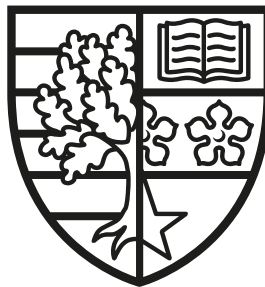


# Relational Knowledge and Representation for Reinforcement Learning

Ng Jun Hao, Alvin

SUBMITTED FOR THE DEGREE OF  
DOCTOR OF PHILOSOPHY

HERIOT-WATT UNIVERSITY



DEPARTMENT OF MATHEMATICS,  
SCHOOL OF MATHEMATICAL AND COMPUTER SCIENCES.

August, 2022

The copyright in this thesis is owned by the author. Any quotation from the thesis or use of any of the information contained in it must acknowledge this thesis as the source of the quotation or information.

## Abstract

In reinforcement learning, an agent interacts with the environment, learns from feedback about the quality of its actions, and improves its behaviour or policy in order to maximise its expected utility. Learning efficiently in large scale problems is a major challenge. State aggregation is possible in problems with a first-order structure, allowing the agent to learn in an abstraction of the original problem which is of considerably smaller scale. One approach is to learn the Q-values of actions which are approximated by a relational function approximator. This is the basis for relational reinforcement learning (RRL). We abstract the state with first-order features which consist of only variables, thereby aggregating similar states from all problems of the same domain to abstract states. We study the limitations of RRL due to this abstraction and introduce the concepts of consistent abstraction, subsumption of problems, and abstract-equivalent problems. We propose three methods to overcome the limitations, extending the types of problems our RRL method can solve. Next, to further improve the learning efficiency, we propose to learn different types of generalised knowledge. The policy is influenced by directed exploration based on multiple types of intrinsic rewards and avoids previously encountered dead ends. In addition, we incorporate model-based techniques to provide better quality estimates of the Q-values. Transfer learning is possible by directly leveraging the generalised knowledge to accelerate learning in a new problem. Lastly, we introduce a new class of problems which considers dynamic objects and time-bounded goals. We discuss the complications these bring to RRL and present some solutions. We also propose a framework for multi-agent coordination to achieve joint goals represented by time-bounded goals by decomposing a multi-agent problem into single-agent problems. We evaluate our work empirically in six domains to demonstrate its efficacy in solving large scale problems and transfer learning.

## **Acknowledgements**

I want to thank my advisor, Dr. Ronald P. A. Petrick, for his mentoring and advice. Ron has been patient and encouraging, and provides the right amount of guidance for me to learn and think for myself. I thank my examiners, Dr. Arash Eshghi and Dr. Siddharth Srivastava, for the insightful discussion and feedback. I am glad to have walked through this long journey with my peers at the Edinburgh Centre for Robotics. I thank them for their friendship, companionship, and encouragement. Also, I am deeply indebted to Hans-Nikolai Vießmann and Helmi Fraser, without whom I will not be able to run my experiments. Last but not least, I am grateful to my family and loved one for their support and understanding.

## Research Thesis Submission

Please note this form should be bound into the submitted thesis.

Name:	JUN HAO ALVIN NG		
School:	SCHOOL OF ENGINEERING AND PHYSICAL SCIENCES		
Version: <i>(i.e. First, Resubmission, Final)</i>	FINAL	Degree Sought:	PhD

### Declaration

In accordance with the appropriate regulations I hereby submit my thesis and I declare that:

1. The thesis embodies the results of my own work and has been composed by myself
2. Where appropriate, I have made acknowledgement of the work of others
3. The thesis is the correct version for submission and is the same version as any electronic versions submitted\*.
4. My thesis for the award referred to, deposited in the Heriot-Watt University Library, should be made available for loan or photocopying and be available via the Institutional Repository, subject to such conditions as the Librarian may require
5. I understand that as a student of the University I am required to abide by the Regulations of the University and to conform to its discipline.
6. I confirm that the thesis has been verified against plagiarism via an approved plagiarism detection application e.g. Turnitin.

### ONLY for submissions including published works


Please note you are only required to complete the Inclusion of Published Works Form (page 2) if your thesis contains published works)

7. Where the thesis contains published outputs under Regulation 6 (9.1.2) or Regulation 43 (9) these are accompanied by a critical review which accurately describes my contribution to the research and, for multi-author outputs, a signed declaration indicating the contribution of each author (complete)
8. Inclusion of published outputs under Regulation 6 (9.1.2) or Regulation 43 (9) shall not constitute plagiarism.

\* Please note that it is the responsibility of the candidate to ensure that the correct version of the thesis is submitted.

Signature of Candidate:		Date:	19/08/2022
-------------------------	---	-------	------------

### Submission

Submitted By <i>(name in capitals)</i> :	JUN HAO ALVIN NG
Signature of Individual Submitting:	
Date Submitted:	19/08/2022

### For Completion in the Student Service Centre (SSC)

Limited Access	Requested	Yes		No		Approved	Yes		No	
<i>E-thesis Submitted (mandatory for final theses)</i>										
Received in the SSC by <i>(name in capitals)</i> :						Date:				

## Inclusion of Published Works

Please note you are only required to complete the Inclusion of Published Works Form if your thesis contains published works under Regulation 6 (9.1.2)

---

### Declaration

This thesis contains one or more multi-author published works. In accordance with Regulation 6 (9.1.2) I hereby declare that the contributions of each author to these publications is as follows:

Citation details	
Author 1	
Author 2	
Signature:	
Date:	

Citation details	
Author 1	
Author 2	
Signature:	
Date:	

Citation details	
Author 1	
Author 2	
Signature:	
Date:	

Please included additional citations as required.

# Contents

List of Tables	vi
List of Figures	ix
List of Algorithms	xvi
Notation	xxiii
<b>1 Introduction</b>	<b>1</b>
1.1 Exploiting The Structure of a Problem . . . . .	3
1.2 Transferring Knowledge . . . . .	4
1.3 Other Aspects of a Problem . . . . .	5
1.4 Overview of Our Approach . . . . .	6
1.5 Objectives and Contributions . . . . .	6
1.6 Structure . . . . .	8
<b>2 Background and Related Work</b>	<b>10</b>
2.1 Problem Representations . . . . .	10
2.1.1 States and Actions . . . . .	11
2.1.2 Markov Decision Processes . . . . .	12
2.1.3 Relational Markov Decision Processes . . . . .	13
2.1.4 Semi-Markov Decision Processes . . . . .	15
2.1.5 Other Extensions of MDPs . . . . .	15
2.1.6 Terminal States, Goals, and Dead Ends . . . . .	16
2.2 Planning Languages . . . . .	16
2.2.1 STRIPS Action Models . . . . .	17
2.2.2 Probabilistic Planning Domain Definition Language . . . . .	17

2.2.3	Relational Dynamic Influence Diagram Language . . . . .	18
2.3	Generative Models . . . . .	19
2.3.1	Maximum Likelihood Model . . . . .	20
2.3.2	Dynamic Bayesian Network . . . . .	20
2.3.3	Model Learning . . . . .	21
2.3.4	Evaluate Correctness of Approximate Models . . . . .	24
2.4	Reinforcement Learning . . . . .	26
2.4.1	Model-Free Reinforcement Learning . . . . .	29
2.4.2	Model-Based Reinforcement Learning . . . . .	30
2.4.3	Measuring Performance . . . . .	34
2.5	Function Approximation . . . . .	36
2.5.1	State Abstraction . . . . .	36
2.5.2	Granularity . . . . .	37
2.5.3	Linear Function Approximation . . . . .	38
2.5.4	Feature Discovery . . . . .	39
2.5.5	Neural Network . . . . .	41
2.6	Relational Reinforcement Learning . . . . .	42
2.6.1	Model-Free RRL . . . . .	43
2.6.2	Model-Based RRL . . . . .	46
2.6.3	Planning Methods . . . . .	47
2.7	Additional Related Work . . . . .	49
2.7.1	Temporal Considerations . . . . .	49
2.7.2	Multi-Agent Coordination . . . . .	50
2.8	Benchmark Domains . . . . .	52
2.8.1	IPPC Benchmark Domains . . . . .	52
2.8.2	Robotic Domains . . . . .	56
2.8.3	Properties of Domains . . . . .	63
2.9	Summary . . . . .	64
<b>3</b>	<b>First-Order Approximation</b>	<b>65</b>
3.1	Online Relational Reinforcement Learning . . . . .	66
3.1.1	Online Feature Discovery . . . . .	68
3.1.2	Adaptive online feature discovery . . . . .	72

