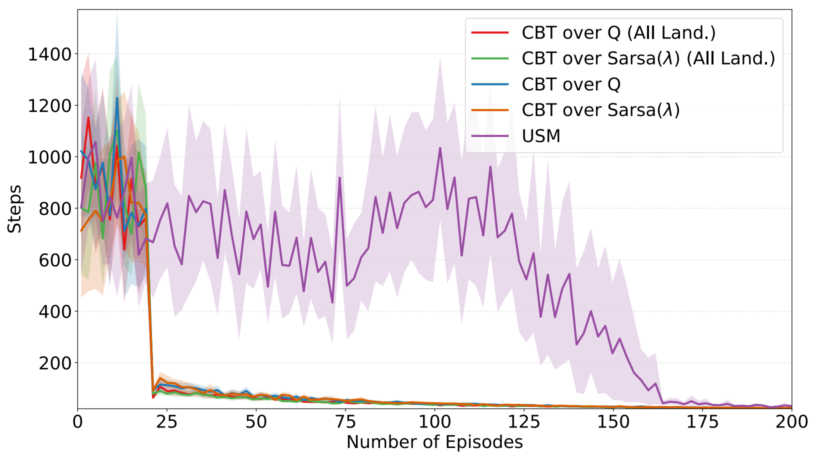
To show that the framework does not assume any underlying learning method, Q-Learning and Sarsa(λ) are interchangeably used in the training of sub-agents. Furthermore, since these domains contain landmark states which are not natural bottlenecks in the domain (For instance, see observation `4`, `9`, `13` and `16` in `downscaled 6 rooms` domain.) two different sets of landmarks are provided to CBT agents in the evaluation of its performance for these different cases. One of the sets includes only bottleneck landmarks that correspond to the door between rooms while the other set contains all of the landmarks. For all cases, the agent is forced to walk randomly for the first 20 episodes.

The construction of the chains is held after the random walk of the agent is completed. Then the sub-agents are trained within 200 episodes. The same linearly decaying ϵ-greedy approach is applied for the sub-agents as well as the main agent. The learning rate (α) is set to 0.05, and the discount factor (γ) is set to 0.9. For the cases with Sarsa(λ) sub-agents, the λ value is chosen as 0.9. During the evaluation of the sub-policies, the same α value is used as in the training of the sub-agents.

Figure 5.2 shows the learning performances of USM and the CBT method with its all configurations regarding the type of sub-agents and given sets of landmarks in the domains. The total number of steps to reach the goal for the methods are averaged over 20 experiments and the error intervals are represented in shaded regions in the plots.



(a) `downscaled 6 rooms`



(b) `downscaled zigzag rooms`

Figure 5.2: The learning performances of the methods in terms of the number of steps needed to reach the goal in downscaled domains

As can be seen in Figure 5.2a, although CBT takes more steps for a random walk in the first 20 episodes, USM learns slower than CBT in all cases for `downscaled 6 rooms` domain. Figure 5.2b depicts that the gap between the performances becomes more drastic when the ambiguity of the environment makes the whole problem more difficult to solve as in the case of `downscaled zigzag domains`. These results indicate that a method using a proper early decomposition of the problem can outperform a memory-based method that tries to grasp the structure of the problem during the overall learning process.

To illustrate the case for learning a sub-problem, Figure 5.3 shows the learning performances of sub-agents using Q-Learning and Sarsa(λ) for the CBT method. The sub-agents are placed into the starting state of the main agent and trained to reach the observation `16` (which corresponds to the first door) in the `downscaled zigzag rooms` domain. After the decomposition of the main problem, constructing a solution for each sub-problem becomes more trivial. Therefore, both of the methods learn the way to their sub-goals without being obligated to cope with aliased states of a separate part of the problem. This would not be the case if these traditional methods are used for a solution in the main problem with its aliased states.

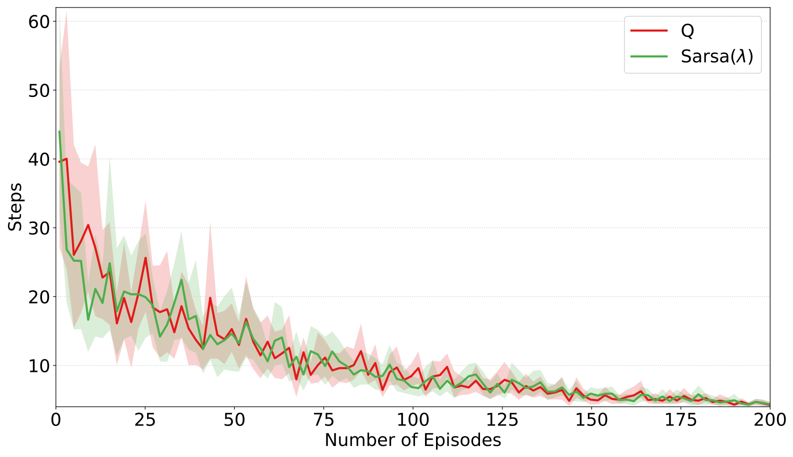


Figure 5.3: The learning performances of sub-agents trained between the starting state and observation `16` (the left-most door) in the `downscaled zigzag rooms` domain

Table 5.2: The average memory usage (MB) in an episode for the methods in downscaled domains

	`downscaled` `6 rooms`$_{FS}$	`downscaled` `zigzag rooms`
CBT over Q	0.422 ± 0.05	0.408 ± 0.056
CBT over Sarsa(λ)	0.46 ± 0.049	0.43 ± 0.052
CBT over Q (All Land.)	0.585 ± 0.081	0.653 ± 0.084
CBT over Sarsa(λ) (All Land.)	0.632 ± 0.086	0.686 ± 0.094
USM	**7.927 ± 0.627**	**31.603 ± 7.096**

Table 5.3: The average elapsed time (sec) in an episode for the methods in downscaled domains

	`downscaled` `6 rooms`$_{FS}$	`downscaled` `zigzag rooms`
CBT over Q	0.007 ± 0.001	0.009 ± 0.001
CBT over Sarsa(λ)	0.007 ± 0.001	0.009 ± 0.001
CBT over Q (All Land.)	0.007 ± 0.001	0.008 ± 0.001
CBT over Sarsa(λ) (All Land.)	0.007 ± 0.001	0.008 ± 0.001
USM	**37.794** ± **7.334**	**339.146** ± **228.191**

Table 5.2 and Table 5.3 show the average memory usage and elapsed time in an episode for the methods in both domains respectively. CBT, with its all configurations, requires significantly less time and memory while USM suffers, especially in the `downscaled zigzag rooms` domain.

Table 5.4: The average time (sec) spent in overall experiments for the methods in downscaled domains

		`downscaled` **`6 rooms`**$_{FS}$	**`downscaled`** **`zigzag rooms`**
CBT over Q	chain const.	0.000053 ± 0.000013	0.000027 ± 0.000003
	sub-agent train.	1.53 ± 0.38	1.01 ± 0.14
	total	13.86 ± 2.90	13.33 ± 1.90
CBT over Sarsa(λ)	chain const.	0.000053 ± 0.000008	0.000035 ± 0.000005
	sub-agent train.	3.16 ± 0.67	2.15 ± 0.44
	total	14.72 ± 2.68	16.55 ± 2.74
CBT over Q (All Land.)	chain const.	0.000073 ± 0.000016	0.000044 ± 0.000007
	sub-agent train.	1.32 ± 0.32	0.93 ± 0.11
	total	18.08 ± 3.54	23.40 ± 4.89
CBT over Sarsa(λ) (All Land.)	chain const.	0.000076 ± 0.000012	0.000042 ± 0.000007
	sub-agent train.	2.68 ± 0.63	1.78 ± 0.35
	total	19.69 ± 3.92	23.36 ± 4.64
USM	total	**7867.98** ± **1556.85**	**69406.98** ± **47171.52**

The execution times of the overall experimentation for the methods are shown in Table 5.4. The measurements include the time required by chain construction and sub-agent training for CBT agents. As can be seen in the table, chain construction time is insignificant. The main reason for that is saving the bottleneck transition history separately during episodic learning, then processing them only once at the end of random walk. The total time requirement of CBT including both the chain construction and the training of the sub-agents is still dramatically less than USM. This is the actual reason why USM is not very practical in larger domains, unlike CBT.

5.3.2 Larger Domains

The second group of experiments was carried out on three sample domains. The first one is the `6 rooms`$_{FS}$ domain [11] that we have followed in our examples in this chapter. The second one is the `zigzag rooms` domain [19], where the agent is subjected to a difficult task that requires actions taken in opposite directions to be able to reach the eastern room where the goal state is placed. Lastly the domain called `6 rooms with locked shortcut` is used in this experimentation setting (see Section 2.5.).

The environment feedback yields two reward values for the domains: 1 upon reaching the goal and 0 for any other steps. The same setting is used in the training of the sub-agents by adjusting their rewarding state according to their sub-task. Similar to the downscaled domains, the initial landmarks of other sub-agents are regarded as pitfalls without introducing any extra punishment.

In each domain, 20 experiments were executed with 5000 episodes for each. Each episode is limited to 5000 steps for all domains. Discount factor (γ) is set to 0.9. In both training of the sub-agents and sub-policy evaluations, the learning rate (α) is chosen as 0.05. In both phases, $\epsilon - greedy$ approach is applied with the ϵ value decaying linearly after each episode from 0.9 down to 0.1. Finally, in the training of the Sarsa(λ) sub-agents, λ is set to 0.9. In the CBT method, the main agents are forced to walk randomly for the first 100 episodes before constructing the chains.

