Learning Options With Exceptions

A THESIS

submitted by

MUNU SAIRAMESH

for the award of the degree

of

MASTER OF SCIENCE

(by Research)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY MADRAS.

March 2017

THESIS CERTIFICATE

This is to certify that the thesis titled LEARNING OPTIONS WITH EXCEPTIONS, submitted by **MUNU SAIRAMESH**, to the Indian Institute of Technology, Madras, for the award of the degree of **Master of Science**, is a bona fide record of the research work done by him under our supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Prof. B. RAVINDRAN
Research Guide
Associate Professor
Dept. of Computer Science and Engineering
IIT-Madras, 600 036

Place: Chennai

Date: 31st MARCH 2017

ACKNOWLEDGEMENTS

I would like to express my deep gratitude to Professor B. Ravindran, my research supervisor, for his patient guidance, enthusiastic encouragement, and useful critiques of this research work. His willingness to give his time so generously is deeply appreciated. Without his support, I would have not been able to complete this thesis. I had to cope up with the loss of both my parents during my research program. Losing parents means losing a part of oneself as no other bond exists like the one with a parent. Mom and dad, I have no words to acknowledge the sacrifices you made and the dreams you had to let go, just to give me a shot at achieving mine. I would also thank my brother who helped me in this difficult time. I would like to thank my grandmother and uncle Subramani, whose advice has been invaluable. I would like to express my appreciation to Professor Deepak Khemani and Professor P Sreenivasa Kumar for their help. Because of them I was able to stay in the AIDB lab during the nights, when I was homeless. My special thanks to all the members of AIDB lab and RISE lab, who offered me both technical and emotional support. Special thanks to Swapna, who like a sister used to listen to my emotional problems and give valuable advice. Finally I want to thank IIT Madras for providing good infrastructure and beautiful campus.

ABSTRACT

KEYWORDS: Policy, Stochastic, Dynamics, Exception

A long standing goal of AI researchers has been to develop learning approaches that enable knowledge accumulation such that it can easily be reused in solving related tasks. Although several tasks may seem similar, it is not easy to reuse knowledge gained in solving a particular task for another task, however similar they may be. This is due to the minute differences in the dynamics involved in the environment. The usual approach to this problem has been to learn good abstractions by ignoring seemingly irrelevant details. But some of these irrelevant details could be used to represent the changes caused by the minute differences in the environment. In this thesis, we use this approach wherein we treat the differences between two tasks as exceptions of one task's solution when applied to the other.

Policy is a function, mapping state to an action. Normally, when exceptions occur near a state, the new optimal policy is not very different from the previous optimal policy. The changes in the optimal policy are usually restricted to a neighborhood of the affected state. Therefore it makes sense to handle only these exceptions as opposed to relearning a new policy. One of the challenges here is to represent these exceptions so as to accommodate cascading changes while making minimal modifications to the model/policy representation.

Detecting an exception is a key issue in such approaches. In a deterministic environment, identifying these changes is trivial since every action is associated with a fixed output (next state/ response from the environment). A violation of this is an exception. On the other hand, in a stochastic environment, differentiating between the occurrence of an exception and a stochastic change in state is hard.

In this thesis, we look at developing compact models of the dynamics that retain sufficient information to detect exceptions and localize them to small regions of the state space. In addition we also explore questions of how to make minimal modification to the model in order to accommodate cascading exceptions.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS

RL	Reinforcement Learning
HRL	Hierarchical Reinforcement Learning
AI	Artificial Intelligence
MDP	Markov Decision Process
SMDP	Semi Markov Decision Process
POMDP	Partially Observable Markov Decision Process
POSMDP	Partially Observable Semi-Markov decision Processes
RDR	Ripple Down Rule
LSTD	Least-Squares Temporal Difference
TD	Temporal Difference
OWE	Options With Exception
TTM	Transition Time Model
AM	Auxiliary Model
LMN	Landmark Network
ALMN	Auxiliary Landmark Network
TPM	Transition Probability Model

NOTATION

$a(t)$	Action taken at time t
$s(t)$	State at time t
$r(t)$	reward got at time t
π	Policy
π^*	Optimal Policy
β	Termination Condition
o	Option
η_s	total number of times a state s has occurred in all the trajectories.
T_t	Transition instance at time t

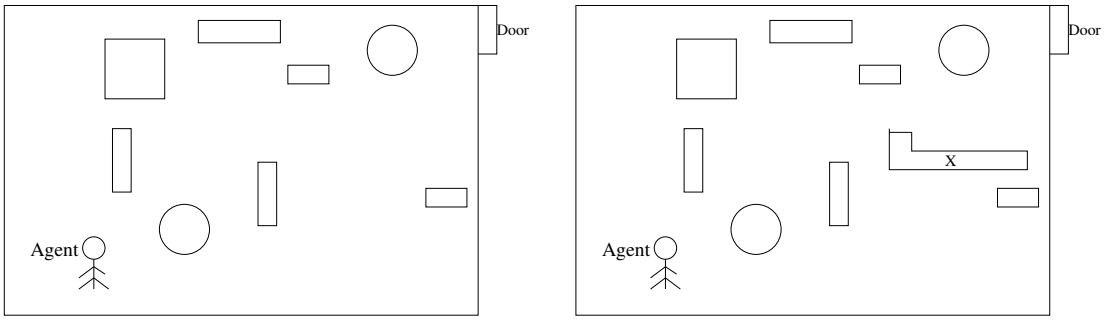
CHAPTER 1

Introduction

The primary objective of *Artificial Intelligence* (AI) is to study human intelligence in order to imitate it in a machine. But in spite of advances made in this area, the notion of intelligence is still not well understood. Hence, the process of seeing, language understanding, reasoning, and other mental process that might appear straightforward and obvious, are far more difficult to imitate in a machine. Though intelligence in generic form is difficult to define, we can understand it better if it is characterized in terms of different functional forms like attention, memory, producing and understanding language, learning, reasoning, problem solving, and decision making. One of the common aspects of intelligence in most of these functional forms is to look for redundancy and then model it as an abstraction, where the notion of redundancy varies across the functional forms.

Machine learning, a branch of AI, focuses on the development of algorithm concerned with the problem of learning. Though there may be different aspects to the problem of learning, the branch of machine learning known as reinforcement learning (RL) is concerned with the development of algorithms to specifically solve the problem of learning based on interaction in order to achieve a goal. *Markov decision processes* or MDPs provide the framework for modeling and solving reinforcement learning problems. Most of the current solution techniques available in the RL community scale poorly with the size of the MDPs. One way of addressing this problem would be to use methods based on *spatio-temporal abstractions*. The abstraction is based on the strategy of "removing irrelevant details". Spatial abstraction is done by the state aggregation methods where the states are grouped together by ignoring distinctions between the states. This is also called as factored representation of the states and the states are represented using the factored variables. The temporal abstraction for example hierarchical RL (HRL) framework (Barto and Mahadevan, 2003), encapsulates the lower level action sequences into a single unit at more abstract level, allowing the decisions to made at coarser level .

Figure 1.1: Agent with different Obstacles en route to the Door(goal)



(a) Without obstacle X (b) With obstacle X

Spatio-temporal abstraction based methods like (MaxQ (Dietterich, 1999), VISA (Jonsson, 2006) and Relativized option (Ravindran and Barto, 2003)) are based on notion of skill specific representation. Dietterich, while proposing the MaxQ framework for value function decomposition has discussed safe state abstraction conditions that would minimally affect the quality of the sub-task policies that are being learned. Jonsson and Barto (Jonsson and Barto, 2000) have looked at the problem of jointly determining spatial and temporal abstractions in factored MDPs. Ravindran and Barto have proposed relativized options that use the notion of partial homomorphisms to determine lossless option specific state representations that satisfy some form of safe state abstraction conditions. Such approaches yield more compact representations of the skill since they do spatial abstractions that are specific to the skill being learned. This allows to considerably cut down the learning time and there by achieve better success rates when solving new problems.

Option (Sutton *et al.*, 1999) is a framework in HRL, where the skills are learned using temporal abstraction. One might need many options to solve a given RL problem. However, the main difficulty in applying these to solve a new set of problems is that problems are seldom encountered in exactly the same form. For example the agent might have to operate in a variety of tasks; or there can be some changes in the environment such as different obstacle configuration (Figure 1.1); or there can be minor changes in the dynamics of the world. In such scenarios one need to address several questions such as whether to use the existing options, modify the existing set of options or to learn new options.

However using the same option again in this situation may be suboptimal or even may result in a failure. Also changing the option may end up in losing the options of

earlier tasks. Further, these changes may create a different spatial abstraction, if the skills are learned with skill specific representation to represent the modified solution thus losing the abstraction of earlier learned skills. Hence we need a framework which can accommodate such changes with minimal variation in the spatial abstraction still maintaining the quality of the solution of the original subtask that the skill was learned on.

The knowledge management community has long dealt with the problem of incremental maintenance of rule bases using the Ripple Down Rule (RDR) (Compton and Jansen, 1990*a*,*b*; Gaines, 1995). The RDR representation allows to minimally modify the existing rule by adding exceptions that are derived from only those training instances that contradict the existing rule. Taking inspiration from this concept, we propose an extension to the options framework called *Options With Exceptions* (OWE). Here, we assumed that new tasks are minor modifications to the earlier existing tasks. In this framework, we also propose a novel method to find the landmarks in the spatial representation of the task. This further allows us to represent the task and provide a novel option management framework that enables us to safely update options. Similar to RDR, this framework allows to modify an option's policy only using instances where the existing policy seems to fail. Thus, in order to maintain minimal variation in the abstraction, changes in the policy are represented by the features extracted from these instances. For example, in the simulated game shown in Figure 1.1a, the introduction of an additional obstacle 'X' (Figure 1.1b) might require changing the policy only around the states with the obstacle. However, this would require us to represent the location of the obstacle as well as the original information that we were considering. In fact, we would like to also accommodate more such changes cascadingly, without degrading the spatial abstraction and any of the solutions learned along the way. While this method may not yield a perfect abstraction, it does help to prevent in making critical changes to the existing abstraction by maintaining the quality of the solution of the original subtask that the skill was learned on.

The recent works like Option-Critic Architecture (Bacon *et al.*, 2017) and Kulkarni *et al.* (2016) learn the policy over the options, and the option policy. However, they neither modify the existing set of options, nor maintain a database like structure for the options.

1.1 Outline of the Thesis

In Chapter 2, we the provide the related work. In Chapter 3, we propose the framework for Options With Exception (OWE). We also provide the notations to be used in the chapter. We give the reasons why methods like UTree McCallum (1995) failed when used as building blocks for OWE. Different building blocks of the framework are given. The first is a structured representation of the option policy that allows for easy addition of exceptions represented by the specific features. The second is the identification of *landmarks* and building a network of landmarks that acts as a compact model of the environment and quickly allows us to narrow down to the region where the option failed. The third is a minimum model for the original policy that allows us to pinpoint the exception state and provides the specific training instances that we need to include in the exception representation. Chapter 4, describes the experiment section, where we provide the results using the grid world and blocks world domain.

In Chapter 5, we summarize and present the possible future work.

CHAPTER 2

Related Work and Background

2.1 Introduction

In this chapter, we provide the background knowledge and the related work needed to formulate the framework of OWE. In the background knowledge we look at reinforcement learning, option framework and ripple down rules.

2.2 Background

2.2.1 Reinforcement Learning

Reinforcement Learning is a computational approach utilized in the area of goal directed learning. It provides a framework to help solve the problem of learning based on interaction. The learner is the agent and decision maker, and interacts with the environment in the form of an action $(a(t))$ at discrete time steps as shown in Figure 2.1. In response, the environment presents a new situation $(s(t))$ to the agent and returns a numerical value called reward $(r(t))$. The agent tries to maximize expected reward over a period of time. Though there are many different ways to formulate the expected reward, here we restrict ourselves to cumulative discounted reward formulation. Hence, the expected reward will be $r(t_1) + \gamma r(t_2) + \gamma^2 r(t_3) + ... + \gamma^{n-1} r(t_n) + ...$, where γ is the discount factor and lies between 0 and 1. Though we have defined the reward in terms of infinite termination time, reinforcement learning is finite horizon problem.

The agent uses the discount factor as a preference for the current reward over the future rewards.

2.2.2 Markov Decision Process

A *Markov decision process* (MDP) provides the simplest framework for modeling and solving reinforcement learning problems. We are using notation from Sutton and Barto (1998). MDP is defined as a tuple $\langle S, A, \Psi, P, R \rangle$, where

(1) S is finite set of states, but can be extended to countable infinite states.

(2) A is finite set of actions that are possible from states in S.

(3) $\Psi \subseteq A \times S$ is set of admissible state action pairs.

(4) P is transition model : $\Psi \times S \rightarrow$ [0,1]. It is denoted as P (s, a, $s^{'}$) = $P^{a}_{s,s'}$ = pr$\{$ $s_{t+1} = s^{'} \mid s_t = s, a_t = a$ $\}$, probability of transitioning from state s to state $s^{'}$ under action a and $\sum_{s'} P(\ s, a, s^{'}) = 1$.

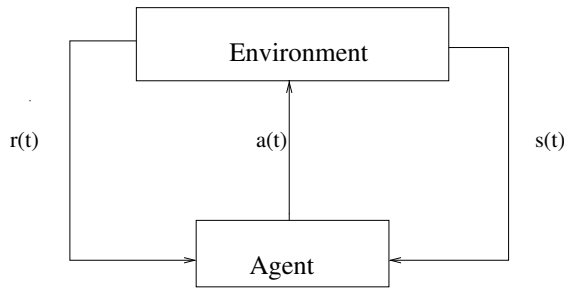
(5) R is reward function : $\Psi \times S \rightarrow \Re$. It is denoted as R(s, a, s'), expected reward for transitioning from the state s to state s' under the action a.

The set of states is described via multi-valued features $X_1, \cdots, X_n$, where each X_i takes value from some finite domain Val(X_i). The state s defines a value $x_i \in$ Val(X_i) for each variable X_i. This representation is called structured representation.

MDPs can be further classified as discrete and continuous MDPs, depending upon the type of values taken by the state and action variables of the MDPs.

The goal of reinforcement learning is to find an optimal policy. A policy π, for an MDP is a mapping $\pi : s \rightarrow a$ that describes the behavior of an agent. Since the MDP dynamics are stationary and as we are not concerned with the finite horizon problem, the policies are assumed to be stationary. Further, a stationary policy can be deterministic

Figure 2.1: Agent Environment Interaction



or stochastic. In a deterministic policy, the same action is picked whenever the agent is in that state, whereas in stochastic policy, actions are picked with some probability from that state. The policy is called ϵ- greedy policy if with probability ϵ the agent explores and with probability $1 - \epsilon$ it acts greedily. While exploring, the agent picks the next state with equal probability.

Further, the environment modeled by the MDP can be deterministic or stochastic depending on the outcome of the action, taken by the agent. In a deterministic environment taking the same action in the same state on two different occasions will result in the same next state. While in a stochastic environment taking the same action in the same state on two different occasions may result in different next states.

Given a policy π, a utility or value function can be associated with each state and is a measure of the expected discounted reward, which the agent will receive following policy π from state s. The value function is denoted as $V^{\pi}(s)$ and written as

$$V^{\pi}(s) = \sum_{a} \pi(s,a) \sum_{s'} P^{a}_{ss'}[R^{a}_{ss'} + \gamma V^{\pi}(s')]$$

called as Bellman equation for state-value function V^{π}.

Optimal value function is denoted as V^{*} and is maximal expected value got by any policy starting at that state. It satisfies Bellman optimality equation.

$$V^{*}(s) = max_{a \in A(s)} \sum_{s'} P^{a}_{ss'}[R^{a}_{ss'} + \gamma V^{*}(s^{'})]$$

Similar to the state-value function, one can define an action-value function. It is denoted as $Q^{\pi}(s,a)$ which means discounted reward which the agent gets from state s, after taking action a and then following policy π. Optimal action-value function denoted as $Q^{*}(s,a)$ is maximal expected value got by starting at state s, taking an action a and then following policy π.

$$Q^{*}(s,a) = \sum_{s'} P^{a}_{ss'}[R^{a}_{ss'} + \gamma max_{a'} Q^{*}(s^{'}, a^{'})]$$

There may be possibly many optimal policies but they share the same optimal state-value function.

$$V(s) = max_{\pi} V^{\pi}(s) \quad \forall s \in S$$

A policy that is greedy with respect to this optimal value function is an optimal policy:

$$\pi^* = greedy(V^*).$$

It is easier to work with the action-value function because given Q^* one does not have to do a one-step-lookahead search to find the optimal action as done with the state-value function. Being greedy relative to optimal action-value function leads to an optimal policy:

$$\pi^* = max_a Q^*(s, a).$$

The sequence of states $\langle s_1, s_2, s_3, \ldots, s_t, s_{t+1}, \ldots \rangle$ are simulated using Q-values by executing the policy $\pi^*(s_t)$ in the state $s_t \in \{s_1, s_2, s_3, \ldots\}$, resulting in the next state s_{t+1} .

2.2.3 Q-Learning

Q-learning (Watkins, 1989) , (Sutton and Barto, 1998) is an off-policy TD control algorithm which is used to update action-value functions:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma max_a Q(s_{t+1}, a) - Q(s_t, a_t)].$$

Its called off-policy because here learned action-value function approximates the optimal action-value function independent of the policy being followed.

2.2.4 Options Framework

Options (Sutton *et al.*, 1999) are used to represent temporal abstraction in reinforcement learning. Given an MDP $\langle S, A, R, P \rangle$, an option is described as $\langle I, \pi, \beta \rangle$, where

1. I : Initiation set where I $\subseteq$ S.

2. $\pi : S \times A \rightarrow [0, 1]$ where π is policy.

3. $\beta : S \rightarrow [0, 1]$ is termination condition.

The option is executed in states $s \in I$ by selecting actions according to policy π. At each state s the option stops with probability $\beta(s)$ or continues with probability $\beta(s) - 1$. Typically the termination condition is deterministic, with the "terminal states" of the option having $\beta(.) = 1$ and the other states having $\beta(.) = 0$. The initiation and termination condition restrict the range over which option policy needs to be defined.

The SMDP Q-learning can be used to update the Q-values. This is given as

$$Q(s, o) \leftarrow Q(s, o) + \alpha[r + \gamma^k max_{o' \in o_{s'}} Q(s', o') - Q(s, o)]$$

where k denotes the number of time steps elapsing between s and s', r denotes the cumulative discounted reward over this time,

In many cases, accurately executing an option's policy does not depend on all state features available to the learning agent. One can do option-specific state abstraction by which irrelevant features specific to each option are ignored leading to compact state representation.

2.2.5 Rules With Exception

Compton and Jansen (1990*a*,*b*); Gaines and Compton (1995) have provided a simple structure which captures the context in which knowledge is obtained from an expert. Usually new rules are added when an expert system misinterprets or fails to interpret a case. For the wrong interpretation, the context is the rule which gave the wrong interpretation. Therefore, the new rule added can fire if the rule that previously fired incorrectly is again satisfied. This is done by including the rule number in the premise of the new rule along with other conditions. In this way, rules are added at the end of the list of rules. This representation of rule is called ripple down rule (RDR).

RDR is very different from the conventional types of representation where the new rules subsume or overlap other rules in the knowledge base. Whereas in RDR, a new rule will be tested after all the rules in the knowledge base have been considered. This

represents the situation exactly the way in which the rules were to be obtained from an expert resulting in the growth of the system very similar to the way in which the knowledge is acquired from an expert.

There are few advantages of using RDR over conventional rules. In the conventional rules, one of the drawbacks is that small changes made can lead to global changes through complex interactions. This in turn makes the development and maintenance of the system tougher. But, if the changes are locally contained in a well-defined manner, the development and maintenance becomes easier. Thus addition of rules to RDR is faster than the conventional rules as it keeps all the rules consistent while incorporating the new knowledge into the system.

The format of rules with exception is

```
if <cond1> then conclusion except
   if <cond2> then ...
else if ...
```

Example:

```
if A and B then rule1              [A, B]
```

If the new case is [A, B, D] then the error occurs in the conclusion which is corrected as:

```
if A and B then rule1 except       [A, B]
   if C then rule2                 [A, B, C]
```

In RDR, rule activation is done in the context of other rule activation. If the antecedent of the parent rule is true and has no dependent rule, then its conclusion is ascertained. However, if it has a dependent rule, then its antecedent is also checked. This goes on until the antecedent of the dependent rule is false. When this occurs, the conclusion of the rule before the rule whose antecedent is false is ascertained. For example,

```
if A and B then rule1 except       [A, B]
   if C then rule2                 [A, B, C]
      if D then rule3              [A, B, C, D]
```

If to the above rule-set, a case [A, B, C] is introduced, the parent rule is checked first. Since it is true, its antecedent is also checked and as it is also true, its dependent antecedent is checked, which is false. Therefore, rule 2 is asserted. In the above rule formation

```
A and B and not C and not D => rule1
A and B and C and not D => rule2
A and B and C and D => rule3
```

2.3 Related Work

Spatial abstraction is important for scaling reinforcement learning. This results in compact representation of the problem. Compact representation not only mean correct representation, it should be easy to learn from sample data and should lead to re usability. Here correct representation means that the quality of solutions of original problem matches with that of the compact representation of the problem. Compact representation of the MDP would require appropriate representation of MDP dynamics. For example, factored MDPs are one approach to represent large MDPs compactly, where the transition probability is represented using dynamic Bayesian network (Dean and Kanazawa, 1989). Further, the ability to find the optimal solutions to the original MDP from the reduced MDP are required. If the optimal solution in the abstract space is also considered optimal in the original space, then the abstraction is called safe state abstraction. Amarel introduced the concept of abstraction in the Missionaries and Cannibles problem (Amarel, 1968). He explored various representations of the problem but selected the one which, while compact, still represents the true nature of the problem.

Skill-specific representation leads to state abstraction where irrelevant features specific to the skill are ignored. For example, while walking out of the room, the features associated with the interior of the room are relevant while the features associated with the exterior of the room are irrelevant. Thus, skill-specific representations lead to compact representations which in turn reduce the computation and accelerate learning. This is because, without state abstraction, one has to learn a separate value function for each state of the world. But the disadvantage is that the abstractions generated could be limited to tasks with similar goals.

MaxQ (Dietterich, 1999) is a HRL method where state abstraction is combined with the decomposition of value functions that are specific to the MaxQ framework. In this framework, MDP M is decomposed into a finite set of subtasks $M0, M1, \ldots, Mn$, where $M0$ is the root task. These subtasks are connected via an acyclic directed graph, implying MaxQ is recursive in nature with the constraint that no subtasks can invoke themselves directly or indirectly. Each subtask can be thought of as discrete-time semi-Markov decision problems where actions taken to solve subtasks consist of primitive actions or policies that solve other subtasks. These primitive actions are at leaf levels and the policies of the subtasks are at intermediate levels. The reward got for executing an action of the subtask is the value function for the executed action. The MaxQ value function for a task consists of two terms; a discounted sum of rewards and a completion function. The discounted sum of rewards is computed until the subtasks terminate. The completion function is updated when the subtask's call is returned and MaxQ learns the value function of all these subtasks simultaneously.

Safe state abstraction is performed by removing the irrelevant state variables within a subtask. The relevant variables are then used to define the value functions and policies which are abstract. To remove the irrelevant variables from the subtask, conditions to identify leaf and subtask irrelevance are used. In the condition identifying leaf irrelevance, a set of state variables X is irrelevant for a primitive action of a MaxQ graph, if all the states, differing only in the value of the state variable X, have the same expected value for the reward function. This leads to a lesser number of relevant variables for the nodes closer to the leaves than the nodes higher up in the MaxQ graph. In the condition identifying subtask irrelevance, a set of state variables Y is irrelevant with respect to subtask relevance, if the state transition probability distribution factors into two sets X and Y for all possible abstract hierarchical policies. The result distribution and termination conditions are only defined for the funnel action. The funnel action is a macro action, which, when used, results in transition from set A to set B, where the size of set A is smaller than the size of set B. In the condition utilizing result distribution irrelevance, a set of state variables Z is irrelevant for the result distribution of an action if the value function of two states have different values under the given policy but has a component of the value function that has the same value for both the states. A termination condition can be used when a subtask guarantees termination of its parent task in a goal state. All these conditions result in safe state abstraction and some of them are

specific to the MaxQ framework.

One of the skill specific learner called VISA algorithm (Jonsson, 2006) decomposes factored MDPs into hierarchies of options. This is done by constructing a causal graph that describes the relation between state variables using Dynamic Bayesian Network model of the factored MDP. Analysis of the state variable changes is used to induce hierarchies of temporal abstractions. The algorithm assumes key state variables to change at different time scales. Temporal abstractions are modeled using option framework. The constraints on state representations along with dependence on a DBN model are drawbacks of the algorithm.

There are various tree-based approaches, which lead to skill-specific state abstraction by learning the compact models of the value function of the abstract states. These abstract states are represented as leaves of the tree. The G-algorithm (Chapman and Kaelbling, 1991) follows a similar approach, applied to the continuous state space, wherein it discretizes the attributes depending upon the distance from the agent and then uses those attributes to discretize the state space by building tree-like structures. It starts with a single node, making no state distinction, but after a fixed number of steps and for each state distinction that could be made, fringe nodes are added beneath the root node. The addition of these distinctions are based on the statistical test on future discounted rewards, i.e., comparisons are made between the immediate rewards, stored in root node and future discounted rewards, stored in fringe node. If the comparison is found useful, then the fringe nodes are upgraded to leaf nodes, with new fringe nodes added beneath the leaf nodes. The statistical test used is t-test. The statistics and Q-value are stored and updated in the leaf nodes.

The parti-game algorithm (Moore and Atkeson, 1995) is similar to the G-algorithm but restricted to high-dimensional continuous deterministic environments. It starts with two cells, with one cell covering the goal region and the other cell covering the rest of the continuous state space. The greedy controller then traverses the cell covering the goal region, and if it fails to transit to the other cell, it then refines the partition. In this process, it builds a tree where the leaves represent the direction. The controller then uses this tree to reach the goal region. The parti-game algorithm is an instance-based learner.

The U-Tree (McCallum, 1995) extended the idea of the G-algorithm to instance-

based learning to solve large partially observable Markov decision problem (POMDP). Normally, while solving POMDP, one has to look at the history of state space until the problem becomes Markov, thereby converting POMDP to MDP. Similarly, the U-Tree looks back in time and records the history of state and action until the state is Markov. This results in partitioning of the set $(A \times O)^H$ where H is the history's index, representing a certain number of experiences, O is a set of observation variables representing the observable features of the state and A is a set of actions. Therefore, the U-Tree considers state distinction with respect to the actions and observation variables over H, resulting in a factored representation of the set $(A \times O)^H$.

The learning agent in the U-Tree encodes the information regarding the agent-environment interaction as a transition instance, which is then added to the end of the transition instance chain. In transition instance $T_t =< T_{t-1}, a_{t-1}, r_t, s_t > s_t$,r_t are observation vectors and rewards at time step t and T_{t-1}, a_{t-1} are transition instances and the action taken at time step $t-1$. Transition instances, whose actions and observation variables match that of the labeled internal nodes of the U-Tree are then stored in the corresponding leaves of the U-Tree. As the leaves of the U-Tree are treated as states, an estimate of $P(s'/s,a)$, $R(s,a)$ and $Q(s,a)$ are stored in the leaves. For each action a taken by the agent, the U-Tree performs one value iteration sweep and updates $Q(s,a)$ stored in the leaf node s and also updates the estimates $P(s'/s,a)$ and $R(s,a)$.

Similar to the G-algorithm, the U-Tree algorithm also starts with a single node, adding fringe nodes beneath the leaf nodes. However, the U-Tree differs from the G-algorithm on the number of split variables considered for the fringe nodes. While the G-algorithm considers one variable as a split variable, the U-Tree considers a set consisting of observation variables and actions which are not already in the path from the root to leaf node. The fringe node is then expanded by all possible permutations of new variables selected from this set to depth z. The maximum value of z depends on the history index H and controls the number of new distinctions that can be made. Depending upon the new distinctions, a fringe node gets its set of instances from its parent leaf node. The U-tree then updates the estimates of $R(s,a)$, $P(s'/s,a)$ for each fringe node, and performs value iteration to calculate the $Q(s,a)$ for each fringe node s, action a pair.

The U-Tree calculates the expected future discounted reward values, $Q(T_i) = Q(T_i) =$

$r_{t+1} + \gamma \sum_{s'} P(s'/s, a)u(s')$ for each instance T_i, for all the fringe nodes and leaf nodes. A non-parametric statistical test, Kolmogorov-Smirinov test, compares the cumulative distributions of the $Q(T_i)$ values of the leaf node's policy with that of the fringe node's policy. This is done to look for the state distinction (fringe node) whose Q-values differ significantly from that of its parent leaf node. If the distributions have enough difference to be statistically significant, the new distinctions are made permanent and the fringe nodes are upgraded to official leaf nodes. This process continues until no more utile distinctions can be made to the tree.

The problem with the U-Tree is that it is computationally highly intensive as a statistical test has to be done for each fringe node / leaf node pair.Also, the number of fringe nodes is a factorial of the number of variables which are selected as a new variable for splitting. Furthermore, there is no concept of roll-back as the distinctions made in the U-Tree are hard-coded, i.e., once the split is made, it cannot be changed.

The U-Tree was further extended by Jonsson and Barto (2000), called the Hierarchical U-Tree or H-Tree. They used the H-Tree to solve problems in partially observable semi-Markov decision processes (POSMDP). POSMDP is got by adding temporarily extended actions to the action set of partially observable decision processes (POMDP). The H-Tree uses options to model activity representing temporarily extended actions. It forms the partition of the set $(op \times O)^H$ where op is the option, i.e., it considers the distinction over the options and observations which are recorded during the execution of the options over the last H time step. The H-Tree represents each option as a U-Tree. Thus, it can represent a single primitive action or a hierarchy of options. In this way, it performs option-specific state abstraction.