3.1.3	Update Q-function Approximation . . . . .	74
3.2	Ground Approximation . . . . .	75
3.3	Consistent Abstraction with First-Order Features . . . . .	77
3.3.1	Initialising First-Order Base Features . . . . .	78
3.3.2	Consistent Abstraction and Abstract-Equivalent Problems . . . . .	82
3.3.3	Augmenting First-Order Base Features . . . . .	85
3.3.4	Conjunctive First-Order Features . . . . .	87
3.4	Grounding First-Order Features . . . . .	88
3.4.1	Granularity and Pitfalls of First-Order Abstraction . . . . .	90
3.4.2	Contextual Knowledge . . . . .	96
3.5	Dealing with Plateaus . . . . .	102
3.5.1	Model-Based Q-value Tree Expansion . . . . .	102
3.5.2	Ensemble of Approximations . . . . .	106
3.6	Transfer Learning . . . . .	107
3.7	Empirical Evaluation and Discussion . . . . .	109
3.7.1	Experimental Setup . . . . .	109
3.7.2	Ablation Study for Contextual Knowledge . . . . .	114
3.7.3	Resolving Plateaus . . . . .	120
3.7.4	Transfer Learning . . . . .	125
3.8	Summary . . . . .	127
<b>4</b>	<b>Generalised Knowledge for RRL</b>	<b>129</b>
4.1	Types of Generalised Knowledge . . . . .	130
4.2	Model Predictions . . . . .	132
4.3	Model-Based Learning and Planning . . . . .	134
4.3.1	Model-Based Feature Selection . . . . .	134
4.3.2	UCT with Model-Free RRL . . . . .	141
4.4	Policy Control . . . . .	146
4.4.1	Learning from Dead Ends . . . . .	146
4.4.2	Intrinsic Motivation . . . . .	154
4.5	Empirical Evaluation and Discussion . . . . .	163
4.5.1	Correctness of Learned Models . . . . .	163
4.5.2	MBFS vs MFFS . . . . .	165



4.5.3	Learning from Dead Ends . . . . .	169
4.5.4	Intrinsic Motivation . . . . .	170
4.5.5	Continual Learning . . . . .	177
4.5.6	Combining Model-Based and Model-Free RL . . . . .	180
4.6	Summary . . . . .	183
<b>5</b>	<b>Dynamic Objects, Time, and Coordination</b>	<b>185</b>
5.1	Service Robot in the Real World . . . . .	186
5.2	Dynamic Object Relational Markov Decision Process . . . . .	187
5.2.1	Dynamic Objects in Service Robot . . . . .	187
5.3	Temporal Considerations . . . . .	188
5.3.1	Durative Actions . . . . .	189
5.3.2	Time-Bounded Goals . . . . .	190
5.3.3	Time-Dependent DORMDP . . . . .	192
5.4	Dealing with TDORMDP Problems . . . . .	193
5.4.1	Dynamic Base Features . . . . .	193
5.4.2	Representing Violated TGs . . . . .	195
5.4.3	Model-Based Methods . . . . .	198
5.4.4	Learning from Hindsight . . . . .	200
5.4.5	Extensions to Online RRL . . . . .	201
5.5	Simulation-to-Simulation Transfer . . . . .	201
5.5.1	Simulated Environment in Gazebo . . . . .	202
5.5.2	Differences between Simulated Environments . . . . .	203
5.6	Empirical Evaluation and Discussion . . . . .	204
5.6.1	Dynamic Objects and TGs . . . . .	205
5.6.2	Sim-to-Sim Transfer . . . . .	208
5.7	Multi-Agent Coordination . . . . .	210
5.7.1	Coordinated Actions . . . . .	211
5.7.2	Decomposition with Temporal Planning . . . . .	214
5.7.3	Auctioning TGs . . . . .	219
5.8	Summary . . . . .	226
<b>6</b>	<b>Discussion</b>	<b>228</b>

<b>7 Conclusions and Future Work</b>	<b>235</b>
7.1 Future Work . . . . .	238
<b>A Proofs</b>	<b>241</b>
A.1 Proofs from Chapter 3 . . . . .	241
A.2 Proofs from Chapter 4 . . . . .	243
<b>B Additional Examples</b>	<b>245</b>
B.1 Examples for MBFS . . . . .	245
<b>C Additional Empirical Results</b>	<b>249</b>
C.1 Empirical Results for Chapter 3 . . . . .	249
C.1.1 Blocks World . . . . .	249
C.1.2 Comparison with Other RRL Methods . . . . .	250
C.1.3 Sensitivity Analysis for Online Feature Discovery . . . . .	251
C.2 Empirical Results for Chapter 4 . . . . .	254
C.2.1 Tuning the Hyperparameter $\nu$ for MBFS . . . . .	254
C.2.2 Sensitivity Analysis for $\beta$ . . . . .	255
C.2.3 Sensitivity Analysis for Number of Rollouts in Dyna . . . . .	257
C.3 Empirical Results for Chapter 5 . . . . .	258
C.3.1 Effect of $\Delta T$ on TDORMDP Problems . . . . .	258
<b>D RDDL Domains</b>	<b>260</b>
D.1 Recon . . . . .	260
D.2 Robot Fetch . . . . .	263
D.3 Robot Inspection . . . . .	265
D.4 Service Robot . . . . .	269
D.5 Blocks World . . . . .	274
<b>Bibliography</b>	<b>277</b>

# List of Tables

3.1	Values of hyperparameters used in experiments. . . . .	112
3.2	The ranking of contextual knowledge based on the total rewards with goal-directedness as a tiebreaker. The number indicates the rank with 1 being the best, the symbol $\checkmark$ ( $\times$ ) indicates that the first-order approximation with contextual knowledge performs better (worse) than the ground approximation, NA indicates that the contextual knowledge is not applicable, and — indicates that the result is not available due to high computational costs. <b>Gr</b> denotes ground context, <b>L</b> denotes location context, and <b>G</b> denotes goal context. . . . .	118
3.3	The contextual knowledge used for each domain in all experiments unless stated otherwise. <b>Gr</b> denotes ground context, <b>G</b> denotes goal context, and <b>L</b> denotes location context. . . . .	118
3.4	Summary of the results for using different methods to resolve plateaus in the first-order approximation. The methods are (A) using decoupled weights, (B) using MQTE, and (C) using an ensemble of ground and first-order approximations. The symbols $\checkmark$ indicates that the method outperforms the ground approximation, $\times$ indicates that it worsens performance (relative to not using it at all), and a number indicates its ranking among all methods if it improves performance. .	122

3.5 Mean and the one standard deviation of six metrics accumulated in 3000 episodes and aggregated over ten problems: “One action” is the number of times MQTE has reduced the number of greedy actions to one, “Lesser actions” is the number of times MQTE has reduced the number of greedy actions but not to one, “No change” is the number of times MQTE has not reduced the number of greedy actions, “Fail to predict” is the number of times MQTE has not reduced the number of greedy actions due to the inability of the generative model to predict the outcomes, “% Effectiveness” is the percentage of times MQTE has reduced the number of greedy actions notwithstanding the failure of the model to predict, and “% Real Effectiveness” is the percentage of times MQTE has reduced the number of greedy actions. The first four metrics are mutually exclusive. . . . . 122

4.1 Average computation time and one standard deviation in seconds for RLFIT to learn the models given training data of 10, 50, and 500 state transitions per symbolic action. The computation time is aggregated over ten independent runs. . . . . 164

4.2 Summary of the performance of the different types of model-free intrinsic rewards. The abbreviations *GND* denotes ground approximation, *FO* denotes first-order approximation, *CT* denotes COUNT (Tabular), and *CL* denotes COUNT (LFA). ✓ indicates that the intrinsic reward improves performance relative to the baseline, × indicates that it worsens performance, and — indicates that it makes no difference. . . . . 174

7.1 Summary of the performance of the methods introduced in this dissertation. ✓ indicates that the method improves performance relative to the baseline (refer to description [Desc.]), × indicates that it did not improve performance, and — indicates that it makes no difference. Empty cells indicate that the method is not applicable for the domain. If there are two symbols in a cell, the first pertains to the ground approximation and the second pertains to the first-order approximation. AA denotes Academic Advising, RC denotes Recon, TT denotes Triangle Tireworld, RF denotes Robot Fetch, RI denotes Robot Inspection, and SR denotes Service Robot. . . . . 237