Table 5.5: The number of distinct sub-policies extracted by the methods in larger domains

	CBT	Chains Provided
`6 rooms`$_{FS}$	11	8
`zigzag rooms`	5	4
`6 rooms with locked shortcut`	21	16

Traditional methods as Q-Learning and Sarsa(λ) by themselves fail to achieve the tasks in these domains. Moreover, USM is far from being practical in these domains as it cannot complete more than a few episodes in days. Therefore, Q, Sarsa(λ) and USM are excluded in this part of experimentation. That is why we used the method from Chapter 4 which uses the provided chains to extract sub-policies. The number of distinct sub-policies extracted by both self-constructed and provided chains is given in Table 5.5. This section of the experimentation aims to determine how the effect of self-constructed chains differs from the effect of the provided ones on the learning performance of the main agent. The Q-Learning agents are trained to achieve the main tasks in the fully observable versions of the domains as a baseline for a fair comparison.

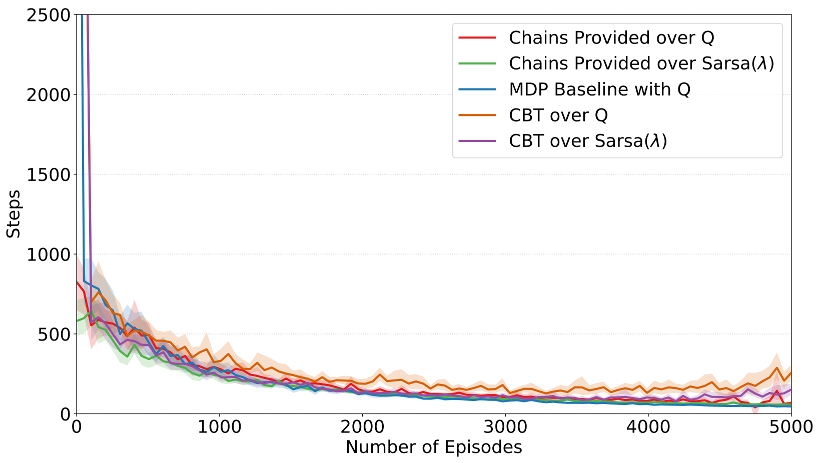


Figure 5.4: The learning performances of the methods in `6 rooms`$_{FS}$ problem in terms of the steps needed to reach the goal

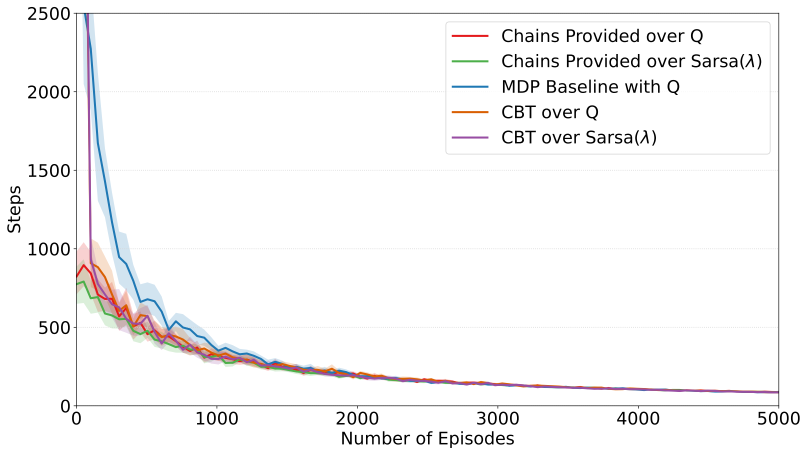


Figure 5.5: The learning performances of the methods in `zigzag rooms` problem in terms of the steps needed to reach the goal

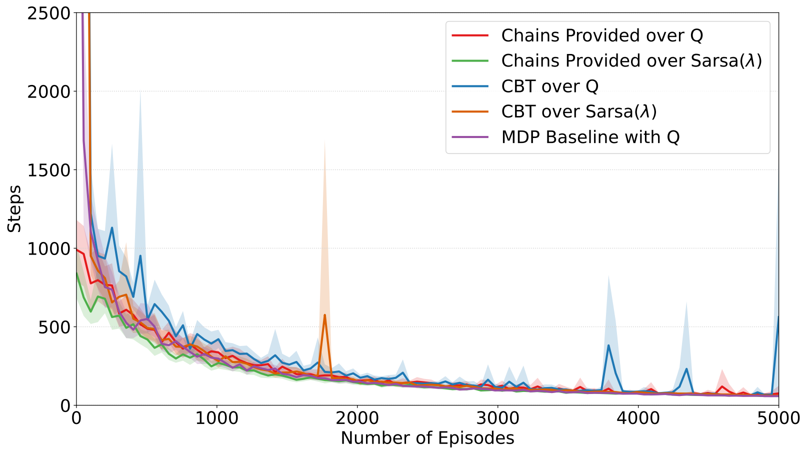


Figure 5.6: The learning performances of the methods in `6 rooms with locked shortcut` problem in terms of the steps needed to reach the goal

Figure 5.4-5.6 show the learning performances of the methods in terms of the steps to achieve the given task. As can be seen from these figures, CBT agent with self-constructed chains catches the main agent with the provided chains almost immediately after its random walk phase is over. Although self-constructed chains result in a slightly unsteady performance in `6 rooms with locked shortcut` domain, especially over Q-learning sub-agents, yet there is no significant difference in the overall performances of the methods.

5.3.3 A Non-Grid Case: Tower of Hanoi

As a non-grid case, we examined the performance of CBT in the well-known puzzle called `Tower of Hanoi` [28] in the last setting of our experimentation in this chapter. Details about the configuration of this puzzle in our settings and the observation semantic that presenting the perception capability of the RL agent for the domain can be found in Section 2.5.

The environment feedback uses two different reward values for the problem: 1 for reaching the goal and 0 for any other steps. The same setting is used in the training of the sub-agents by adjusting their rewarding state according to their sub-task. Like the previous experiments, the initial landmarks of other sub-agents are regarded as pitfalls without introducing any extra punishment.

20 experiments were carried out with 1000 episodes for each. If the agent fails to achieve the task in 750 steps, the episode is terminated anyway. Learning rate and discount factor are set to 0.05 and 0.9 respectively for all methods and all sub-agents. Similar to previous cases the action of the agent or a sub-agent of the CBT method is selected by a ϵ-greedy approach where ϵ value is linearly decayed after each episode from its initial value of 0.9 to 0.1. λ value is used as 0.9 for Sarsa(λ) agents. The main agents of CBT are forced to walk randomly for the first 100 episodes before constructing the chains like in the larger domains. The training of the sub-agents includes 1000 episodes in the same manner as the main agents.

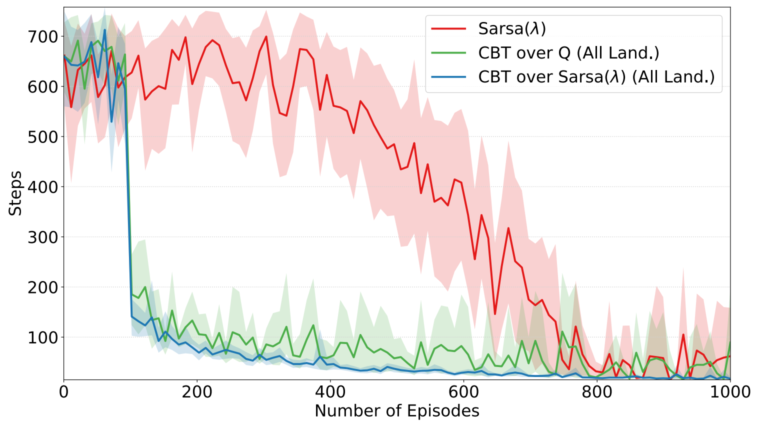


Figure 5.7: The learning performances of the methods in `Tower of Hanoi` problem in terms of the steps needed to reach the goal

Figure 5.7 shows the number of steps needed by the methods to reach the goal state from the initial state in the `Tower of Hanoi` problem. The Q-Learning fails to achieve the task by itself, hence its results are excluded on purpose. Sarsa(λ) method constructs a late solution for the problem while CBT quickly solves the problem by using one of these methods for the training of the sub-agents.

5.4 Summary and Discussion

In this chapter, self-construction of the bottleneck transition chains is achieved by a simple recursive method. With this method, the necessity of provided chains is eliminated for the CBT method and its learning agent. Experimentation shows that there is no significant difference between the learning performances when the agent has its own chains or chains that is provided by the problem designer.

As the automatic construction of the bottleneck transition chains with the provided landmarks is completed, it is time to examine the case where the learning agent is expected to identify the landmarks in the problem by itself. Hence, the next chapter covers the integration of CBT method with a Diverse Density based approach for the automatic landmark identification.

CHAPTER 6

AUTOMATIC LANDMARK IDENTIFICATION WITH DIVERSE DENSITY FOR BOTTLENECK TRANSITION CHAINS

In this chapter, as a candidate method to identify the landmarks in the given problem, Diverse Density (DD) [33] will be integrated to the CBT method. In this way, the overall process can be completed with the minimization of human interference. The idea of using DD in the context of RL is not novel as it used by McGovern et al. to identify subgoals in fully observable RL context [34]. However, as the original DD method has its own drawbacks, instead of using this version, its variant that is boosted with Concept Filtering (DDCF) [35] is chosen as the base method for the landmark identification phase. Therefore, this chapter starts with the summaries of DD and DDCF methods.

Although DDCF is more successful than the original DD method by finding landmarks with higher quality, it is still noisy as it falsely identifies non-landmark states as landmarks. Since the sub-agents of CBT method must be provided initial and goal states of sub-problems, any noise in identified landmarks cannot be tolerated by the proposed framework. Hence, any refinement on the results of landmark identification would be crucial in CBT's overall learning performance. As a result, the attempts to improve the identification process such as balancing the number of positive and negative bags in DD and normalization based on occurrences of observations are also covered in this chapter. Finally, the integration of all components for the proposed CBT framework is provided.