The differences between the U-Tree and the H-Tree are very limited. Transition instances used to train the H-Tree are extensions of the instances of the U-Tree. The only difference in the transition instance of the H-Tree when compared to the U-Tree is in reward. In H-Tree, reward is the sum of discounted rewards received during the execution of the option as compared to rewards received during the execution of actions stored in the U-Tree. Transition instances of the H-Tree also store an extra term called the duration of the execution of the option. The H-Tree uses SMDP Q-learning to perform value iteration to estimate $Q(s, op)$ of each leaf-option pair (s, op). One interesting point highlighted by the authors was about hierarchical memory. Generally, there

is no method to select the history index H in the U-Tree. Although the H-Tree doesn't define a method to select H, hierarchical organizations make it easier to remember important decision points. This is because of transition instances which act like memory. The H-Tree framework was also extended to MDP, with H's value recorded as 1.

The continous U-Tree (Uther and Veloso, 1998) was an extension of the discrete U-Tree to continuous valued features. In this method, discretization of continuous values is not needed since it is done while the U-Tree is being built. The agent views the world as a continuous state called sensory input. Sensory input is a vector of real value. The U-tree leads to a factored state space where the abstract state is an area in sensory input space. It starts with a single node depicting no discretization of sensory input. A binary tree is then formed where each internal node is labeled with an attribute and decides the way sensory input has to be divided. Leaves of this binary tree correspond to abstract states. The transition instance is recorded similar to the discrete U-tree and is represented as $\langle I, a, I', r\rangle$, where I, I' are sensory input, a is action and r is reward. This transition instance is then used to generate data-points represented as tuples $\langle I, a, q(I, a)\rangle$ where $q(I, a)$ is the expected reward of performing this transition. It updates the action value function, $q(I, a) = r + \gamma V(s')$ where s' is the abstract state in which the transition instances are stored. The continuous U-Tree performed two statistical tests, Kolmogorov-Smirinov and sum-squared error, to compare the distributions. Once the tree was built, the result was a standard discrete reinforcement learning problem and finding $Q(s, a)$ for every abstract state s was similar to that done by the U-Tree.

The idea of the U-Tree looking back in time for a required number of time steps to make a current decision was used to formalize Hierarchical Short-Term Memory (Hernandez-gardiol and Mahadevan, 2000). This solved large partially-observable sequential decision tasks and specially perceptually-aliased tasks. It used short-term memory to make decisions at each decision-making point which was represented as the Nearest Sequence Memory (McCallum, 1995) or the U-Tree. The Nearest Sequence Memory algorithm was used to evaluate the closeness between the current state and the past states experienced by the agent according to the matched length of the suffix chain preceding the current state, and the action was selected from the longest match. Combining hierarchical activity with short-term memory allowed the agent to add memory at more informative levels of abstraction. A Hierarchical Abstract machine was used

to implement the hierarchy of activities while SMDP Q-learning was used to update the Q-value of the current state. They found that multi-level Hierarchical Short-Term Memory, where each abstract level had memory, outperformed the flat and hierarchical methods which did not use memory because memory-based SMDP Q-learning methods rapidly propagated delayed rewards across long decision sequences.

A relativized option framework (Ravindran and Barto, 2003) is a compact representation where the family of related subtasks are represented using a single subtask whose solution gives the solution for the entire family of related subtasks. A relativized option does symmetry-based reductions, by abstracting away the repeated portions in the tasks by specific subtasks called option-MDP. Let M be an MDP and $M^{'}$ be the reduced model got by minimization of M. Homomorphism mapping from M to $M^{'}$ is a mapping where equivalent states and actions of M are mapped to the same state and the same action respectively in $M^{'}$. The transition probability of $M^{'}$ is the sum of the transition probability of states of M that are mapped to the same state in $M^{'}$. A state-action pair that has the same image under homomorphism has the same expected reward. Subtasks restrict the state-action space and when viewed with respect to the original MDP, homomorphism over these restricted spaces is called partial homomorphism. This partial homomorphic image models the similarity among the family of related subtasks and is called an option MDP as it is a partial homomorphic image of an MDP with respect to options. The option MDP is the reduced representation of the subtask. S, A and P of the option MDP are same as the original MDP but for reward functions, which are chosen based on option subtasks. As the relativized option is defined over the option MDP, separate state abstraction is done for each option. To learn relativized options, it is assumed that the option MDP and the transformation are given beforehand.

The traditional RL framework assumes that the 'critic' evaluates the agent's behavior as a part of the environment. In intrinsically motivated RL (Singh *et al.*, 2004) the environment is split into external and internal environment. The critic is now part of the internal environment. The inspiration behind the framework is to encapsulate the organism motivation system within the internal environment. The rewards are extrinsic and intrinsic, given by the external and internal environment respectively. The authors also assume that the salient events are given beforehand, and the reward given to achieve an event is intrinsically proportional to the error in prediction of the option.

Higher level options use the options corresponding to the 'salient' event options using intra option learning methods (McGovern, 2002). Limitation of this work is that the notion of 'interesting and novelty' is much more complex than their assumption of salient events. Since the salient events are assumed to be known beforehand, there is no notion of option discovery.

Linear options (Sorg and Singh, 2010) is a linear representation of the option. Linear method are used to approximate the value function as a linear combination of the state feature vectors. In this work the authors represent the state feature vector in a n dimensional space. A linear option is a option where $I \in \mathbb{R}^n$, $\pi : \mathbb{R}^n \times A \rightarrow [0, 1]$ and $\beta : \mathbb{R}^n \rightarrow [0, 1]$. The behavioral policy evaluation is done using SMDP-LSTD, which is an extension of MDP-LSTD method for policy evaluation of primitive actions. Although linear options provide better generalization over states, their use has so far been explore mainly in LSTD style methods, which have quadratic complexity. This makes their convergence slow as compared to the conventional linear complexity of the TD algorithms.

Kulkarni *et al.* (2016) extends the work (Singh *et al.*, 2004), where they use function approximation (deep neural network) to represent the option policies (controller) and the policy over the options (meta controller). The meta controller finds the policy over the goals by maximizing the expected future extrinsic rewards. The goals here can be thought of as the sub goals. The meta controller hands the control to the controller, and the controller uses the current state and the goal to find the policy that maximizes the expected future intrinsic reward. When the controller reaches the goal, it hands the control back to the meta controller, which again calls the controller with the current state and the new goal. Limitations here are that the subgoals are given beforehand to the option learners.

In Option-Critic Architecture (Bacon *et al.*, 2017), the authors overcome the limitations of the work (Kulkarni *et al.*, 2016). They use policy gradient methods to learn the option policies and the termination conditions of the option. The idea of policy gradient method is to increase the probability of the actions which would increase the expected discounted reward. The option goal which is a subgoal in a task is a state with the property of the 'bottleneck' (Simsek *et al.*, 2005; Menache *et al.*, 2002). So the option policy would be to optimize the reward to achieve the subgoal but will ignore the

effect of those actions in hindsight. The authors come up with the "Intra-option policy gradient theorem" which actually says that the local changes (primitive actions) are going to have an effect on the global expected discounted reward. Hence this theorem says take better primitive actions more often in the option. In the "Termination gradient theorem", the intuition is that when the advantage function is large then lengthen the option or cut down the option when the option is suboptimal. The advantage function with respect to an action measures the difference between expected return by taking action from the state with the expected return from the state. The limitation of this work is the assumption that all options apply everywhere.

The recent works like Option-Critic Architecture (Bacon *et al.*, 2017) and Kulkarni *et al.* (2016) learn both the policy over the options, and the option policy. However, they do not modify the existing option, whereas we develop an option representation that can accomodate small changes in the option policy. The changes do not modify the original policy thus maintaining a database like structure for the options.

2.3.1 Conclusion

In this chapter, we looked at the background work and the related work needed for the next chapter "Options with exception". In the next chapter, we propose the framework of options with exceptions and the explain different building blocks of this framework. We also provide the experimental setup and the results to evaluate the framework.

CHAPTER 3

Options With Exceptions

3.1 Introduction

In this chapter, we propose the Options With Exceptions (OWE) framework and address several key issues towards its formulation. Let us consider a partitioned gridworld domain with partitions ('P1', 'P2', 'P3' and 'P4') and an agent as shown in Figure 3.1. The actions available to the agent are North, South, East, West. An action would result in the transition to the expected state with a probability of 0.9. It results in a transition to any one of the unintended directions with a probability of 0.1. A executed trajectory from the start position to goal position using the optimal policy is indicated by the line through 'P1' and 'P2'. Let an obstacle 'O' be placed near the partition 'P1' as shown in Figure 3.2, leading to the failure of the policy (action 'West') which is called an *exception*. Usually, when an exception happens, a new policy is learnt between the start position and the goal. However, since the changes in the policy are restricted to a few states, one can look at learning a policy from the point where the original policy failed, state 'A', to the nearest point where the two policies agree which is indicated as 'L2'. A executed trajectory of this new policy is represented using the dashed line in Figure 3.2. Similarly when an obstacle 'R' is placed near partition 'P2' blocking the original policy 'West', the new policy is learnt, which is denoted using double dotted line in Figure 3.3.

One key issue to be addressed in such a framework is "how to detect when an exception has occurred". In a deterministic environment this is easy since each action has only one outcome and we can readily detect when the expected outcome does not occur. In a stochastic environment identifying such a failure of expectation is non-trivial. One would have to learn a probabilistic model of the expected dynamics under the current policy and then devise a test to detect when there is significant deviation from the expectation. While one could take recourse to many of the model estimation methods, building and maintaining a database of such models for all possible situations is time

Figure 3.1: A partitioned gridworld example with the partitions P1, P2, P3 and P4. The agent learns the policy to reach the goal from the start.

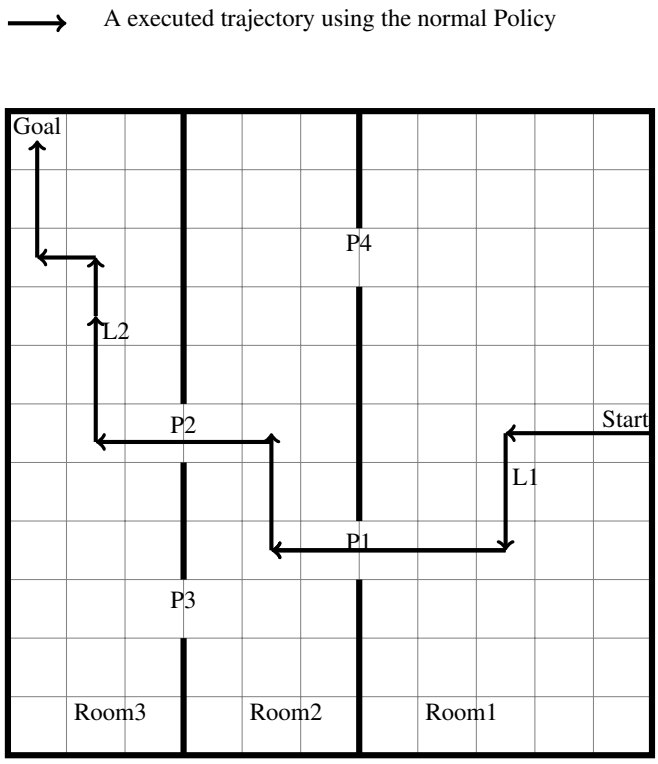


Figure 3.2: A partitioned gridworld example showing the exception caused by the introduction of an obstacle O near partition P1.

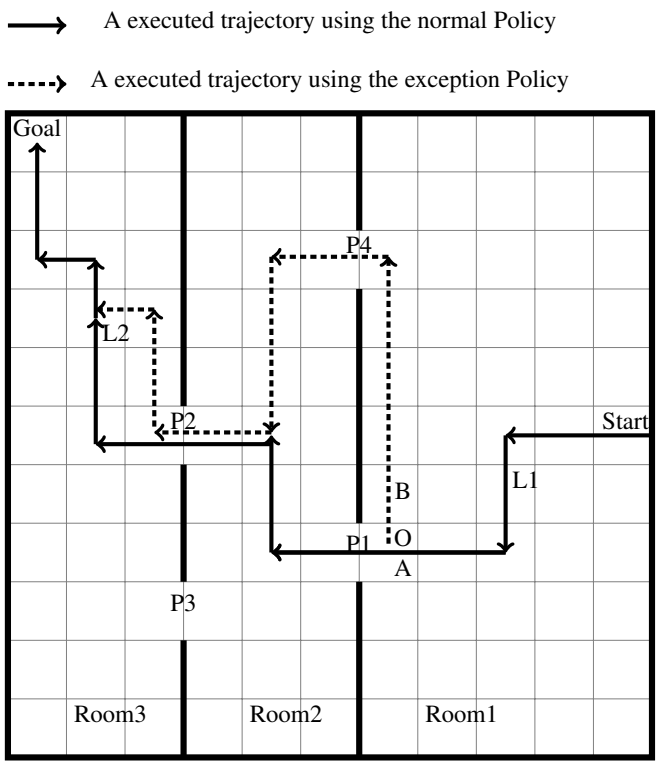
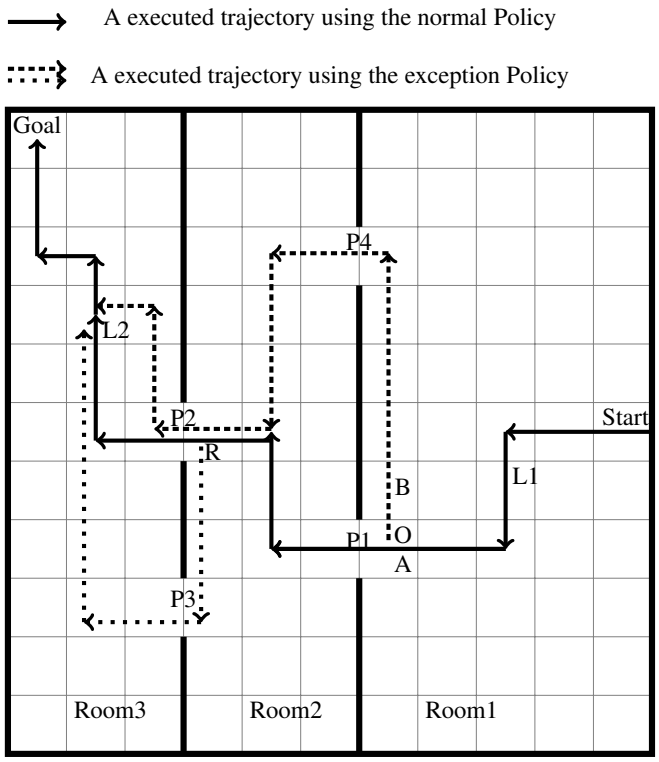


Figure 3.3: A partitioned gridworld example showing the exception caused by the introduction of the obstacles O and R near partition P1 and P2 respectively.



consuming and not feasible. So we look at developing compact models of the dynamics that retain sufficient information to detect exceptions and localize them to small regions of the state space. In addition we also explore questions of how to make minimal modification to the model in order to accommodate cascading exceptions.

We look at problem definition in section 3.2, followed by choice of algorithm in section 3.3. In section 3.4, we look at Transition Time Model. Finally in section 3.5 we conclude.

3.2 Problem Definition

An option $O = \langle I, \pi, \beta \rangle$ is a policy fragment that represents a solution to a frequent subproblem encountered in a domain. Often, it is hard to find subproblems that are exactly the same. Though I and β might be same for the subproblem, the policy π might be different. Thus using the same option again could be suboptimal. These differences in π, however small, need to be accounted for in the reused policy. The goal is to develop an option representation that can accommodate small changes in π while

maintaining the original π. Further, the representation should be able to accommodate cascading changes to π.

3.3 Framework and Choice of Algorithms

Let us consider the partitioned gridworld shown in Figure (3.1), with the assumption of a deterministic environment in the room. In a deterministic environment, taking the same action in the same state on two different occasions will result in the same next state. For example at the state 'A', the action 'East' taken by the agent would always result in a state that represents the passage through 'P1' leading to the other side of the room. But when an obstacle 'O' is placed near the partition 'P1' as shown in Figure 3.2, the action 'East' would result in a failure. This failure is easy to detect as the agent fails to cross the partition 'P1'. However, one might still need to model the result of the action to detect the failure. This model can be of a simple form, where one needs to store the next state for each state-action pair that one encounters on the path from start to goal.

In a stochastic environment, identifying the failure mentioned in the above paragraph is non-trivial. Because in a stochastic environment taking the same action in the same state on two different occasions may result in different next states. Further, there are three basic types of outcomes possible due to a failure in a stochastic domain. These are lack of transitions, transitions to arbitrary neighboring states, and transitions to arbitrary non-neighboring states. For example in Figure (3.2), placing the object 'O' near 'P1' will result in the failure, lack of a change of state. In order to identify these patterns of outcomes, one needs to store the transition dynamics for each action. Though there might be different models to capture the transition dynamics, the basic model, similar to deterministic environment would be to store a set of next states along with their respective transition probabilities for each state-action pair.

However, building and maintaining a database of such models for all possible state-action pairs is time consuming and not feasible for stochastic environment. Thus we need a framework which can compactly store the information regarding the transition dynamics and is useful in detecting failures and localizing them to small regions of the state space. We also need to add and safely update the policies. Hence we propose a

framework called *Transition Time Model*, which addresses all the above issues. In the next section, we enumerate the difficulties faced in our initial attempts while implementing OWE using U-Tree.

3.3.1 Discussion

Initially, we looked at using U-Tree representation (McCallum, 1995) to model OWE as it does skill specific state representation along with on-line learning. In addition, a U-Tree is essentially a suffix tree, which is a variant of decision trees and supports incremental learning of representations. However, U-Trees do not represent the policies explicitly and at any point use the action with the maximum Q value in the abstract state corresponding to the current leaf of the U-Tree.

Hence even local changes in policy result in global changes, due to the change in the value functions. This results in the original policy being adversely affected. Furthermore it also changes the state abstraction extensively resulting in unnecessarily complex representations. For example in Figure 3.3, the states whose policies fail, are predominantly near the place, where the obstacle 'R' is placed. But when the U-Tree learns the new policy through the partition 'P3', the value function for most of the states will change. This will lead to splits in the U-Tree for all the states represented by the U-Tree in 'Room3' and 'Room2', though the policies are not changed in these states.

Thus we have to look at alternate mechanisms for implementing the OWE framework. We use an explicit representation for the policy using suffix trees in conjunction with a "Transition Time Model" which we will explain in the next section.

3.3.2 Notation

***Definition*:** A *trajectory* is a sequence of states $\langle s_1, s_2, s_3 \ldots \rangle$ through which an agent moves.

***Definition*:** *st-trajectory* is a trajectory where the agent starts from the start state and follows the optimal policy to reach the destination. Q-value table is used to simulate this trajectory as described in the section 2.2.2.

***Definition*:** *st-exception-trajectory* is a trajectory where the agent starts from the start

state and follows the optimal policy but the trajectory might contain states where the optimal policies fail. The *st-exception-trajectory* , unlike *st-trajectory*, might never reach the destination. Hence, we need to specify the threshold $thresh_{no-state}$, which is used to terminate the trajectory when the length of the trajectory is greater than the threshold.

***Definition*:** η_s is the total number of times a state s has occurred in all the trajectories.

***Definition*:** A *transition instance* is the tuple $\langle s_t, a_t, s_{t+1} \rangle$, where s_{t+1} is the resultant state when the agent follows the action a_t in current state s_t.

***Definition*:** A *transition chain* is a sequence of transition instances where the resultant state (s_{t+1}) in the transition instance (T_t) is same as the current state of the next transition instance (T_{t+1}) i.e., { T_1=$\langle$ s_1, a_1, s_2 $\rangle$, T_2=$\langle$ s_2, a_2, s_3 $\rangle$ ... T_t=$\langle$ s_t, a_t, s_{t+1} $\rangle$, T_{t+1}=$\langle$ s_{t+1}, a_{t+1}, s_{t+2} $\rangle$ ... }.

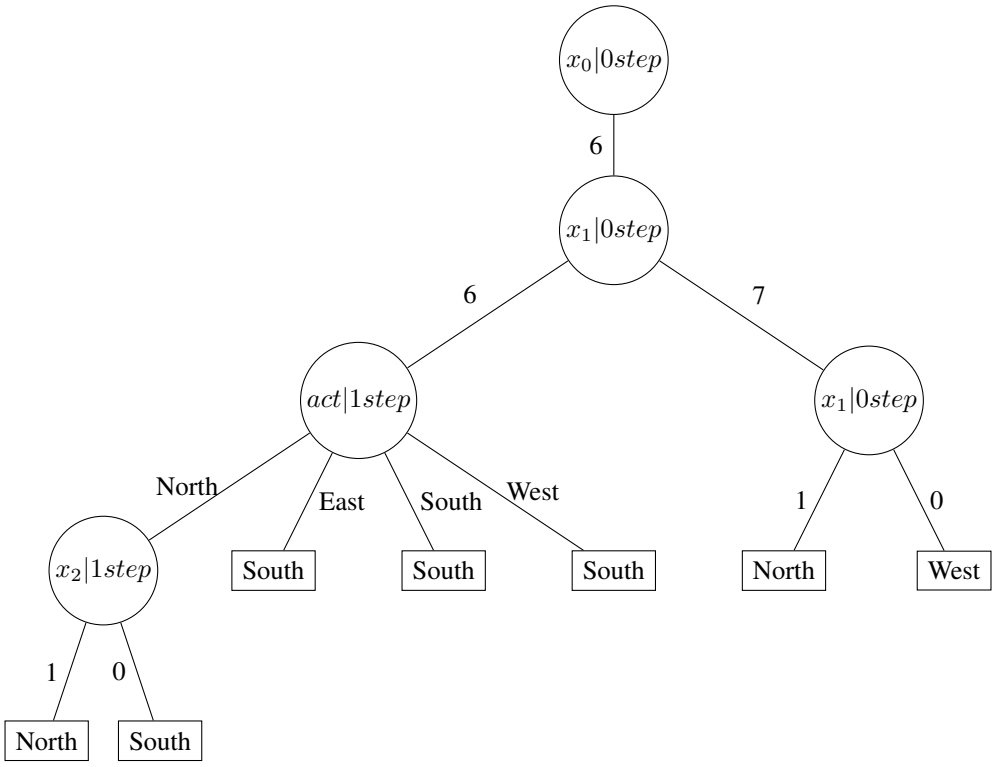
3.3.3 Policy Representation

One of the key issues in OWE framework is to address the problem of policy representation, so that modifications can be made without affecting the base policy. Further, we also need context specific updation of the policies. In this section, we address these issues and propose a representation for policies.

Usually, the policies are implicitly represented using Q-values. But if the agent learns a new policy, then modifying the policy will result in changing the value function. For example in Figure 3.2, the representation of the new policy between the states 'A' and 'L1' through partitions 'P1' and 'P2' might affect the representation of the original policy if it is represented using Q-values. However the values of the value function take a longer time and more experience to converge to useful values. Hence, a representation is required that is computationally efficient. Further, the changes to the policies should not alter the existing policies and also the changes should result in a representation with minimum variation from the existing one. Here by minimum variation we mean that required changes are made to the affected part of the representation and rest of the representation is left untouched.

Though there are more than one method to do it, we look at a method in which the

Figure 3.4: An example of a suffix tree, where the current state $\langle x_0 = 6, x_1 = 6, x_2 = 0\rangle$(H = 0) of the agent is 'B' and the previous state $\langle x_0 = 6, x_1 = 7, x_2 = 1\rangle$(H = 1) of the agent is 'A' with an obstacle 'O'. The action taken at state 'B' is dependent on the previous action $act|1step$. The features of a state for example the i^{th} feature of a state, is represented with the history information(H) as $feature_i|Hstep$. Similary the action is represented with the history H as $act|Hstep$.



representation of the new policy is conditioned on the features that caused the changes. For example in Figure 3.2, new policy between the states 'A' and 'L1' through partitions 'P1' and 'P2' can be represented by conditioning it on the feature corresponding to obstacle 'O'. Let the initial policy and the new policy represent the policy of the state before and after the obstacle is introduced in the domain. Note that in Figure (3.1,3.2,3.3), the states can be represented as a tuple $\langle x_0, x_1, x_2 \rangle$, where x_0, x_1 indicates the position of the agent in the domain and x_2 indicate the presence of the obstacle. Therefore, the state and the position of the agent are the same. Though an obstacle will affect the policy of the state containing it, it will also affect the policies of the neighboring states. For example in Figure 3.2, let 'B' be the state which is above the state containing the obstacle 'O'. Initial policy of the agent at state 'B' is south. However as the new policy at state 'B' is north, the change in policy should be represented using the feature of 'O' as shown in Figure 3.4. Similarly the neighbors of 'B' also have to represent the changes in their policy. In general we might also require a small amount of history in representing these changes. But as we move farther away from point 'O', the policies might not get affected. Thus the representation should be able to make the required changes to the affected parts of the existing representation and leave the rest of representation unaffected. This can be achieved using a suffix tree.

Definition : A suffix tree T for an m-character string S is a rooted directed tree with exactly m leaves where

(1) Each internal node, other than the root, has at least two children and each edge is labeled with a non-empty substring of S.

(2) No two edges out of a node can have edge-labels which begin with the same character.

(3) The key feature of the suffix tree is that for any leaf i, the concatenation of the edge-labels on the path from the root to leaf i exactly spells out the suffix of S that starts at position i i.e., $S[i..m]$.

The suffix tree provides a rich representation that supports context specific updation and thereby accommodating the required changes in a minimal way. Our policy representation uses a modified form of a suffix tree, where the past states and actions are stored as internal nodes of the suffix tree, and the leaf nodes store the policy.

Definition : A *history space* $(\tilde{A} \times \tilde{S})^H$=\{ s_{t-H}, . . ., a_{t-2}, s_{t-1}, a_{t-1} \} where H is the history index denoting the number of time steps in the past from the current time, is a sequence of past states $\tilde{S}$ = \{ s_{t-H}, . . ., s_{t-2}, s_{t-1}\} visited and past actions $\tilde{A}$ = \{ a_{t-H}, . . . a_{t-2}, a_{t-1} \} taken by the agent.

For example, if the agent's current state is 'B', then the history space one time step in the past is $(\tilde{A} \times \tilde{S})$, where the variable $\tilde{S}$ is the state with 'O' and the variable $\tilde{A}$ is the action in that state. Suffix can be thought of as a sequence in the history space. For example in the previous example, the suffix will be the values of $\tilde{S}$ and $\tilde{A}$. We have included the action in the history space because it helps to capture the context of learning a new policy in a more generic manner, thus allowing us to condition on it along with the state features. This formulation is very useful as some of the states which are farther away from 'B' might represent its new policy conditioned on the policies of its neighboring states. In general if the history space is $(\tilde{A} \times \tilde{S})^H$ then the internal nodes of the suffix tree are from the set \{ s_{t-H}, . . ., a_{t-2}, s_{t-1}, a_{t-1} \}. Another advantage which suffix tree provides is that of incremental learning as done with decision tress.

The transition instances required to train the suffix tree are provided with the help of the transition time model which is explained in the subsequent section.

3.4 Transition Time Model

One of the chief components of the *Option With Exceptions* (OWE) framework is a *Transition Time Model* that records the transition time between certain distinguished states, hereafter known as *landmarks*.

In the partitioned gridworld example , as shown in Figure (3.1,3.2) the initial policy of the agent whose task is to reach the goal from the start, is represented by the lines through partitions 'P1' and 'P2'. Though the policy fails at the state 'A' because of the obstacle 'O', the policy of the states along the line segment from the start to state 'A' agrees with the initial policy. Similarly policies of the states along the line from the 'L2' to goal also agree. Thus one has to come up with a method, to find the nearest position where the two policies agree, from the point where the policies failed. In order to pinpoint such a point, one has to compare the two policies at each position, which

is computationally intensive and unrealistic. Hence, we look at a generic method of landmarks and networks formed by interconnecting these landmarks. The idea behind this is to partition the *st-trajectory* in terms of landmarks, thus providing the abstraction of the trajectory.

Definition : A *Region* $\langle s_t, s_{t+1}, s_{t+2} \ldots s_{t+m} \rangle$ is a subsequence of the trajectory $\langle s_1, s_2, s_3 \ldots s_t, s_{t+1}, s_{t+2} \ldots s_{t+m}, s_{t+m+1} \ldots \rangle$.

Definition : A *Path* $\langle s_1, s_2, s_3 \ldots s_n \rangle$ is a trajectory that starts at the start state s_1 and terminates at the goal state s_n.

Definition : *Landmarks* are places or sites that are visited often and are used to either find the way backward or forward. Using the same notion, we define a landmark for a region as a state that is mostly visited when paths are traversed (successful or unsuccessful) through that region. The notion of a subgoal is different from that of a landmark.

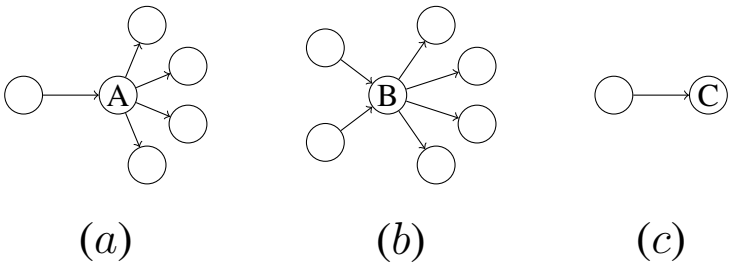
Definition : *Subgoal* is a funnel state (Asadi and Huber, 2005) or a bottleneck (McGovern and Barto, 2001) state that is visited on all successful paths but not on unsuccessful paths. Thus a subgoal may be a landmark whereas vice-versa may not be true. For example, in a two room connected by a door, the state corresponding to the doorway can be a subgoal and landmark but the state near some distinguished object might be a landmark, but not a subgoal.

Definition : *Exception region* is a region of *st-exception-trajectory* where the policy fails. Landmarks help us to approximately find the region where the policy disagrees by identifying the landmarks which cover this region of trajectory. In Figure 3.2, the exception region lies within landmark pair $L1$ and $L2$ and we can assume the point at which the policy agrees as $L2$. This allows to work abstractly, reducing the computation cost. It also provides an option management framework that allows us to safely update the options.

3.4.1 Identification of Landmarks

We propose two heuristics to identify landmarks:

Figure 3.5: Spiked states



Spiked State Method While traveling, most people try to associate landmarks with junctions. A junction would aid in allowing people to move between paths.