# List of Figures

2.1	The PPDDL action model for <code>move_vehicle</code> in <code>Triangle Tireworld</code> . . . . .	17
2.2	An excerpt of the domain file for <code>Triangle Tireworld</code> written in RDDL. $\dot{}$ denotes omitted parts of the domain file. . . . .	18
2.3	A portion of the DBN for a problem of <code>Triangle Tireworld</code> . Green rectangles denote actions at time step $t$ , blue (orange) ovals denote state predicates at time step $t$ ( $t + 1$ ). . . . .	21
2.4	Action models for <code>move_vehicle(<math>WP_1, WP_2</math>)</code> . Numeric values in parentheses represent the probabilities of effects. . . . .	25
2.5	The RL framework. . . . .	26
2.6	The four phases in MCTS: selection, expansion, simulation, and back-propagation. . . . .	31
2.7	A state for the problem TT3 of the <code>Triangle Tireworld</code> domain. A green circle denotes a location with a spare tire and an arrow denotes the direction of traversal allowed. . . . .	55
2.8	An environment in Gazebo for the <code>Service Robot</code> domain. A TIAGo robot fulfils a task from the person $p1$ who requested assistance by bringing an item $o1$ to the location $wp5$ . . . . .	60
3.1	An illustration of initialising a set of first-order base features for a symbolic action and the grounding of first-order features. To evaluate first-order features, they are grounded with consideration to the state $s_t$ , the objects in the action $a_1$ , and contextual knowledge. . . . .	79

3.2	Illustration of the relation between the ground approximation and the first-order approximation. Three ground approximations are learned in different problems with non-overlapping state-action spaces and are abstracted to a first-order approximation. A state-action space is represented by a large circle with dark shade. A state-action pair is represented by a small circle with light shade. . . . .	80
3.3	A problem of <b>Robot Fetch</b> which requires the robot to start by placing any of the objects, $o1$ , $o2$ , or $o3$ , at the location $wp1$ instead of their respective goal locations as none of their goal locations are empty. . .	92
3.4	A problem of <b>Recon</b> where there is a plateau in state $s_1$ for a first-order approximation. Left: the numberings represent the locations $wp1$ , $wp2$ , $\dots$ , $wp9$ . Center: state $s_1$ . Right: state $s_2$ . . . . .	95
3.5	Ablation study involving large scale problems from six domains. The sample complexity is indicated by the total undiscounted rewards received in each episode. . . . .	115
3.6	Ablation study involving large scale problems from six domains. The goal-directedness is measured by the cumulative number of goal states reached minus the cumulative number of dead ends reached. . . . .	116
3.7	Ablation study involving large scale problems from six domains. The computational cost is measured by the cumulative computation time taken in seconds. . . . .	117
3.8	Performance of different methods for resolving plateaus in the first-order approximation. The methods are (1) using decoupled weights ( $\tilde{Q}^{fo} (DW)$ ), (2) using MQTE ( $\tilde{Q}^{fo} (MQTE)$ ), and (3) using an ensemble of ground and first-order approximations ( $\tilde{Q}^{gnd} + \tilde{Q}^{fo}$ ). The baselines are ground approximation ( $\tilde{Q}^{gnd}$ ) where plateaus are unlikely and first-order approximation ( $\tilde{Q}^{fo}$ ) where plateaus are not resolved. . . . .	121

3.9 Results for transfer learning.  $\tilde{Q}^{gnd}$  ( $\tilde{Q}^{fo}$ ) represents the ground (first-order) approximation, and  $\tilde{Q}^{gnd} + \tilde{Q}^{fo}$  represents an ensemble. The subscripts  $\perp$  and  $\top$  denote that  $\tilde{Q}^{fo}$  is learned in and transferred from a small scale problem.  $\perp$  denotes that  $\tilde{Q}^{fo}$  is not updated in the large scale problem while  $\top$  denotes that  $\tilde{Q}^{fo}$  continues to be updated in the large scale problem. . . . . 125

4.1 A toy problem with two state predicates: X and Y. The solid and dotted arrows denote two actions. A red arrow denotes the action is optimal. The numerical values are the immediate rewards for executing the actions. On the left, there is no abstraction. On the right, Y is abstracted away resulting in two abstract states. . . . . 141

4.2 The four approaches in which the Q-values from a Q-function approximation,  $\tilde{Q}$ , can be combined with UCT. (A) The policy considers the Q-values of actions at the root node which is a mix of the Q-values estimated by UCT and by  $\tilde{Q}$ . (B) During the simulation phase, the rollout policy is generated from  $\tilde{Q}$ . (C) During backpropagation, the value of the last node in the rollout, which is not a terminal state, is estimated by  $\tilde{Q}$ . (D) During backpropagation, the Q-value of the leaf node is the mix of the return from the rollout and  $\tilde{Q}(s_L, a)$  where  $s_L$  is the state in the leaf node. . . . . 142

4.3 Trajectories in episodes 1 (top row) and 2 (middle row and bottom row) of the problem TT3 from Triangle Tireworld.  $s_{i,j}$  and  $a_{i,j}$  denote the state and action executed, respectively, at episode  $i$ , time step  $j$ . Each subfigure illustrates a state that an action is executed in. The next state is illustrated in the next subfigure. Locations are represented by circles. A green circle indicates that the location has a spare. A circle with a border indicates that the vehicle is at that location. A blue border indicates that the vehicle has a spare while a red border indicates otherwise. A dotted border indicates that the vehicle has a flat tire while a solid border indicates otherwise. . . . . 151



4.4 A framework for an ensemble of Q-function approximations which learns at different abstraction levels and from extrinsic ( $r_{ext}$ ) and intrinsic ( $r_{int}$ ) rewards. Multiple types of intrinsic rewards can be considered and are categorised into four types, depending on how they fit into the framework. . . . . 155

4.5 Average variational distance and one standard deviation for each of the ten model learned by RLFIT given the training data of 10, 50, and 500 state transitions per symbolic action. The average variational distance is aggregated over ten independent runs. . . . . 165

4.6 Comparing the number of features added to  $\Phi$  between MFFS and MBFS. The true model is used by MBFS to determine the base features. 166

4.7 Comparing the total undiscounted rewards received in each episode between MFFS and MBFS. The true model is used by MBFS to determine the base features. . . . . 167

4.8 Comparing the total undiscounted rewards received in each episode between the true model (T) and the learned model (L) for MBFS. . . . 168

4.9 Performance of LDE and its variants for the problems TT6 (top) and RI2 (bottom). For LDE-DT-F0 (**transfer**), first-order dead end situations which are learned in the small scale problems, TT3 and RI1, are transferred to the large scale problems, TT6 and RI2. . . . . 169

4.10 Comparing different policies when using intrinsic reward. The intrinsic reward used is TDE. Extrinsic and intrinsic approximations are first-order approximations. . . . . 171

4.11 Comparing different types of intrinsic rewards. Extrinsic and intrinsic approximations are ground approximations. COUNT (Tabular) denotes the visit count intrinsic reward is approximated by a tabular representation while COUNT (LFA) denotes that it is approximated by a ground approximation. . . . . 172

4.12	Comparing different types of intrinsic rewards. Extrinsic and intrinsic approximations are first-order approximations. <code>COUNT</code> (Tabular) denotes the visit count intrinsic reward is approximated by a tabular representation while <code>COUNT</code> (LFA) denotes that it is approximated by a first-order approximation. . . . .	173
4.13	<code>Dyna</code> uses a model to generate imagined observations which an extrinsic ground ( $\tilde{Q}^{gnd}$ ) or first-order approximation ( $\tilde{Q}^{fo}$ ) learns from. The models are either the true models (T) or learned from a set of training data which contains either 50 ( $L \sim 50$ ) or 500 ( $L \sim 500$ ) state transitions per symbolic action. . . . .	175
4.14	Comparing different combinations of intrinsic reward approximated with a first-order approximation. The extrinsic approximation is also a first-order approximation. <code>COUNT</code> denotes the visit count intrinsic approximated with a tabular representation. . . . .	177
4.15	A different problem is attempted in every 300 episodes. For each problem (except for the first), Q-functions are transferred from the previous problem unless stated otherwise with the subscript $\perp$ . . . . .	178
4.16	The total undiscounted rewards received in each episode for <code>THTS</code> , <code>LQ-RRL</code> , and combinations of model-based and model-free RL methods, <code>RRL+Dyna</code> and <code>RRL+UCT</code> . . . . .	180
4.17	The goal-directedness for <code>THTS</code> , <code>LQ-RRL</code> , and combinations of model-based and model-free RL methods, <code>RRL+Dyna</code> and <code>RRL+UCT</code> . . . . .	181
5.1	A PDDL2.1 action model for <code>put_down</code> in <code>Service Robot</code> . . . . .	189
5.2	Simulated environments in Gazebo for the <code>Service Robot</code> domain. On the left is the environment for one person ( <code>SR1</code> ) and on the right is the environment for three people ( <code>SR2</code> and <code>SR3</code> ). . . . .	202