6.1 Diverse Density

Diverse Density (DD) is one of most common Multiple Instance Learning (MIL) methods [33]. As the name implies, in MIL problems, the instances are not individually labeled. Yet, they are labeled in two groups as positive and negative *bags*. A bag is called positive if it contains at least one positive instance (or *target concept*). Likewise, the sample groups with no positive instances are labeled as negative bags. Then DD value is calculated with the probability of c_t being the target concept given the positive and negative bags:

$$DD(c_t) = Pr(c_t|B_1^+, B_2^+, ..., B_m^+, B_1^-, B_2^-, ...B_n^-), \tag{6.1}$$

where B_i^+ and B_j^- are i^{th} positive bag and j^{th} negative bags respectively. Assuming that prior is uniform for each bag, and they are conditionally independent, DD value of a target concept can be expressed as below by the means of repeated application of Bayes' rule:

$$\prod_{i=1}^{m} Pr(c_t|B_i^+) \prod_{i=1}^{n} Pr(c_t|B_i^-). \tag{6.2}$$

Using `noisy-or` model is generally preferred in the calculation of the probabilities for each bag. In this model, they are defined as follows:

$$Pr(c_t|B_i^+) = 1 - \prod_{i=1}^{p} (1 - Pr(B_{ij}^+ \in c_t)), \text{ and} \tag{6.3}$$

$$Pr(c_t|B_i^-) = \prod_{i=1}^{p} (1 - Pr(B_{ij}^- \in c_t)), \tag{6.4}$$

where B_{ij} and p are the j^{th} instance in i^{th} bag B_i and the total number of instances in the given bag respectively. Lastly, the probability of B_{ij} being the correct concept, which is donated as $Pr(B_{ij} \in c_t)$ in the equations above, is obtained as Gaussian probability based on the distance between the particular instance and the target concept.

McGovern et al. identifies the target concepts with the highest DD values as the sub-goals in a fully observable RL problem [34]. Demir et al. takes a step forward in the application of DD on RL problems by introducing the concept filtering as explained in next section [35].

6.2 Diverse Density with Concept Filtering

The original DD approach uses the distances between the instances and the target concept. Providing these distances beforehand is not practical for most RL problems since it requires prior knowledge of problem for the learning agent especially while we are trying to construct a model-free approach to achieve the given task.

Another issue with the original DD method is that the instances which are far from being a bottleneck in the agent's solution are included in the calculation over and over again. To overcome this inefficiency along with the requirement for prior knowledge mentioned above, Demir et al. proposed Diverse Density with Concept Filtering (DDCF) [35]. DDCF builds a graph from the agent's own experience instead of using a prior graph at hand for the given RL problem. Then by measuring the concepts for being a bottleneck in this graph it eliminates the non-target concepts.

In the DDCF method, the graph is build with the nodes each of which is an observation that the agent has encountered. If the agent has a transition from a particular observation to another one, then the nodes of these two observations are connected in this graph. The concepts, as observations in this case, are filtered according to a graph metric called *Congestion Ratio (CR)* which is defined as:

$$CR^{(k)}(c) = \frac{BC(c)}{CC^{(k)}(c)} \, , \tag{6.5}$$

where $BC(c)$ and $CC^{(k)}(c)$ are the *Bridging Coefficient* and *k-Clustering Coefficient* values for the given concept respectively. These values are also graph metrics by themselves and defined as follows.

Bridging Coefficient is proposed by Hwang et.al to measure given node for the number of connections it brings to its immediate neighbors:

$$BC(v) = \frac{d^{-1}(c)}{\sum_{i \in N(c)} d^{-1}(i)} , \tag{6.6}$$

where $d(c)$ is the degree of node c and $N(c)$ denotes the set of immediate neighbors of this node [36].

As the other graph metric used in CR, *k-Clustering Coefficient* is proposed by Jiang et al. as an extension of Clustering Coefficient [37] by including not only immediate neighbors but also all neighbors up to depth k. This value is used to determine how well clustered the given node is, and calculated as below:

$$CC^{(k)}(c) = \frac{2 \times e^{(k)}}{n^{(k)} \times (n^{(k)} - 1)} , \tag{6.7}$$

where $e^{(k)}$ and $n^{(k)}$ represents the number of edges in k-neighborhood of node c and the number of nodes within this neighborhood [38].

DDCF recalculates CR for each concept for a new experience introducing a new node or a new edge to the graph. In other words, CR values are not calculated for each calculation of DD values redundantly. After CR values are obtained, local maxima in the constructed graph are kept in the calculations of DD values. The rest of the nodes are filtered out from this calculation as they are less likely to be a bottleneck in the agent's trajectory.

Since the constructed graph includes the distances for each pair of observations, DDCF eliminates the requirement for prior knowledge of transitions in the problem. Obviously this graph is not initially complete, however, by executing the DDCF after some episodes, the agent may have sufficient experience to build a complete graph.

6.3 Proposed Refinements on Identified Landmarks

Although DDCF eliminates some non-target concepts, identified landmarks with this method are still noisy especially including some non-landmark observations that are still aliased but not broadly seen in the problem environment. From the agent's point of view with its limited capability to perceive the environment, these observations have identical properties with the actual landmarks. Therefore, while constructing the graph from the random trajectories of the agent, distinguishing them from the actual landmarks becomes extremely challenging for DD methods, even when the interaction between landmarks are simple in the given partially observable RL problem.

Application of random walk is required to complete the transition graph as early as possible in the learning process. However, with random trajectories, the number of positive bags becomes much less than the number of negative bags as well. Therefore, balanced bags approach is proposed to compensate the shortage of positive bags. Algorithm 3 explains the steps of this approach in details. For every episode of the learning agent, the current difference between the numbers of positive and negative bags is kept. This difference is bounded in both ends with some threshold so that neither positive bags nor negative bags dominate the calculation of DD values. However, in order not to lose any information from earlier experiences, the bags that are not immediately included in the identification process are saved to be used later for the case of change in the difference in their favor as shown between lines 20-28 in the algorithm.

One other challenge for both DD and DDCF method is that when some of the landmarks are far from the goal state, the agent may face with them many times within a unsuccessful episodes no matter how crucial that landmark is to construct the solution. This directly affects the calculation of DD values coming from negative bags for that particular landmark. DD based methods are obligated to fail once no negative bags are provided. Hence, instead of eliminating the negative bags completely, we prefer to use some weight in the range of (0,1] to balance the value coming from these negative bags with the value that belongs the positive ones. This weight can be adjusted depending on the given problem. Obviously if the weight is 1 then the calculation is done exactly as in the DD and DDCF methods.

Algorithm 3 DD with Balanced Bags

```
1: procedure ProcessBags
2:     require: $H$                                   ▷ Episode History
3:     require: $\tau$                                ▷ Tolerance
4:     require: $B^+$, $B^-$                          ▷ Sets of Positive & Negative Bags
5:     require: $BB^+$, $BB^-$                        ▷ Sets of Backed Up Positive & Negative Bags
6:     $\Delta \leftarrow |B^+| - |B^-|$
7:     if H is successful then
8:         if $|\Delta| < \tau$ then
9:             $B^+ \leftarrow B^+ \cup \{H\}$
10:        else
11:            $BB^+ \leftarrow BB^+ \cup \{H\}$
12:        end if
13:    else
14:        if $|\Delta| < \tau$ then
15:            $B^- \leftarrow B^- \cup \{H\}$
16:        else
17:            $BB^- \leftarrow BB^- \cup \{H\}$
18:        end if
19:    end if
20:    $\Delta \leftarrow |B^+| - |B^-|$                ▷ Recalculation of Diff.
21:    if $|\Delta| > 0$ and $|BB^-| > 0$ then
22:        $H_i^- \leftarrow$ Select and remove a random history from $BB^-$
23:        $B^- \leftarrow B^- \cup \{H_i^-\}$
24:    end if
25:    if $|\Delta| < 0$ and $|BB^+| > 0$ then
26:        $H_i^+ \leftarrow$ Select and remove a random history from $BB^+$
27:        $B^+ \leftarrow B^+ \cup \{H_i^+\}$
28:    end if
29: end procedure
```

Along with the application of weighting the DD values coming from negative bags, overall DD values are normalized according the occurrence of the particular observation. Therefore, the normalized value of visitation is added to the DD calculation as a factor. The value for the given observation o can be formulated as below:

$$NVV(o) = 1 - \frac{F(o) - \min_{o' \in O} F(o')}{\max_{o' \in O} F(o') - \min_{o' \in O} F(o')}, \tag{6.8}$$

where $F(o)$ and O denotes the frequency of given observation o and the set of observations respectively.

Although these precautions refine the identified landmarks, the complete elimination of the noise is not guaranteed within the capability of DD based methods. As long as the second component of CBT framework constructs the correct chains, that is not crucial for the overall performance. However, any noise in the constructed chains results in failure for CBT, since it disturbs the accurate partitioning of the given RL problem with hidden states by introducing ambiguous starting and ending observations as landmarks that bound each sub-problem.

6.4 Integration of All CBT's components

Despite being noisy, with the implementation of DD based landmark identification, the complete integration of CBT's components can be realized for the learning agent. The ideal process of CBT framework can be seen in Figure 6.1. This figure depicts the case where set of chains constructed by CBT agent contains at least one *noise-free* and *complete* bottleneck transition chain.

This approach may bring the question "Why are the chains verified instead of landmarks?". There are two main reasons to follow such an approach. First of all, the identified landmarks are subject to a second elimination while looking for the associations between other identified landmarks during the construction of the bottleneck transition chains. Therefore, this gives the CBT agent another chance to avoid the noise of non-landmark observations without the guidance of a problem designer.