Definition : *Spiked State* s_t is a state with the property $\eta_{s_{t-1}} < \eta_{s_t}$ and $\eta_{s_t} > \eta_{s_{t+1}}$ 3.3.2, where s_{t-1}, s_t, s_{t+1} are the states that are visited by the agent at successive time instances. Hence the spiked state has the highest frequency of visits across paths when compared to other neighboring states. It normally occurs at the junction of many paths.

Definition : *Co-occurring Spiked States* are set of states that spiked in the same path.

Spiked state heuristic uses the notion of spiked state with the constraint that the paths are sampled with random start state. From the set of spiked states, we choose those states that co-occcured with the spiked states in a maximum number of paths. We use this heuristic to identifiy landmarks in the entire domain because it uses a random start state to sample the *st-trajectory*, thus allowing us to capture the actual notion of landmarks.

One of the problems with this heuristic is that it requires the domain to have large number of different paths through the states which enables us to find the required pattern to identify the spiked states. For example in Figure 3.5, states A and B will satisfy the criteria of spiked state compared to state C as they have multiple different paths passing through them. The states A or B should have large number of neighbors in order to be highly connected. But most of the RL domains do not satisfy this criterion, thus leading to an alternate method called mean-to-variance ratio.

Mean-to-Variance Ratio Method This heuristic uses a simpler computational approach when compared to a spiked state. Here we find the landmarks only on the *st-*

trajectory. Paths are assumed to start from the same start state. Let t_s^i be the time at which the state s is encountered while traversing the path i. The time at which each state s occurs in the path along with its mean, $\mu_s = \frac{\sum_i t_s^i}{Totalnumberofpaths}$ and variance $\sigma_s = \sum_i (t_s^i - \mu_s)^2$ is extracted for all the states in the paths. The states which are closer to start are almost always visited hence they have low variance as compared to the other states in the path. Similarly the states near the goal have high mean. The states are then sorted in an increasing order of mean-to-variance ratio. By this the states near the goal state and the start state are avoided. This analysis is done for the states which occur in nearly all the paths. From this list, landmarks are then selected, satisfying the criteria of minimum and maximum distance between them. This leads to uniform distribution of landmark on *st-trajectory*.

3.4.2 Construction of the Transition Time Model

Once the landmarks have been identified, we build a Transition Time Model to capture the connectivity information between the landmarks. To build a Transition Time Model, landmarks need to be uniformly distributed throughout the *st-trajectory*. The model also works with non-uniformly distributed landmarks, but then it will lead to non-uniform division of *st-trajectory*. This will further lead to learning varied length of policies between the landmarks, though exception is caused by small change in the policies. In order to localize an exception within a small region and to make minimal changes in the policies, we need the landmarks to be distributed uniformly. We specify a bound on the distance between landmarks. Small region can be described in terms of the bounded distance between landmarks having a lower and an upper limit.

Definition : A *Landmark Network* denoted as l_{graph} is a labeled directed acyclic graph described by the tuple $\langle l_{list}, l_{edge}, L_f \rangle$ where l_{list} is a list of landmarks, and $l_{edge} \subseteq l_{list} \times l_{list}$ is the set of edges; $(u, v) \in l_{edge}$ implies that landmark v is followed directly after the landmark u in many *st-trajectories*. The edge (u, v) is present in l_{graph} if the percentage of *st-trajectories* in which v directly follows u is greater than the threshold $thresh_{no-traj}$. The label function, L_f assigns to each element

in l_{edge} a list of real numbers

$$\begin{aligned} L_f: \quad l_{edge} \quad &\rightarrow \mathbb{R}^{|paths(u,v)|} \\ (u,v) \quad &\mapsto (t_1^{u,v}, t_2^{u,v}, t_3^{u,v} \ldots, t_{|paths(u,v)|}^{u,v}). \end{aligned}$$

Here $\mid paths(u,v) \mid$ represents the number of different paths between landmarks u and v leading to multiple average transition times between the same pair of landmarks. This happens because, after an exception, the modification of the landmark network leads to the addition of new paths between landmarks which may be already connected. The average transition time along the i^{th} path between the landmarks u and v is denoted by $t_i^{u,v}$. The average transition time between any two landmarks u and v is the average of the difference t_v^i - t_u^i, as discussed in Section 3.4.1, and is extracted for all the landmark pairs that are part of the same path *i*

Definition : A Transition Time Model consists of the tuple $\langle l_{graph}, T_f \rangle$ where T_f maps an edge in l_{graph} to a list of thresholds, i.e.,

$$\begin{aligned} T_f: \quad l_{edge} \quad &\rightarrow \mathbb{R}^{|paths(u,v)|} \\ (u,v) \quad &\mapsto rv_1^{u,v}, rv_2^{u,v}, rv_3^{u,v}, \cdots, rv^{|paths(u,v)|}. \end{aligned}$$

The threshold $rv_i^{u,v}$ is the "relative variability" of the average transition time for the path i between the landmarks u and v and is estimated by $\frac{\sigma_i^{u,v}}{t_i^{u,v}} \times 100$. The standard deviation($\sigma_i^{u,v}$) is calculated for the transition time along the i^{th} path.

3.4.3 Identification of the Exception Region

The landmark network in the Transition Time Model helps to find the exception region. Let the current Transition Time Model be denoted as TTM. An RL agent follows a policy corresponding to some exception which results in many exception trajectories. If a landmark $l^i \in l_{list}$ is not visited within a *reasonable* deviation of the transition time stored in the Transition Time Model from its predecessor landmark $l^{i-1} \in l_{list}$, then a failure occurs. Let $l_{list}^{ex} \subseteq l_{list}$ be the set of landmarks that were visited in *st-exception-trajectories*.

Definition : An Auxiliary Landmark Network is a tuple $\langle l^{ex}_{set}, l^{ex}_{edge}, L^{ex}_{f} \rangle$ where $l^{ex}_{edge} \subseteq l^{ex}_{set} \times l^{ex}_{set}$ is the set of edges; $(u, v) \in l^{ex}_{edge}$ implies that landmark v followed directly after the landmark u in many *st-exception-trajectories*. Edge (u, v) is considered between the landmarks u and v if the percentage of *st-exception-trajectories* in which v directly follows u is greater than the threshold $thresh_{no-traj}$. The Auxiliary Model is populated with the average transition time($\tilde{t}^{u,v}$) extracted from the *st-exception-trajectories*. The label function L^{ex}_{f} assigns to l^{ex}_{edge} a real number

$$\begin{aligned} L^{ex}_{f} : \quad l^{ex}_{edge} \quad &\rightarrow \mathbb{R} \\ (u, v) \quad &\mapsto \{\tilde{t}^{u,v}\}. \end{aligned}$$

Definition : An Auxiliary Model consists of $\langle l_{graph}, A_f \rangle$, where the label function A_f assigns to each element in l_{edge} a label from the set $\{RC, ER, NR\}$.

$$\begin{aligned} A_f : \quad l_{edge} \quad &\rightarrow \{RC, ER, NR\} \\ (u, v) \quad &\mapsto \{RC, ER\} \ if \ (u, v) \in l_{edge} \cap l^{ex}_{edge} \\ (u, v) \quad &\mapsto \{NR\} \ otherwise. \end{aligned}$$

Here RC, ER and NR denotes *"Reached Correctly"*, *"Error in Reaching"* and *"Not Reachable"* respectively. Let the Auxiliary Model be denoted as AM. In order to find the pair of landmarks between which exception occurs, we need to compare the average transition time provided by TTM with the average transition time stored in AM for each pair of landmarks $(u, v) \in l_{edge}$. There are two different scenarios which an RL agent might encounter while transitioning between the two landmarks u and v. In the first scenario, the agent does not reach or reaches landmark v occasionally. This is defined as failure in *reachability criteria*. The reachability criterion is satisfied if the proportion of *st-exception-trajectories* in which v directly follows u is higher than the threshold $thresh_{no-traj}$. If the *reachability criteria* between u and v is not satisfied, the label function A_f of the Auxiliary Model AM labels the edge (u, v) as "***NR***".

For example in Figure 3.6, let the $thresh_{no-traj}$ be .5 and 100 trajectories reach landmark o. If the number of trajectories reaching the landmark p is 55 and the rest 45 reaches landmark q, then (a) the *reachability criteria* is satisfied by the landmark *p*,

Figure 3.6: Identification of exception landmark: possible combinations of labelled edges.

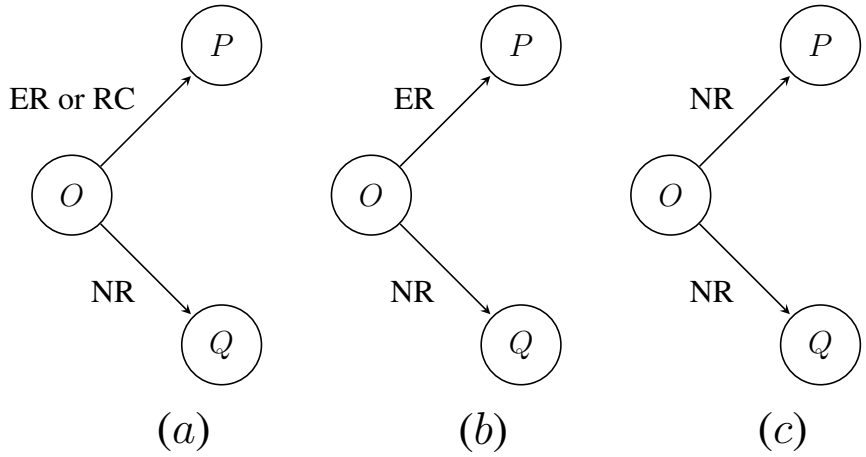
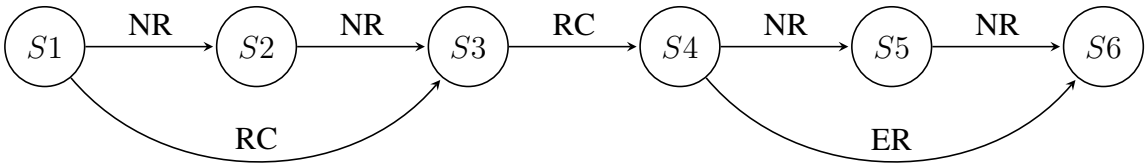


Figure 3.7: Identification of exception landmark: example of a Transition Time Model



implying the proportion of trajectories in which the RL agent reaches landmark *p* from landmark *o* is greater than the $thresh_{no-traj}$. But the landmark *q* is labelled as "*NR*" as the propotion of trajectories from *o* to *q* is lower than the $thresh_{no-traj}$.

In the second scenario, while transitioning from landmark u to v, an agent may satisfy the *reachability criteria* but then v might not be visited within a *reasonable* deviation of the transition time stored in the TTM from u. Since we have stored the average transition time for each path between the landmarks as a list in TTM, we need to compute the percentage increase between $\tilde{t}^{u,v}$ and $t_i^{u,v}$ for all paths . This is done by computing

$$CIt_i^{u,v} = \frac{|\tilde{t}^{u,v} - t_i^{u,v}|^2}{t_i^{u,v}} \times 100$$

called *Change In Landmark Distance* ($CIt_i^{u,v}$) for the i^{th} path between the landmarks u and v. Each $CIt_i^{u,v}$ corresponds to a different path i between the two landmarks. Existence of $CIt_i^{u,v}$ value less than or equal to $rv_i^{u,v}$ for any $i \leq \mid paths(u,v) \mid$ assures the nonexistence of the exception in the region covered by the corresponding landmark.

This is called as *variability criteria* where

$$\begin{aligned} A_f(u,v) &= RC \ if \ \exists i \leq \mid paths(u,v) \mid \ such \ that \ CIt_i^{u,v} \leq rv_i^{u,v} \\ &= ER \ otherwise. \end{aligned}$$

For example, Figure 3.6 (b) shows the failure of *variability criteria* where $A_f(o,p) =$ "ER".

In our implementation, we initialize $A_f(u,v) =$ "*NR*", $\forall (u,v) \in l_{list} \times l_{list}$. The *variability criteria* for each $(u,v) \in l_{edge}^{ex}$ is checked only if the *reachability criteria* is satisfied by the landmark pair (u,v). Hence, the *variability criteria* results in an edge label being changed from "*NR*" to either "*RC*" or "*ER*" by the label function A_f, depending upon the success or the failure of the *variability criteria*.

The Auxiliary Model with the edges labeled by the label function A_f is further used to find the exception region. Nonexistence of the exception is assured if there exists a unique path consisting of a sequence of edges labeled as "*RC*" between the start and the terminal landmarks. Exception is reported when one of the outgoing edges from the current landmark is labeled as "*ER*" as shown in Figure 3.6 (b) or all the outgoing edges are labeled as "*NR*" as shown in Figure 3.6 (c). Such a landmark is then called an exception landmark ($l_{exception}$) and the sequence of states visited under the current policy starting from the exception landmark is called exception region. For example, in Figure 3.7, s_4 is the exception landmark in the sequence of landmarks s_1, s_2, s_3, s_4, s_5 and s_6. The exception region, starts from the landmark s_4 and might end up at landmark s_5 or s_6.

Thus, Transition Time Model and Auxillary Model provides an approximate location of an exception, further leading to identification of the exact point where the exception happens.

3.4.4 Identification of Exception State

The Transition Time Model identifies the exception region where the current policy fails. An exception can happen because of changes in the dynamics of the domain which are caused by either the introduction of a new object in the domain or a change

in the values of domain's features. For example in Figure 3.2, the obstacle 'O' can be thought of as a new object in the domain. Similarly, if the door is present at 'A', with value 0 representing the *door* as open and 1 as closed, then the change in dynamics is caused by the feature *door*. In either case, there are some features of the exception state which reflect the change in the domain.

Definition : *Exception state* is a state where the first exception happens and it is denoted as $s_{exception}$.

This characterization of exception state is needed because of cascading nature of failures. In order to pinpoint such a state, we need to come up with a method which can do a local search in the exception region to find the exception state. Further, the features of the exception state are used by suffix tree to represent the new policy by conditioning it on these features. For example in Figure 3.2, the exception state is a state where the feature corresponding to obstacle 'O' has a value *present* whereas the rest of states has the value *absent* for the same feature.

In order to find the exception state in the exception region, one needs to compare the policy at each state in the exception region. But since the domain can be stochastic, the policy needs to be stored along with the set of resultant states and their respective transition probability, in a model called *Transition Probability Model*.

Definition : The Transition Probablity model TPM consists of $P_{s,s'}^{\pi(s)}$ (2.2.2) where s is the state, π is the policy at s, and s' is the resultant state.

There are three basic types of outcomes due to an exception caused by the change in the dynamics of the domain. These are (a) lack of transitions, (b) transitions to arbitrary neighboring states, and (c) transitions to arbitrary non-neighboring states, as shown in Figure 3.8. Other outcomes of exceptions that are possible are combinations of these. Thus, in order to find the exception state, we search for these three patterns of outcomes using the transition model. Let the *Transition Probability Model* learned from the transition instance derived using the *st-trajectory* be denoted as $TPM_{st-traj}$. This stores the state transition model for states with the assumption that exception has not yet happened. Similarly the *Transition Probability Model* learned from the transition instance derived using the *st-exception–trajectory* is denoted as $TPM_{st-ex-traj}$, which

stores the state transition model after an exception has happened. Since the exception region starts from the exception landmark, we iterate sequentially over the *st-exception-trajectories* starting from the exception landmark.

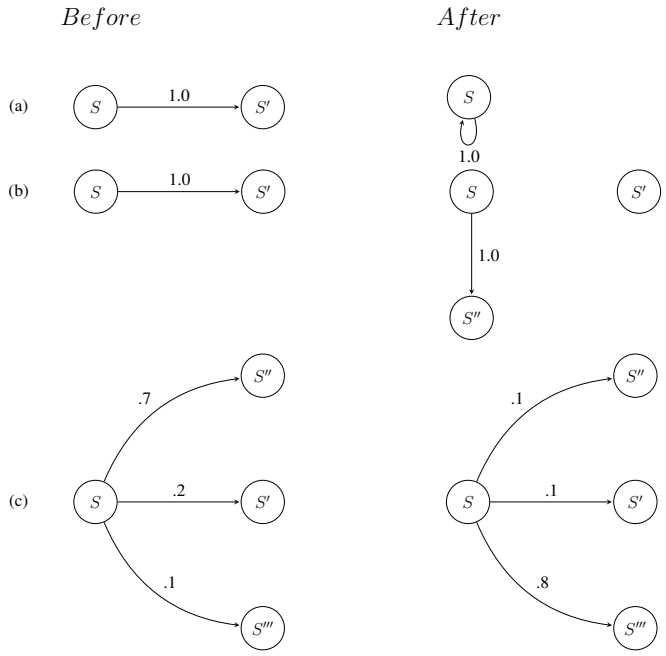
Let the entry corresponding to some $(s, \pi(s), s') \in TPM_{st-traj}$ be denoted as $p_{s'}$ and corresponding entry in $TPM_{st-ex-traj}$ be denoted as $p'_{s'}$. For all possible transitioned states s' from the state s under the policy $\pi(s)$, there is entry in $TPM_{st-traj}$. The distribution of s' can be modelled as a binomial distribution where one bin consists of state s' and the other bin consists of rest of transitioned states. Let $\sigma_{s'} = \sqrt{n * p_{s'} * (1 - p_{s'})}$ be the standard deviation of s', where n is total number of samples corresponding to transition from state s under policy π stored in $TPM_{st-traj}$. Similarly n' is total number of samples corresponding to transition from state s under policy π stored in $TPM_{st-ex-traj}$. If $|n * p_{s'} - n' * p'_{s'}| \geq \sigma_{s'}$ for any s', then failure is ascertained for the state s. We sample same number of trajectories before and after exception in order to construct $TPM_{st-traj}$ and $TPM_{st-ex-traj}$. In certain cases where for some resultant state s', the entry $(s, \pi(s), s')$ might either be present in $TPM_{st-traj}$ or $TPM_{st-ex-traj}$ but not in both. In such cases we assume the transition probability to be 0 where it is not present and then compare the corresponding entries. The state s where the model fails is called *exception state*.

After the exception state has been identified, the new policy has to be learnt and the Transition Time Model has to be updated.

3.4.5 Subtask Option and Updation of Transition Time Model

In the previous section we looked at a method which gave the first state where the current policy deviates from expected behaviour, called exception state. In order to learn a new policy, i.e., exception policy, we also need to know the point where the current policy and the exception policy agrees after the failure. We identify these points with existing landmarks used in the definition of the transition time model. Since it is difficult to identify these landmarks apriori, we come up with a list of possible landmarks called potential destination landmarks (l_{dest}), that could be points where the policy agrees. One way of defining l_{dest} is to consider all landmarks in the ordered list, l_{list} which come after $l_{exception}$ as potential destination landmarks.

Figure 3.8: Different types of exceptions. Lack of transistion is shown as (a), Transistion to arbitary non neighbour is shown as (b) and Transistion to arbitary neighbour is shown as (c)



An exception policy is learned between the exception state and some potential destination landmark using Q-learning. The destination landmark chosen by the RL agent would be the one to which it finds the least expensive path from the exception state. Let $O_{s_{exception} \rightarrow l_{term}}$ be the subtask option denoting $\langle I, \pi, \beta \rangle$, where π is the exception policy and $\beta(l_{term})$ = 1 when $l_{term} \in l_{dest}$. Multiple executions of the option policy are stored as transition chain which are then used to retrain the suffix tree corresponding to the current policy to represent the exception policy. The exception policy is used to update the $TPM_{st-traj}$ by adding the transition model for the exception state and its neighbours. During the updation of the $TPM_{st-traj}$, we follow a procedure similar to the one described in Section 3.4.4. If in the state s, the deviation of transistion probability is beyond the standard deviation then we update the entries of the state s in $TPM_{st-traj}$. The l_{graph} is updated to include the new $t^{u,v}_{|paths(u,v)|+1}$ between the exception landmark u and the destination landmark v. This is done by following the old policy from the exception landmark to the exception state and then further using exception policy to reach the l_{term}. This new path might add edges between landmarks which were not connected earlier or might add new transition times between already connected landmarks. The $rv^{u,v}_{|paths(u,v)|+1}$ is also calculated and stored in the TTM.

3.4.6 Description of algorithm

Algorithm 1 Pseudo-code for OWE learning agent

1: Let π be the optimal policy learned using Q-learning between the start and the goal state
2: [lm_{list}, LMN, TTM, $TPM_{st-traj}$, t_{sfx}] = Initialize-Data-structures(π)
3: l_{ex} = start state // Initialize exception landmark
4: **while** $l_{ex} != null$ **do**
5: *st-exception-trajectories* = $\emptyset$ //List to store *st-exception-trajectories*
6: Simulate policy using t_{sfx} and store it in *st-exception-trajectories*
7: Identify the landmarks $\in lm_{list}$, which *st-exception-trajectories* traverses through and call it as lm^{ex}_{set}
8: ALMN = Initialize-Auxiliary-Model(*st-exception-trajectories*, lm^{ex}_{set}) //Initialize data structure ALMN
9: $l_{exception}$ = Identify-the-exception-landmark(TTM, ALMN)
10: **if** $l_{exception}$!= null **then**
11: $s_{exception}$ = Identify-the-exception-state($TM_{st-traj}$, *st-exception-trajectories*, $l_{exception}$)
12: **if** $s_{exception}$!= null **then**
13: Learn the new subtask option, $O_{s_{exception} \rightarrow l_{term} \in l_{dest}}$ using Q-learning
14: Update TTM, $TPM_{st-traj}$ with the *st-trajectories* simulated using Q-table
15: Update t_{sfx} using *st-trajectories*
16: **end if**
17: **end if**
18: **end while**

The *Pseudo-code for the OWE learning agent* [Algorithm 1] gives the detailed overview of the learning agent. The agent learns the optimal policy π, between the start and goal state using the Q-learning algorithm. The procedure *Initialize-Data-structures* initializes the data structure for the implementation of the landmark network (LMN), transition time model (TTM) and the transition model ($TPM_{st-traj}$). The variable $l_{exception}$ stores the exception landmark and is initialized with the start state. In order to build the auxiliary landmark network (ALMN), the trajectories are simulated using the policy represented by the suffix tree t_{sfx} and these trajectories are stored in the list *st-exception-trajectories* as shown in lines 5-6. The landmarks from the list lm_{list} that are visited in the *st-exception-trajectories* are stored in the list lm^{ex}_{set}. These are further used along with the *st-exception-trajectories* to initialize the auxiliary landmark network in the procedure *Initialize-Auxiliary-Model* as shown in line 8. The exception landmark is found using the procedure *Identify-the-exception-landmark* and *if statement* is entered in line 10. Within the *if statement* the exception region is explored by the agent to find the exception state which is stored in the variable $s_{exception}$. This is

done by comparing the transition probabilities of the states in the procedure *Identify-the-exception-state* in line 11. If the exception state is found, then the agent learns the new policy ($O_{s_{exception} \rightarrow l_{term} \in l_{dest}}$) between the $s_{exception}$ and the nearest landmark (l_{term}) from the list of landmarks (l_{dest}) which comes after the exception landmark in the list lm_{set}. The agent then updates the TTM, $TPM_{st-traj}$ and the t_{sfx} with the *st-trajectories* simulated using Q-table as shown in lines 14-15.

Algorithm 2 Initialize-Data-structures(π)

Input: policy: π

Output: lm_{list}: landmarks, LMN: landmark network, TTM: transition time model, $TPM_{st-traj}$: transition probability model, suffix tree: t_{sfx}

1: *st-trajectories* = $\emptyset$ //List to store *st-exception-trajectories*
2: Simulate policy π and store it in *st-trajectories*
3: Create transition instance chain using *st-trajectories*
4: Learn suffix tree t_{sfx} using the transition instance chain
5: lm_{list} = Identify-landmarks(*st-trajectories*).
6: temp = hashtable // Data structure used to initialize landmark network and transition time model
7: LMN = Initialize-Landmark network(*st-trajectories*, lm_{list}, temp) //Initialize data structure LMN
8: TTM = Initialize-Transition-Time-Model(LMN, temp)//Initialize data structure TTM
9: $TPM_{st-traj}$ = transition model for each $(state, action)$ in *st-trajectories*.
10: return lm_{list}, LMN, TTM, $TPM_{st-traj}$, t_{sfx}

The procedure *Initialize-Data-structures* [Algorithm 2], initializes the data structure for the policy representation, landmark network, transition time model and the transition model. The agent simulates the trajectories by following the policy π and stores them in the *st-trajectories* shown in line 2. The transition instance is created for each tuple $\langle s, a \rangle$ in the trajectory and the transition instance chain is formed using these transition instances. The transition instance chain is used to learn the suffix tree (t_{sfx}) which represents the agents policies. The *st-trajectories* are also used to find the landmarks using the *spiked state method* or *mean to variance ratio method* and the landmarks are stored as a list lm_{list} as shown in line 5. The temporary hash table *temp* is used to store the transition time in procedure *Initialize-Landmark network* and the temp is further used in procedure *Initialize-Transition-Time-Model*. The LMN, TTM and $TPM_{st-traj}$ are initialized in lines 7-9.

The algorithm *Identify-landmarks* [Algorithm 3] selects the final list of landmarks

Algorithm 3 Identify-landmarks(*st-trajectories*)

Input: policy: *st-trajectories*

Output: lm_{list}:Landmark list.

1: minimum-distance-candidate-landmarks = $\emptyset$ //List to store candidate landmarks.
2: Add start and end states to minimum-distance-candidate-landmarks.
3: From the list of candidate landmarks, select the landmark with the increasing order of mean to variance ratio.
4: Add the selected landmark to the list of minimum-distance-candidate-landmarks if its distance is greater than all the existing landmarks of the list.
5: The maximum distance between the landmark constraint could be checked on the minimum-distance-candidate-landmarks and removing those landmarks which does not satisfy the constraint.

from the list of candidate landmarks. We here use mean to variance method to get the list of candiate landmarks. This problem can be thought as an subset selection problem that has to satisfy certain constraints. The first constraint is that the landmarks are selected in a increasing order of mean to variance ratio. The second constraint is that the distance between the two consecutive landmarks is bounded between the minimum and the maximum limit. The problem can be also formulated as an minimization problem. Let $f(S) = \sum_{n=1}^{N} p_j$, where $j \in S$ for $j = 1 \ldots N$, where p_j is the priority value associated with the candidate landmark j. The minimization problem would be min $f(S)$ s.t $B_1 \leq |i - j| \leq B_2$ where $i, j \in$ candidate landmarks set and B_1 and B_2 are lower and upper bound respectively. Since the problem is hard, we used greedy approach to find the solution as described in the algorithm. Sometimes the approach fails and the landmarks violate either the minimum or maximum limit. But the failures are local and are not cascading.

The algorithm *Initialize-Landmark network* [Algorithm 4] describes the procedure to initialize the LMN. We start by initializing the list of landmarks in LMN with the landmarks in lm_{list}. The list l_{edge} stores the edges between the landmarks and is initialized as an empty list. We also initialize hash table L_f to store the average transition time between the landmarks. The for loop in lines 2-8 adds the edge between the landmarks. After initialization we iterate through each trajectory stored in the *st-trajectories* and extract the transition time between the landmarks u and v as shown in lines 9-15. The transition time between the landmarks u and v is stored in the hash table temp with the key (u, v). The edge list (LMN.l_{edge}) stored in the LMN is pruned by removing the edge corresponding to the landmarks u and v if number of trajectories in which u is followed

Algorithm 4 Initialize-Landmark network(*st-trajectories*, lm_{list}, temp)

Input: *st-trajectories*: list of trajectories, lm_{list}: list of landmarks, temp: hash table to store data

Output: LMN: landmark network.

LMN = < l_{list} = lm_{list}, l_{edge} = $\emptyset$, L_f = empty hash table > // Data structure for landmark network
for u in LMN.l_{list} **do**
  for v in LMN.l_{list} **do**
    if index(u in LMN.l_{list}) $\leq$ index(v in LMN.l_{list}) **then**
      add (u, v) to LMN.l_{edge}
    end if
  end for
end for
for $i = 1$ to number(*st-trajectories*) **do**
  for $(u, v) \in$ LMN.l_{edge} **do**
    if landmark u followed by v occurs in *st-trajectories*(i) **then**
      add transition time between (u, v) to temp(u, v)
    end if
  end for
end for
for $(u, v) \in$ LMN.l_{edge} **do**
  if count(temp(u, v)) > $thresh_{no-traj}$ **then**
    add average(temp(u, v)) to LMN.$L_f(u, v)$ // Average transition time stored in $L_f((u, v))$.
  else
    remove (u, v) from LMN.l_{edge}
  end if
end for
return LMN

by v does not satisfy the threshold criteria ($thresh_{no-traj}$). The average transition time is calculated for the remaining edges by using the hash table temp. This value is stored in the hash table L_f with the key (u, v).

Algorithm 5 Initialize-Transition-Time-Model(LMN, temp)

Input: LMN: landmark network, temp: hash table to store data
Output: TTM: transition Time model

1: TTM = < LMN, A_f = empty hash table > // Data structure for TTM
2: **for** $(u, v) \in$ TTM.LMN.l_{edge} **do**
3: calculate rv using TTM.LMN.$L_f(u, v)$ and temp(u, v) // rv is relative variability
4: add rv to the list TTM.$T_f(u, v)$
5: **end for**
6: return TTM

The algorithm *Initialize-Transition-Time-Model* [Algorithm 5] calculates the relative variability of the transition time and stores in the TTM. We start by initializing the TTM with the LMN and a empty hash table A_f. The list of relative variability of the transition time between the landmarks are stored in A_f. The for loop in lines 2-5 calculates the relative variability using the value in TTM.LMN.L_f and the data stored in the temp for each edge in the LMN.l_{edge}. This is stored in the TTM.T_f.

The procedure to initialize the ALMN is described in the algorithm *Initialize-Auxiliary-Landmark-Network* [Algorithm 6]. In line 1, the landmark list of the ALMN is initialized with the landmarks stored in the lm^{ex}_{set}. This is followed by the addition of edges between the landmarks stored in the lm^{ex}_{set}. The hash table L^{ex}_f stores the average transition time between the landmarks and is initialized lines 3-7. After the initialization we iterate through each trajectory stored in the *st-exception-trajectories* and extract the transition time between the landmarks u and v as shown in lines 8-14. The transition time between u and v is stored in the hash table temp-ex with the key (u, v). The for loop in lines 15-21 prunes the edge list (ALMN.l^{ex}_{edge}) by removing the edge between the landmarks u and v if the number of trajectories in which the landmark u is followed by v does not satisfy the threshold criteria. The average transition time is calculated for the remaining edges using the hash table temp-ex and is stored in the hash table L^{ex}_f with the corresponding key (u, v).