5.3	Snapshots of a robot executing some actions in the Gazebo environment. The blue rays emanating from the robot is its laser scan. (1): The robot localises. (2) and (2a): The robot finds a person using its camera where (2a) is the camera view. (3): The robot moves towards the person. (4): The robot talks to the person and receives some tasks. (5) and (5a): The robot picks up an item. (6) and (6a): The robot gives the item to the person and completes the task. . . . .	203
5.4	Results for learning from scratch and transfer learning between different classes of problems. The source problems are randomised RMDP problems of <b>SR1</b> ( $P_{\emptyset}^{SR1}$ ). The target problems are randomised TDOR-MDP problems of <b>SR1</b> (top row) and <b>SR3</b> (bottom row), and can have dynamic objects ( $P_{DO}$ ), TGs ( $P_T$ ), or both ( $P_{TDO}$ ). . . . .	205
5.5	Performance of different methods in solving TDORMDP problems of <b>SR3</b> . The first-order approximation is learned in <b>SR1</b> and transferred to <b>SR3</b> . If stated in the legend, either the oracle or learned model is used. . . . .	208
5.6	Result for Sim-to-Sim transfer. . . . .	208
5.7	A multi-agent decomposition framework for multi-agent coordination which consists of a multi-agent module (MAM) and single-agent modules (SAMs). The green arrows denote information pertaining to the single-agent problems, the purple arrows denote feedback from the SAMs, the red arrows denote actions executed by the SAMs, and the blue arrows denote observations from the environment. . . . .	211
5.8	The MAM of a multi-agent decomposition framework which decomposes a multi-agent problem to a set of single-agent problems using information from a temporal planner. A purple arrow denotes feedback from a SAM and a green arrow denotes information sent to a SAM. . . . .	214
B.1	DBN representing the transition function for a particular problem of <b>Recon</b> which is described in Example 40. . . . .	248

C.1	Comparing other RRL methods with LQ-RRL on randomised problems of <b>Blocks World</b> which involve ten blocks. “Ground” denotes the ground approximation which serves as a point of reference. . . . .	250
C.2	Sensitivity analysis for the hyperparameter $\xi$ for iFDD+. The total undiscounted rewards received in each episode is shown. . . . .	252
C.3	Sensitivity analysis for the hyperparameter $\xi$ for iFDD+. The total number of features added to $\Phi$ is shown. . . . .	252
C.4	Sensitivity analysis for the hyperparameter $\tau$ for $\tau$ -iFDD+. The total undiscounted rewards received in each episode is shown. . . . .	253
C.5	Sensitivity analysis for the hyperparameter $\tau$ for $\tau$ -iFDD+. The total number of features added to $\Phi$ is shown. . . . .	253
C.6	The number of base features or first-order base features given by MFFS and MBFS for different values of $\nu$ . Legend: <i>Small (Large)</i> stands for a small (large) scale problem of the domain, <i>GND</i> stands for ground approximation, and <i>FO</i> stands for first-order approximation. . . . .	255
C.7	Sensitivity analysis for the coefficient for intrinsic reward, $\beta$ . The intrinsic reward used is TDE. Both the extrinsic and intrinsic approximations are first-order approximations. . . . .	256
C.8	Dyna uses the true models to generate imagined observations which an extrinsic first-order approximation learns from. Different number of rollouts, $N_{sim}$ , are tested with a simulated horizon $H_{sim} = 40$ . For $N_{sim} = 0$ , Dyna is not used. . . . .	257
C.9	Dyna uses the learned models to generate imagined observations which an extrinsic first-order approximation learns from. The models are learned from a set of training data which contains 500 state transitions per symbolic action. Different rollouts $N_{sim}$ are tested with a simulated horizon $H_{sim} = 40$ . $N_{sim} = 0$ implies that Dyna is not used. . . . .	258
C.10	Effect of $\Delta T$ on performance in randomised problems of SR3. All problems have dynamic objects and time-bounded goals except for those with $\Delta T = \infty$ . The values of $\Delta T$ are in seconds. . . . .	259

# List of Algorithms

1	Online RRL with ensemble of Q-function approximations . . . . .	66
2	Online feature discovery (iFDD+) . . . . .	68
3	Adaptive online feature discovery ( $\tau$ -iFDD+) . . . . .	72
4	Update Q-function approximation . . . . .	74
5	Initialise a set of first-order features for a symbolic action . . . . .	78
6	Evaluate a set of first-order features . . . . .	88
7	Apply an ordered set of substitutions . . . . .	88
8	Model-Based Q-value Tree Expansion (MQTE) . . . . .	110
9	Evaluate state node for MQTE . . . . .	111
10	Evaluate value of action node for MQTE . . . . .	111
11	Generalised-knowledge-assisted online RRL . . . . .	130
12	Model-based Feature Selection (MBFS) . . . . .	134
13	Get neighbours of an action . . . . .	135
14	Get neighbours of a state predicate . . . . .	135
15	Find dead end traps . . . . .	152
16	Using Dyna to initialise a Q-function approximation . . . . .	161
17	Reactive plan execution . . . . .	218
18	Multi-agent coordination for weakly coordinated actions . . . . .	219
19	Sequential single-item auctions of TGs . . . . .	220

# Notation

$A$	A set of discrete actions
$\dot{A}$	A set of durative actions
$\bar{A}$	A set of abstract actions
$\mathcal{A}$	A set of symbolic actions
$\dot{\mathcal{A}}$	A set of symbolic durative actions
$A_{greedy}$	A set of actions with maximal Q-values in a particular state
$A_{\hat{a}}$	A set of discrete actions resulting from the grounding of the symbolic action $\hat{a}$
$\mathcal{A}$	Action abstraction function
$\bar{a}$	Abstract action
$a$	Action
$\dot{a}$	Durative action
$\hat{a}$	Symbolic action
$\mathcal{C}$	A set of types
$\mathcal{C}$	Variable or object type
$CK$	Contextual knowledge for grounding first-order features
$D$	Domain
$\mathcal{D}$	Action duration distribution

NOTATION

$e$	Base of the natural logarithm
$e_t(s, a)$	Eligibility trace for a state-action pair $(s, a)$ at time step $t$
$\mathcal{G}$	A set of graphs constructed by MBFS
$\mathcal{G}$	A set of goal predicates
$\dot{\mathcal{G}}$	A set of time-bounded goals
$g$	Goal predicate
$\dot{g}$	Time-bounded goal
$H$	Time horizon
$H_{MQTE}$	Maximum depth MQTE can expand to
$H_{rollout}$	Length of a rollout during simulation phase of UCT
$H_{sim}$	Simulated horizon for Dyna
$\mathcal{M}$	Generative model
$\tilde{\mathcal{M}}$	Approximate model
$\mathcal{M}_{MLM}$	Generative model which uses a maximum likelihood model
$\mathcal{M}_{DBN}$	Generative model which uses parameterised Dynamic Bayesian Networks
$N_{known}$	Threshold for maximum likelihood model to make a prediction
$N_{sim}$	Number of rollouts for Dyna
$N_{\Phi}$	Maximum number of features which can be added per time step by $\tau$ -iFDD+
$\mathcal{O}$	A set of objects each associated with a type
$\mathcal{P}$	A set of state predicates
$\mathcal{P}^+$	A set of positive literals