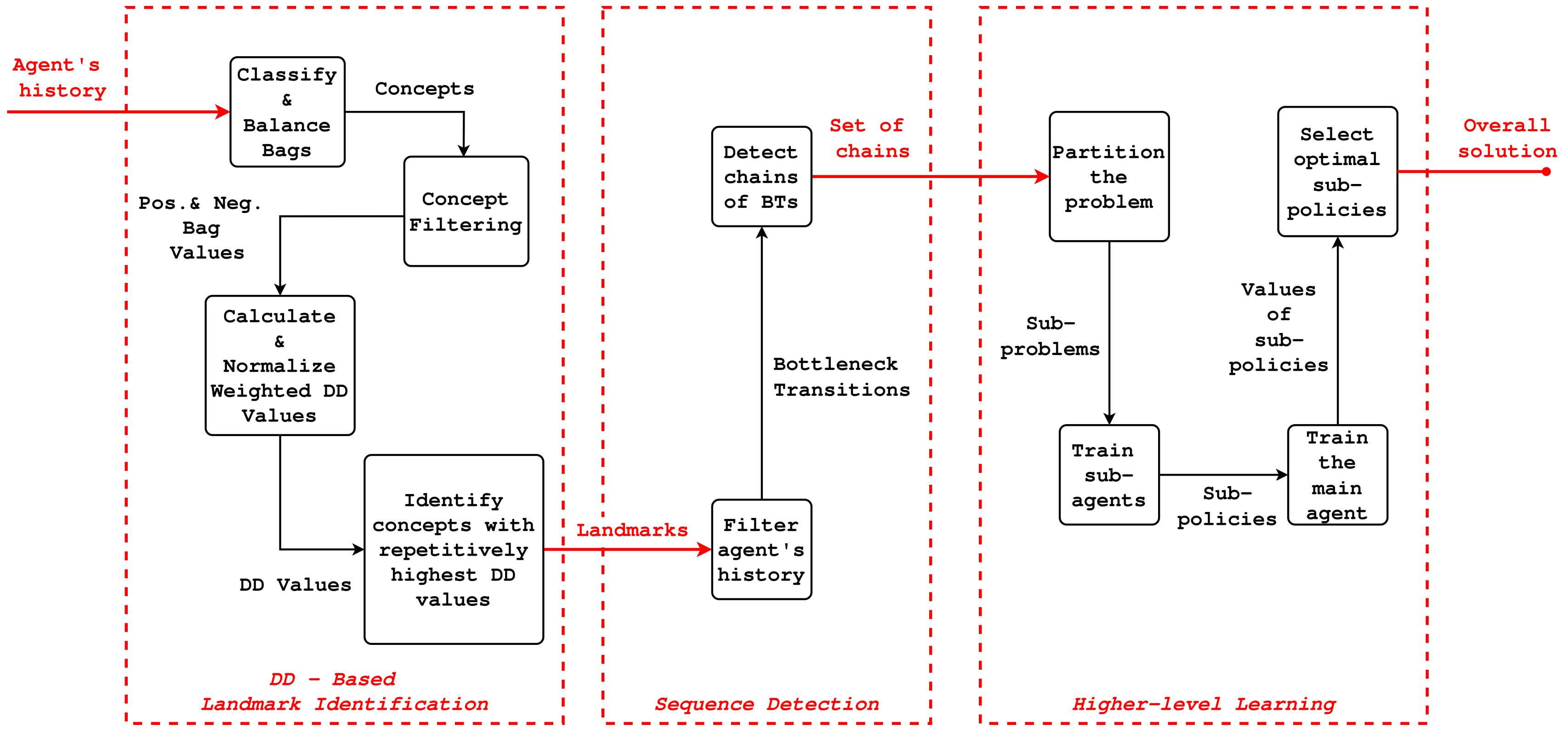


Figure 6.1: The ideal process of CBT framework

Secondly, while checking the completeness of chains, we are not trying to find an exact match between the found and actual bottleneck transition sequences. As long as the learning agent finds a sequence that leads it to the solution of the problem, it can use the chains including non-essential transitions to partition the problem. This may cause an inefficiency in the overall learning performance, however, considering that the traditional methods fail to solve the problems attacked in the scope of this study, this inefficiency can be tolerated while we are trying to keep the human interference as minimal as possible.

The learning agent starts to interacting with the environment by doing a random walk. Its history is fed to the component that is responsible for identification of the landmarks as seen in the Figure 6.1. Then applying DD-based approach with our refinements, identified landmarks are provided to second component to construct the chains. As the initial state of the agent is fixed in our formulation of the problems, the agent is assumed to be aware of the initial landmark, since its identification is beyond the capabilities of DD. Finally, by the means of these chains higher-level learning is achieved with the decomposition of the problem and evaluation of sub-policies for each sub-problem as applied in the earlier phases of CBT framework.

6.5 Experiments

Experiments are carried out in four different domains to evaluate the learning performance of CBT with both automatic landmark identification and the self-constructed chains. Three of them are the larger ones that we conventionally follow from the earlier phases of CBT method, namely `6 rooms`$_{FS}$, `zigzag rooms`, and `6 rooms with locked shortcut`. The last one is the puzzle called `Tower of Hanoi` (see Section 2.5.).

Despite all our efforts, CBT agent fails to construct a noisy-free and complete sequence of bottleneck transitions in `Tower of Hanoi` domain. Therefore, the experiments in this domain is not reported, instead the reasons behind this failure are discussed in the next section.

The results in the rest of the domains are provided both excluding and including the failing cases. In this way, the chance to examine the ideal case is obtained as well as the average case.

6.5.1 Settings and Results

The environment feedback yields two reward values for the domains: 1 upon reaching the goal and 0 for any other steps. The same setting is used in the training of the sub-agents by adjusting their rewarding state according to their sub-task. Similar to the downscaled domains, the initial landmarks of other sub-agents are regarded as pitfalls without introducing any extra punishment.

In each domain, 20 experiments were executed with 5000 episodes for each. Each episode is limited to 5000 steps for `zigzag rooms` domain and 10000 steps for the rest. Discount factor (γ) is set to 0.9. In both training of the sub-agents and sub-policy evaluations, the learning rate (α) is chosen as 0.05. In both phases, $\epsilon - greedy$ approach is applied with the ϵ value decaying linearly after each episode from 0.9 down to 0.1. In the CBT method, the main agents are forced to walk randomly for the first 300 episodes for `6 rooms` and its variant. In `zigzag rooms` domain same thing applied in the first 200 episodes. DD-based landmark identification is executed in earlier phase of random walk where the last 100 episodes are reserved for the self-construction of bottleneck transition chains in all domains. Parameters used in DD-based landmark identification are provided in Table 6.1.

Table 6.1: Parameters used in DD-based landmark identification for the sample domains

Parameter	**6 rooms$_{FS}$**	**zigzag rooms**	**6 rooms with locked shortcut**
Success threshold (to classify bags)	4000	5000	5000
λ (to discount running avg.)	0.95	0.95	0.95
θ (as threshold for running avg.)	15	17	10
R_N (neighborhood radius for DDCF)	1	1	1
W_n (weight for neg. bags)	0.75	1	0.5

As we have adequately examined the Q-learning and Sarsa(λ) as the base methods of the sub-agents' training in the previous chapters, this chapter only includes the Q-learning as the underlying learning method of CBT's sub-agents.

Figure 6.2-6.4 show the learning performances of the CBT method equipped with the DD-based landmark identification for the cases where at least one noise-free and complete sequence is obtained.

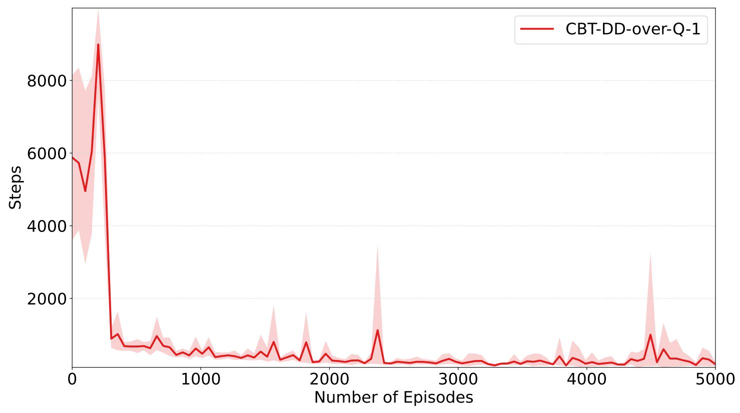


Figure 6.2: The learning performances of the CBT method integrated with DD-based landmark identification in `6 rooms`$_{FS}$ problem in terms of the steps needed to reach the goal

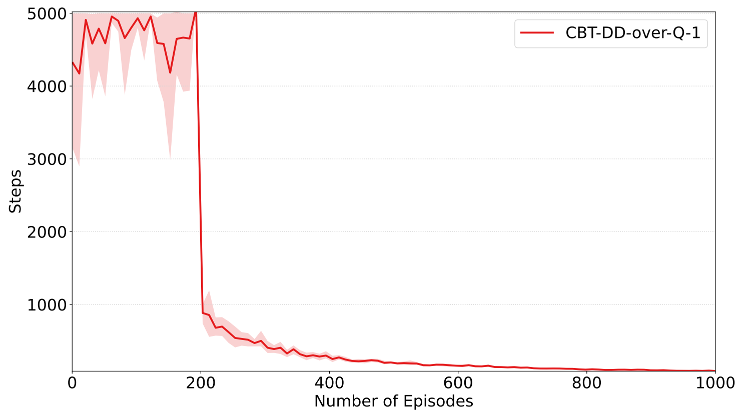


Figure 6.3: The learning performances of the CBT method integrated with DD-based landmark identification in `zigzag rooms` problem in terms of the steps needed to reach the goal

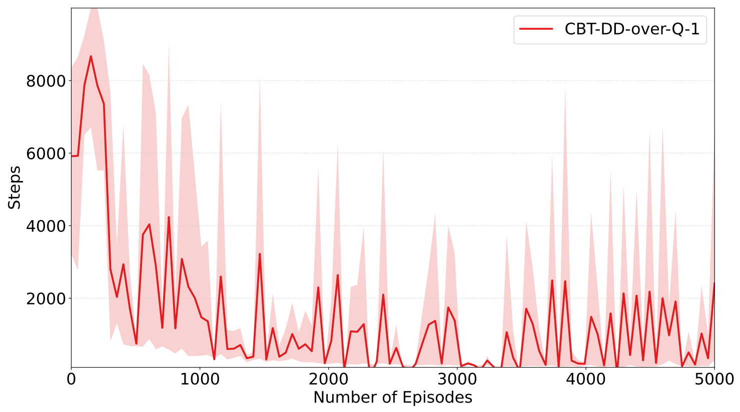


Figure 6.4: The learning performances of the CBT method integrated with DD-based landmark identification in `6 rooms with locked shortcut` problem in terms of the steps needed to reach the goal

The percentages of experiments where at least one non-noisy and complete chain is constructed is as follows:

- `6 rooms`$_{FS}$ $\rightarrow$ 50%
- `zigzag rooms` $\rightarrow$ 40%
- `6 rooms with locked shortcut` $\rightarrow$ 15%

Table 6.2 shows the learning performances of the method in terms of average rewards the agent gets for both the ideal and the average cases. Although the experiments are terminated within the failure case, the agent is assumed to end an episode by reaching the step limit in the calculation of average case. It is clear that as the landmark identification is improved, the agent will achieve better learning.