Identify-the-exception-landmark [Algorithm 7] is a procedure to find the exception

Algorithm 6 Initialize-Auxiliary-Landmark-Network(ALMN, *st-exception-trajectories*, lm^{ex}_{set})

Input: *st-exception-trajectories*: list of exception trajectories, lm^{ex}_{set}: list of landmarks
Output: ALMN: auxiliary landmark network.

1: ALMN = < l^{ex}_{list} = lm^{ex}_{set}, l^{ex}_{edge} = $\emptyset$, L^{ex}_{f} = empty hash table > // Data structure for auxiliary landmark network.
2: temp-ex = empty hash table //To store list of transition time
3: **for** u in ALMN.l^{ex}_{set} **do**
4: **for** v in ALMN.l^{ex}_{set} **do**
5: add (u, v) to ALMN.l^{ex}_{edge}
6: **end for**
7: **end for**
8: **for** $i = 1$ to number(*st-exception-trajectories*) **do**
9: **for** $(u, v) \in$ ALMN.l^{ex}_{edge} **do**
10: **if** landmark u followed by v occurs in *st-trajectories*(i) **then**
11: add transition time between (u, v) to temp-ex(u, v)
12: **end if**
13: **end for**
14: **end for**
15: **for** $(u, v) \in$ ALMN.l^{ex}_{edge} **do**
16: **if** count(temp(u, v)) > $thresh_{no-traj}$ **then**
17: add average(temp-ex(u, v)) to ALMN.$L^{ex}_{f}(u, v)$ // Average transition time stored in $L^{ex}_{f}(u, v)$.
18: **else**
19: remove (u, v) from ALMN.l^{ex}_{edge}
20: **end if**
21: **end for**
22: return ALMN

Algorithm 7 Identify-the-exception-landmark(TTM, ALMN)

Input: TTM: Transition time model, ALMN: Auxiliary landmark network
Output: landmark: exception landmark

AM = < TTM.LMN, A_f = Hashtable > // Data structure for Auxiliary model.
for (u, v) in TTM.LMN.l_{edge} **do**
  AM.$A_f(u, v)$ = "NR"
end for
for $(u, v) \in$ ALMN.l_{edge} **do**
  found = false
  for $i = 1$ to$\|paths(u, v)\|$ **do**
    Extract $t_i^{u,v}$ value from the TTM.LMN.$L_f(u, v)$ and $rv_i^{u,v}$ from TTM.T_f(u,v)
    $\tilde{t}^{u,v}$ = ALMN. $L_f^{ex}(u, v)$
    $CIt_i^{u,v} = \frac{|\tilde{t}^{u,v} - t_i^{u,v}|}{t_i^{u,v}} \times 100$
    if $CIt_i^{u,v} \leq rv_i^{u,v}$ **then**
      AM.$A_f(u, v)$ = "RC"
      found = true
      break
    end if
  end for
  if not found **then**
    AM.$A_f(u, v)$ = "ER"
  end if
end for
Search in the order of landmarks in TTM.LMN.l_{list} for the landmark whose outgoing edge is labeled as "ER" or whose all the outgoing edges are labeled as "NR"

if landmark not found **then**
  return null
else
  return landmark
end if

landmark. The AM is initialized with the landmark network (TTM.LMN) in line 1. The for loop in lines 2-4 initializes A_f with the "NR" for each edge in the landmark network (TTM.LMN.l_{edge}). The outer for loop in line 5 iterates over the edge list stored in the auxiliary landmark network (ALMN.l_{edge}). The inner for loop in lines 7-16 iterate over the number of paths (*paths(u,v)*) and for each path i, the average transition time($t_i^{u,v}$ and $\tilde{t_i}^{u,v}$) between the landmarks u and v is extracted from the landmark network (TTM.LMN.$L_f(u,v)$) and the auxiliary landmark network (ALMN.$L_f^{ex}(u,v)$). The change in the transition time $CIt_i^{u,v}$ is calculated and is compared with the i^{th} component ($rv_i^{u,v}$) of the relative variability stored in the transition time model (TTM.$T_f(u,v)$) in lines 10-11. The hash table $AM.A_f(u,v)$ is modified to "RC" in line 12 when the $CIt_i^{u,v}$ is lesser then the $rv_i^{u,v}$ indicating the existence of i^{th} path between u and v in the auxiliary landmark network. If no component ($rv_i^{u,v}$) of the relative variability is lesser than the $CIt_i^{u,v}$, then the $AM.A_f(u,v)$ is modified to "ER" indicating the existence of erroneous i^{th} path between u and v in the auxiliary landmark network as shown in line 18. The landmark whose outgoing edge is labeled with "ER" or all the outgoing edges are labeled with "NR" is searched using the $AM.A_f$.

Algorithm 8 Identify-the-exception-state($TPM_{st-traj}$, *st-exception-trajectories*, $l_{exception}$)

Input: $TPM_{st-traj}$: transition probability model, *st-exception-trajectories*: list of exception trajectories,
Output: exception state: s

1: $TPM_{st-ex-traj}$ = transition model for $(s,a) \in$ *st-exception-trajectories* .
2: **for** $(s,a) \in$ *st-exception-trajectories* **do**
3: Compare the entry of (s,a) in $TPM_{st-traj}$ with that of in $TPM_{st-ex-traj}$ for the failure of transition probability.
4: **if** found **then**
5: return s
6: **end if**
7: **end for**
8: return null

In the procedure *Identify-the-exception-state* [Algorithm 8], the transition model ($TPM_{st-traj}$) is compared with the transition model ($TPM_{st-ex-traj}$) extracted from the *st-exception-trajectories* as shown in line 1. The for loop in lines 3-8 iterate over each state, action (s,a) pair in the *st-exception-trajectories*. The first state in the *st-exception-trajectories* where the transition models fails is returned as *exception state*.

3.5 Conclusion

In this chapter, we proposed a framework in which exceptions can be added to already learned options without disturbing the original policies. While the framework succeeds in identifying small changes required to the options, we still have some distance to go before building a complete skill management system. In the next chapter we describe the experimental domains and the results of our framework on these domains.

CHAPTER 4

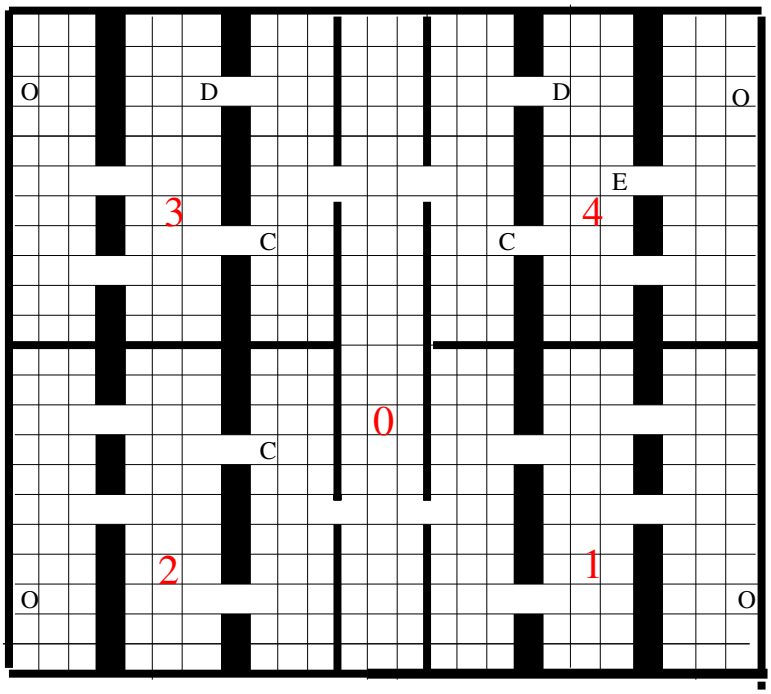
Experiments and Results

In this chapter, we test our framework on Grid world and Blocks world domain. We descirbe the domains and the results in detail.

4.1 Grid World

4.1.1 Experimental setup

Figure 4.1: A simple room domain with different objects.The rooms are as numbered 1, 2, 3 and 4. The task is to collect all the diamonds, O in the room



Our goal is to learn options in one room and re-use in other rooms, adapting them suitably to accommodate exceptions. In this scenario as shown in Figure 4.1, the agent's goal is to collect the diamond(O) in every room by occupying the same square as the diamond. Each of the rooms is a 11 by 11 grid with certain obstacles in it. The actions available to the agent are N, S, E, W. An action would result in the transition to the

indicated direction with a probability of 0.9. It results in a transition to any one of the unintended directions with a probability of 0.1.

The state is described by the following features: the room number the agent is in, with 0 denoting the corridor, the x and y co-ordinates within the room or corridor and boolean variables *have*(i), i = 1,..., 4, indicating possession of diamond in room i. The goal is any state of the form $\langle 0, ..1, 1, 1, 1\rangle$, where 0 indicates the corridor and 1 indicates possession of diamonds in the respective rooms. The state abstraction for the option is $\langle x, y, have\rangle$ as explained in Section 2.2.4. In each room the diamond is placed in (0,2). The agent starts from (10,5) in each room. The base policy is learnt in room 1 with no exceptions. The agent used a learning rate of .05, discount rate of .9 and ϵ greedy exploration, with an ϵ of 0.1. The results were averaged over 10 independent runs. The trials were terminated either on completion of the task or after 1500 steps.

4.1.2 Landmarks extraction

Initially we ran spiked state method described in Section 3.4.1 to find the landmarks. 10000 trajectories were simulated from a random start state that terminated at the goal state. The set of states along with their frequency count are shown in Table 4.1. The state (0, 2, 1) and (2, 5, 1) has the characteristics shown in sub figure B in Figure 3.5. It is obvious in (0, 2, 1) because it is the state which represents the agent fetching the diamond, hence it occurs in all the trajectories. From the state (2, 5, 1), if the agent stochastically transitions to (3, 5, 1), then the agent follows the path through the lower slit. Hence through both the states there are multiple different trajectories, representing the characteristics of spiked state.

But since the number of states that spiked are few and also are not uniformly distributed over the state space, we use mean-to-variance ratio to find the landmarks. States that occur in more than 80 % of the trials only were considered as candidate landmarks. Three sets of experiments were performed with a different bounded distance (in time) between the landmarks, resulting in a different number of landmarks for each set of experiments.

Since in this domain, number of steps taken by the agent was approximately around 43 steps, the l_{min}(minimum distance between the two landmarks) and l_{max}(maximum

distance between the two landmarks) were chosen as 16 and 32 , 8 and 16, 4 and 8 respectively for Experiment set GW-Exp-1, Experiment set GW-Exp-2 and Experiment set GW-Exp-3. The number of trajectories used to collect the information regarding the landmarks and transition times between the landmarks was 3000. Though we had different set of ordering for each experiment, we give example for one experiment from each experiment set. The landmarks in Experiment GW-Exp-1 with $thresh_{no-traj}$ = .8, Experiment GW-Exp-2 with $thresh_{no-traj}$ = .8 and Experiment GW-Exp-3 with $thresh_{no-traj}$ = .6 are selected respectively from the Tables 4.2, 4.3 and 4.4. These are marked with bold letters in the table. Though the landmarks are selected in increasing order of the mean to variance ratio, we have labelled the landmarks the order in which they are visited by the agent. The landmarks are shown in the domain as in Figures 4.2, 4.3 and 4.4. The number of candidates for the landmarks will usually depend on the nature of the domain. If the domain consist of multiple different trajectories through the state space then naturally the number of states which will be captured by 80% threshold will be fewer resulting in fewer number of candidates for landmarks. In this domain as there are no optimal multiple disjoint trajectories between the start and the goal state, we have large number of candidates for the landmarks.

The problem of adding the landmark is generally hard as we have to satisfy both l_{min}, l_{max} constraint and also we need to add in the increasing order of mean to variance ratio. For example when l_{min} = 8 and l_{max} = 16, then one cannot add any landmark between the start and the end landmark satisfying both the l_{min} and l_{max} constraints. Here the distance between the start and end landmark is 43. We apply algorithm Identify-landmarks. Some times this method fails as landmarks maybe removed for not satisfying the l_{max} constraint. This happens particularly, when the distance between the landmarks are very close with respect to the total distance(distance between the start and the end landmark), for example in Experiment GW-Exp-3, but still the failure affects very few landmarks. In Experiments GW-Exp-1 and GW-Exp-2, it rarely fails as the landmark distance are large and in this case mostly l_{min} condition automatically satisfies the l_{max} condition.

Table 4.1: Information regarding spiked states

Result 1		
previous state	spiked state	next state
(0, 3, 0) , η = 8635	(0, 2, 1) , η = 10000	(0 ,3 ,1), η = 9460
(2, 4, 1) , η = 8771	(2, 5, 1) , η = 9990	(3, 5, 1), η = 9596
Result 2		
previous state	spiked state	next state
(0, 3, 0) , η = 8075	(0, 2, 1) , η = 10000	(0, 3, 1), η = 9519
(1, 5, 1) , η = 9058	(2, 5, 1) , η = 9175	(3, 5, 1), η = 8715

4.1.3 Exceptions

We introduce multiple objects in successive trials in order to cause exceptions. As described in Section 3.4.4, we model 3 types of exceptions - C, D, E. The objects were introduced in the domain in the order “C", “D" and “E". For the purpose of learning nested OWE, the objects were introduced in the specific order. The mentioned representation is with respect to room 2 in Figure 4.1 where the coordinates are written as (row, column). The object “C" is placed at coordinate (row = 8, column =7) of the gridworld and it blocks the action west. The object “D" is placed at coordinate (6,2) of the gridworld and it randomly moves the agent to one of the coordinates (7,2), (6,1), (5,2) and (5,3). The object “E" is placed at coordinate (4,5) and it randomly moves the agent to one of the coordinates (4,3), (4,7), (6,5) and (2,5).

Within the experiment set we further perform the experiments depending upon the $thresh_{no-traj}$. This value affects the identification of the exception between the landmarks, particulary when the exceptions are of type “E" or “D" and are close to the landmarks. We explain this later in this section.

In the graphs, the new policies are plotted for various obstacles. The new policy was a concatenation of the path given by the suffix tree from the start state to the exception state or the terminal state, followed by the path given by Q-learning from the exception state to the potential terminal landmarks, followed by a path given by the suffix tree from the potential terminal landmarks to the terminal state. The maximum number of steps were assigned as 100 for the simulation of the path from start to exception state. Similar number of steps were assigned for the path given by the suffix tree from the potential terminal landmarks to the terminal state. The trials for the exception policy

Table 4.2: Set of candidates for landmark arranged in increasing order of mean-to-variance ratio for Experiment GW-Exp-1

State	mean(m)	variance(v)	m/v	
7, 7, 0	6.2849462366	3.6614131692	1.7165356506	
3 ,5, 0	13.9794820047	8.0557512285	1.7353418208	
0 ,2, 1	**21.417**	**12.053111**	**1.7768856522**	**L2**
2 ,5, 0	15.1937436932	8.5161128936	1.7841172238	
2 ,4, 0	16.3632986627	9.1325014517	1.7917652408	
10 ,5, 1	**42.9716666667**	**23.8588638889**	**1.8010776568**	**L3**
3 ,5, 1	28.953480589	16.0644356767	1.8023341231	
9 ,5, 1	41.6075388027	22.9723600179	1.8112000147	
2 ,3, 0	17.5678724232	9.6814104481	1.8145984531	
6 ,7, 0	7.5507392473	4.1553556363	1.8171102327	
4 ,5, 1	30.1904283802	16.5986098567	1.8188528221	
5 ,7, 0	8.8077876106	4.8313729814	1.823040292	
2 ,5, 1	27.622244489	15.1415359108	1.8242696548	
2 ,4, 1	26.323379461	14.4023448793	1.8277148396	
0 ,3, 0	20.1269609632	11.0068652261	1.8285824846	
4 ,5, 0	12.5620585267	6.8482341748	1.8343500245	
0 ,3, 1	22.4870956016	12.2403823678	1.8371236229	
5 ,6, 0	10.0427509294	5.4662020978	1.8372447176	
1 ,3, 0	18.8519040903	10.2411141459	1.840805973	
6 ,6, 1	33.8115325077	18.0979939362	1.8682475321	
4 ,6, 0	11.2024399543	5.976936479	1.8742779003	
2 ,3, 1	24.9956043956	13.3274532061	1.8754974419	
8 ,7, 1	37.7594623276	20.1091030714	1.8777298119	
8 ,6, 1	38.9851576994	20.7508001143	1.8787303374	
7 ,7, 1	36.4941634241	19.3642213288	1.8846181731	
6 ,5, 1	32.5089424572	17.1892668443	1.8912349638	
8 ,5, 1	40.1518590998	21.1855493813	1.8952474811	
6 ,7, 1	35.1057658295	18.4879292596	1.8988479097	
5 ,5, 1	31.2759633028	16.444945173	1.9018587763	
1 ,3, 1	23.6557852738	12.4332028382	1.9026300449	
8 ,7, 0	4.906922043	2.3901940208	2.0529387992	
9 ,7, 0	3.4158730159	1.0540337617	3.2407624311	
9 ,6, 0	2.2363984674	0.4877939255	4.5847197976	
10, 6, 0	1.0599634369	0.1060936001	9.9908329595	
10 ,5, 0	**0**	**0**	**Inf**	**L1**

Figure 4.2: Set of landmarks shown in the domain for Experiment GW-Exp-1.

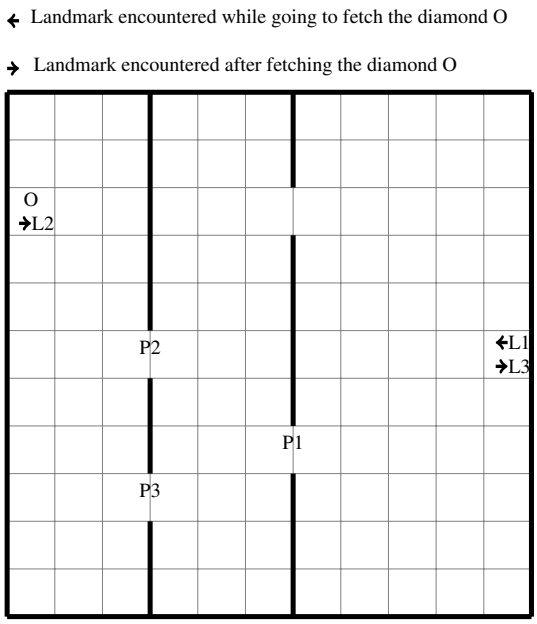


Table 4.3: Set of candidates for landmark arranged in increasing order of mean-to-variance ratio Experiment GW-Exp-2

State	mean(m)	variance(v)	m/v	
8 ,7, 0	4.7316001465	5.5650907753	0.8502287451	
7 ,7, 0	6.1215671915	6.6677552883	0.9180851616	
6 ,7, 0	7.3687294031	7.2236138744	1.0200890484	
6 ,6, 0	**8.4869530309**	**8.0049482029**	**1.0602133601**	**L2**
5 ,6, 0	9.6645833333	8.5712456597	1.127558784	
4 ,6, 0	11.0191769876	9.4530880961	1.1656695543	
4 ,5, 0	12.4582210243	9.8715698811	1.2620303735	
3 ,5, 0	13.8490566038	10.9118522097	1.2691756026	
2 ,4, 0	16.3073785688	12.6771164874	1.2863633923	
2 ,5, 0	15.1164591047	11.6821626219	1.2939778014	
0 ,4, 0	**18.8396778917**	**13.8447214189**	**1.3607841806**	**L3**
1 ,4, 0	17.5644366197	12.7789465136	1.3744823645	
0 ,3, 0	20.1099824869	14.4089196144	1.3956620638	
0 ,2, 1	21.4263333333	15.1272398889	1.4164073215	
0 ,3, 1	22.6228873239	15.7785606774	1.4337738268	
1 ,3, 1	23.806379822	16.51444001	1.4415493233	
2 ,3, 1	25.0742230347	17.1261179978	1.4640926238	
2 ,4, 1	26.3509817672	17.917835642	1.4706565175	
6 ,5, 1	**32.5127996749**	**21.8256183707**	**1.4896622457**	**L4**
3 ,5, 1	28.9856235373	19.4277438355	1.4919706469	
6 ,6, 1	33.6397621071	22.5303815917	1.4930844367	
5 ,5, 1	31.3830022075	20.9625808322	1.4970963002	
4 ,5, 1	30.2420595119	20.0477261496	1.5085032231	
2 ,5, 1	27.6003333333	18.2765998889	1.5101459517	
7 ,7, 1	36.2963818322	24.0068460313	1.5119179664	
8 ,6, 1	38.7850542387	25.5765383759	1.5164309442	
8 ,7, 1	37.5369515012	24.5165329699	1.5310872686	
6 ,7, 1	34.9453636014	22.7026708855	1.5392622206	
10, 5, 1	**42.826**	**27.5830573333**	**1.5526197652**	**L5**
9 ,6, 1	39.8994345719	25.5751046884	1.5600888074	
10 ,6, 1	41.10479423	21.740821309	1.8906734776	
8 ,6, 0	3.3414023372	0.8016414391	4.1682006121	
9 ,6, 0	2.1937953263	0.4269880935	5.1378372368	
9 ,5, 0	1.0738304094	0.1444028718	7.4363507858	
10, 5, 0	**0**	**0**	**Inf**	**L1**

Figure 4.3: Set of landmarks shown in the domain for Experiment GW-Exp-2.

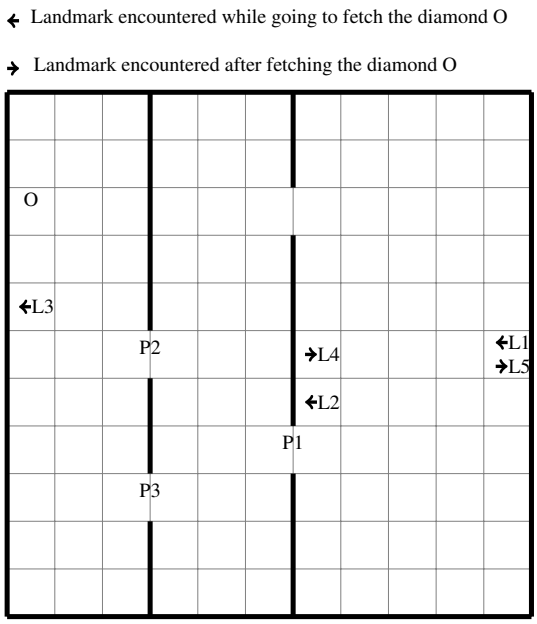


Table 4.4: Set of candidates for landmark arranged in increasing order of mean-to-variance ratio Experiment GW-Exp-3

State	mean(m)	variance(v)	m/v	
3, 5, 0	**13.9889742733**	**9.7603194624**	**1.4332496315**	**L3**
4 ,5, 0	12.631807551	8.7745579422	1.4395947504	
2 ,5, 0	15.2335449382	10.19838025	1.4937219995	
2 ,4, 0	16.4488078541	10.8890343432	1.5105846245	
0 ,2, 1	**21.475**	**13.5780416667**	**1.5815977391**	**L4**
0 ,3, 0	20.2093969663	12.7197489149	1.588820432	
1 ,4, 0	17.6622097115	11.0991278406	1.5913150984	
7 ,7, 1	**36.7267226891**	**23.0207816962**	**1.5953725279**	**L7**
0 ,4, 0	18.9261595547	11.8605958261	1.5957174355	
5 ,7, 1	34.0062043796	21.2273337684	1.6020007388	
10, 5, 1	**43.087**	**26.8707643333**	**1.6034899293**	**L8**
8 ,7, 1	37.9700840336	23.6552394887	1.6051447736	
9 ,7, 1	39.2215909091	24.3521758781	1.6105990325	
1 ,2, 1	22.644037988	14.0576033922	1.6108035884	
6 ,7, 1	35.3368067227	21.8072335005	1.6204167632	
1 ,3, 1	23.8119038535	14.633448531	1.6272243554	
10, 7, 1	40.4641130532	24.7686079991	1.6336853914	
2 ,4, 1	**26.3638773192**	**16.1103043909**	**1.6364605336**	**L5**
10, 6, 1	41.7329718739	25.5003666	1.6365636043	
3, 5, 1	29.0950150552	17.6597576863	1.6475319521	
2, 3, 1	25.0503624571	15.1970057824	1.6483748717	
4 ,5, 1	30.3268651723	18.0902150307	1.6764236976	
2, 5, 1	27.7161936561	16.4510065468	1.684771906	
5, 5, 1	**31.5375886525**	**18.7166721996**	**1.6849997861**	**L6**
5, 6, 1	32.7051372273	19.1495091472	1.7078838406	
7, 7, 0	**6.174020473**	**3.2415135569**	**1.9046721121**	**L2**
6, 7, 0	7.4221673138	3.7575319071	1.9752772558	
6, 6, 0	8.5757695343	4.2111098064	2.0364630534	
6, 5, 0	9.7567001675	4.5978403674	2.1220180319	
5, 5, 0	10.9737521515	5.0290012398	2.1820937455	
8, 7, 0	4.7822096717	2.0178691731	2.3699304868	
9, 7, 0	3.4367816092	0.9779277955	3.5143510852	
9, 6, 0	2.2071269488	0.3676403722	6.0034944907	
10, 6, 0	1.0513104467	0.0730409184	14.3934450696	
10, 5, 0	**0**	**0**	**Inf**	**L1**

Figure 4.4: Set of landmarks shown in the domain for Experiment GW-Exp-3.

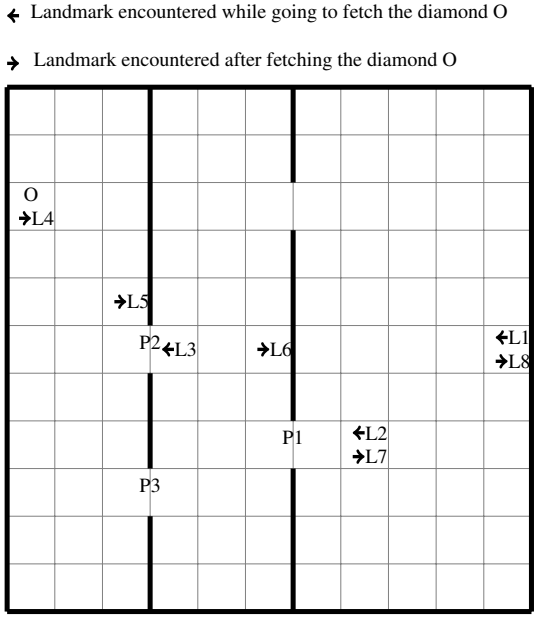


Table 4.5: Selected Landmarks

Experiment		
GW-Exp-1	GW-Exp-2	GW-Exp-3
($thresh_{no-traj}$ = .9)	($thresh_{no-traj}$ = .8)	($thresh_{no-traj}$ = .7)
(10, 5, 0), (0, 3, 0), (10, 5, 1)	(10, 5, 0), (4, 6, 0), (0, 3, 0) (4, 5, 1), (10, 5, 1)	(10, 5, 0), (8, 7, 0), (5 ,6, 0) (3, 5, 0), (0, 2, 1), (2, 5, 1) (6, 5, 1), (8, 7, 1), (10, 5, 1)
($thresh_{no-traj}$ = .8)	($thresh_{no-traj}$ = .7)	($thresh_{no-traj}$ = .6)
(10, 5, 0), (0, 2, 1), (10, 5, 1)	(10, 5, 0), (6, 6, 0), (0, 4, 0) (6, 5, 1), (10, 5, 1)	(10, 5, 0), (7, 7, 0), (3 ,5, 0) (0, 2, 1), (2, 4, 1), (5, 5, 1) (7, 7, 1), (10, 5, 1)
($thresh_{no-traj}$ = .7)	($thresh_{no-traj}$ = .6)	($thresh_{no-traj}$ = .5)
(10, 5, 0), (0, 3, 0), (10, 5, 1)	(10, 5, 0), (5, 6, 0), (1, 2, 0) (4, 6, 1), (10, 5, 1)	(10 ,5, 0), (8 ,7 ,0), (5 ,6, 0) (0, 5, 0), (0, 3, 1), (1, 3, 1) (8, 7, 1) (10, 5, 1)

that is from exception state to potential landmark were terminated either on completion of the task or after 1300 steps.