NOTATION

$\mathbf{P}^\#$	A set of positive and negative literals
$\mathcal{P}$	A set of symbolic state predicates
$\widehat{\mathcal{P}}$	A set of lifted state predicates or first-order base features
$\mathbf{P}_D$	A set of problems consisting of every problem of the domain $D$
$P$	Problem
$p$	State predicate
$\widetilde{\mathcal{Q}}$	Ensemble of approximations
$\widetilde{\mathcal{Q}}_{int}$	Ensemble of intrinsic approximations
$\widetilde{\mathcal{Q}}^{fo}$	First-order approximation
$\widetilde{\mathcal{Q}}^{gnd}$	Ground approximation
$\widetilde{\mathcal{Q}}_{int}$	Intrinsic approximation
$Q^*$	Optimal Q-function
$\widetilde{Q}$	Q-function approximation
$Q(s, a)$	Q-value for an action $a$ in the state $s$
$\widetilde{\mathcal{Q}}(s, a)$	Q-value for an action $a$ in the state $s$ estimated by an ensemble of approximations $\widetilde{\mathcal{Q}}$
$\widetilde{Q}(s, a)$	Q-value for an action $a$ in the state $s$ estimated by a Q-function approximation $\widetilde{Q}$
$\widetilde{Q}^{UCT}(s, a)$	Q-value for an action $a$ in the state $s$ estimated by UCT
$\mathcal{R}$	Parameterised reward function
$\dot{\mathcal{R}}$	Time-dependent parameterised reward function
$\mathcal{R}$	Reward function
$\dot{\mathcal{R}}$	Time-dependent reward function



NOTATION

$\mathbf{r}_{int}$	A set of intrinsic rewards
$r$	Immediate reward
$r_{ext}$	Extrinsic reward
$r_{int}$	Intrinsic reward
$\mathbf{S}$	A set of discrete states
$\bar{\mathbf{S}}$	A set of abstract states
$\dot{\mathbf{S}}$	A set of states each augmented with the elapsed time
$\mathcal{S}$	State abstraction function
$SW$	Episode to switch the Q-function approximation which a policy is generated from
$s$	State
$\bar{s}$	Abstract state
$T$	Continuous time
$T_{dur}$	Action duration
$T^{-}$	Start time of the time bound when a goal can be achieved
$T^{+}$	End time of the time bound when a goal can be achieved
$\mathcal{T}$	Parameterised transition function
$\mathcal{T}$	Transition function
$\dot{\mathcal{T}}$	Time-dependent parameterised transition function
$\dot{\mathcal{T}}$	Time-dependent transition function
$t$	Discrete time step
$V_{sim}$	Total discounted return for a sampled trajectory in UCT
$\mathbf{w}$	A set of weights for an ensemble of approximations

## NOTATION

$\alpha$	Learning rate
$\beta$	Weight of intrinsic reward
$\Gamma$	State trajectory
$\gamma$	Discount factor
$\Delta T$	Amount of time given to achieve time-bounded goals
$\delta$	TD error
$\epsilon$	Probability of the $\epsilon$ -greedy policy selecting a random action
$\eta$	Relevance of a candidate feature
$\boldsymbol{\eta}$	Relevances of candidate features
$\theta$	Weight of a feature
$\boldsymbol{\theta}$	Weight vector
$\Lambda_{leaf}$	Mixing parameter for the leaf node of the UCT search tree
$\Lambda_{root}$	Mixing parameter for the root node of the UCT search tree
$\lambda$	Hyperparameter for TD( $\lambda$ ) methods
$\nu$	Depth of connections for MBFS
$\mathbb{E}$	Experience buffer
$\Xi$	Observation of the form $(s, a, r, s')$
$\boldsymbol{\xi}$	A set of discovery thresholds for each action, used by iFDD+ and $\tau$ -iFDD+
$\xi$	Discovery threshold for iFDD+ and $\tau$ -iFDD+
$\pi$	Policy
$\pi^*$	Optimal policy
$\pi_{sum}$	Policy which considers the aggregation of the Q-values from each Q-function approximation in an ensemble

## NOTATION

$\pi_{switch}$	Policy which considers the Q-values from a different Q-function approximation in an ensemble depending on a condition
$\pi(a s)$	Probability of the policy $\pi$ to select an action $a$ in the state $s$
$\sigma$	A set of substitutions
$\bar{\sigma}$	An ordered set of substitutions
$\sigma_\phi$	A set of possible substitutions for the first-order feature $\phi$
$\sigma$	Substitution
$\sigma_{goal}$	Substitution due to goal context
$\sigma_{ground}$	Substitution due to ground context
$\sigma_{location}$	Substitution due to location context
$\tau$	Maximum number of features in terms of percentage which can be added per time step by $\tau$ -iFDD+
$\tau_{temp}$	Temperature for the softmax policy
$\Phi$	A set of features
$\Phi_a$	A set of features for the action $a$
$\Phi^c$	A set of candidate features
$\Phi_{active}$	A set of active features in $\Phi$
$\Phi(s, a)$	Feature vector
$\phi$	Feature
$\phi(s, a)$	Value of a feature $\phi$ for the state-action pair $(s, a)$
$\chi$	Failure buffer
$\chi$	Dead end situation or first-order dead end situation
$\emptyset$	Empty set

*NOTATION*

$\star\mathcal{C}$  Free variable of type  $\mathcal{C}$

$\infty$  Infinity

$\wp$  Power set

# Chapter 1

## Introduction

The ability to perceive, deliberate, and act is a form of intelligence necessary to solve sequential decision-making problems. Some examples of such problems are:

- A person plans for a holiday and has some preferences on destinations but also wants to keep within budget.
- In logistics applications, resources must be deployed efficiently to deliver goods before their deadline.
- Robots must avoid hazards while optimising their actions to achieve some tasks in the face of uncertainty.

A common theme lies across decision-making problems: an agent, be it a human, a robot, or a computer software, interacts with its environment by executing some actions, changing the state of the environment, and in return receives some rewards as feedback on its actions. A state describes the situation the agent is in and contains information required for decision making. While the ability to make sound decisions comes naturally to humans, it has to be (artificially) replicated in machines.

Decision-theoretic planning is the computational process of reasoning and deliberation over what sequence of actions to execute to maximise a utility such as the total rewards received. We are interested in decision making for problems at the task level where actions are high-level (e.g., move to that door) rather than low-level (e.g., actuate the left motor by 15 degrees). Domain-independent planning or automated planning [58] solves decision-making problems across a diversity of domains and does not require any domain-specific knowledge. A planner uses a model of the interaction between an agent and its environment to predict the consequences or

outcomes of executing an action. The planner searches for a plan, a sequence of actions, which yields the highest expected utility. Different planners address different types of sequential decision-making problems. For example, temporal planners such as [12, 28] deal with temporal constraints while probabilistic planners such as [87, 93] deal with uncertainty in dynamics. We construct intelligent machines by incorporating planners in their software, allowing them to make decisions autonomously based on their perceptions of the states they are in.

If the true model is not known, planners cannot be used. Reinforcement learning (RL) [168] is an alternative approach to solve decision-making problems. RL is one of three main areas of machine learning. The other two areas, supervised learning and unsupervised learning, differ from RL in the types of problems they are applied to; supervised learning deals with classification and regression while unsupervised learning deals with pattern recognition. The objective of RL is to learn a policy which tells the agent what to do in each state such that the expected utility is maximised. Offline RL improves its policy given a set of observations which it assumes is available beforehand. Thus, agents typically do not need to interact with the environment to learn. In the absence of prior observations, offline RL cannot be used as it does not deal with data (i.e., observations) collection. This is in contrast with online RL: an agent acts in its environment and learns from the resulting observations to improve its policy over time. Online RL learns by trial-and-error, going through an iterative learning process where it incrementally improves its policy which in turn gives more meaningful observations that allows the agent to improve the policy further.

This dissertation introduces a new online RL algorithm which exploits the structure of a problem to solve the problem more efficiently by learning and utilising different forms of relational knowledge. In the remainder of this chapter, we provide an intuition for why this is possible, describe motivating examples for more complex types of problems, and present an overview of our methods and contributions.

## 1.1 Exploiting The Structure of a Problem

A state variable describes one aspect of the state such as a fact, a property of an object, or a relation between objects. For example, one state variable describes the location of a robot while another describes the destination a package needs to be delivered to. In large scale problems where there are many objects, each state is described by a multitude of state variables. This results in an exponential growth of the number of states which is known as the curse of dimensionality [9]. This poses a major issue to RL. Since actions can lead to different outcomes, the agent should attempt every action multiple times in each state in order to learn the optimal policy. This is the exploration phase where the agent is more concerned with learning about its environment than to maximise its utility. Intuitively, how would an agent know how to act in a state it has never seen before? This naive approach has several drawbacks. First, it is exceedingly rare for an agent to reach desirable states through random exploration in large scale problems where the number of states can be intractable. The sample complexity [81], or the amount of observations required to achieve near-optimal behaviour, is prohibitively high in such problems. Second, exploration can also be expensive and impractical. Consider a robot operating in a crowded office space. Not only does each action takes minutes or even longer to execute, it is unsafe or infeasible for the robot to explore liberally. Lastly, it is not always possible for an agent to revisit a particular state over and over again to try a different action.