Table 6.2: Average reward per step in the sample domains for both ideal and average cases of automated CBT method

	6 rooms$_{FS}$	**zigzag rooms**	**6 rooms with locked shortcut**
Ideal case	0.0013364	0.0007430	0.0006198
Average case	0.0000914	0.0000217	0.0000296

6.6 Summary and Discussion

In this chapter the automation of overall CBT framework is completed despite the noisy identification of the landmarks in the given problems. This issue is inherited by the chosen DD method. Although several refinements are applied on this method, the noise could not be completely eliminated.

Whereas the DD method has its drawbacks on landmark identification, sometimes the problem can be really difficult by having non-bottleneck yet rare observations. The corners in the navigational domains are good examples for this type of observation that is falsely identified.

Yet, the worst scenario arises in the `Tower of Hanoi` problem where there exists non-landmark bottleneck states (see Section 2.5.3.). For instance, the aliased states covered in Figure 2.30 yield the same observation yet only one of them has a critical role in the solution.

Nevertheless, CBT framework is empirically proven to be successful in the problems where the solution without an external memory is difficult or impossible to obtain. Hence, the automated CBT method has a potential to succeed in RL problems where the states are hidden from the learning agent. Therefore, the studies following this one will focus on landmark identification with more quality to provide a higher chance for the noise-free and complete chains.

CHAPTER 7

CONCLUSIONS

This thesis proposes two different approaches for the RL problem with hidden states where the agent is assumed to be oblivious to the underlying MDP model. Within the limited perception of the agent, it can utilize the frequencies of its significant experiences in order to complete its task.

Based on this idea, a selective memory mechanism, namely Compact Frequency Memory (CFM) is presented within the scope of this study. Since the ambiguous states are statistically seen more often than the less aliased ones, eliminating them leads to a small sized memory which is quite useful in the construction of compact state estimations. Hence, an RL agent equipped with CFM, deals with the ambiguity of the environment more efficiently and a faster learning is obtained in this way. As a result, CFM is much more scalable than the instance based methods that use extensive memory to distinguish aliased states in the given problem.

Although CFM is highly efficient and successful in simple tasks with and/or without a larger state space, it fails to catch long term dependencies, if a critical transition from the agent's history is overwritten by a new infrequent transition. Therefore, a possible future direction for this approach could be the construction of an additional trace mechanism to distinguish the useful transitions that have a critical role in the problem and keep them in the memory as long as the agent needs it.

As the second part of the thesis, by Chains of Bottleneck Transitions (CBT), given RL problem is divided into smaller sub-problems that have less ambiguous states than the original. In this way, challenging partially observable problems turn out to be easier to solve. Even the tasks, for which traditional methods fail to succeed, can be accomplished by this divide-and-conquer based approach. Furthermore, it is empirically shown by experiments that if the agent is equipped with the correct order of the bottleneck transitions, the solution can be constructed in the earlier stages of the learning process without any need for additional external memory.

Once CBT is proven to be useful, as the next step of the study, self-construction of the chains for given landmarks has been realized by the proposed method. Searching from last seen landmarks in an episode through the first ones, this method completes the chains via following the most associated landmarks. Since the method itself is prone to be noisy, the found chains can be longer than it should be. However, considering the fact that it may be impossible to solve these problems without the decomposition based on chains, such noises can be tolerated. Nevertheless, further improvement can be made in order to find complex sequences in the domains where there are more landmarks and the solution require complicated and overlapping paths. Lastly, one other future work for the self-construction of the chains could be handling the problem that requires cyclic visitation of landmarks.

Finally, to complete the self-exploratory behavior of an RL agent, the overall method is automatized by the integration of Diverse Density (DD) algorithm that identifies landmarks without prior knowledge of the domain. To improve the quality of found landmarks, the idea of Concept Filtering [35, 19] is applied to the instances coming from the agent's own experience to eliminate the noise, namely, irrelevant observations. Further improvement on DD has been achieved by balancing the number of negative and positive bags. Although found landmarks are subject to another filtering in the construction of chains phase, it is more convenient to feed the constructor component with completely noise-free data. Hence, achieving highly accurate landmark identification could be the ultimate aim in the future of this study.

REFERENCES

[1] A. K. McCallum, *Reinforcement Learning with Selective Perception and Hidden State*. PhD thesis, University of Rochester, 1996.

[2] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction Second Edition: 2014, 2015*. The MIT Press, 2014.

[3] J. N. Tsitsiklis and B. Van Roy, "An analysis of temporal-difference learning with function approximation," *Report LIDS-P-2322). Laboratory for Information and Decision Systems, Massachusetts Institute of Technology, Tech. Rep.*, 1996.

[4] S. P. Singh, T. Jaakkola, and M. I. Jordan, "Reinforcement learning with soft state aggregation," in *Advances in neural information processing systems*, pp. 361–368, 1995.

[5] P. Dayan and G. E. Hinton, "Feudal reinforcement learning," in *Advances in neural information processing systems*, pp. 271–278, 1993.

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

[7] M. Wiering and J. Schmidhuber, "Hq-learning," *Adaptive Behavior*, vol. 6, no. 2, pp. 219–246, 1997.

[8] 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.

[9] E. A. Mcgovern and A. G. Barto, *Autonomous discovery of temporal abstractions from interaction with an environment*. PhD thesis, University of Massachusetts at Amherst, 2002.

[10] O. Simsek, *Behavioral building blocks for autonomous agents: description, identification, and learning*. University of Massachusetts Amherst, 2008.

[11] I. Menache, S. Mannor, N. Shimkin, *et al.*, "Q-cut-dynamic discovery of sub-goals in reinforcement learning," in *13th European Conference on Machine Learning Proceedings*, vol. 14 of *Machine Learning: ECML 2002*, pp. 295–306, Springer, 2002.

[12] L. Chrisman, "Reinforcement learning with perceptual aliasing: the perceptual distinctions approach," in *Proceedings of the Tenth National Conference on Artificial Intelligence*, AAAI'92, pp. 183–188, 1992.

[13] R. Regier, O. Price, A. Hadi, Z. Faltersack, and A. Nuxoll, "Aro: A memory-based approach to environments with perceptual aliasing.," in *AAAI Spring Symposium: Combining Machine Learning with Knowledge Engineering (1)*, 2020.

[14] A. Siddique, W. N. Browne, and G. M. Grimshaw, "Frames-of-reference-based learning: Overcoming perceptual aliasing in multistep decision-making tasks," *IEEE Transactions on Evolutionary Computation*, vol. 26, no. 1, pp. 174–187, 2022.

[15] L.-J. Lin and T. M. Mitchell, *Memory approaches to reinforcement learning in non-Markovian domains*. Carnegie-Mellon University. Department of Computer Science, 1992.

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

[17] 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, 1998.

[18] M. R. James and S. Singh, "Sarsalandmark: An algorithm for learning in pomdps with landmarks," in *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems - Volume 1*, AAMAS '09, (Richland, SC), pp. 585–591, International Foundation for Autonomous Agents and Multiagent Systems, 2009.

[19] A. Demir, E. Çilden, and F. Polat, "Automatic landmark discovery for learning agents under partial observability," *The Knowledge Engineering Review*, vol. 34, p. e11, 2019.

[20] D. Steckelmacher, D. M. Roijers, A. Harutyunyan, P. Vrancx, H. Plisnier, and A. Nowé, "Reinforcement learning in pomdps with memoryless options and option-observation initiation sets," *arXiv preprint arXiv:1708.06551*, 2017.

[21] G. Theocharous and S. Mahadevan, *Hierarchical learning and planning in partially observable Markov decision processes*. Michigan State University. Department of Computer Science & Engineering, 2002.

[22] 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*, From Animals to Animats 3: SAB'94, pp. 238–245, 1994.

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

[24] R. F. Pereira, N. Oren, and F. Meneguzzi, "Landmark-based approaches for goal recognition as planning," *Artificial Intelligence*, vol. 279, p. 103217, 2020.

[25] H. Aydın, E. Çilden, and F. Polat, "Compact frequency memory for reinforcement learning with hidden states," in *International Conference on Principles and Practice of Multi-Agent Systems*, pp. 425–433, Springer, 2019.

[26] H. Aydın, E. Çilden, and F. Polat, "Using chains of bottleneck transitions to decompose and solve reinforcement learning tasks with hidden states," *Future Generation Computer Systems*, vol. 133, pp. 153–168, 2022.

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

[28] A. M. Hinz, "The tower of hanoi," *Enseign. Math*, vol. 35, no. 2, pp. 289–321, 1989.

[29] C. J. C. H. Watkins, *Learning from Delayed Rewards.* PhD thesis, King's College, Cambridge, UK, May 1989.

[30] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger, "Deep reinforcement learning that matters," in *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018* (S. A. McIlraith and K. Q. Weinberger, eds.), pp. 3207–3214, AAAI Press, 2018.

[31] D. Ron, Y. Singer, and N. Tishby, "Learning probabilistic automata with variable memory length," in *Proceedings of the Seventh Annual Conference on Computational Learning Theory*, COLT '94, (New York, NY, USA), pp. 35–46, ACM, 1994.