4.1.4 Results and Discussion

In Experiment GW-Exp-1, while going to fetch the diamond, the exception caused by “D" was not identified. This happened because of the change in the number of steps caused by the object “D" was around 1.5 on an average and this change becomes negligible compared to the distance between the landmarks. This is reflected when CIT is calculated and is lesser than rv as shown in Table 4.6 under the column 1. But when object “E" was introduced in the domain the exception was caught. Though the exception was caused by the object E, the object “D" also got identified as an exception state by the minimum model. This happened because both “D" and “E" lay within the same pair of landmarks and the state with the obstacle “D" is visited first compared to the state with the obstacle “E". The exception policy learnt from the exception state at (6,2) to the next landmark passes through coordinate (4,5). The object “E" at coordinate (4,5) further randomly increases or decreases the number of steps in the exception policy. For example if the agent moves to coordinate (2,5) or (4,7) from (4,5) then there is decrease in number of steps in the exception policy. But if the agent moves to (4,3) or (6,5) it can again come to (4,5) further increasing the number of steps. This can be seen in the

Table 4.6: Details of exception caused by object D

Experiment	
GW-Exp-1	GW-Exp-2
($thresh_{no-traj}$ = .9)	($thresh_{no-traj}$ = .8)
u = (10, 5, 0) , v = (0, 3, 0) $t_2^{u,v}$ = 26.96, $\tilde{t}^{u,v}$ = 27.08 $rv_2^{u,v}$ = 10.02, $CIt_2^{u,v}$ = .45	u = (10, 5, 0) , v = (4, 6, 0) $t_2^{u,v}$ = 18.2, $\tilde{t}^{u,v}$ = 18.03 $rv_2^{u,v}$ = 12.36, $CIt_2^{u,v}$ = 0.50
($thresh_{no-traj}$ = .8)	($thresh_{no-traj}$ = .7)
u = (10, 5, 0) , v = (0, 2, 1) $t_2^{u,v}$ = 25.6, $\tilde{t}^{u,v}$ = 27.42 $rv_2^{u,v}$ = 11.02, $CIt_2^{u,v}$ = 7.10	u = (10, 5, 0) , v = (6, 6, 0) $\tilde{t}^{u,v}$ = NR
($thresh_{no-traj}$ = .7)	($thresh_{no-traj}$ = .6)
u = (10, 5, 0) , v = (0, 3, 0) $t_2^{u,v}$ = 25.12, $\tilde{t}^{u,v}$ = 27.38 $rv_2^{u,v}$ = 11.56, $CIt_2^{u,v}$ = 8.99	u = (10, 5, 0) , v = (5, 6, 0) $\tilde{t}^{u,v}$ = NR

Table 4.7: Details of exception caused by object E after the agent fetches the diamond

Experiment
GW-Exp-1
($thresh_{no-traj}$ = .9)
u = (0, 3, 0) , v = (10, 5, 1) $t_2^{u,v}$ = 17.8, $\tilde{t}^{u,v}$ = 21.08 $rv_2^{u,v}$ = 11.0, $CIt_2^{u,v}$ = 18.0
($thresh_{no-traj}$ = .8)
u = (0, 2, 1) , v = (10, 5, 1) $t_2^{u,v}$ = 19.08, $\tilde{t}^{u,v}$ = 22.25 $rv_2^{u,v}$ = 11.35, $CIt_2^{u,v}$ = 16.6
($thresh_{no-traj}$ = .7)
u = (0, 3, 0) , v = (10, 5, 1) $t_2^{u,v}$ = 18.83, $\tilde{t}^{u,v}$ = 22.38 $rv_2^{u,v}$ = 11.56, $CIt_2^{u,v}$ = 18.85

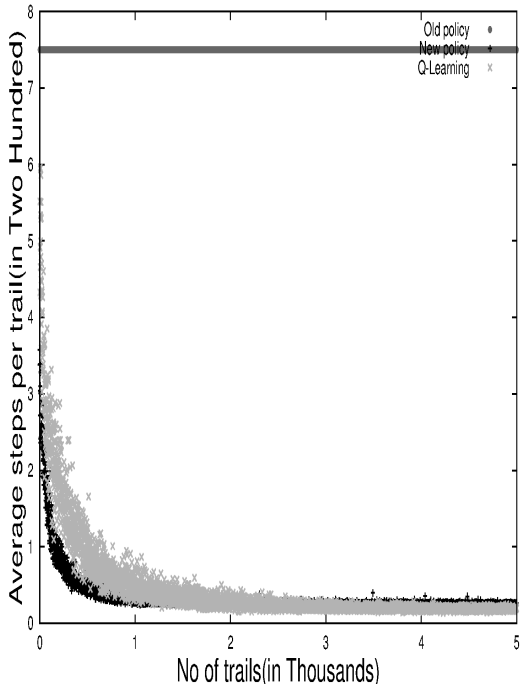
graph of new policy as shown in sub figure b of Figures 4.5, 4.6 and 4.7 where we can see points above the average.

Similar thing happens after the agent fetches the diamond and returns through the coordinate (4,5) where object “E" is placed. But in the graph of new policy we can see few points above the average in sub-figure c of Figures 4.5, 4.6 and 4.7 as compared to sub-figure b because the agent can come back again to (4,5) only if it moves back to (2,5). If it moves to (4,3), (6,5) and (4,7), it generally does not come back to (4,5) or only occasionally comes back because of the stochasticity.

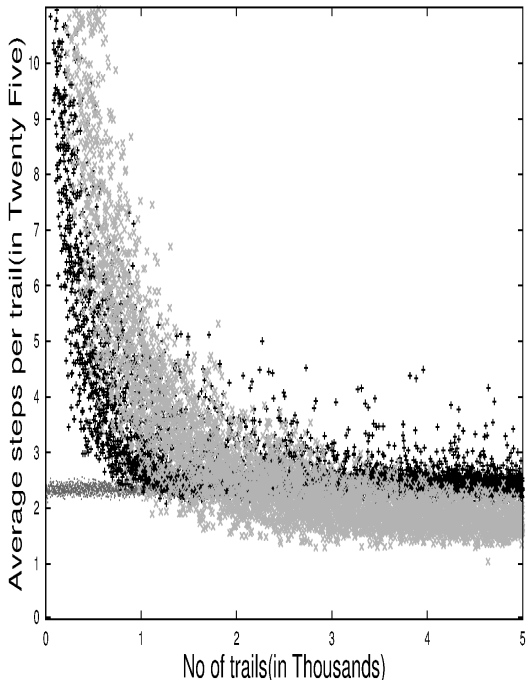
The new policy converges faster compared to the policy learned using Q learning. But the new policy is mostly similar to the old policy or is worse than the old policy because of the nature of the object “E" and its position in the domain. It seems that on average the new policy which goes through coordinate (3,8) takes more number of steps as compared to the old policy which most of time passes through the partition near (4,5) by moving to (2,5) from (4,5). We will further explain this in the end of this section. As shown in sub-figure a of Figures 4.5, 4.6 and 4.7, the old policy is far worse than the new policy. This purely happens because of the nature of the object“C" and the position it is placed. We also find that the average number of steps for the new policy was greater than that of Q Learning because the new policy was constrained to go through the landmarks.

In Experiments GW-Exp-2 and GW-Exp-3 we find that the exception by the object “D" is caught because the landmark which is near (6,2) is occasionally visited by the agent because of the stochasticity of the object “D". For example in Table 4.6 under the column Experiment GW-Exp-2, with the exception of the category $thresh_{no-traj}$ = .8 all the other experiment shows similar result because of object “D". This happens because after the exception by the object “D", the landmark (4, 6, 0) still lies on the path taken by the agent. Whereas (6, 5, 0) and (5, 6, 0) are close to the object “D" and once the agent moves randomly away from the object “D", it takes the path on which landmark (6, 5, 0) and (5, 6, 0) does not lie. Thus out of the 3000 trajectories, around 1220 trajectories reached next landmark from object “D". Similar pattern happens in all the experiments in the experiment set 3. For Experiments GW-Exp-2 and GW-Exp-3, the next landmark after the exception caused by object “E" was mostly not reachable. For example when the landmark was at (4, 6, 1) the agent visited it only 42 times.

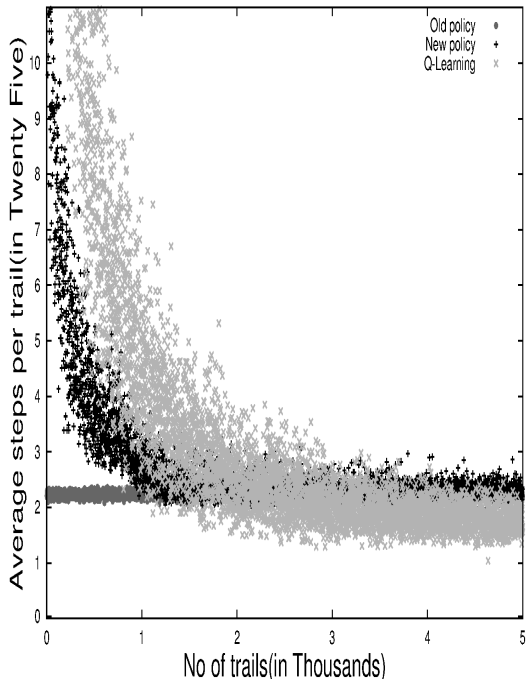
Figure 4.5: Comparison of old, new and policy learnt using Q-learning for Experiment GW-Exp-1($thresh_{no-traj}$ = .7)



(a) Exception caused by the object C

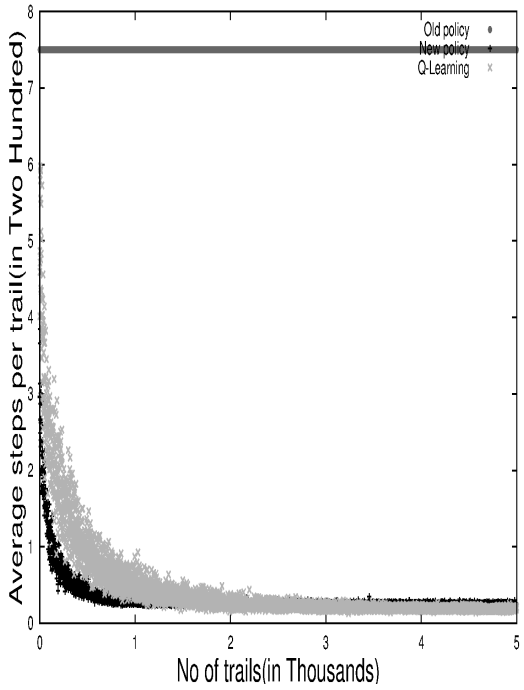


(b) Exception caused by the object D and E while going to fetch the diamond

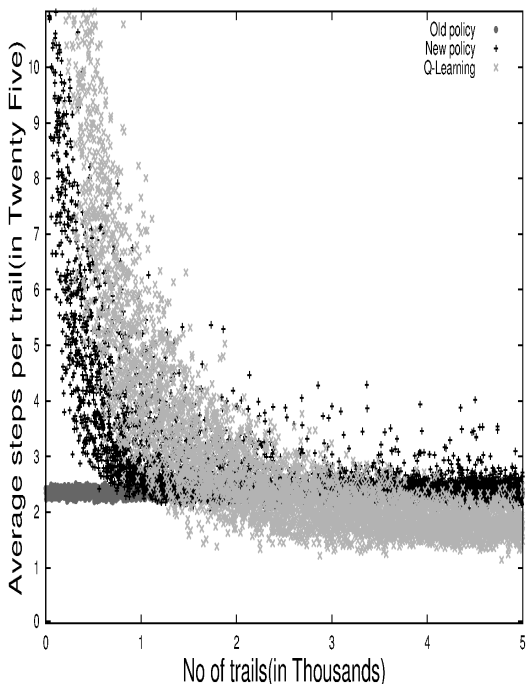


(c) Exception caused by the object E after fetching the diamond

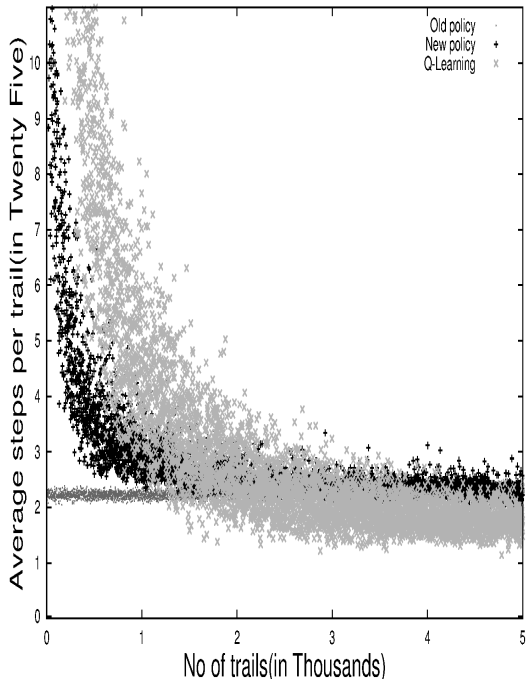
Figure 4.6: Comparison of old, new and policy learnt using Q-learning for Experiment GW-Exp-1 ($thresh_{no-traj}$ = .8)



(a) Exception caused by the object C

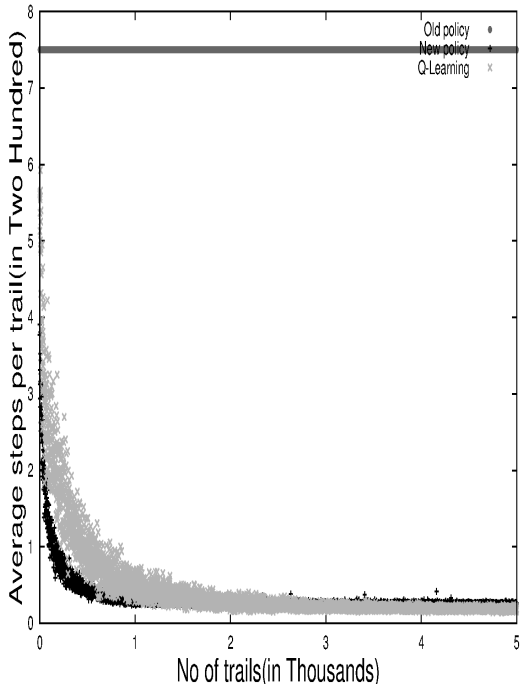


(b) Exception caused by the object D and E while going to fetch the diamond

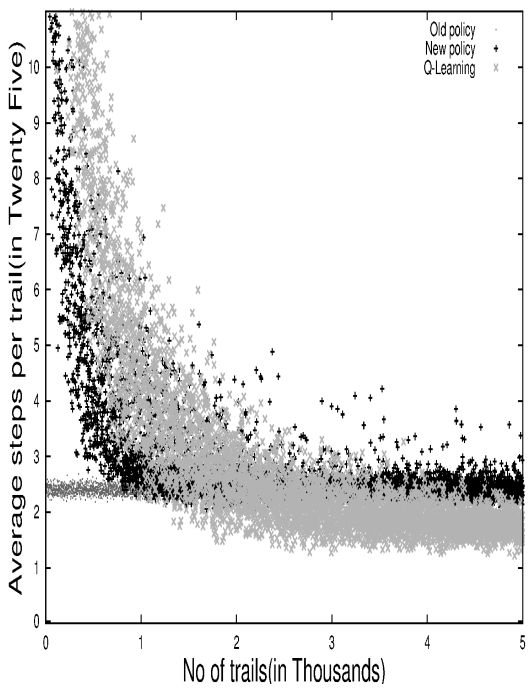


(c) Exception caused by the object E after fetching the diamond

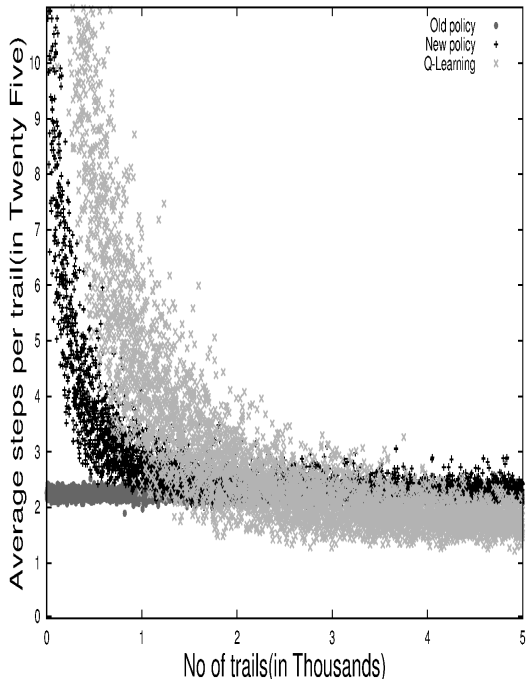
Figure 4.7: Comparison of old, new and policy learnt using Q-learning for Experiment GW-Exp-1 ($thresh_{no-traj}$ = .9)



(a) Exception caused by the object C



(b) Exception caused by the object D and E while going to fetch the diamond



(c) Exception caused by the object E after fetching the diamond

The lowering of the $thresh_{no-traj}$ for the Experiments GW-Exp-2 and GW-Exp-3 was reasonable because when the exception policy is learnt after the exception, the number of trajectories passing through the landmarks near the object “E" and “D" decreases as compared to total number of trajectories. For example when the landmark is at (4, 6, 1), after the exception policy is learnt, the number of trajectories that pass through this landmark is 2100 out of a total of 3000 trajectories. The analysis of the graphs are similar to experiment “1".

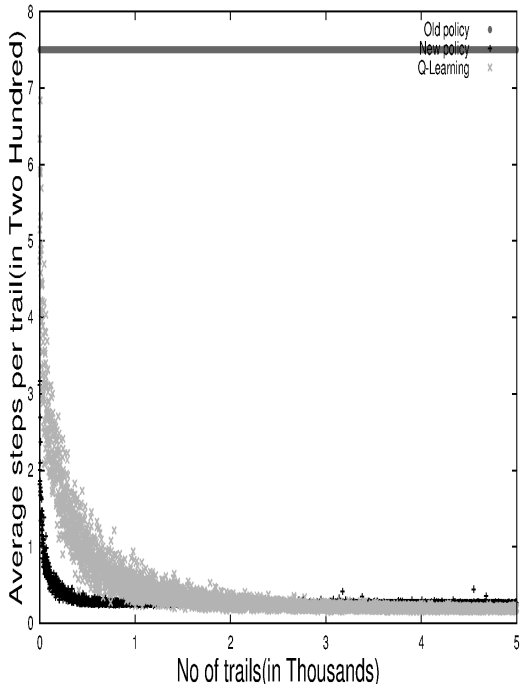
When comparing the graphs across the Experiments GW-Exp-1, GW-Exp-2 and GW-Exp-3 we find we find that the new policy learned with OWE converges much more quickly than Q learning for Experiments GW-Exp-2 and GW-Exp-3 as compared to GW-Exp-1. This indicates that as the distance between the landmarks decreases the performance is far better compared to when the landmarks are far away.

Table 4.8: Transfer back of the policy to old domain(Avg No of steps) for Experiment GW-Exp-1

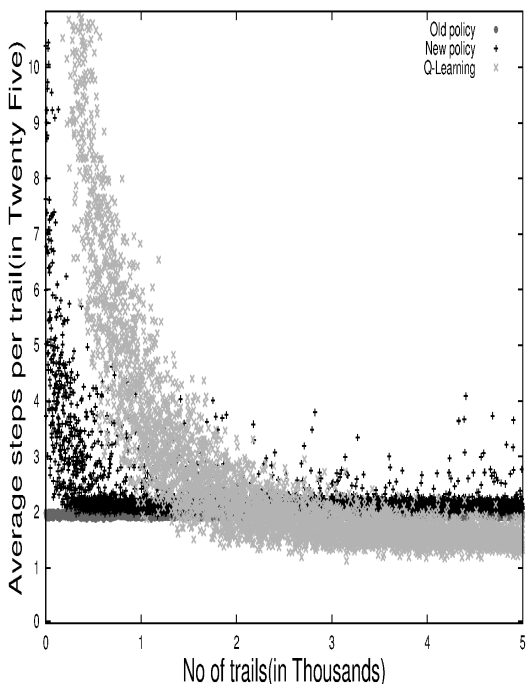
		Initial Environment	Object C	Object C,D
$thresh_{no-traj}$ = .7	Initial Environment	45.36		
	After Object C	44.673	49.971	
	After Object D	45.124	50.55	52.957
	After Object E	45.521	50.65	52.899
$thresh_{no-traj}$ = .8	Initial Environment	45.12		
	After Object C	44.528	50.875	
	After Object D	44.473	50.605	52.961
	After Object E	43.866	50.771	53.103
$thresh_{no-traj}$ = .9	Initial Environment	45.59		
	After Object C	45.582	50.542	
	After Object D	45.324	50.632	53.712
	After Object E	44.904	50.56	53.36

When the OWE policies were used in the original domains, i.e., without exceptions or fewer exceptions, the average number of steps taken to complete the task did not change appreciably, as shown in Tables 4.8, 4.9 and 4.10. The average was taken over 1000 trails and the maximum length of the trajectory is 500. The trajectories are sampled using the ϵ-greedy policy. In these tables, the rows indicate transfer of policies to the original domain after a particular object was introduced in the domain and the columns indicate different objects present in the domain at that time. There is random-

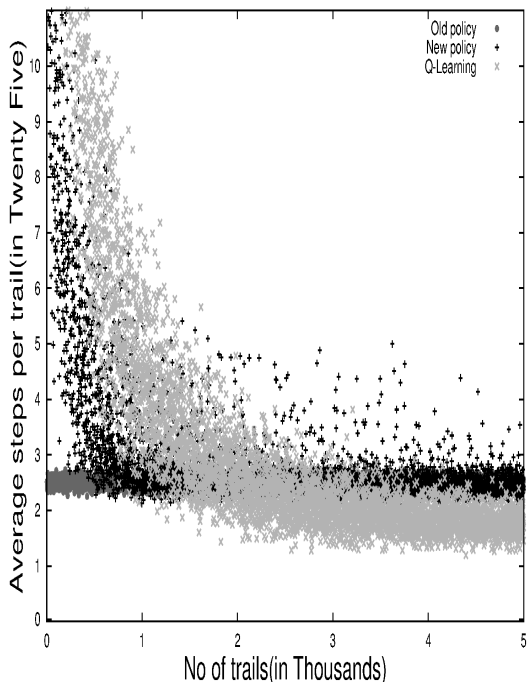
Figure 4.8: Comparison of old, new and policy learnt using Q-learning for Experiment GW-Exp-2 ($thresh_{no-traj}$ = .6)



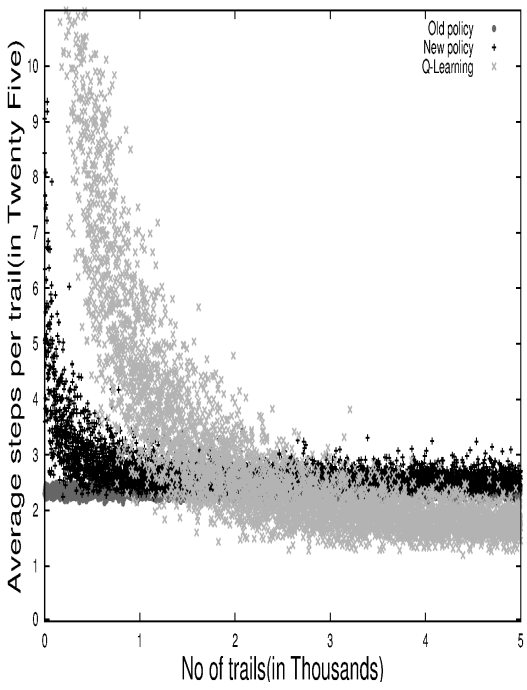
(a) Exception caused by the object C



(b) Exception caused by the object D

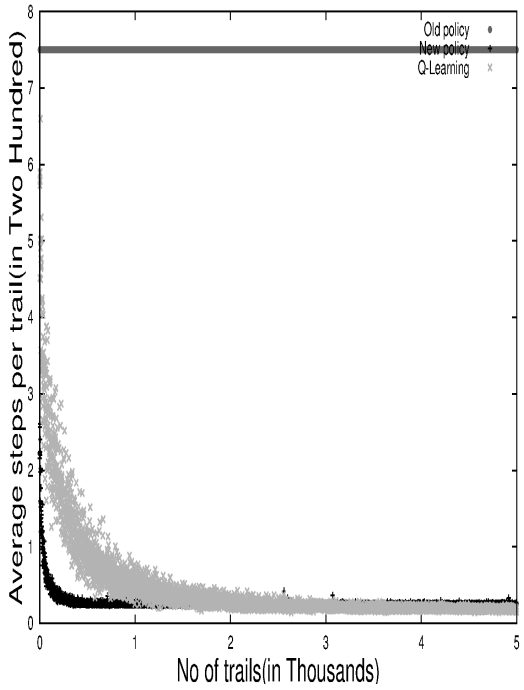


(c) Exception caused by the object E while going to fetch the diamond

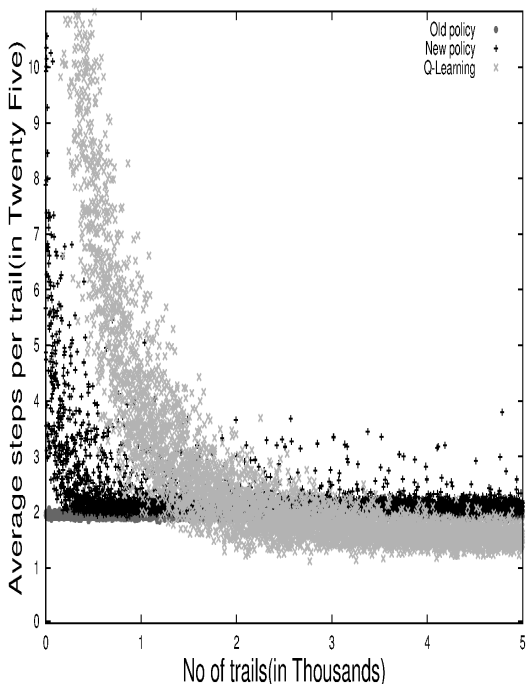


(d) Exception caused by the object E after fetching the diamond

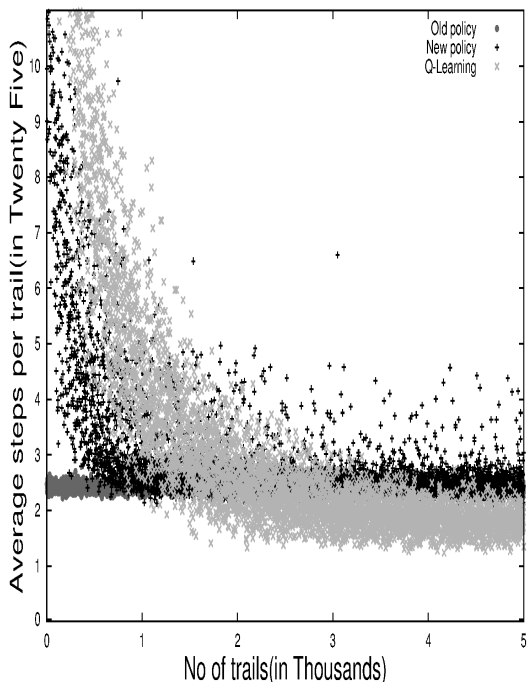
Figure 4.9: Comparison of old, new and policy learnt using Q-learning for Experiment GW-Exp-2 ($thresh_{no-traj}$ = .7)



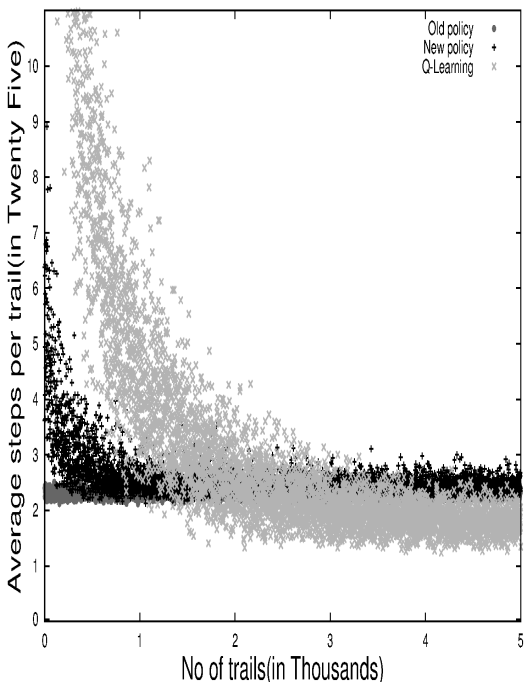
(a) Exception caused by the object C



(b) Exception caused by the object D

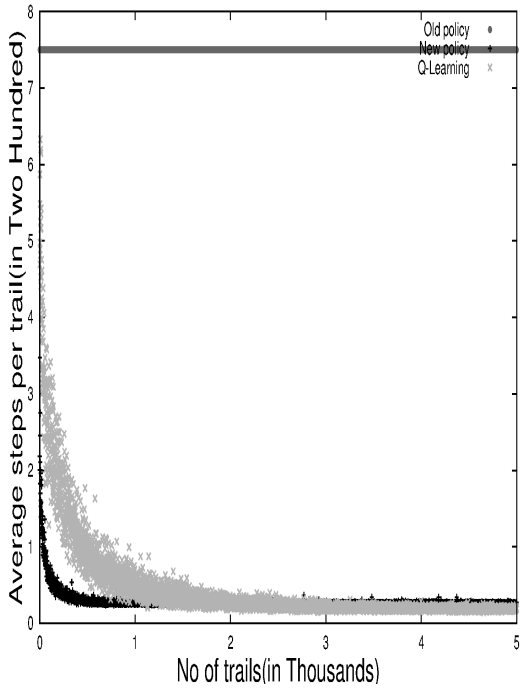


(c) Exception caused by the object E going to fetch the diamond

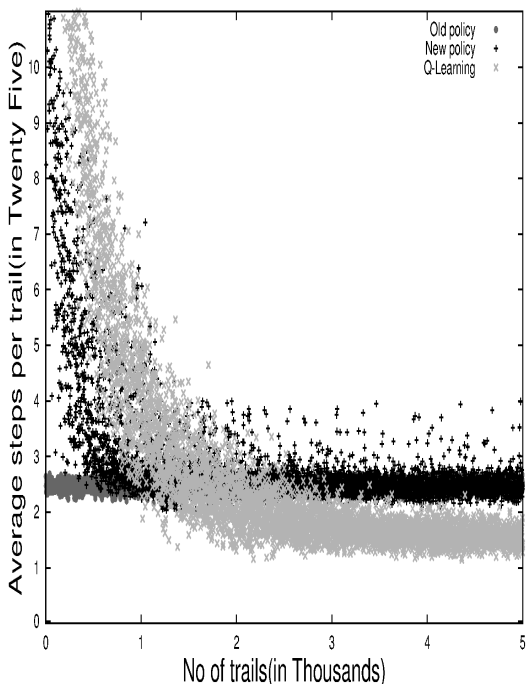


(d) Exception caused by the object E after fetching the diamond

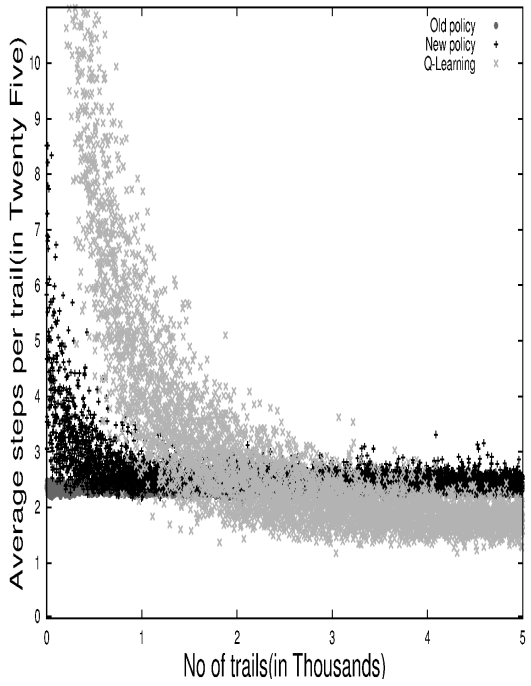
Figure 4.10: Comparison of old, new and policy learnt using Q-learning for Experiment GW-Exp-2 ($thresh_{no-traj}$ = .8)