The above naive approach considers an exact representation of the problem. That is, each state is treated as a unique state and every observation only applies for the state in which it was obtained. If the problem is structured such that acting similarly in some states produces similar outcomes, then generalisation of the observations is possible and can be useful in reducing the sample complexity. For example, a robot picks up an apple from a table. It is now holding onto the apple and the apple is no longer on the table. There is a ball on another table. A similar outcome is observed when the robot picks up the ball. If we abstract away the identities of the objects involved, that is the apple, the ball, and the tables, then the effects of the two actions are identical: when the robot picks up an item from a table, the item is no longer on the table and is in the robot's gripper. Also, the states

in which this action can produce the observed effects are states where the robot is by a table where an item is on. This perspective of looking at states and actions with no regards to objects but rather variables leads to a relational representation of the problem. For problems with a first-order or relational structure (i.e., some actions change the relations or properties of all objects of the same type in a similar manner), relational representations can be used instead of exact representations for efficient learning.

A relational reinforcement learning (RRL) method [47] is an RL method which learns in a relational representation of the problem. RRL methods reduce the sample complexity as they learn in a smaller abstract representation of the problem rather than an exact representation. To see why this makes learning more efficient, we consider the same example as before. Suppose that the goal is for a robot to bring an apple to a specific table and the robot has learned how to do so. Now, a ball is introduced to the environment and the goal is to bring it to another table. The robot has never attempted this before. In an exact representation, the second goal is a new goal which the robot has to learn to achieve. In a relational representation, both goals are the same since the identities of objects does not matter. Thus, the robot knows how to achieve the second goal without any further learning despite seeing a ball for the first time. This is due to the generalisation property of RRL. By aggregating multiple different but similar scenarios as one abstract scenario, learning is made more efficient. The agent no longer needs to revisit the same state over and over again as there are other states similar to it. Generalised knowledge learned from an observation in a particular state can be applied to other similar states for more informed decision making.

## 1.2 Transferring Knowledge

Transfer learning [174, 176, 192] is another form of intelligence where we retrieve relevant knowledge, possibly adapt it, and apply it to perform a new task. Instead of learning from scratch, we are able to solve new problems by leveraging on previously learned knowledge. RRL enables transfer learning if the representations of different problems of the same domain are abstracted to the same relational representation.



From the perspective of the agent, these problems are one single problem. Now, learning in one problem helps the agent to solve another problem quickly. This opens up the possibility of curriculum learning [10] where the agent learns in gradually more complex problems. To see why this can be beneficial, recall the curse of dimensionality for large scale problems. Since online RRL incrementally builds up the knowledge of an agent based on current knowledge, without any prior knowledge, the agent executes some random actions until something meaningful happens. This almost never happens in large scale problems. In curriculum learning, the agent starts by learning in small scale problems before using the acquired knowledge to solve large scale problems. Another way to look at curriculum learning is to train an agent in simulated environments before deploying them in the real world. Besides accelerated learning in the real world, this also mitigates the risk of damages due to sub-optimal behaviours and reduces the amount of data collection required in the real world which can be expensive.

### 1.3 Other Aspects of a Problem

There are many types of complexities in decision-making problems. We describe some of them which are relevant to this dissertation.

- The state could change as a result of an external factor rather than an agent's action. Such **exogenous events** cannot be controlled by the agent. Instead, the agent has to learn to adapt and react to them.
- There are problems where the agent could get stuck in a **dead end**. For example, a mobile robot runs out of energy and can no longer operate. Decision making now needs to deal with avoiding dead ends which can have dire consequences.
- It is plausible that some environments have many objects (e.g., a library with books and patrons). It is unlikely that the agent has full knowledge of every object. Also, it is often impractical to consider all of them for the purpose of decision making. The agent is now placed in a situation where the existence of some objects are initially unknown.
- The notion of time is inherent in some problems such as robots operating in

the real world. For example, a robot takes time to execute an action or a goal needs to be achieved by a deadline. The state an agent is in has to consider the **passage of time**. While an agent might be able to revert any changes made to its environment, it cannot travel back in time.

- Some problems have tasks that cannot be achieved by one agent. Coordination between **multiple agents** are required to achieve these tasks. Decision making is complicated by the fact that agents must consider the actions of other agents in addition to their own actions.

## 1.4 Overview of Our Approach

In this dissertation, we propose an online RRL method to solve decision-making problems under uncertainty without knowledge of the true model. Our motivation is as described in preceding sections. Our method is domain-independent, avoiding any domain-specific or expert knowledge, and learns and utilises multiple types of generalised knowledge which can be transferred to any problem of the same domain. This generalised knowledge allows us to incorporate planning-type techniques into our RRL method, provide efficient, guided exploration in place of random exploration, and avoid dead ends [118, 123]. We also investigate the limitations of RRL in solving relational problems with certain properties [120–122]. We consider transfer learning between not just different problems but also different classes of problems and simulated environments . To this end, we introduce a new class of problems which considers additional complexities, examine the impact they have on RRL, and propose solutions to solve such problems. Our software is released publicly at <https://github.com/njunhao/GKRRL>.

## 1.5 Objectives and Contributions

RRL can be made more efficient by learning and utilising various types of generalised knowledge to reason with different aspects of the decision-making problem. This also overcomes the limitations of RRL and extends the types of problems it can solve. The generalised knowledge can be transferred to any problem of the same domain

to guide RRL and improve its performance.

We enumerate each objective of this dissertation which is accompanied by the contributions made to achieve it:

- Many of the prior RRL work use relational decision trees to approximate Q-functions which are ill-suited for online RL. While remedies have been proposed, the inherent issue remains. We propose an online RRL method which uses first-order features, thereby sidestepping the issue plaguing relational decision trees. To the best of our knowledge, our work is the first of its kind as similar work use supervised learning. We also incorporate several RL techniques to improve the learning efficiency and stability.
- While the increased abstraction due to RRL gives generalisation, it has drawbacks which limits the applicability of RRL. We examine the causes of the limited representational capacity associated with RRL and its ramifications on performance. We propose three solutions. First, first-order features are grounded with contextual knowledge [122]. Second, an ensemble of Q-function approximations with different abstraction levels is used for decision making [120]. Lastly, a model-based method performs multi-steps lookahead to make a more informed deliberation on action choices [121].
- RL and RRL methods typically take a long time to converge and achieve (near-)optimal performance. This makes them impractical or prohibitively expensive for some applications. We improve the learning efficiency in four ways. First, relational models are learned. We adapt existing model-based methods to either initialise the Q-function approximation or perform multi-steps lookahead for better decision making. Second, agents perform guided exploration by learning from multiple types of intrinsic rewards. Third, observed dead ends are represented in a first-order representation which generalises to unseen situations, allowing agents to avoid similar dead ends in the future [118, 123]. Lastly, the aforementioned knowledge used by the three methods are generalised which can be transferred to another problem to accelerate learning.
- Model-based RL methods have better initial performance than model-free RL methods while the latter have better asymptotic performance. We combine both approaches to obtain both of their strengths. While this approach is

not new, it has not been done before in the context of RRL. Since learning the true models can be more difficult than solving the problem itself, the learned models are (possibly poor) approximations of the true models. We evaluate the efficacy of our approach using models which are learned offline from observations.

- Prior RRL methods are tested in a few types of problems which are also rather simple. We propose three new domains for robotic applications. Extensive empirical studies are conducted on these domains in addition to three benchmark domains.
- We want to extend RRL methods to solve more complex types of problems which are more representative of real world applications. We introduce a new class of problems which involves dynamic objects and time-bounded goals. We examine its complications on RRL methods and propose solutions to solve this class of problems.
- RL agents often perform well in the environments they are trained in but perform poorly in a test environment due to the discrepancies between the two environments. To instil confidence in RL methods for real world applications, it is crucial to show that they perform well in test environments as well. For this purpose, we build a simulated environment in Gazebo [89] for one of our robotic domains. We demonstrate transfer learning between different simulated environments and classes of problems for our RRL method.
- Our proposed class of problems introduces time-bounded goals which can naturally extend to address temporally-coordinated actions between multiple agents. We propose a multi-agent decomposition framework which uses either a temporal planner or an auction algorithm for multi-agent coordination to achieve joint goals [24].