[32] E. Çilden and F. Polat, "Generating memoryless policies faster using automatic temporal abstractions for reinforcement learning with hidden state," in *2013 IEEE 25th International Conference on Tools with Artificial Intelligence*, pp. 719–726, IEEE, 2013.

[33] O. Maron and T. Lozano-Pérez, "A framework for multiple-instance learning," in *Advances in neural information processing systems*, pp. 570–576, 1998.

[34] A. McGovern and A. G. Barto, "Automatic discovery of subgoals in reinforcement learning using diverse density," *Computer Science Department Faculty Publication Series*, p. 8, 2001.

[35] A. Demir, E. Cilden, and F. 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.

[36] 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, 2008.

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

[38] 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.

APPENDIX A

APPENDIX

The further details about the experimentation of the CBT method are given in the following tables.

Table A.1: The number of distinct sub-policies extracted by CBT in remaining sample domains

`downscaled 6 rooms`	Only Doors	12
	All Landmarks	27
`downscaled zigzag rooms`	Only Doors	5
	All Landmarks	12
`Tower of Hanoi`		36

Table A.2: The average memory usage (MB) in an episode for the methods in remaining sample domains

	6 rooms$_{FS}$	**zigzag rooms**	**6 rooms with locked shortcut**	**Tower of Hanoi**
CBT over Q	2.519 ± 0.082	2.31 ± 0.061	7.814 ± 0.14	3.06 ± 0.083
CBT over Sarsa(λ)	2.608 ± 0.063	2.371 ± 0.071	8.024 ± 0.17	3.086 ± 0.114

Table A.3: The average time (sec) spent in overall experiments for the methods in remaining sample domains

		6 rooms$_{FS}$	**zigzag rooms**	**6 rooms with locked shortcut**	**Tower of Hanoi**
CBT over Q	Chain Construction	0.000061 ± 0.000004	0.000122 ± 0.000006	0.000237 ± 0.000013	0.000101 ± 0.000015
	Sub-agent Training	106.19 ± 11.54	47.18 ± 9.10	1050.69 ± 199.28	26.57 ± 17.54
	Total	254.32 ± 75.72	156.90 ± 28.60	1208.67 ± 206.75	463.87 ± 90.84
CBT over Sarsa(λ)	Chain Construction	0.000055 ± 0.000010	0.000123 ± 0.000006	0.000241 ± 0.000017	0.000096 ± 0.000019
	Sub-agent Training	365.19 ± 78.81	251.32 ± 53.45	1320.58 ± 286.78	27.68 ± 6.87
	Total	454.98 ± 93.14	357.79 ± 71.69	1452.88 ± 301.63	475.76 ± 85.62

Table A.4: The chains constructed by CBT in `downscaled 6 rooms` domain

Problem	**Chain**	**Occurrence**
downscaled 6 rooms	(S, east) → (o17, north) → (o18, east) → (o20, east) → (o22, south) → G	1
	(S, east) → (o17, south) → (o19, east) → (o21, east) → G	8
	(S, south) → (o17, south) → (o19, east) → (o21, east) → G	24
	(S, east) → (o18, east) → (o20, east) → (o22, south) → G	25
	(S, south) → (o18, east) → (o20, east) → (o22, south) → G	10
	(S, east) → (o18, west) → (o17, south) → (o19, east) → (o21, east) → G	6
	(S, south) → (o17, north) → (o18, east) → (o20, east) → (o22, south) → G	4
	(S, south) → (o18, west) → (o17, south) → (o19, east) → (o21, east) → G	2
downscaled 6 rooms (All Landmarks)	(S, east) → (o9, east) → (o18, east) → (o20, east) → (o4, south) → (o22, south) → (o13, south) → G	3
	(S, south) → (o17, south) → (o16, east) → (o19, east) → (o21, east) → G	4
	(S, east) → (o9, east) → (o18, east) → (o20, east) → (o22, south) → G	1
	(S, east) → (o9, south) → (o17, south) → (o16, east) → (o19, east) → (o21, east) → G	11
	(S, east) → (o9, east) → (o18, east) → (o20, east) → (o4, south) → (o22, south) → G	8
	(S, east) → (o9, south) → (o17, north) → (o18, east) → (o20, east) → (o4, south) → (o22, south) → G	2
	(S, south) → (o9, east) → (o18, east) → (o20, east) → (o22, south) → (o13, east) → G	1
	(S, south) → (o9, south) → (o17, south) → (o16, east) → (o19, east) → (o21, east) → G	6
	(S, south) → (o17, south) → (o16, east) → (o19, east) → (o21, east) → (o13, east) → G	1
	(S, south) → (o9, east) → (o18, east) → (o20, east) → (o22, south) → G	1
	(S, south) → (o9, south) → (o17, south) → (o19, east) → (o21, east) → G	1
	(S, south) → (o9, east) → (o18, east) → (o20, east) → (o4, east) → (o22, south) → G	1
	(S, south) → (o9, south) → (o17, south) → (o16, east) → (o19, east) → (o21, east) → (o13, east) → G	1

Table A.5: The chains constructed by CBT in `downscaled 6 domains` (cont.)

Problem	Chain	Occurrence
`downscaled 6 rooms` **(All Landmarks)**	(S, east) → (o9, east) → (o18, east) → (o20, east) → (o4, south) → (o22, south) → (o13, east) → G	3
	(S, east) → (o9, east) → (o18, west) → (o17, south) → (o16, east) → (o19, east) → (o21, east) → G	3
	(S, east) → (o9, east) → (o18, east) → (o20, east) → (o22, south) → (o13, east) → G	1
	(S, south) → (o9, east) → (o18, west) → (o17, south) → (o16, east) → (o19, east) → (o21, east) → G	1
	(S, south) → (o9, east) → (o18, east) → (o20, east) → (o4, south) → (o22, south) → G	7
	(S, east) → (o9, south) → (o17, south) → (o16, east) → (o19, east) → (o21, east) → (o22, south) → (o13, east) → G	1
	(S, south) → (o18, east) → (o20, east) → (o4, south) → (o22, south) → G	1
	(S, south) → (o17, south) → (o16, east) → (o19, east) → (o21, east) → (o13, south) → G	1
	(S, south) → (o9, east) → (o18, east) → (o20, east) → (o4, south) → (o22, south) → (o13, east) → G	2
	(S, east) → (o18, east) → (o20, east) → (o4, south) → (o22, south) → (o13, south) → G	2
	(S, south) → (o9, south) → (o17, south) → (o19, east) → (o21, east) → (o13, east) → G	1
	(S, south) → (o17, south) → (o19, east) → (o21, east) → G	3
	(S, east) → (o18, east) → (o20, east) → (o4, south) → (o22, south) → G	3
	(S, south) → (o9, south) → (o17, south) → (o16, east) → (o19, east) → (o21, east) → (o13, south) → G	1
	(S, east) → (o9, south) → (o17, south) → (o16, east) → (o19, east) → (o21, east) → (o13, south) → G	1
	(S, south) → (o9, east) → (o18, east) → (o20, east) → (o4, south) → (o22, south) → (o13, south) → G	1
	(S, east) → (o17, south) → (o16, east) → (o19, east) → (o21, east) → G	1
	(S, south) → (o9, south) → (o17, south) → (o16, east) → (o19, east) → (o21, east) → (o13, west) → G	1
	(S, east) → (o9, east) → (o18, east) → (o20, east) → (o4, east) → (o22, south) → G	1
	(S, south) → (o9, south) → (o17, south) → (o19, east) → (o21, east) → (o13, south) → G	1

Table A.6: The chains constructed by CBT in `downscaled zigzag rooms` domain

Problem	**Chain**	**Occurrence**
`downscaled zigzag rooms`	(S, north) → (o16, east) → (o17, east) → (o18, east) → G	23
	(S, east) → (o16, east) → (o17, east) → (o18, east) → G	17
`downscaled zigzag rooms` **(All Landmarks)**	(S, north) → (o16, east) → (o14, east) → (o15, east) → (o17, east) → (o12, east) → (o11, east) → (o18, east) → G	2
	(S, north) → (o16, east) → (o15, east) → (o17, east) → (o12, east) → (o11, north) → (o18, east) → G	4
	(S, north) → (o16, east) → (o15, east) → (o17, east) → (o12, north) → (o18, east) → G	4
	(S, east) → (o16, east) → (o15, east) → (o17, east) → (o12, east) → (o11, north) → (o18, east) → G	2
	(S, east) → (o16, east) → (o14, east) → (o15, east) → (o17, east) → (o12, north) → (o18, east) → G	5
	(S, east) → (o16, east) → (o14, east) → (o15, east) → (o17, east) → (o12, east) → (o11, north) → (o18, east) → G	6
	(S, north) → (o16, east) → (o14, east) → (o15, east) → (o17, east) → (o12, north) → (o18, east) → G	2
	(S, east) → (o16, east) → (o14, east) → (o15, east) → (o17, east) → (o12, east) → (o11, east) → (o18, east) → G	2
	(S, north) → (o16, east) → (o14, east) → (o15, east) → (o17, east) → (o12, east) → (o11, north) → (o18, east) → G	9
	(S, north) → (o16, east) → (o15, east) → (o17, east) → (o12, east) → (o11, east) → (o18, east) → G	2
	(S, east) → (o16, east) → (o15, east) → (o17, east) → (o12, north) → (o18, east) → G	2