(a) Exception caused by the object C

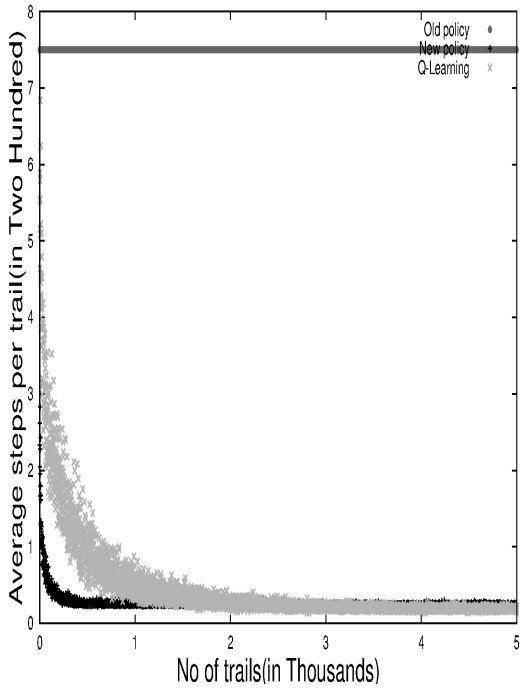


(b) Exception caused by the object D and E while going to fetch the diamond

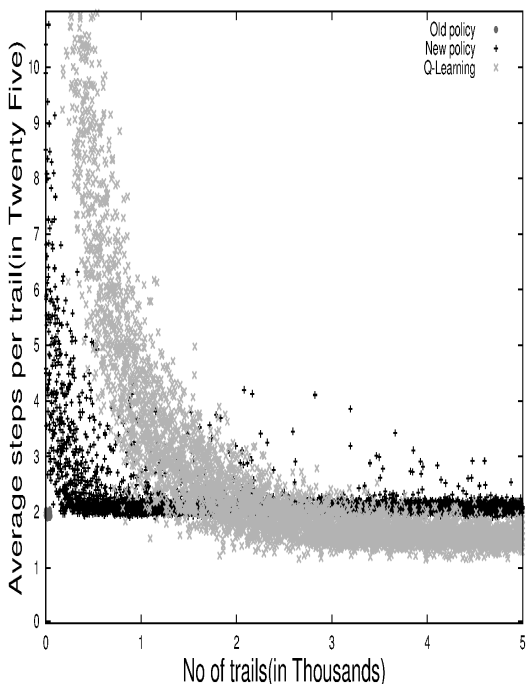


(c) Exception caused by the object E after fetching the diamond

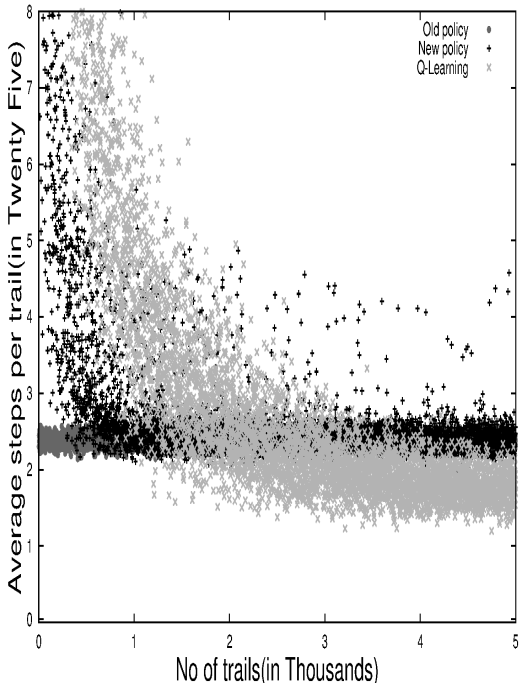
Figure 4.11: Comparison of old, new and policy learnt using Q-learning for Experiment GW-Exp-3 ($thresh_{no-traj}$ = .5)



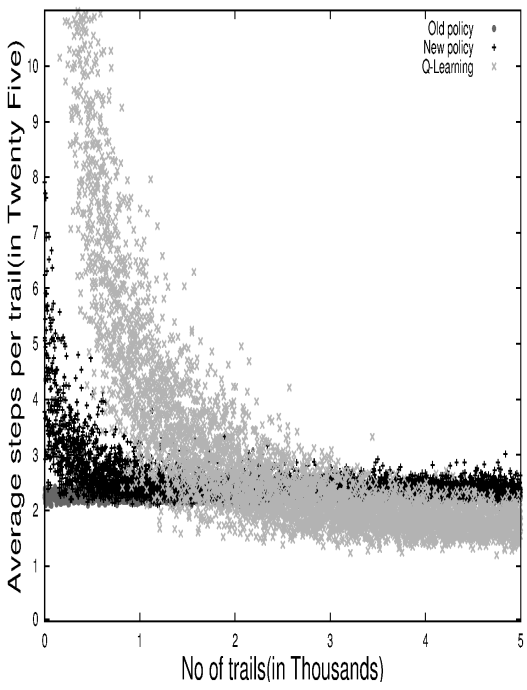
(a) Exception caused by the object C



(b) Exception caused by the object D

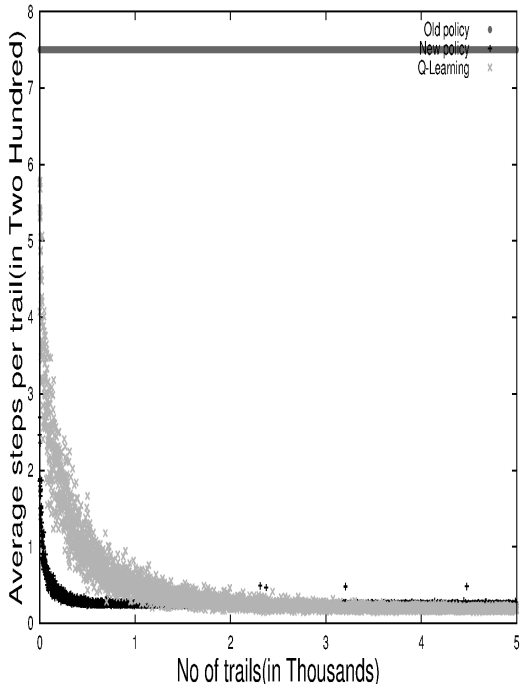


(c) Exception caused by the object E while going to fetch the diamond

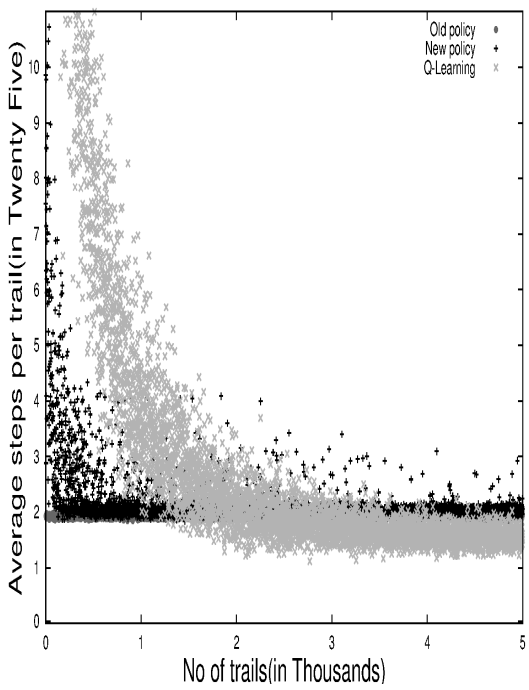


(d) Exception caused by the object E after fetching the diamond

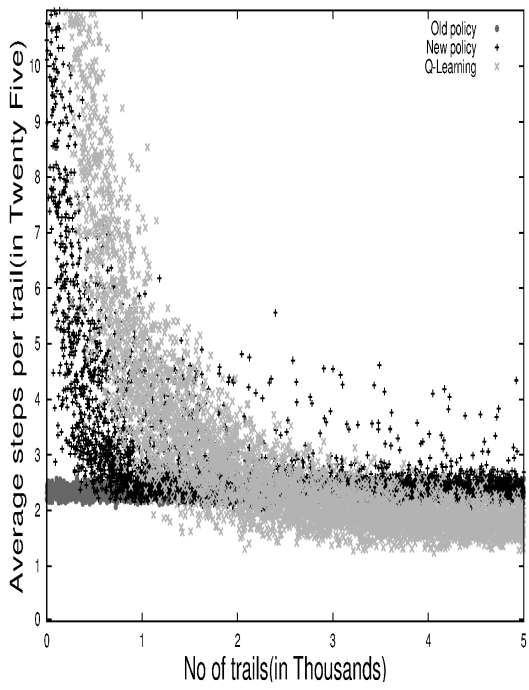
Figure 4.12: Comparison of old, new and policy learnt using Q-learning for Experiment GW-Exp-3 ($thresh_{no-traj}$ = .6)



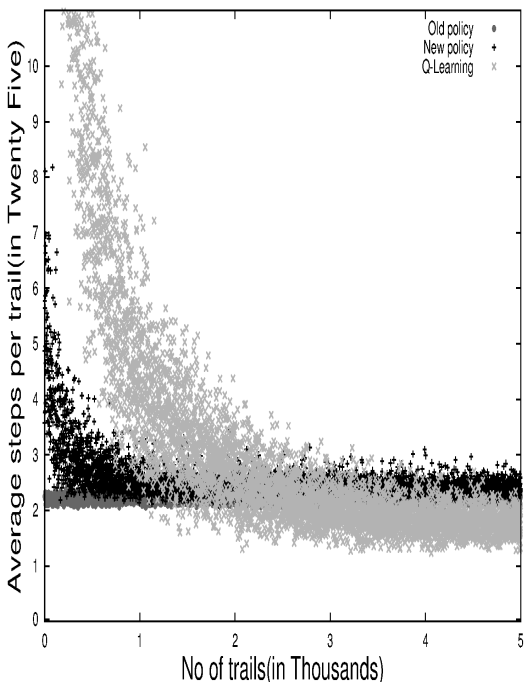
(a) Exception caused by the object C



(b) Exception caused by the object D

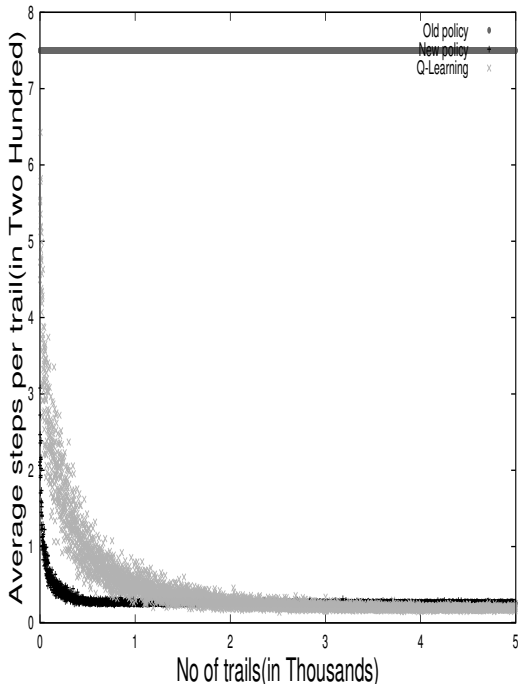


(c) Exception caused by the object E whilegoing to fetch the diamond

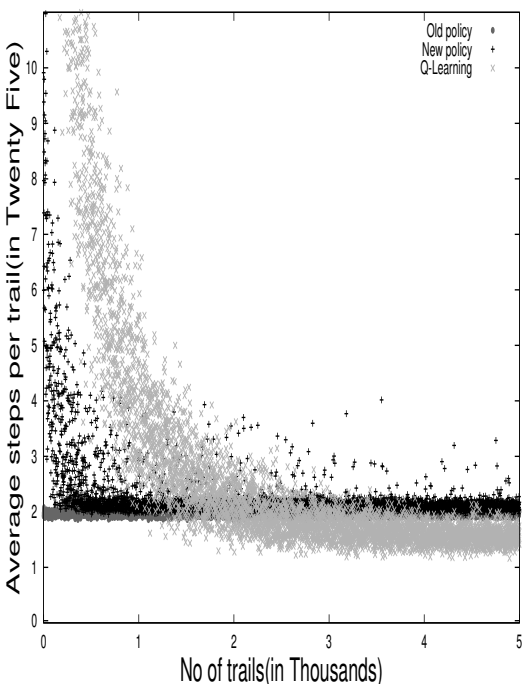


(d) Exception caused by the object E afterfetching the diamond

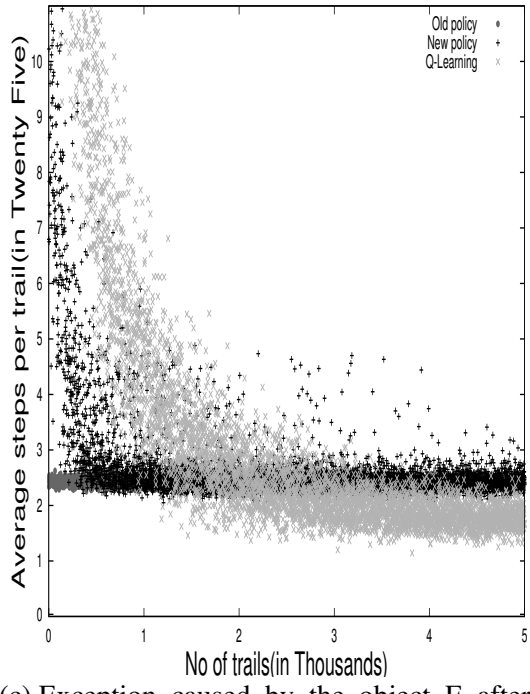
Figure 4.13: Comparison of old, new and policy learnt using Q-learning for Experiment GW-Exp-3 ($thresh_{no-traj}$ = .7)



(a) Exception caused by the object C



(b) Exception caused by the object D



(c) Exception caused by the object E after fetching the diamond

Table 4.9: Transfer back of the policy to original domains (Avg No of steps) for Experiment GW-Exp-2

		Initial Environment	Object C	Object C,D
$thresh_{no-traj}$ = .6	Initial Environment	45.47		
	After Object C	45.301	54.5	
	After Object D	45.094	54.571	56.421
	After Object E	44.599	54.475	56.404
$thresh_{no-traj}$ = .7	Initial Environment	44.55		
	After Object C	44.709	53.995	
	After Object D	44.623	53.943	56.875
	After Object E	44.086	53.866	56.574
$thresh_{no-traj}$ = .8	Initial Environment	44.13		
	After Object C	43.866	53.912	
	After Object D	44.168	54.321	55.48
	After Object E	44.15	54.314	55.779

Table 4.10: Transfer back of the policy to the original domains (Avg No of steps) for Experiment GW-Exp-3

		Initial Environment	Object C	Object C,D
$thresh_{no-traj}$ = .5	Initial Environment	44.10		
	After Object C	43.712	54.62	
	After Object D	43.749	54.594	56.849
	After Object E	43.923	54.352	56.807
$thresh_{no-traj}$ = .6	Initial Environment	44.50		
	After Object C	43.897	54.927	
	After Object D	43.961	54.69	56.97
	After Object E	44.12	54.88	56.619
$thresh_{no-traj}$ = .7	Initial Environment	45.10		
	After Object C	43.812	54.194	
	After Object D	44.281	54.762	54.97
	After Object E	44.409	54.481	55.046

ness in the domain in the execution of the action as explained in the domain section and also because of ϵ =.1. Because of this randomness, a state may exist in the trajectory that might not be represented by the suffix tree. That is why in the table, we see some form of randomness.

As the agent is constrained to visit each landmark, we have a longer new policy in the domain with larger number of landmarks as compared to that with smaller number of landmarks. For example, in Experiment GW-Exp-1 (Table 4.8) in column “Object C” and “Object C,D”, the average number of steps was lower as compared to the Experiments GW-Exp-2 (Table 4.9) and GW-Exp-3 (Table 4.10).

Because of the above reason, we can see that the old policy is some times better than the new policy(exception policy) especially when the exception is caused by the object “D" and “E" in Experiments GW-Exp-2 and GW-Exp-3. Though the old policy is better than the new policy, one can only know this after learning the new policy.

4.2 Blocks World

4.2.1 Experimental setup

A blocks world, as shown in Figure 4.14 , contains a table, a fixed set of blocks and a robot arm. Given the initial configuration of blocks, the agent, using the robot arm, has to arrange the blocks as depicted in the final configuration. Each block can be in one of the 3 positions; either on the table, on top of the other block or in the robot's arm. The set of predicates are ***on(?Block,?Block)***, ***clear(?Block)***, ***ontable(?Block)*** and ***holding(?Block)***. Here **?Block** is the variable corresponding to the name of the block. The predicate on(A,B) is true if block A is on block B, clear(A) is true if the top of block A is empty, ontable(A) is true if block A is on the table, and holding is true if the robot arm is holding some block. The set of possible actions in this world are ***Stack(?Block,?Block)*** , ***UnStack(?Block,?Block)***, ***PutDown(?Block)*** and ***PickUp(?Block)***. Stack and UnStack actions need two blocks as indicated by the number of arguments. The pickUp action results in picking up the block from the table. PutDown action puts the block on the table. In order to introduce stochasticity in the world, we assume that an action results in no effect with a probability of 0.1.

Figure 4.14: A blocks world

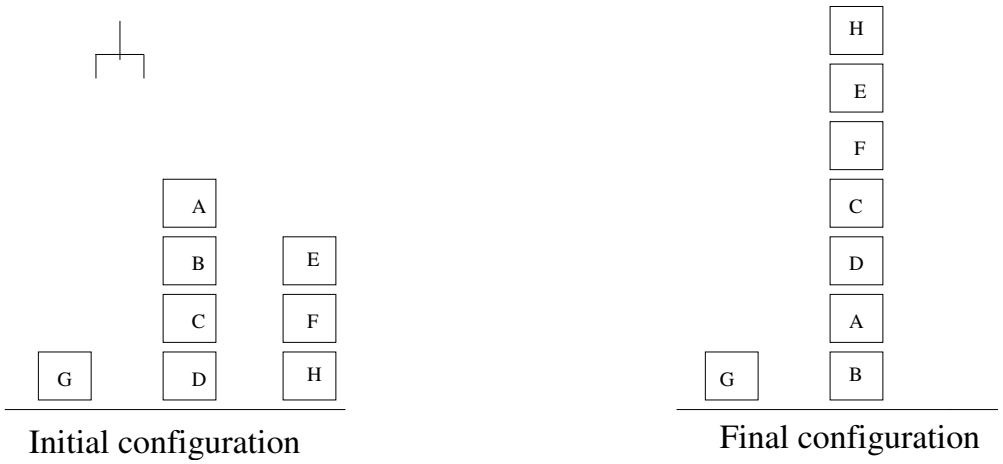


Table 4.11: Experiments for the block worlds domain

	Ordering of exceptions	$thresh_{no-traj}$	5-10	10-15
Independent	Damage(C) followed Damage(F)	.7	BW-Exp-1	BW-Exp-2
	Damage(C) followed Damage(F)	.9	BW-Exp-3	BW-Exp-4
	Damage(F) followed Damage(C)	.9	BW-Exp-5	BW-Exp-6
Nested	Damage(C) followed Damage(F)	.9	BW-Exp-7	BW-Exp-8
	Damage(F) followed Damage(C)	.9	BW-Exp-9	BW-Exp-10

The agent used a learning rate of .01, discount rate of .9 and ϵ- greedy exploration, with an ϵ of 0.001. The results were averaged over 5 independent runs. The trials were terminated either on completion of the task or after 10000 steps.

4.2.2 Exceptions

The state is described by the predicates which are true in the current time step. To introduce the exception, we need to ensure a policy action is invalidated, i.e., the policy is made to fail. From the set of 4 possible actions, only actions Stack and PutDown can be invalidated. UnStack cannot be invalidated because if the optimal policy contains unStack, then every other optimal policy will also contain unStack. Hence unStack is necessary to acheive the task. Similary if the block is on the table then pickUp cannot be invalidated as this is the scenario, when the optimal policy consists of pickUp. From the final configuration, as shown in Figure 4.14, we can conclude that Stack(H,E), Stack(E,F), Stack(F,C), Stack(C,D), Stack(D,A), Stack(A,B), PutDown(B) and PutDown(G) cannot be invalidated as they are necessary to achieve the final configuration. The other actions which occurs as policies and can be invalidated are PutDown(E), PutDown(F) and PutDown(C).

In order to invalidate these actions, we assume that if a block is damaged then it cannot be put down by the robot on the table. However, it can be stacked on some other block. As shown in Table 4.11, different sets of experiments where performed with different ordering of the damaged blocks, C and F. Block E was not damaged because the action PutDown(E) occurs quite early in the optimal policy, hence learning the exception policy would lead to learning the optimal policy from the start state(first landmark) to the end state(last landmark).

4.2.3 Landmarks extraction

We ran spiked state method to find the landmarks. 5000 trajectories were sampled starting from a random start state and ending at the end state. Most of times we either got sub figure (a) or (b) of Figure 4.15 as spiked state. These states were consecutive in the trajectories. Since the number of state that spiked are very less and also not uniformly distributed over the state space, we use mean-to-variance ratio to find the landmarks. We did two set of experiments. In the first set of experiment the actions were invalidated independent of each other i.e., either Damage(F) or Damage(C) is invalidated. This is shown as row, labeled "Independent" in Table 4.11. In the second set of experiments, the exceptions were introduced in sequential manner so that we can have the notion of nested exceptions. This is shown as row, labeled "Nested" in Table 4.11. Here the ordering of the exception matters. For example if Damage(F) occurs before Damage(C) in the optimal policy, then we need to invalidate Damage(C) and then invalidate Damage(F). Here when we are invalidating Damage(F), Damage(C) also remains invalidated. As the ordering is important, we further did two set of experiment depending upon the ordering of the action Damage(F) and Damage(C).

Within each set of the above experiments, two sets of experiments were further performed with a different bounded distance (in time) between the landmarks, resulting in a different number of landmarks for each set of experiments. The l_{min} and l_{max} were selected depending upon the number of steps taken by the agent to achieve the task and required distance between the landmarks. Since in this domain, number of steps taken by the agent was approximately around 22 steps, the l_{min} and the l_{max} were chosen

Table 4.12: Set of candidates for landmark arranged in increasing order of mean-to-variance ratio

State	mean(m)	variance(v)	m/v	
OTC(A),C(B), O(B,C), O(C,D), OT(D), O(E,F), O(F,H), OT(H), OTC(G), C(E)	2.223	0.257548	8.632556	
H(A),C(B), O(B,C), O(C,D), OT(D), O(E,F), O(F,H), OT(H), OTC(G), C(E)	1.109	0.125785	8.816584	
OTC(A),OTC(B), C(C), O(C,D), OT(D) OTC(F), OTC(H), OTC(G), OTC(E))	8.868	0.985909	8.994741	
OTC(A),C(B), O(B,C), O(C,D), OT(D) H(F), O(F,H), OTC(H), OTC(G), OTC(E))	5.555	0.616896	9.005173	
OTC(A),C(B), O(B,C), O(C,D), OT(D) C(F), O(F,H), OT(H), OTC(G), OTC(E)	4.445	0.493393	9.009180	
O(A,B), O(C,D), O(D,A) O(F,C),H(H), OTC(G), O(E,F)	23.337	2.539431	9.189853	
O(A,B), O(C,D), O(D,A) O(F,C),O(H,E), OTC(G), O(E,F)	**24.452**	**2.660362**	**9.191228**	**L3**
O(A,B), H(C), O(D,A) OTC(F), OTC(H), OTC(G), OTC(E)	16.661	1.811863	9.195871	
O(A,B), OTC(C), O(D,A) OTC(F), OTC(H), OTC(G), OTC(E)	15.537	1.689565	9.196070	
OTC(A),C(B), O(B,C), O(C,D), OT(D) OTC(F), OTC(H), OTC(G), OTC(E),	6.656	0.72313	9.204551	
O(A,B), O(C,D), O(D,A) O(F,C), OTC(H), OTC(G), O(E,F)	22.223	2.412973	9.210075	
H(A),OTC(B), C(C), O(C,D), OT(D) OTC(F), OTC(H), OTC(G), OTC(E))	9.977	1.082152	9.21989	
O(A,B), O(C,D), O(D,A) O(F,C), OTC(H), OTC(G), H(E)	21.108	2.275263	9.277312	
O(A,B), O(C,D), O(D,A) OTC(F), OTC(H), OTC(G), OTC(E)	17.771	1.914739	9.280983	
O(A,B), O(C,D), O(D,A) O(F,C), OTC(H), OTC(G), OTC(E)	19.993	2.146293	9.315440	
O(A,B), O(C,D), O(D,A) H(F), OTC(H), OTC(G), OTC(E)	18.881	2.022933	9.332981	
O(A,B), OTC(C), H(D) OTC(F), OTC(H), OTC(G), OTC(E)	**14.421**	**1.543388**	**9.344123**	**L2**
OTC(A),H(B), C(C), O(C,D), OT(D) OTC(F), OTC(H), OTC(G), OTC(E)	7.7522	0.828104	9.361446	
O(A,B), OTC(C), OTC(D) OTC(F), OTC(H), OTC(G), OTC(E)	13.306	1.419505	9.37429	
OTC(A),C(B), O(B,C), O(C,D), OT(D) C(F), O(F,H), OT(H), OTC(G), H(E)	3.321	0.349822	9.495881	
O(A,B), C(C), O(C,D), OT(D) OTC(F), OTC(H), OTC(G), OTC(E))	11.091	1.012453	10.95461	
O(A,B), H(C), OTC(D) OTC(F), OTC(H), OTC(G), OTC(E)	12.196	1.009543	12.08111	
O(A,B),O(B,C),O(C,D), OT(D), O(E,F), O(F,H),OT(H), OTC(G), C(E), C(A)	**0**	**0**	**Inf**	**L1**

Table 4.13: Set of candidates for landmark arranged in increasing order of mean-to-variance ratio

State	mean(m)	variance(v)	m/v	
OTC(D), C(F), OTC(E),O(F,H), OT(H), OTC(G), OTC(A), OTC(B),H(C)	**7.007**	**1.46212**	**4.79233**	**L2**
OTC(D), C(F), OTC(E),O(F,H), OT(H), OTC(G), OTC(A), OTC(B), OTC(C)	8.0	1.605	4.98442	
C(C),O(C,D), OT(D), C(F),OTC(E), O(F,H) OT(H), OTC(G), OTC(A), OTC(B)	6.003	1.185518	5.06614	
C(C),O(C,D), OT(D), C(F), OTC(E), O(F,H) OT(H), OTC(G), OTC(A), H(B)	5.0	0.92092	5.43478	
C(B),O(B,C),O(C,D), OT(D), C(F), OTC(E), O(F,H) OT(H), OTC(G), OTC(A)	4.0	0.69602	5.75263	
O(D,A),O(C,D), OT(D), O(F,C), H(E),C(F) O(F,H), OTC(H), OTC(G),O(A,B), O(C,D)	18.0	3.06	5.88235	
C(B),O(B,C),O(C,D), OT(D), C(F), OTC(E), O(F,H) OT(H), OTC(G), H(A)	3.001	0.48548	6.18556	
O(D,A),O(C,D), OT(D), H(F), OTC(E),C(C) O(F,H), OTC(H), OTC(G),O(A,B)	**16.0**	**2.37033**	**6.75011**	**L3**
O(D,A),O(C,D), OT(D), O(F,C), OTC(E), C(F),OTC(H), OTC(G),O(A,B), O(C,D)	17.0	2.5	6.8	
O(A,B),O(B,C),O(C,D), OT(D), C(F), OTC(E), O(F,H) OT(H), OTC(G), C(A)	2.006	0.27794	7.202881	
O(D,A),O(C,D), OT(D), C(F), OTC(E) O(F,H), OT(H), OTC(G),O(A,B), O(C,D)	15.041	2.07419	7.251047	
O(D,A), OTC(D), C(F), OTC(E) O(F,H), OT(H), OTC(G),O(A,B), H(C)	14.037	1.85561	7.564841	
O(A,B),O(B,C),O(C,D), OT(D), C(F), H(E), O(F,H) OT(H), OTC(G), C(A)	1.001	0.13146	7.61421	
O(D,A),O(C,D), OT(D), C(F), OTC(E) O(F,H), OT(H), OTC(G),O(A,B), OTC(C)	13.031	1.65586	7.869249	
H(D),O(C,D), OT(D), C(F), OTC(E) O(F,H), OT(H), OTC(G),O(A,B), OTC(C)	12.021	1.45676	8.251203	
O(D,A),O(C,D), OT(D), O(F,C), O(E,F),C(E) O(H,F), C(H), OTC(G),O(A,B), O(C,D)	**22.0**	**2.65233**	**8.29458**	**L4**
O(D,A),O(C,D), OT(D), O(F,C), O(E,F),C(E) O(F,H), H(H), OTC(G),O(A,B), O(C,D)	21.0	2.45733	8.54584	
C(D),O(C,D), OT(D), C(F), OTC(E) O(F,H), OT(H), OTC(G),O(A,B), OTC(C)	11.018	1.25208	8.8	
O(D,A),O(C,D), OT(D), O(F,C), O(E,F),C(E) OTC(H), OTC(G),O(A,B), O(C,D)	20.0	2.273	8.798944	
C(D),O(C,D), OT(D), C(F), OTC(E) OT(H), OTC(G),H(A), OTC(B), OTC(C)	10.011	1.11878	8.947211	
O(A,B),O(B,C),O(C,D), OT(D), O(E,F), O(F,H),OT(H), OTC(G), C(E), C(A)	**0.0**	**0.0**	**Inf**	*L1*

as 5 and 10 , 10 and 15 as shown under the columns 5-10 and 10-15 in Table 4.11. The number of trajectories used to collect the information regarding the landmarks and transition times between the landmarks was 3000. Though we had a different ordering for each experiment, we give the example of two Tables 4.12 and 4.13. The landmarks as shown in sub figure (b) of Figure 4.16 under experiment BW-Exp-8 and sub figure (b) of Figure 4.18 under experiment BW-Exp-7 are selected from the Table 4.12 and Table 4.13 respectively. In the table OTC implies *ontable and clear*, O implies *on*, C implies *clear*, OT implies *ontable* and H implies *holding*.