## 1.6 Structure

The remainder of this dissertation consists of six chapters. Chapter 2 contains descriptions of three new domains while Chapters 3, 4, and 5 contain original research. Parts of the work in this dissertation have been published before in [24, 117–123].

The six chapters are structured as follows:

- Chapter 2 (Background and Related Work): This chapter provides background information for all chapters and reviews state-of-the-art work in related fields.
- Chapter 3 (First-Order Approximation): This chapter introduces our RRL method and investigates the conditions necessary for generalisation. We examine the pros and cons of learning in abstract spaces and propose solutions to mitigate the drawbacks.
- Chapter 4 (Generalised Knowledge for RRL): This chapter explores different types of generalised knowledge and methods to learn and utilise them for increased learning efficiency. These methods do not require any expert knowledge, relying on the same feedback signal from the previous chapter to extract additional valuable information.
- Chapter 5 (Dynamic Objects, Time, and Coordination): This chapter presents a new class of problems which considers dynamic objects and time-bounded goals. It builds on and extends the work from the previous chapters to solve this new class of problems. Empirical experiments demonstrate transfer learning across different classes of problems and across different simulated environments. The chapter concludes with a framework which extends our work for multi-agent coordination.
- Chapter 6 (Discussion): In this chapter, we reflect on our work and discuss some lessons learned.
- Chapter 7 (Conclusions and Future Work): This chapter summarises and concludes our work and suggests some possible directions for future work.

# Chapter 2

## Background and Related Work

This chapter provides some background for the subsequent chapters. First, we describe various aspects of knowledge representation: problem representations, planning languages, and generative models. A problem representation, which can be written in a planning language, is given as an input to a planning or RL method which then solves the problem. A model is one part of the problem representation and is required by model-based methods. Second, we introduce RL, function approximations for RL, and RRL. RL is the field of study for solving decision-making problems and RL methods operate over problem representations. Third, we discuss state-of-the-art work and compare them with ours. Fourth, we describe six benchmark domains, three of which are newly introduced domains, that will be used in our empirical studies. These domains and their problems are written in a planning language. Lastly, the chapter concludes with a summary of how these topics fit in with our work.

### 2.1 Problem Representations

Suppose that a vehicle needs to move from an initial location to a goal location. A reward is given for reaching the goal location. Along the way, the vehicle might get a flat tire and cannot move until the tire is replaced. Actions affect the state of the environment, and consequently the actions which should be executed next. This is a sequential decision-making problem [58] where an agent has to reason over the outcomes due to its actions and deliberate over which action to execute in each time

step. The above example is a problem for the **Triangle Tireworld** domain [103] which shall be described formally in Section 2.8.1 after the prerequisite background is introduced. We shall use this problem throughout the chapter as examples.

There are two fundamental problems in sequential decision making. The first is a **learning problem** where the environment, or rather a model of the environment, is initially unknown. The agent has to interact with the environment and learn from observations in order to improve its decision making. The second is a **planning problem** where a model of the environment is known. The agent computes with this model and deliberates over the outcomes due to its actions to make a decision. In this dissertation, we are interested in the learning problem.

In the remainder of this section, we describe various types of representations for decision-making problems. A problem representation is given as an input to a planning or RL method which operates over it to solve the problem.

### 2.1.1 States and Actions

Before looking at problem representations, we describe states and actions which are ubiquitous in decision-making problems. A state is a conjunction of **literals** which are positive or negative state predicates.

#### Definition 1 (State Predicates)

*A state predicate  $\mathbf{p}(x_1, \dots, x_n)$  consists of a predicate symbol  $\mathbf{p}$  and possibly some terms  $x_1, \dots, x_n$ . The arity of a state predicate is the number of terms it has. Each term is associated with a type and holds an object of the same type. A state predicate is a boolean state variable which represents a fact, a property of an object, or a relation between two or more objects. A symbolic state predicate has variables in its terms instead of objects. It can be ground to a state predicate by substituting variables with objects.*

Given a set of state predicates  $\mathbf{P}$ , a state  $s$  is represented as  $s = \bigwedge_{i=1}^{|\mathbf{P}|} p_i$  where  $p_i$  is the  $i$ -th literal.<sup>1</sup> Assuming that all possible combinations of state predicates or their negation are valid states, then the size of the set of states  $\mathbf{S}$  is  $|\mathbf{S}| = 2^{|\mathbf{P}|}$ . Similar to state predicates, an action  $\mathbf{a}(y_1, \dots, y_n)$  consists of a symbol  $\mathbf{a}$  and possibly

---

<sup>1</sup>A symbol with boldface denotes a set and  $|\cdot|$  denotes the cardinality of a set.

some terms  $y_1, \dots, y_n$  which are objects while a symbolic action has variables as terms.

### Example 1 (State Predicates and Actions)

In *Triangle Tireworld*, a symbolic state predicate  $\text{vehicle\_at}(WP)$  represents the fact that the vehicle is at a location  $WP$  where  $\text{vehicle\_at}$  is a symbol and  $WP$  is a variable of type  $wp$ .<sup>2</sup> The state predicate  $\text{vehicle\_at}(la1a1)$  is obtained by substituting  $WP$  with  $la1a1$ , an object of type  $wp$ . Suppose that in a state  $s_0$ , the vehicle is at the location  $la1a1$ . Then,  $s_0 = \text{vehicle\_at}(la1a1) \wedge \neg \text{vehicle\_at}(la1a2) \wedge \dots$ . The symbolic action  $\text{move\_vehicle}(WP_1, WP_2)$  moves the vehicle from  $WP_1$  to  $WP_2$ . Likewise,  $WP_1$  and  $WP_2$  are variables of type  $wp$ . By executing the action  $\text{move\_vehicle}(la1a1, la1a2)$  in  $s_0$ , the vehicle moves from  $la1a1$  to  $la1a2$  and the next state is  $s_1 = \neg \text{vehicle\_at}(la1a1) \wedge \text{vehicle\_at}(la1a2) \wedge \dots$ .

## 2.1.2 Markov Decision Processes

Markov decision processes (MDPs) model fully-observable environments for sequential decision making under uncertainty.

### Definition 2 (Markov Decision Processes)

A finite horizon MDP is a tuple  $(\mathbf{S}, \mathbf{A}, \mathcal{T}, \mathcal{R}, s_0, H, \gamma)$  where  $\mathbf{S}$  is a set of discrete states,  $\mathbf{A}$  is a set of discrete actions,  $\mathcal{T} : \mathbf{S} \times \mathbf{A} \times \mathbf{S} \rightarrow [0, 1]$  is the transition function which defines a probability distribution over possible next states after executing an action,  $\mathcal{R} : \mathbf{S} \times \mathbf{A} \rightarrow \mathbb{R}$  is the reward function which specifies rewards for executing actions in states,  $s_0$  is the initial state,  $H$  is the time horizon or the maximum number of time steps, and  $\gamma \in (0, 1]$  is the discount factor.

An MDP is a propositional representation as states can be represented as vectors of attribute-value pairs where the attribute is a state predicate and its value is binary (i.e., true or false). A state holds all information necessary for the transition function to determine the next state following the execution of an action in a state. This is the Markov property and the state is a Markov state. As a consequence of the Markov property, only the current state rather than the history of states is considered to select an action. Many RL methods assume Markov states [168]. We discuss RL in

---

<sup>2</sup>Uppercase letters denote variables and lowercase letters denote objects and types.



## Section 2.4.

Our work addresses episodic learning problems. In an **episode** of a finite horizon MDP, the agent interacts with the environment starting from the initial state  $s_0$  for  $H$  time steps. We assume that the agent can start from the same initial state for multiple consecutive episodes. For a learning problem, more than one episode is often required to improve the decision-making capability of an agent.