Table A.7: The chains constructed by CBT in larger domains

Problem	**Chain**	**Occ.**
`6 rooms`$_{\texttt{FS}}$	(S, east) → (o37, east) → (o39, east) → (o41, south) → G	22
	(S, south) → (o36, south) → (o38, east) → (o40, east) → G	22
	(S, south) → (o37, east) → (o39, east) → (o41, south) → G	18
	(S, east) → (o36, south) → (o38, east) → (o40, east) → G	17
	(S, east) → (o37, west) → (o36, south) → (o38, east) → (o40, east) → G	1
`zigzag rooms`	(S, east) → (o36, east) → (o37, east) → (o38, east) → G	57
	(S, north) → (o36, east) → (o37, east) → (o38, east) → G	63
`6 room with locked shortcut`	((S, l), east) → ((o36, l), north) → ((B, u), north) → ((o37, u), east) → ((o39, u), east) → ((o41, u), south) → ((o40, u), west) → (G, u)	34
	((S, l), east) → ((o36, l), north) → ((B, u), west) → ((o36, u), south) → ((o38, u), east) → (G, u)	34
	((S, l), north) → ((o36, l), north) → ((o37, l), east) → ((o39, l), east) → ((o41, l), south) → ((o40, l), west) → (G, l)	46
	((S, l), north) → ((o36, l), north) → ((B, u), west) → ((o36, u), south) → ((o38, u), east) → (G, u)	28
	((S, l), north) → ((o36, l), north) → ((B, u), north) → ((o37, u), east) → ((o39, u), east) → ((o41, u), south) → ((o40, u), west) → (G, u)	30
	((S, l), north) → ((o36, l), north) → ((B, u), north) → ((o36, u), south) → ((o38, u), east) → (G, u)	8
	((S, l), north) → ((o36, l), north) → ((B, u), west) → ((o37, u), east) → ((o39, u), east) → ((o41, u), south) → ((o40, u), west) → (G, u)	10
	((S, l), east) → ((o36, l), north) → ((o37, l), east) → ((o39, l), east) → ((o41, l), south) → ((o40, l), west) → (G, l)	34
	((S, l), east) → ((o36, l), north) → ((B, u), west) → ((o36, u), south) → ((S, u), east) → ((o38, u), east) → (G, u)	2
	((S, l), east) → ((o36, l), north) → ((B, u), west) → ((o37, u), east) → ((o39, u), east) → ((o41, u), south) → ((o40, u), west) → (G, u)	6
	((S, l), north) → ((o36, l), north) → ((B, u), west) → ((o36, u), south) → ((S, u), east) → ((o38, u), east) → (G, u)	2
	((S, l), north) → ((o36, l), north) → ((B, u), west) → ((o36, u), south) → ((S, u), north) → ((o38, u), east) → (G, u)	2
	((S, l), east) → ((o36, l), north) → ((B, u), north) → ((o36, u), south) → ((o38, u), east) → (G, u)	4

Table A.8: The chains constructed by CBT in `Tower of Hanoi` domain

Problem	Chain	Occ.
Tower of Hanoi	$(S, a_{AB}) \rightarrow ((4, 3, 1), a_{CB}) \rightarrow ((4, 1, 0), a_{AC}) \rightarrow ((0, 1, 4), a_{BC}) \rightarrow ((1, 3, 4), a_{BC}) \rightarrow G$	5
	$(S, a_{AC}) \rightarrow ((4, 1, 3), a_{BC}) \rightarrow ((4, 0, 1), a_{AB}) \rightarrow ((0, 4, 1), a_{CB}) \rightarrow ((1, 4, 0), a_{BC}) \rightarrow ((1, 0, 4), a_{AC}) \rightarrow ((3, 1, 4), a_{AC}) \rightarrow G$	3
	$(S, a_{AB}) \rightarrow ((4, 3, 1), a_{CA}) \rightarrow ((4, 1, 0), a_{AC}) \rightarrow ((0, 1, 4), a_{BC}) \rightarrow ((1, 3, 4), a_{BC}) \rightarrow G$	20
	$(S, a_{AC}) \rightarrow ((4, 1, 3), a_{BA}) \rightarrow ((4, 0, 1), a_{AB}) \rightarrow ((0, 4, 1), a_{CB}) \rightarrow ((1, 4, 3), a_{AC}) \rightarrow ((1, 4, 0), a_{BC}) \rightarrow ((1, 0, 4), a_{AC}) \rightarrow ((3, 1, 4), a_{AC}) \rightarrow G$	1
	$(S, a_{AC}) \rightarrow ((4, 1, 3), a_{BA}) \rightarrow ((4, 0, 1), a_{AB}) \rightarrow ((0, 4, 1), a_{CB}) \rightarrow ((1, 4, 3), a_{AC}) \rightarrow ((3, 4, 1), a_{CB}) \rightarrow ((1, 4, 0), a_{BC}) \rightarrow ((1, 0, 4), a_{AB}) \rightarrow ((3, 1, 4), a_{AC}) \rightarrow G$	1
	$(S, a_{AC}) \rightarrow ((4, 1, 3), a_{BC}) \rightarrow ((4, 0, 1), a_{AB}) \rightarrow ((0, 4, 1), a_{CA}) \rightarrow ((3, 4, 1), a_{CB}) \rightarrow ((1, 4, 0), a_{BC}) \rightarrow ((1, 0, 4), a_{AB}) \rightarrow ((3, 1, 4), a_{AC}) \rightarrow G$	1
	$(S, a_{AB}) \rightarrow ((4, 3, 1), a_{CA}) \rightarrow ((4, 1, 0), a_{AC}) \rightarrow ((0, 1, 4), a_{BA}) \rightarrow ((1, 3, 4), a_{BC}) \rightarrow G$	7
	$(S, a_{AC}) \rightarrow ((4, 1, 3), a_{BA}) \rightarrow ((4, 0, 1), a_{AB}) \rightarrow ((0, 4, 1), a_{CB}) \rightarrow ((1, 4, 3), a_{CB}) \rightarrow ((3, 4, 1), a_{CB}) \rightarrow ((1, 4, 0), a_{BC}) \rightarrow ((1, 0, 4), a_{AB}) \rightarrow ((3, 1, 4), a_{AC}) \rightarrow G$	1
	$(S, a_{AB}) \rightarrow ((4, 1, 3), a_{BA}) \rightarrow ((4, 0, 1), a_{AB}) \rightarrow ((0, 4, 1), a_{CA}) \rightarrow ((1, 4, 0), a_{BC}) \rightarrow ((1, 0, 4), a_{AC}) \rightarrow ((3, 1, 4), a_{AC}) \rightarrow G$	1
	$(S, a_{AC}) \rightarrow ((4, 3, 1), a_{CA}) \rightarrow ((4, 1, 0), a_{AC}) \rightarrow ((0, 1, 4), a_{BC}) \rightarrow ((1, 3, 4), a_{BC}) \rightarrow G$	8
	$(S, a_{AC}) \rightarrow ((4, 1, 3), a_{BA}) \rightarrow ((4, 0, 1), a_{AB}) \rightarrow ((0, 4, 1), a_{CA}) \rightarrow ((1, 4, 0), a_{BC}) \rightarrow ((1, 0, 4), a_{AC}) \rightarrow ((3, 1, 4), a_{AC}) \rightarrow G$	3
	$(S, a_{AC}) \rightarrow ((4, 1, 3), a_{BA}) \rightarrow ((4, 0, 1), a_{AB}) \rightarrow ((0, 4, 1), a_{CA}) \rightarrow ((1, 4, 3), a_{CB}) \rightarrow ((3, 4, 1), a_{CB}) \rightarrow ((1, 4, 0), a_{BC}) \rightarrow ((1, 0, 4), a_{AB}) \rightarrow ((3, 1, 4), a_{AC}) \rightarrow G$	2
	$(S, a_{AC}) \rightarrow ((4, 1, 3), a_{BA}) \rightarrow ((4, 0, 1), a_{AB}) \rightarrow ((0, 4, 1), a_{CA}) \rightarrow ((3, 4, 1), a_{CB}) \rightarrow ((1, 4, 0), a_{BC}) \rightarrow ((1, 0, 4), a_{AC}) \rightarrow ((3, 1, 4), a_{AC}) \rightarrow G$	3
	$(S, a_{AC}) \rightarrow ((4, 1, 3), a_{BC}) \rightarrow ((4, 0, 1), a_{AB}) \rightarrow ((0, 4, 1), a_{CA}) \rightarrow ((1, 4, 3), a_{CB}) \rightarrow ((0, 1, 0), a_{BA}) \rightarrow ((3, 4, 1), a_{CA}) \rightarrow ((1, 4, 0), a_{BC}) \rightarrow ((1, 0, 4), a_{AB}) \rightarrow ((3, 1, 4), a_{AC}) \rightarrow G$	1
	$(S, a_{AC}) \rightarrow ((4, 1, 3), a_{BA}) \rightarrow ((4, 0, 1), a_{AB}) \rightarrow ((0, 4, 1), a_{CB}) \rightarrow ((1, 4, 3), a_{CB}) \rightarrow ((0, 1, 0), a_{BC}) \rightarrow ((3, 4, 1), a_{CB}) \rightarrow ((1, 4, 0), a_{BC}) \rightarrow ((1, 0, 4), a_{AC}) \rightarrow ((3, 1, 4), a_{AC}) \rightarrow G$	1
	$(S, a_{AC}) \rightarrow ((4, 1, 3), a_{BC}) \rightarrow ((4, 0, 1), a_{AB}) \rightarrow ((0, 4, 1), a_{CA}) \rightarrow ((1, 4, 3), a_{CB}) \rightarrow ((3, 4, 1), a_{CB}) \rightarrow ((1, 4, 0), a_{BC}) \rightarrow ((1, 0, 4), a_{AC}) \rightarrow ((3, 1, 4), a_{AC}) \rightarrow G$	1
	$(S, a_{AC}) \rightarrow ((4, 1, 3), a_{BA}) \rightarrow ((4, 0, 1), a_{AB}) \rightarrow ((0, 4, 1), a_{CB}) \rightarrow ((1, 4, 3), a_{AB}) \rightarrow ((3, 4, 1), a_{CB}) \rightarrow ((1, 4, 0), a_{BC}) \rightarrow ((1, 0, 4), a_{AB}) \rightarrow ((3, 1, 4), a_{AC}) \rightarrow G$	1
	$(S, a_{AC}) \rightarrow ((4, 1, 3), a_{BA}) \rightarrow ((4, 0, 1), a_{AB}) \rightarrow ((0, 4, 1), a_{CA}) \rightarrow ((1, 4, 3), a_{CB}) \rightarrow ((3, 4, 1), a_{CB}) \rightarrow ((1, 4, 0), a_{BC}) \rightarrow ((1, 0, 4), a_{AC}) \rightarrow ((3, 1, 4), a_{AC}) \rightarrow G$	1

Table A.9: The chains constructed by CBT in `Tower of Hanoi` domain (cont.)