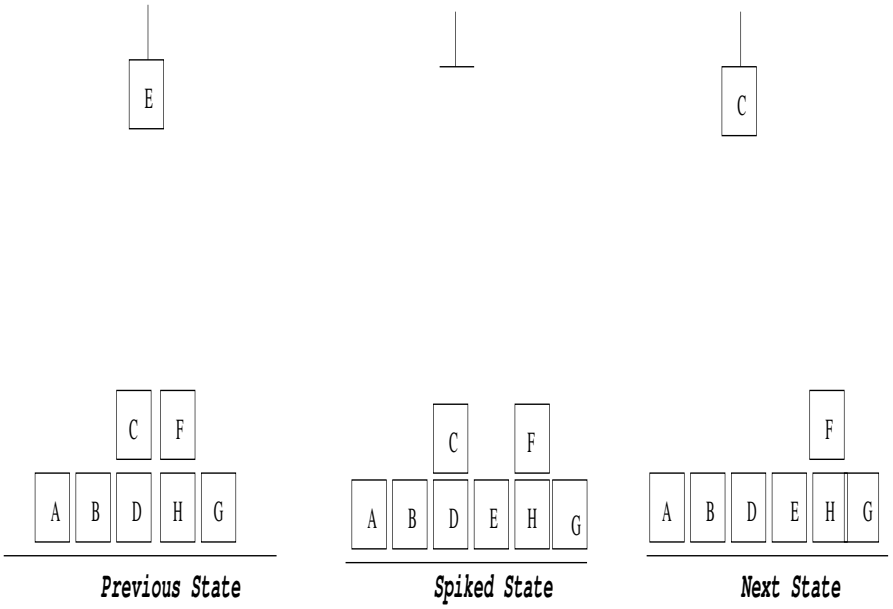
4.2.4 Results and Discussion

Within the experiment set "Independent" and "PutDown(C) followed PutDown(F)", we further perform the experiments depending upon the $thresh_{no-traj}$. This value will not affect the results as the exception is the result of the blocked action. But still we did the experiment with $thresh_{no-traj}$= .7 and $thresh_{no-traj}$= .9 and found in both cases the landmarks were not reachable as the action was blocked. The landmarks selected by mean-to-variance ratio are shown in Figures 4.16, 4.17 and 4.18.

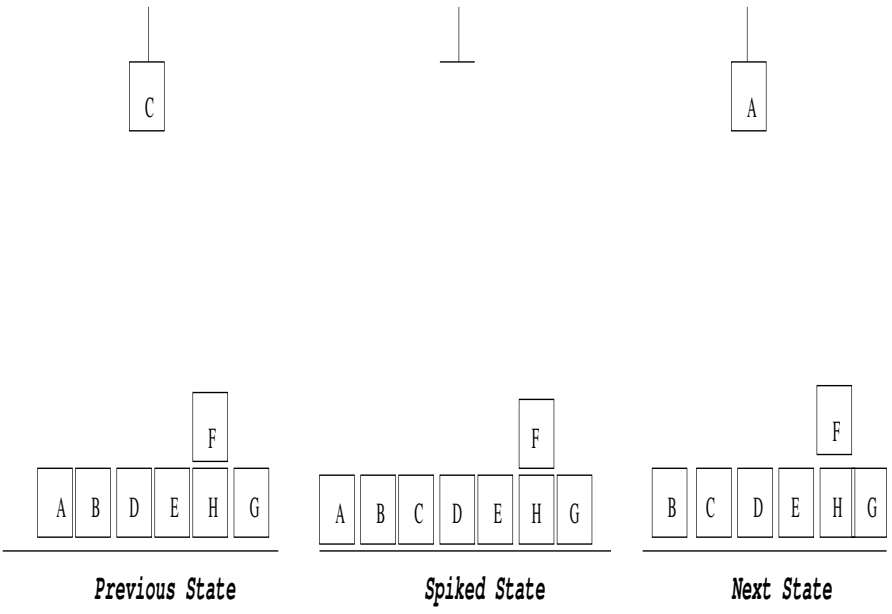
The maximum number of steps were assigned as 100 for the simulation of the path from start to exception state. Similar number of steps were assigned from the potential terminal landmarks to the terminal state. The trials for the exception policy that is from exception state to potential landmark were terminated either on completion of the task or after 9800 steps.

When learning the exception policy, only few landmarks called as the valid landmarks would be considered from the candidate set of possible terminal states for the exception policy. The valid landmarks are those landmarks in which the corresponding pickUp action of the invalidated PutDown action has been executed. The absence of valid landmarks other than the goal state leads to global changes in the domain. This can be observed in all the experiments, sub figure (a), (b), (c), (d) of Figure 4.19, (c) and (d) of Figure 4.21 and (c), (d) of Figure 4.22 where the landmark distance was 10-15 steps and the candidate set of terminal states consist of only the goal state. Similarly, when the landmark distance is 5-10, we can see the similar pattern as shown in sub figure (b), (d) of Figure 4.19 and (c) of Figure 4.21. This happens because the Put-

Figure 4.15: Spiked States

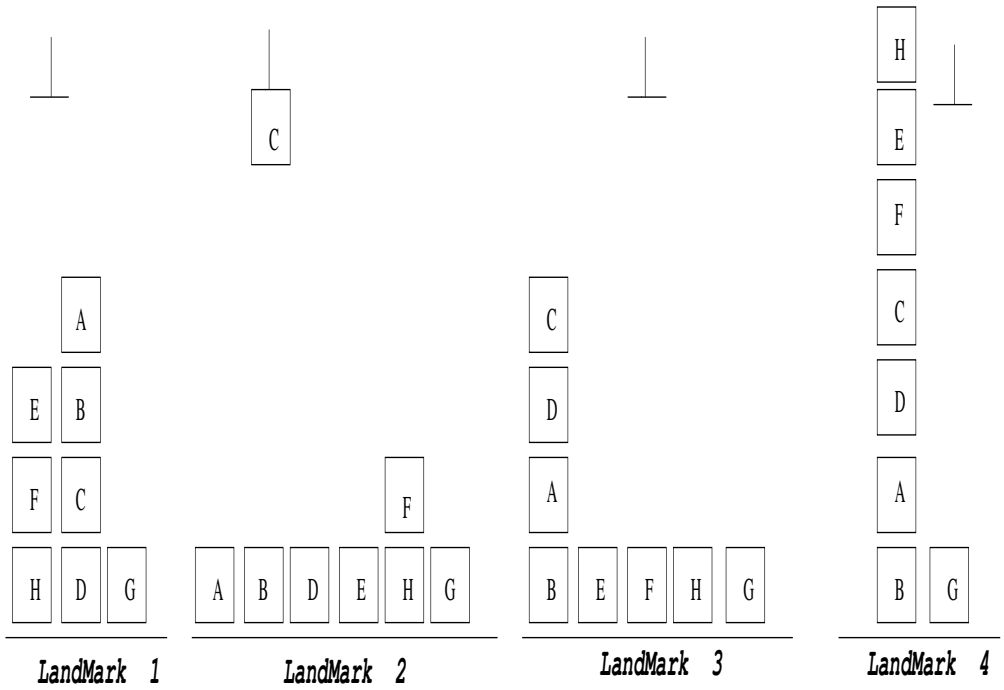


(a) η(Previous state) = 4501,η(Spiked state) = 4955,η(Next state) = 4450

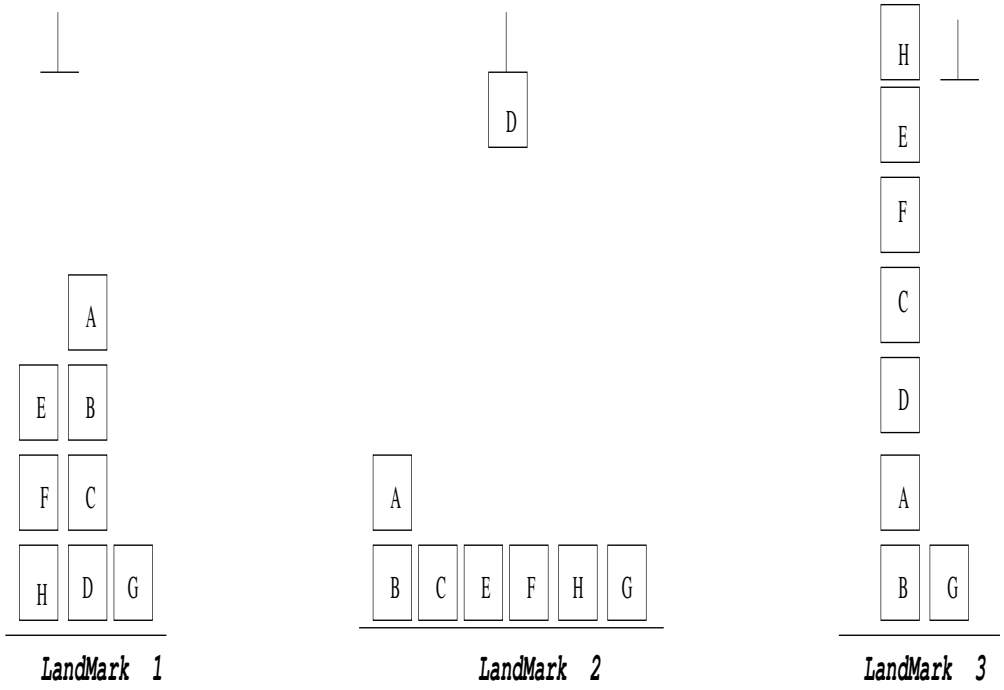


(b) η(Previous state) = 4670,η(Spiked state) = 4677,η(Next state) = 4466

Figure 4.16: Landmarks for Experiments BW-Exp-1,BW-Exp-2,BW-Exp-6, BW-Exp-8 and BW-Exp-10

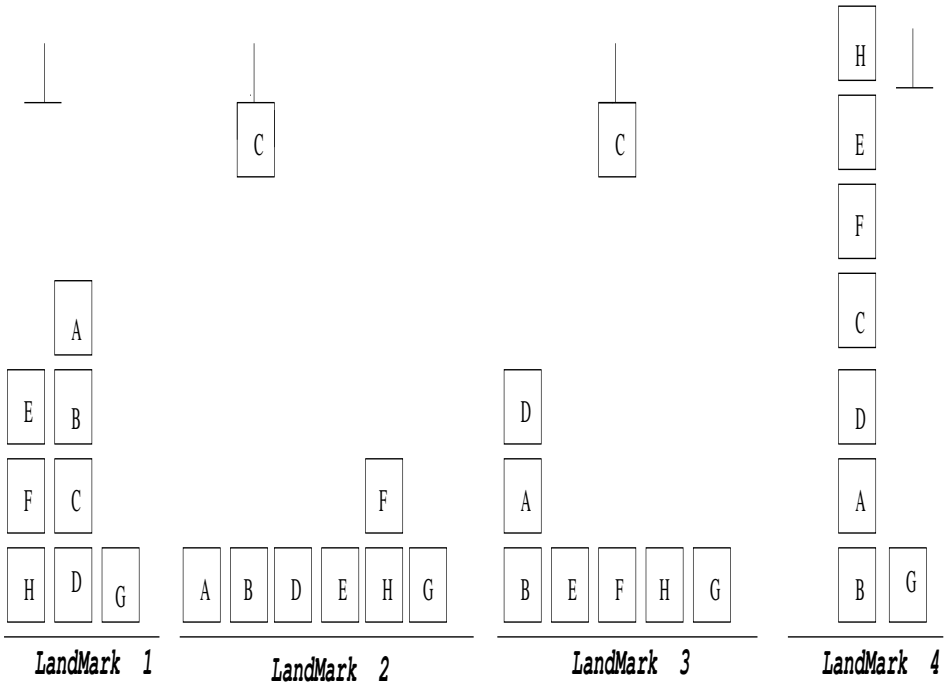


(a) BW-Exp-1

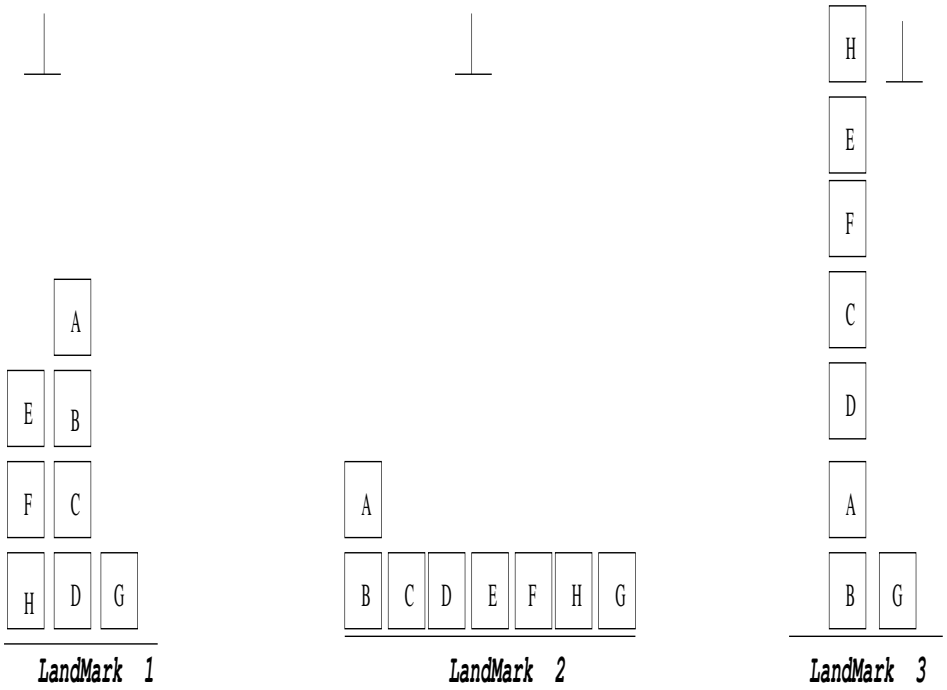


(b) BW-Exp-2,BW-Exp-6,BW-Exp-8 BW-Exp-10

Figure 4.17: Landmarks for Experiments BW-Exp-3 and BW-Exp-4

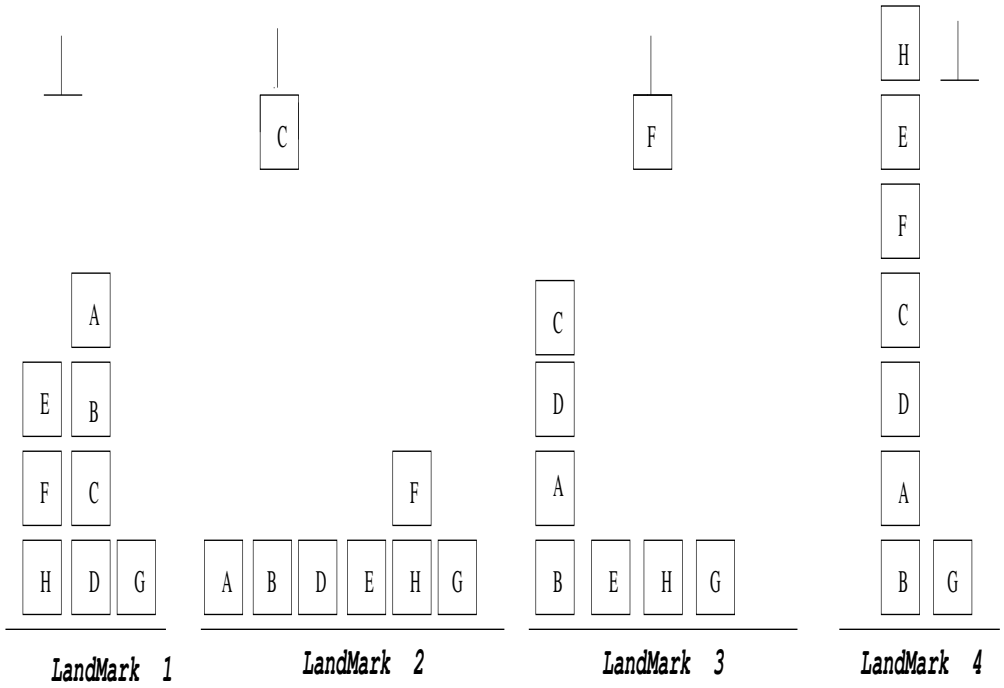


(a) BW-Exp-3

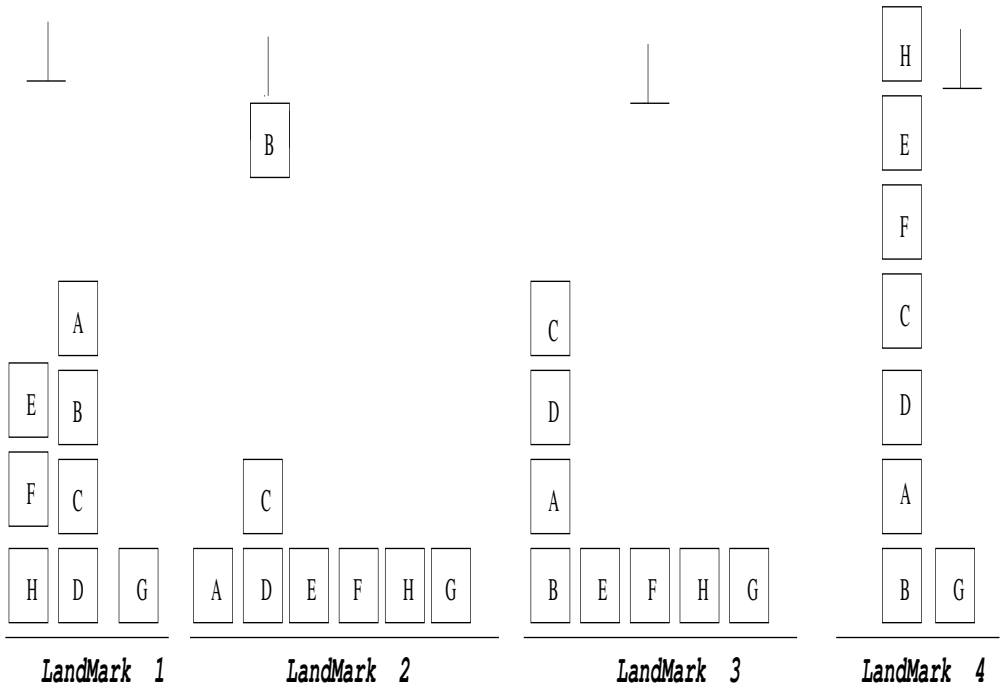


(b) BW-Exp-4

Figure 4.18: Landmarks for Experiments BW-Exp-5,BW-Exp-7 and BW-Exp-9



(a) BW-Exp-5,BW-Exp-9



(b) BW-Exp-7

Down(F) action occurs very early in the optimal policy. Invalidation of PutDown(C) gives similar results in some experiments as shown in sub figure (b) in Figures 4.21 and 4.23. In all these cases the exception policy was at least better than the policy obtained by Q learning.

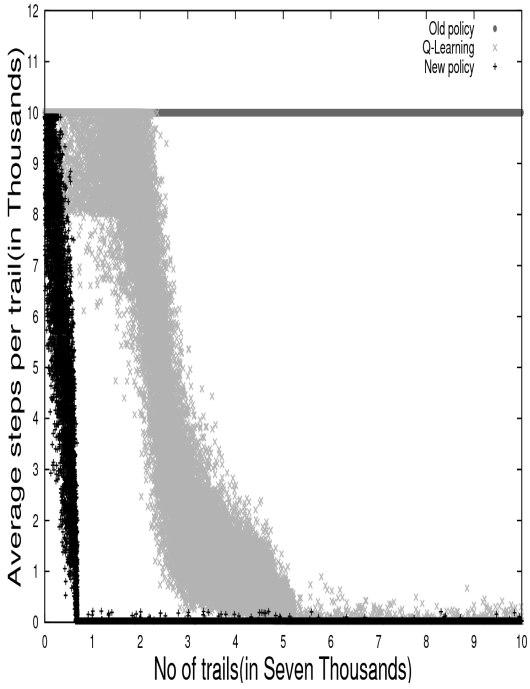
In all the experiments, where the landmark distance is 5-10, the exception policy learnt after the first exception is far better than the Q learning. This happened because the candidate set of landmarks consisted of valid potential terminal states different from the goal state. The exception policy can terminate at one of these valid landmarks. This can be seen in sub figure (a) and (c) of Figure 4.19, (a) of Figures 4.22 and 4.23.

Table 4.14: Transfer back of the policy to original domains (Avg No of steps)

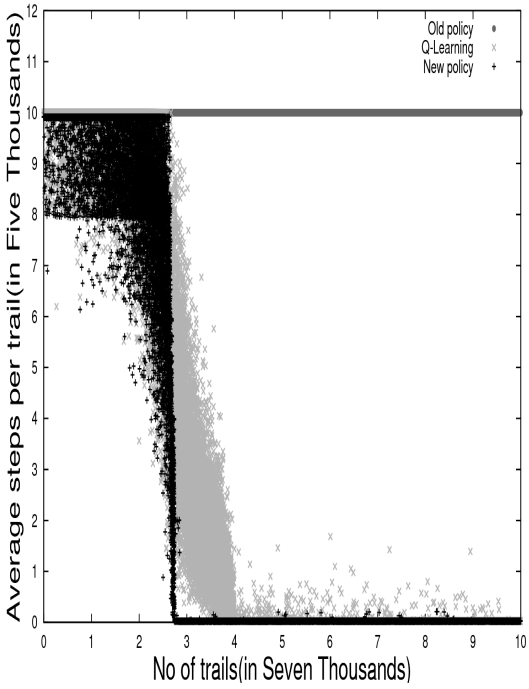
		Initial Environment	Damage (C)
BW-Exp-1	Initial Environment	25.321	
	Damage C	25.755	25.796
	Damage (F)	25.324	25.858
BW-Exp-2	Initial Environment	25.89	
	Damage (C)	25.755	25.294
	Damage (F)	25.324	25.376
BW-Exp-3	Initial Environment	25.09	
	Damage (C)	25.543	25.359
	Damage (F)	25.764	25.96
BW-Exp-4	Initial Environment	25.89	
	Damage (C)	25.7768	25.32
	Damage (F)	25.756	25.869
BW-Exp-7	Initial Environment	25.851	
	Damage (C)	25.433	25.87
	Damage (F)	25.488	25.934
BW-Exp-8	Initial Environment	25.89	
	Damage (C)	25.694	25.498
	Damage (F)	25.55	25.584

When the OWE policies were used in the original domains, i.e., without exceptions or fewer exceptions, the average number of steps taken to complete the task did not change appreciably, as shown in Tables 4.13, 4.14. The average was taken over 1000 trails and the maximum length of the trajectory is 500. In Table 4.3, the rows indicate the transfer of policies to the old domain after a particular object was introduced in the domain and the columns indicate the different objects present in the domain at that time. The values under the column, “Initial Environment” , “Damage(C)” in Table 4.14 and “Intial Environement” , “Damage(F)” in Table 4.15)are same because the damaged block could be placed on some other block and end state could be achieved. Hence

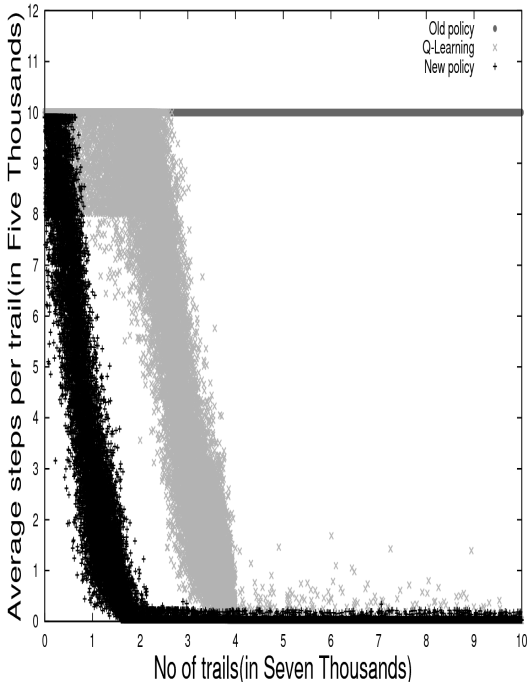
Figure 4.19: Comparison of old, new and policy learnt using Q-learning for BW-Exp-1 and BW-Exp-3



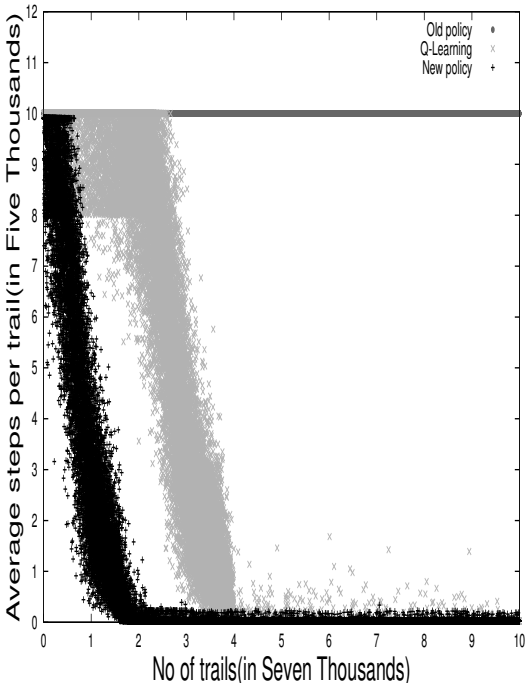
(a) Exception caused by the damaged Block C in BW-Exp-1



(b) Exception caused by damaged Block F in BW-Exp-1

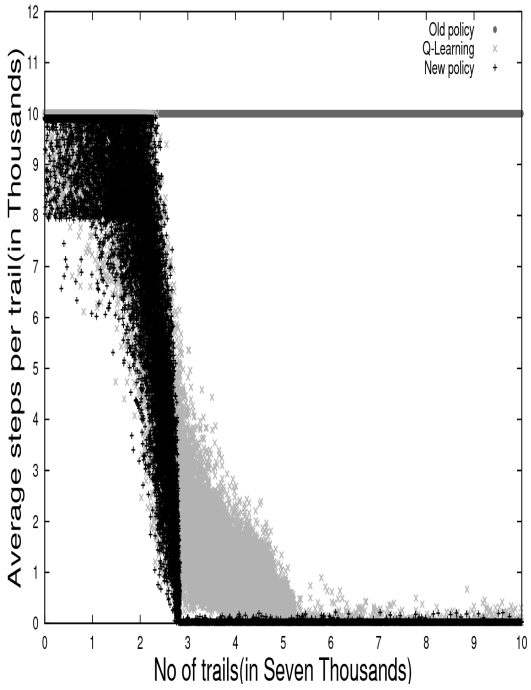


(c) Exception caused by damaged Block C in BW-Exp-3

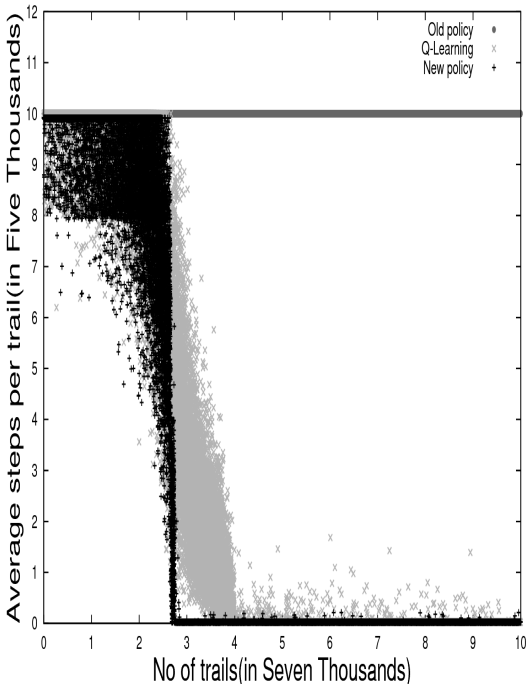


(d) Exception caused by damaged Block F in BW-Exp-3

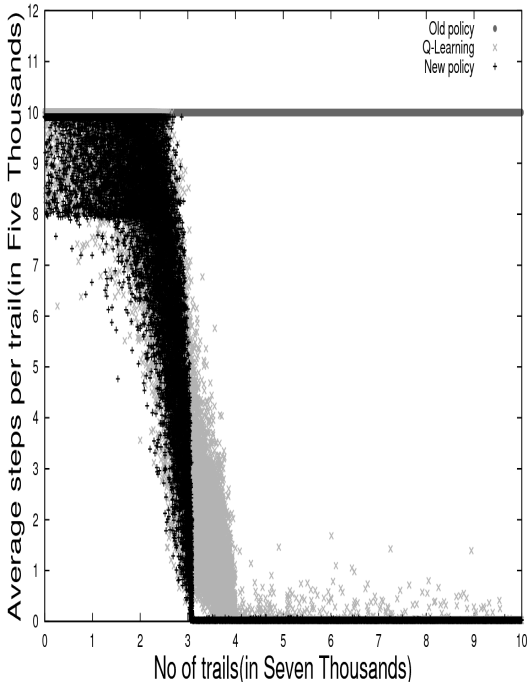
Figure 4.20: Comparison of old, new and policy learnt using Q-learning for BW-Exp-2 and BW-Exp-4



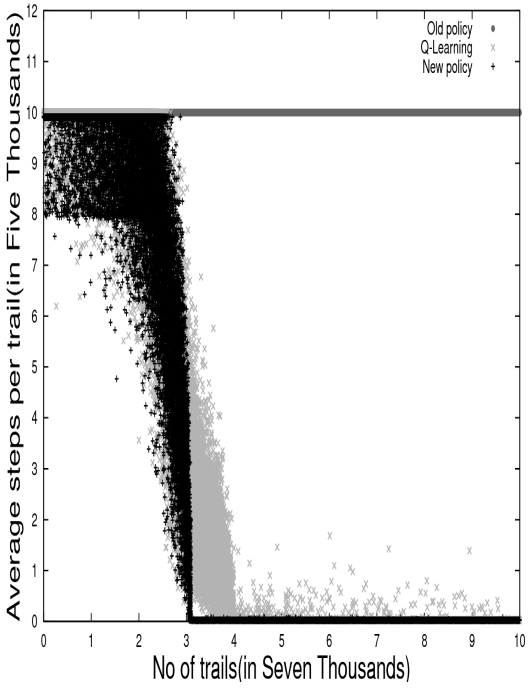
(a) Exception caused by the damaged Block C in BW-Exp-2



(b) Exception caused by damaged Block F in BW-Exp-2

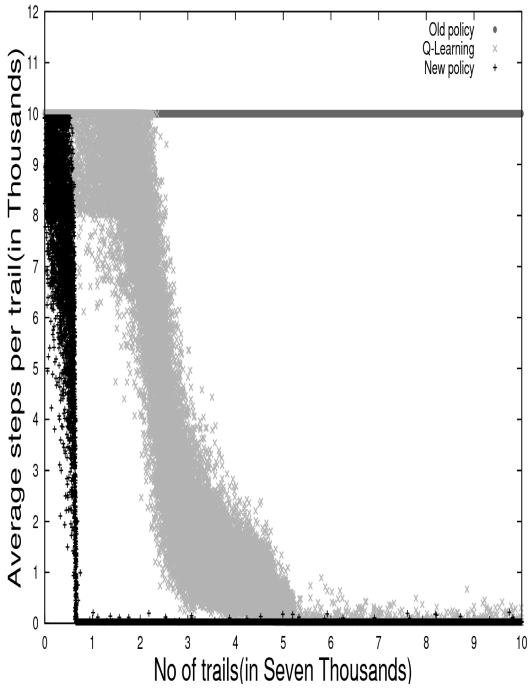


(c) Exception caused by damaged Block C in BW-Exp-4

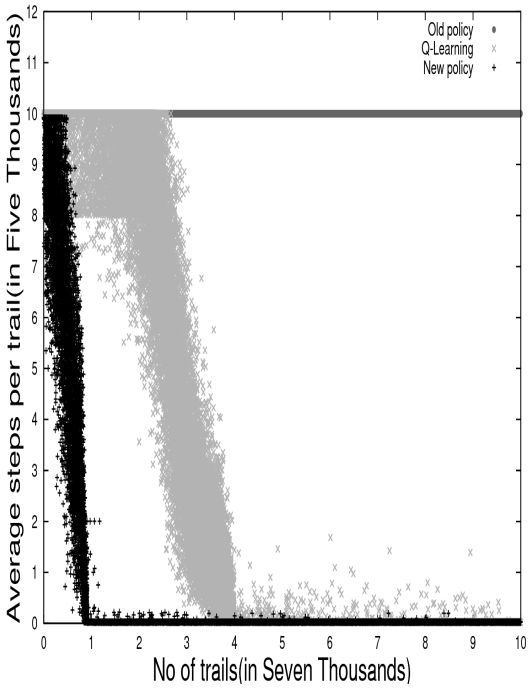


(d) Exception caused by damaged Block F in BW-Exp-4

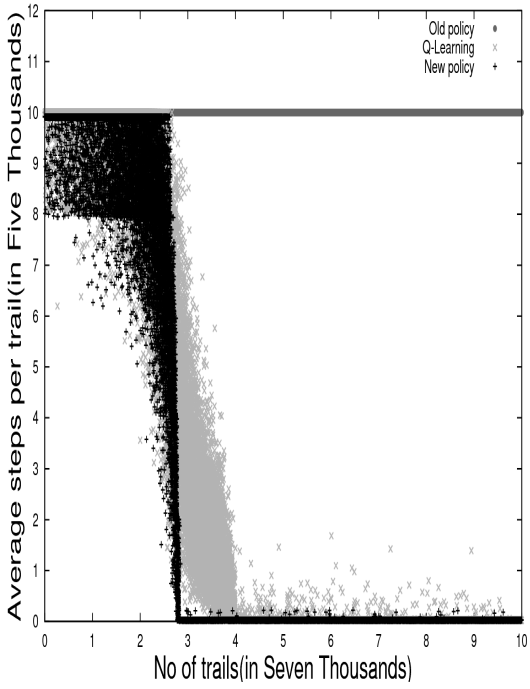
Figure 4.21: Comparison of old, new and policy learnt using Q-learning for BW-Exp-5 and BW-Exp-6



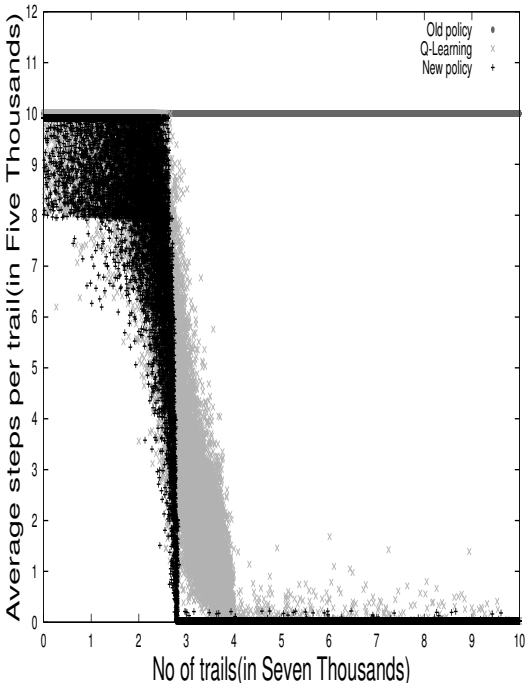
(a) Exception caused by the damaged Block F in BW-Exp-5



(b) Exception caused by damaged Block C in BW-Exp-5

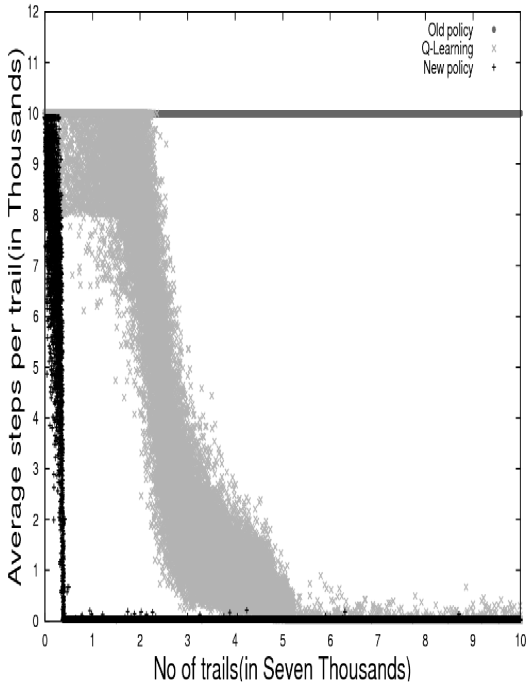


(c) Exception caused by damaged Block F in BW-Exp-6

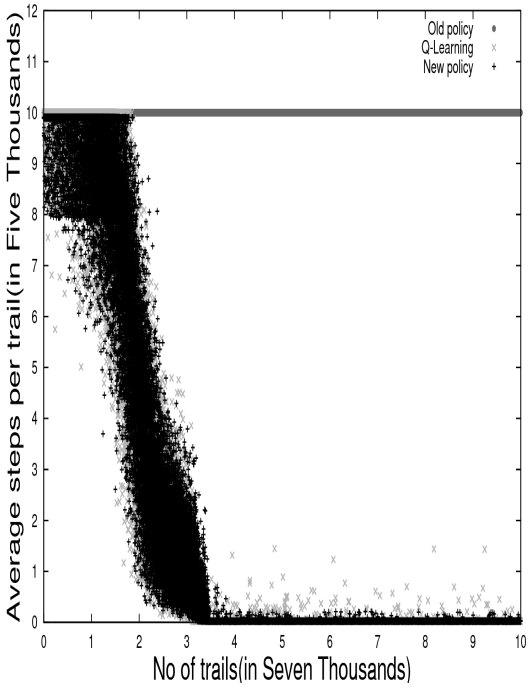


(d) Exception caused by damaged Block C in BW-Exp-6

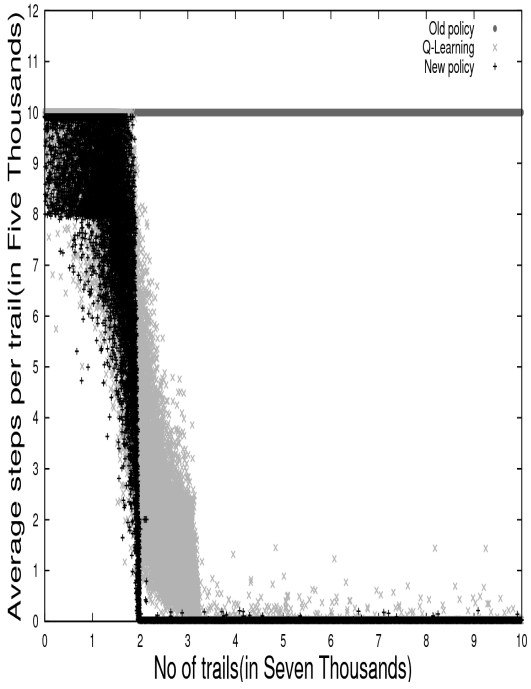
Figure 4.22: Comparison of old, new and policy learnt using Q-learning for BW-Exp-7 and BW-Exp-8



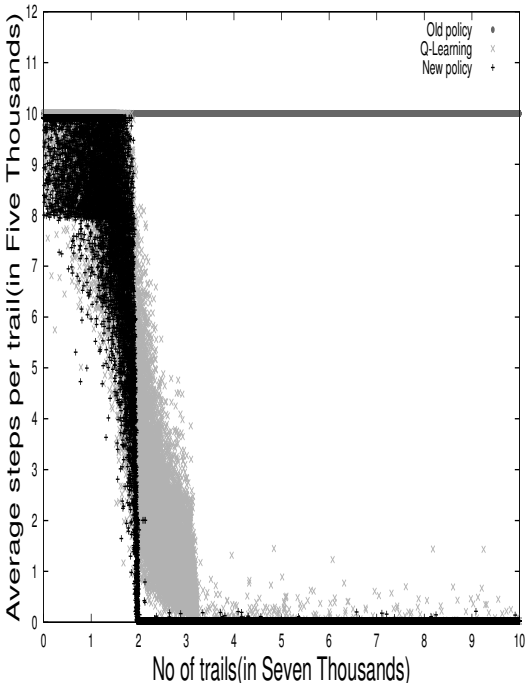
(a) Exception caused by the damaged Block F in BW-Exp-7



(b) Exception caused by damaged Block F and C in BW-Exp-7

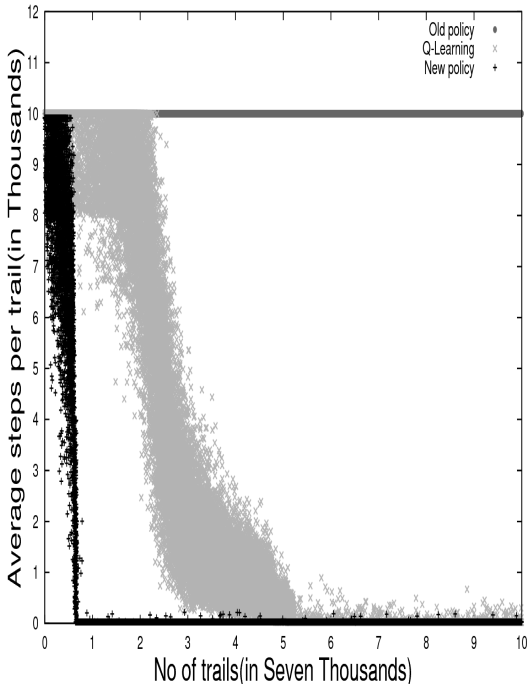


(c) Exception caused by damaged Block F in BW-Exp-8

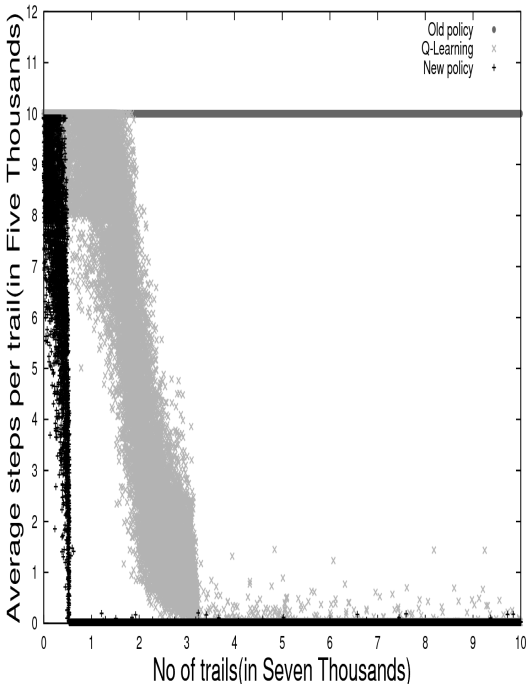


(d) Exception caused by damaged Block F and C in BW-Exp-8

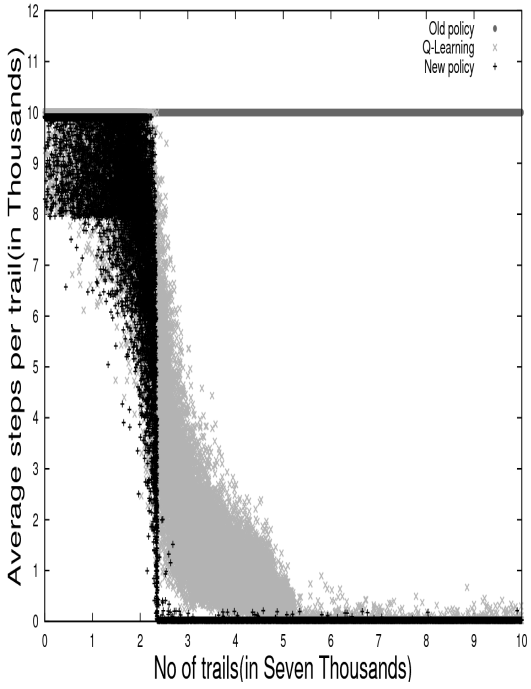
Figure 4.23: Comparison of old, new and policy learnt using Q-learning for BW-Exp-9 and BW-Exp-10



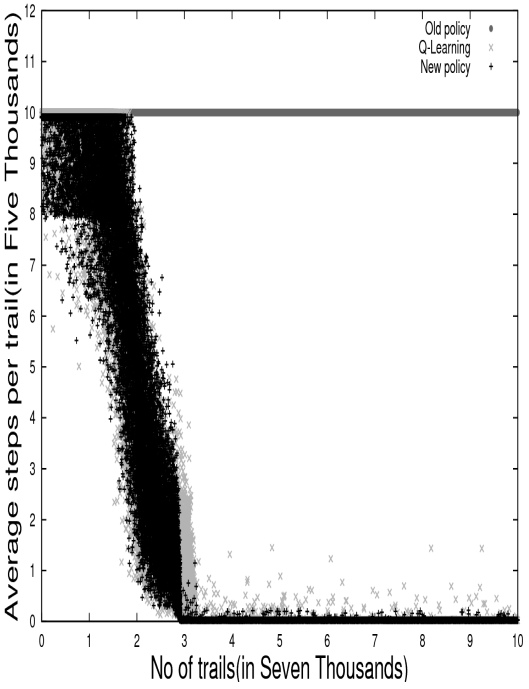
(a) Exception caused by the damaged Block C in BW-Exp-9



(b) Exception caused by damaged Block F and C in BW-Exp-9



(c) Exception caused by damaged Block C in BW-Exp-10



(d) Exception caused by damaged Block F and C in BW-Exp-10

Table 4.15: Transfer back of the policy to original domains (Avg No of steps)

		Initial Environment	Damage(F)
BW-Exp-5	Initial Environment	25.561	
	Damage (F)	25.902	25.413
	Damage (C)	25.543	25.685
BW-Exp-6	Initial Environment	25.89	
	Damage (F)	25.186	25.52
	Damage (C)	25.756	25.855
BW-Exp-9	Initial Environment	25.89	
	Damage (F)	25.483	25.428
	Damage (C)	25.466	25.629
BW-Exp-10	Initial Environment	25.89	
	Damage (F)	25.833	25.854
	Damage (C)	25.5	25.96

blocking the putDown action doesn't changes the no of steps . Though block "G" was provided as a replacement for the table, so that damaged block could be placed on it, it was rarely used by the agent.

There is randomness in the domain in the execution of the action as explained in the domain section and also because of ϵ =.001. Because of this randomness, a state may exist in the trajectory that might not be represented by the suffix tree. That is why in the table, we see some form of randomness.

4.3 Conclusion

In this chapter, we tested the *OWE* framework with the Grid World and Blocks World domain. In the experiments we evaluated the Transition Time Model and learned the exception policy. We found that exception policy learned is better than the original policy. We also found that because of the complex exception structure, the policies learned using the framework might be quite far away from the optimal policy.

CHAPTER 5

Conclusion and Future Directions

Spatio-temporal abstraction of skills with the notion of skill specific representation is an approach for solving large scale RL problems. These skill specific representations cut down the learning time and achieve a better success rate when applied to new problems. But seldom the new set of problems are exactly of the same form on which the skill specific representation is learned on. Transferring the skill specific abstraction to the entirely different domain may not work as the old policy may break down completely. However the transfer may work if the domain is slightly different from the domain on which representation is learned on. This slight difference might be in the form of small changes in the environment such as change in the configuration or change in the dynamics of the world. Applying the old policy here might be suboptimal or may result in failure. This failure is attributed to a small change in the policy rather than complete breakdown of the policy. In such a scenario repairing the old policy might be better option than learning a new policy from scratch.

In order to repair the old policy, one needs to detect when and where the failure has occurred and then change the failed policy. But changing the old policy might end up in losing the policies of earlier tasks. Further, these changes may create a different spatial abstraction if the skills are learned with skill specific representation to represent the modified solution, thus losing the abstraction of earlier learned skills. Hence we need a framework which can accommodate such changes with minimal variation in the spatial abstraction still maintaining the quality of the solution of the original subtask that the skill was learned on.

5.1 Contributions

We proposed novel methods to find the landmarks in the spatial representation of the task. Landmarks are places or sites that are visited often and are used to either find the way backward or forward. Thus we define a landmark for a region as a state that

is mostly visited when paths are traversed (successful or unsuccessful) through that region. The first method is called Spiked State method. This is based on the notion of junction which most people try to associate as landmarks while traveling. The heuristic is based on this idea, where the junction of many paths is called a spiked state. One of the problems with this heuristic is that it requires the domain to have large number of different paths through the states which enables us to find the required pattern to identify the spiked states. The second heuristic is called Mean-to-Variance ratio, were we find the landmarks only on the *st-trajectory*. This problem can be formulated as a constrained subset selection problem. The first constraint is that the landmarks are selected in increasing order of mean to variance ratio. The second constraint is that distance between two consecutive landmarks is bounded between the minimum and the maximum limit. We also formulated this problem as an minimization problem. We used greedy approach to find the solution as described in the algorithm. Sometimes the approach fails and the landmarks violate either the minimum or maximum limit. But the failures are local and are not cascading. Since *OWE* framework can also work with non uniform distribution of landmarks on the *st-trajectory*, the local failure does not affect.

Landmarks further allow us to represent the task abstractly and provide a novel option management framework that enables us to safely update options. This framework is called Transition Time Model and it helps to find the exception region. Transition Time Model consists of a landmark network which captures the connectivity information between the landmarks. The model stores the average transition time and the relative variability of the transition time between pair (u, v) of landmarks, where the pair is formed by the landmark u that follows directly the landmark v in many st-trajectories. Similarly an Auxiliary Model is built using the landmark network. It stores the average transition time extracted from the *stexception- trajectories*. Change In Landmark Distance measure compares the average transition time extracted from the Transition Time Model and the Auxiliary model between the pair of landmarks. When this measure deviates above the relative variability, exception is triggered between the pair of landmarks. Once the exception region is found, the Transition Probability Model identifies the first exception state where the current policy fails.

Since we are looking at skill specific representation, the option representation should be able to make the required changes to the affected parts of the existing representation

and leave the rest of representation unaffected. This can be achieved using a suffix tree. Similar to Ripple Down Rules, this framework allows to modify an option's policy using instances only where the existing policy seems to fail. This results in context specific updates. In order to maintain minimal variation in the abstraction, changes in the policy are represented by the features extracted from these instances.

5.2 Future Work

While the framework succeeds in identifying small changes required to be made to the options, we still have some distance to go before building a complete skill management system. Regarding the identification of exception, in the experiments we found that though exception is triggered, the exception policy learned remains the same as old one. This happens for particular types of obstacles placed in the domain. More principled approach would be to automatically identify when an exception would suffice and when a new option is warranted.

The nested exceptions lead to more deeper suffix tree which further leads to huge computation while storing and retrieving the policies. Usually the addition of exception policy to the current set of policies might be suboptimal policy. With the complex exception structure the polices learned using the framework might be quite far away from the optimal policy. A more principled approach would be to find the threshold, which will allow us to decide when to discard the current exception policies and learn a new policy from scratch. A less stringent approach would be to flatten the exception structure in the suffix tree without disturbing the exception.

The proposed framework cannot handle the notion of dynamic obstacles. In fact, a moving obstacle would not cause an exception as the state in which the obstacle is present might not even be a part of the sampled trajectories. One has to look at the expanded representation of the state space to capture the exception caused by the dynamic obstacle, which is not handled in this thesis.

REFERENCES

1. **Amarel, S.**, On representations of problems of reasoning about actions. *In* **D. Michie** (ed.), *Machine Intelligence 3*. American Elsevier Publisher, 1968, pp 131–171.

2. **Asadi, M.** and **M. Huber**, Autonomous subgoal discovery and hierarchical abstraction for reinforcement learning using monte carlo method. *In* **M. M. Veloso** and **S. Kambhampati** (eds.), *Proceedings, The Twentieth National Conference on Artificial Intelligence and the Seventeenth Innovative Applications of Artificial Intelligence Conference, July 9-13, 2005, Pittsburgh, Pennsylvania, USA*. AAAI Press / The MIT Press, 2005, pp 1588–1589. ISBN 1-57735-236-X.

3. **Bacon, P.**, **J. Harb**, and **D. Precup**, The option-critic architecture. *In* **S. P. Singh** and **S. Markovitch** (eds.), *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, pp 2017, San Francisco, California, USA.*. AAAI Press, 2017, pp 1726–1734.

4. **Barto, A. G.** and **S. Mahadevan** (2003). Recent advances in hierarchical reinforcement learning. *Discrete Event Dynamic Systems*, **13**(1-2), pp 41–77.

5. **Chapman, D.** and **L. P. Kaelbling**, Input generalization in delayed reinforcement learning: An algorithm and performance comparisons. *In Proceedings of the 12th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'91. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1991, pp 726–731. ISBN 1-55860-160-0.

6. **Compton, P.** and **R. Jansen**, Knowledge in context: A strategy for expert system maintenance. *In Proceedings of the Second Australian Joint Conference on Artificial Intelligence*, AI '88. Springer-Verlag New York, Inc., New York, NY, USA, 1990*a*, pp 292–306. ISBN 0-387-52062-7.

7. **Compton, P.** and **R. Jansen** (1990*b*). A philosophical basis for knowledge acquisition. *Knowl. Acquis.*, **2**(3), pp 241–257. ISSN 1042-8143.

8. **Dean, T.** and **K. Kanazawa** (1989). A model for reasoning about persistence and causation. *Comput. Intell.*, **5**(3), pp 142–150. ISSN 0824-7935.

9. **Dietterich, T. G.**, State abstraction in MAXQ hierarchical reinforcement learning. *In* **S. A. Solla**, **T. K. Leen**, and **K. Müller** (eds.), *Advances in Neural Information Processing Systems 12, [NIPS Conference, Denver, Colorado, USA, November 29 - December 4, 1999]*. The MIT Press, 1999, pp 994–1000. ISBN 0-262-19450-3.

10. **Gaines, B.**, Transforming rules and trees into comprehensible knowledge structure. *In* **U. Fayyad**, **G. Piatetsky-Shapiro**, **P. Smyth**, and **R. Uthurusamy** (eds.), *Knowledge Discovery in Databases II*. Cambridge, Massachusetts, AAAI/MIT press, 1995, pp 205–226.

11. **Gaines, B. R.** and **P. Compton** (1995). Induction of ripple-down rules applied to modeling large databases. *Journal of Intelligent Information Systems*, **5**(3), pp 211–228. ISSN 1573-7675.

12. **Hernandez-gardiol, N.** and **S. Mahadevan**, Hierarchical memory-based reinforcement learning. *In Advances in Neural Information Processing Systems (NIPS)*. MIT Press, 2000, pp 1047–1053.

13. **Jonsson, A.** (2006). *A Causal Approach to Hierarchical Decomposition in Reinforcement Learning*. Ph.D. thesis, Department of Computer Science, University of Massachusetts, Amherst, USA.

14. **Jonsson, A.** and **A. G. Barto**, Automated state abstraction for options using the u-tree algorithm. *In* **T. K. Leen**, **T. G. Dietterich**, and **V. Tresp** (eds.), *Advances in Neural Information Processing Systems 13, Papers from Neural Information Processing Systems (NIPS) 2000, Denver, CO, USA*. MIT Press, 2000, pp 1054–1060.

15. **Kulkarni, T. D.**, **K. Narasimhan**, **A. Saeedi**, and **J. Tenenbaum**, Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. *In* **D. D. Lee**, **M. Sugiyama**, **U. von Luxburg**, **I. Guyon**, and **R. Garnett** (eds.), *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*. 2016, pp 3675–3683.

16. **McCallum, A. K.** (1995). *Reinforcement Learning with Selective Perception and Hidden State*. Ph.D. thesis, Department of Computer Science, The College of Arts and Science, University of Rocheater, USA.

17. **McGovern, A.** (2002). *Autonomous Discovery of Temporal Abstraction from Interaction with An Environment*. Ph.D. thesis, Department of Computer Science, University of Massachusetts, Amherst, USA.

18. **McGovern, A.** and **A. G. Barto**, Automatic discovery of subgoals in reinforcement learning using diverse density. *In* **C. E. Brodley** and **A. P. Danyluk** (eds.), *Proceedings of the Eighteenth International Conference on Machine Learning (ICML 2001), Williams College, Williamstown, MA, USA, June 28 - July 1, 2001*. Morgan Kaufmann, 2001, pp 361–368. ISBN 1-55860-778-1.

19. **Menache, I.**, **S. Mannor**, and **N. Shimkin**, Q-cut - dynamic discovery of sub-goals in reinforcement learning. *In Proceedings of the 13th European Conference on Machine Learning*, ECML '02. Springer-Verlag, London, UK, UK, 2002, pp 295–306. ISBN 3-540-44036-4.

20. **Moore, A. W.** and **C. G. Atkeson** (1995). The parti-game algorithm for variable resolution reinforcement learning in multidimensional state-spaces. *Machine Learning*, **21**(3), pp 199–233.

21. **Ravindran, B.** and **A. G. Barto**, Relativized options: Choosing the right transformation. *In* **T. Fawcett** and **N. Mishra** (eds.), *Machine Learning, Proceedings of the Twentieth International Conference (ICML 2003), August 21-24, 2003, Washington, DC, USA*. AAAI Press, 2003, pp 608–615. ISBN 1-57735-189-4.

22. **Simsek, Ö.**, **A. P. Wolfe**, and **A. G. Barto**, Identifying useful subgoals in reinforcement learning by local graph partitioning. *In* **L. D. Raedt** and **S. Wrobel** (eds.), *Machine Learning, Proceedings of the Twenty-Second International Conference (ICML 2005), Bonn, Germany, August 7-11, 2005*, volume 119 of *ACM International Conference Proceeding Series*. ACM, 2005, pp 816–823. ISBN 1-59593-180-5.

23. **Singh, S. P.**, **A. G. Barto**, and **N. Chentanez**, Intrinsically motivated reinforcement learning. *In Advances in Neural Information Processing Systems 17 [Neural Information Processing Systems, NIPS 2004, December 13-18, 2004, Vancouver, British Columbia, Canada]*. 2004, pp 1281–1288.

24. **Sorg, J.** and **S. P. Singh**, Linear options. *In* **W. van der Hoek**, **G. A. Kaminka**, **Y. Lespérance**, **M. Luck**, and **S. Sen** (eds.), *9th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2010), Toronto, Canada, May 10-14, 2010, Volume 1-3*. IFAAMAS, 2010, pp 31–38. ISBN 978-0-9826571-1-9.

25. **Sutton, R.** and **A. Barto**, *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 1998.

26. **Sutton, R. S.**, **D. Precup**, and **S. P. Singh** (1999). Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artif. Intell.*, **112**(1-2), pp 181–211.

27. **Uther, W. T. B.** and **M. M. Veloso**, Tree based discretization for continuous state space reinforcement learning. *In Proceedings of the Fifteenth National/Tenth Conference on Artificial Intelligence/Innovative Applications of Artificial Intelligence*, AAAI '98/IAAI '98. American Association for Artificial Intelligence, Menlo Park, CA, USA, 1998, pp 769–774. ISBN 0-262-51098-7.

28. **Watkins, C. J. C. H.** (1989). *Learning from Delayed Rewards*. Ph.D. thesis, Cambridge University, Cambridge, England.

LIST OF PAPERS BASED ON THESIS

1. Munu Sairamesh and Balaraman Ravindran. "Options with Exceptions" in 9th European Workshop, EWRL 2011, pp 165-176, Lecture Notes in Computer Science, volume-7188, Springer Berlin Heidelberg .