Problem	**Chain**	**Occ.**
`Tower of Hanoi`	$(S, a_{AB}) \rightarrow ((4, 3, 1), a_{CA}) \rightarrow ((4, 1, 0), a_{AC}) \rightarrow ((0, 1, 4), a_{BA}) \rightarrow ((3, 1, 4), a_{AC}) \rightarrow G$	1
	$(S, a_{AB}) \rightarrow ((4, 3, 1), a_{CB}) \rightarrow ((4, 0, 1), a_{AB}) \rightarrow ((0, 4, 1), a_{CB}) \rightarrow ((1, 4, 3), a_{CB}) \rightarrow ((3, 4, 1), a_{CB}) \rightarrow ((1, 4, 0), a_{BC}) \rightarrow ((1, 0, 4), a_{AC}) \rightarrow ((3, 1, 4), a_{AC}) \rightarrow G$	1
	$(S, a_{AB}) \rightarrow ((4, 1, 3), a_{BC}) \rightarrow ((4, 0, 1), a_{AB}) \rightarrow ((0, 4, 1), a_{CA}) \rightarrow ((1, 4, 3), a_{AC}) \rightarrow ((1, 4, 0), a_{BC}) \rightarrow ((1, 0, 4), a_{AC}) \rightarrow ((3, 1, 4), a_{AC}) \rightarrow G$	1
	$(S, a_{AC}) \rightarrow ((4, 1, 3), a_{BA}) \rightarrow ((4, 0, 1), a_{AB}) \rightarrow ((0, 4, 1), a_{CB}) \rightarrow ((3, 4, 1), a_{CB}) \rightarrow ((1, 4, 0), a_{BC}) \rightarrow ((1, 0, 4), a_{AC}) \rightarrow ((3, 1, 4), a_{AC}) \rightarrow G$	1
	$(S, a_{AC}) \rightarrow ((4, 1, 3), a_{BA}) \rightarrow ((4, 0, 1), a_{AB}) \rightarrow ((0, 4, 1), a_{CA}) \rightarrow ((1, 4, 0), a_{BC}) \rightarrow ((1, 0, 4), a_{AB}) \rightarrow ((3, 1, 4), a_{AC}) \rightarrow G$	1
	$(S, a_{AC}) \rightarrow ((4, 3, 1), a_{CA}) \rightarrow ((4, 1, 0), a_{AC}) \rightarrow ((0, 1, 4), a_{BA}) \rightarrow ((3, 1, 4), a_{AC}) \rightarrow G$	2
	$(S, a_{AC}) \rightarrow ((4, 1, 3), a_{BC}) \rightarrow ((4, 0, 1), a_{AB}) \rightarrow ((0, 4, 1), a_{CB}) \rightarrow ((1, 4, 3), a_{CB}) \rightarrow ((0, 1, 0), a_{BA}) \rightarrow ((3, 4, 1), a_{CB}) \rightarrow ((1, 4, 0), a_{BC}) \rightarrow ((1, 0, 4), a_{AC}) \rightarrow ((3, 1, 4), a_{AC}) \rightarrow G$	1
	$(S, a_{AC}) \rightarrow ((4, 1, 3), a_{BC}) \rightarrow ((4, 0, 1), a_{AB}) \rightarrow ((0, 4, 1), a_{CB}) \rightarrow ((1, 4, 3), a_{CB}) \rightarrow ((3, 4, 1), a_{CB}) \rightarrow ((1, 4, 0), a_{BC}) \rightarrow ((1, 0, 4), a_{AB}) \rightarrow ((3, 1, 4), a_{AC}) \rightarrow G$	1
	$(S, a_{AC}) \rightarrow ((4, 1, 3), a_{BA}) \rightarrow ((4, 0, 1), a_{AB}) \rightarrow ((0, 4, 1), a_{CB}) \rightarrow ((1, 4, 3), a_{CB}) \rightarrow ((0, 1, 0), a_{BA}) \rightarrow ((3, 4, 1), a_{CA}) \rightarrow ((1, 4, 0), a_{BC}) \rightarrow ((1, 0, 4), a_{AC}) \rightarrow ((3, 1, 4), a_{AC}) \rightarrow G$	1
	$(S, a_{AB}) \rightarrow ((4, 1, 3), a_{BC}) \rightarrow ((4, 0, 1), a_{AB}) \rightarrow ((0, 4, 1), a_{CA}) \rightarrow ((1, 4, 0), a_{BC}) \rightarrow ((1, 0, 4), a_{AC}) \rightarrow ((3, 1, 4), a_{AC}) \rightarrow G$	1
	$(S, a_{AC}) \rightarrow ((4, 1, 3), a_{BC}) \rightarrow ((4, 0, 1), a_{AB}) \rightarrow ((0, 4, 1), a_{CB}) \rightarrow ((1, 4, 3), a_{CB}) \rightarrow ((3, 4, 1), a_{CB}) \rightarrow ((1, 4, 0), a_{BC}) \rightarrow ((1, 0, 4), a_{AC}) \rightarrow ((3, 1, 4), a_{AC}) \rightarrow G$	1
	$(S, a_{AC}) \rightarrow ((4, 1, 3), a_{BA}) \rightarrow ((4, 0, 1), a_{AB}) \rightarrow ((0, 4, 1), a_{CA}) \rightarrow ((3, 4, 1), a_{CA}) \rightarrow ((1, 4, 0), a_{BC}) \rightarrow ((1, 0, 4), a_{AB}) \rightarrow ((3, 1, 4), a_{AC}) \rightarrow G$	1
	$(S, a_{AC}) \rightarrow ((4, 1, 3), a_{BC}) \rightarrow ((4, 0, 1), a_{AB}) \rightarrow ((0, 4, 1), a_{CA}) \rightarrow ((1, 4, 0), a_{BC}) \rightarrow ((1, 0, 4), a_{AC}) \rightarrow ((3, 1, 4), a_{AC}) \rightarrow G$	1
	$(S, a_{AC}) \rightarrow ((4, 3, 1), a_{CA}) \rightarrow ((4, 1, 0), a_{AC}) \rightarrow ((0, 1, 4), a_{BC}) \rightarrow ((3, 1, 4), a_{AC}) \rightarrow G$	1
	$(S, a_{AB}) \rightarrow ((4, 1, 3), a_{BA}) \rightarrow ((4, 0, 1), a_{AB}) \rightarrow ((0, 4, 1), a_{CB}) \rightarrow ((1, 4, 3), a_{CB}) \rightarrow ((0, 1, 0), a_{BC}) \rightarrow ((3, 4, 1), a_{CB}) \rightarrow ((1, 4, 0), a_{BC}) \rightarrow ((1, 0, 4), a_{AC}) \rightarrow ((3, 1, 4), a_{AC}) \rightarrow G$	1
	$(S, a_{AC}) \rightarrow ((4, 1, 3), a_{BA}) \rightarrow ((4, 0, 1), a_{AB}) \rightarrow ((0, 4, 1), a_{CB}) \rightarrow ((1, 4, 3), a_{CB}) \rightarrow ((3, 4, 1), a_{CB}) \rightarrow ((1, 4, 0), a_{BC}) \rightarrow ((1, 0, 4), a_{AC}) \rightarrow ((3, 1, 4), a_{AC}) \rightarrow G$	1
	$(S, a_{AC}) \rightarrow ((4, 1, 3), a_{BC}) \rightarrow ((4, 0, 1), a_{AB}) \rightarrow ((0, 4, 1), a_{CA}) \rightarrow ((1, 4, 0), a_{BC}) \rightarrow ((1, 0, 4), a_{AB}) \rightarrow ((3, 1, 4), a_{AC}) \rightarrow G$	1
	$(S, a_{AB}) \rightarrow ((4, 3, 1), a_{CA}) \rightarrow ((4, 1, 0), a_{AC}) \rightarrow ((0, 1, 4), a_{BC}) \rightarrow ((1, 0, 4), a_{AC}) \rightarrow ((3, 1, 4), a_{AC}) \rightarrow G$	1

CURRICULUM VITAE

PERSONAL INFORMATION

Surname, Name: Aydın, Hüseyin
Nationality: Turkish (TC)
Email: huseyin@ceng.metu.edu.tr

EDUCATION

Degree	Institution	Year of Graduation
M.S.	Dep. of Comp. Eng., METU	2017
B.S.	Dep. of Comp. Eng., METU	2015
High School	Ankara Atatürk Anatolian High School	2010

PROFESSIONAL EXPERIENCE

Year	Place	Enrollment
Oct 2015 - Present	Dep. of Comp. Eng., METU	T. & R. Assistant
Nov 2013 - Dec 2014	SRDC	P.T. Software Developer

PUBLICATIONS

International Conference Publications

H.Aydın, E. Çilden, F. Polat "Using chains of bottleneck transitions to decompose and solve Reinforcement Learning tasks with hidden states", *Future Generation Computer Systems*, Vol. 133, pp.153-168 August, 2022

H.Aydın, E. Çilden, F. Polat, “Compact Frequency Memory for Reinforcement Learning with Hidden States", *In PRIMA 2019,* 22^{th} *International Conference on Principle and Practice of Multi-agent Systems*, October, 2019, Torino, ITALY

H.Aydın, E. Çilden, F. Polat, “Using Transitional Bottlenecks to Improve Learning in Nearest Sequence Memory Algorithm", *In ICTAI 2017,* 29^{th} *International Conference on Tools with Artificial Intelligence*, November, 2017, Boston, MA, USA

M. Gençtürk, E. Alpay, G. B. L. Ertürkmen, A. Doğaç, H. Aydın, “Self-Management of Patients with Severe Arthritis through a Personal Health System: the Turkish Case Study in the PALANTE Project”, *In eChallenges 2014 Conference*, October, 2014, Belfast, IRELAND