**Example 2 (A Learning Problem for Triangle Tireworld)**

*In **Triangle Tireworld**, the goal is to reach the goal location upon which a positive immediate reward is given. For simplicity, we assume that the immediate reward is zero otherwise. Following Example 1, the agent starts from the initial state  $s_0 \in \mathbf{S}$  and executes an action  $a_0 = \text{move\_vehicle}(la1a1, la1a2)$  at time step  $t = 0$ . The agent receives an immediate reward  $r_0 = 0$  for this and is now in the next state  $s_1$  at  $t = 1$ .<sup>3</sup> Due to this sparse reward signal and no knowledge of the model, the agent has no information on how to achieve the goal and resorts to executing random actions, or random exploration, until the goal is achieved. Unless the problem is trivially simple, it is highly unlikely to achieve the goal through random exploration in one episode or even in many episodes.*

An **observation** is a tuple  $(s_t, a_t, r_t, s_{t+1})$  where  $s_t$  is the state the agent was in at time step  $t$ ,  $a_t$  is the action executed at  $t$ ,  $r_t$  is the immediate reward received at  $t$ , and  $s_{t+1}$  is the next state the agent is in at  $t+1$ . In Example 2, the agent acquires an observation  $(s_0, \text{move\_vehicle}(la1a1, la1a2), 0, s_1)$  at time step  $t = 0$ . Typically, an agent learns from observations over time to solve a learning problem. We discuss this in details in Section 2.4.

**2.1.3 Relational Markov Decision Processes**

The transition and reward functions can be represented by tables or tabular forms.  $\mathcal{T}$  is a  $|\mathbf{S}| \times |\mathbf{A}| \times |\mathbf{S}|$  matrix where an element stores  $\mathcal{T}(s'|s, a)$ , the probability of observing  $s'$  after executing  $a$  in  $s$ . Similarly,  $\mathcal{R}$  is a  $|\mathbf{S}| \times |\mathbf{A}|$  matrix which stores the reward  $\mathcal{R}(s, a)$  for every state-action pair. This exact or flat representation makes no assumptions about the structures of  $\mathcal{T}$  and  $\mathcal{R}$ . For large state-action spaces,

---

<sup>3</sup>We use the subscript to denote the time step associated with an entity. For example,  $s_i$  and  $r_i$  are the state and immediate reward at time step  $t = i$ , respectively.

tabular forms are impractical as the number of elements scales exponentially with the number of state predicates. That is, tabular forms incur a high space complexity. We discuss space complexity in Section 2.4.3.

Factored MDPs [15] are one approach to represent large scale MDPs compactly if the models (i.e.,  $\mathcal{T}$  and  $\mathcal{R}$ ) are structured. First,  $\mathcal{T}$  is a factored transition function if the transition of each state predicate is determined independently of each other, conditioned on the state and action:

$$\mathcal{T}(s'|s, a) = \prod_i \mathcal{T}(p'_i|s, a), \quad (2.1)$$

where  $\mathcal{T}(p'_i|s, a)$  is a discrete probability distribution for a state predicate  $p_i$  and  $p'_i$  is the value of  $p_i$  in  $s'$  (i.e., at the next time step). If  $\mathcal{T}(p'_i|s, a)$  depends only on a small number of state predicates  $\mathbf{P}^- \subset \mathbf{P}$ , then  $\mathcal{T}$  is expressed as:

$$\mathcal{T}(s'|s, a) = \prod_i \mathcal{T}(p'_i|\mathbf{P}^-, a). \quad (2.2)$$

In this case, Dynamic Bayesian Network (DBN) is one type of representations for  $\mathcal{T}$ . DBNs shall be discussed in Section 2.3.2. Second,  $\mathcal{R}$  is a factored reward function if it can be decomposed as a sum of localised reward functions, each of which depends on an action and a subset of  $\mathbf{P}$ . A factored MDP is an MDP but with factored transition and reward functions.

Some problems are relational in nature where actions change the relations between an arbitrary number of interacting objects. The interaction between an agent and its environment is governed by the properties of objects and relations between objects. Following Example 1, the symbolic action `move_vehicle( $WP_1, WP_2$ )` moves the vehicle from  $WP_1$  to  $WP_2$ . The same outcome applies regardless of what objects the variables are substituted with—the vehicle is now at  $WP_2$  instead of  $WP_1$ .<sup>4</sup> The repeated structure in the model can be represented compactly rather than enumerated explicitly in propositional representations such as MDPs and factored MDPs. We refer to problems with a relational structure as **relational problems**. A relational Markov decision process (RMDP) is a first-order representation of a factored MDP which generalises over objects through the use of variables and represents

---

<sup>4</sup>For simplicity but without loss of generality, we ignore preconditions and probabilistic effects.

relational problems. We use the formalism of RMDPs from [53, 109].

### Definition 3 (Relational Markov Decision Processes)

An RMDP is a tuple  $(\mathbf{C}, \mathbf{P}, \mathbf{A}, \mathbf{O}, \mathcal{T}, \mathcal{R}, s_0, H, \gamma)$  where  $\mathbf{C}$  is a set of classes or types,  $\mathbf{P}$  is a set of symbolic state predicates,  $\mathbf{A}$  is a set of symbolic actions,  $\mathbf{O}$  is a set of objects and each object is associated with a type  $C \in \mathbf{C}$ ,  $\mathcal{T}$  is the parameterised transition function,  $\mathcal{R}$  is the parameterised reward function,  $s_0$  is the initial state,  $H$  is the time horizon, and  $\gamma$  is the discount factor.

An MDP can be constructed from an RMDP where  $\mathbf{P}$  is the grounding of  $\mathbf{P}$  with  $\mathbf{O}$ ,  $\mathbf{A}$  is the grounding of  $\mathbf{A}$  with  $\mathbf{O}$ , and  $\mathcal{T}$  and  $\mathcal{R}$  are the grounding of  $\mathcal{T}$  and  $\mathcal{R}$ , respectively.  $\wp$  denotes the power set. Compact representations such as RMDPs play a key role in solving learning problems efficiently. In Example 2, the agent explores randomly until the goal is achieved which is highly inefficient. In Section 2.6, we introduce RL methods which utilise compact representations to learn more efficiently.

#### 2.1.4 Semi-Markov Decision Processes

A semi-Markov decision process (SMDP) [139] is an extension of an MDP where actions take some time to execute.

### Definition 4 (Semi-Markov Decision Processes)

A SMDP is a tuple  $(\mathbf{S}, \mathbf{A}, \mathcal{T}, \mathcal{R}, \mathcal{D}, s_0, H, \gamma)$  where the elements are the same as those in an MDP (see Definition 2) except for the action duration distribution  $\mathcal{D} : \mathbf{S} \times \mathbf{A} \times \mathbf{S} \rightarrow \mathbb{R}$ . The stochastic action duration  $T_{dur}$  of an action  $a$  starting in a state  $s$  and finishing in  $s'$  is given by  $\mathcal{D} : T_{dur} \sim \mathcal{D}(s, a, s')$ .

#### 2.1.5 Other Extensions of MDPs

There are numerous extensions of MDPs proposed such as first-order MDPs [16], dynamic object RMDPs [109], and object-oriented MDPs [38]. In Section 2.7.1, we discuss some of them that are related to our work.

### 2.1.6 Terminal States, Goals, and Dead Ends

An episode terminates when there is no remaining time horizon or when a terminal state is reached.

#### **Definition 5 (Terminal States, Goals, and Dead Ends)**

*A terminal state is an absorbing state in which executing any action will only lead back to itself; the immediate reward for executing any action in a terminal state is zero. A problem has at least one goal. A goal state is a terminal state where all goals are achieved. A terminal state where at least one goal is not achieved is a dead end.*

Our definition of dead ends is different from [102] which defines dead ends as states where the goal state is unreachable. We consider only terminal states as dead ends (i.e., at least one goal is not achieved and executing any action does not change the state). If a goal can no longer be achieved, then the goal state is unreachable. In [102], this is a dead end. However, since some problems have more than one goal, there remains incentive (i.e., rewards) to attempt to achieve the remaining goals. Thus, we do not consider states where the goal state is unreachable as dead ends. In other words, we are interested in a different type of problems from [102] where agents are allowed to attempt to achieve any remaining goals even when the goal state is not reachable.

## 2.2 Planning Languages

Planning languages are human-interpretable languages used to write domains and problems. A set of problems can be specified for each domain; these problems have some common characteristics associated with the domain such as the same sets of symbolic state predicates and actions. We refer to these problems as problems of the domain. A domain file and a problem file are given to an algorithm which parses them into appropriate data structures. For example, a planning language can be used to describe an MDP or RMDP without any programming required.

```

(:action move_vehicle
:parameters (?from - wp ?to - wp)
:precondition (and (vehicle_at ?from) (ROAD ?from ?to) (not flattire))
:effect (and (vehicle_at ?to)
             (not (vehicle_at ?from))
             (probabilistic 0.25 (not (not flattire))))
)

```

Figure 2.1: The PPDDL action model for `move_vehicle` in `Triangle Tireworld`.

## 2.2.1 STRIPS Action Models

Action models based on the Stanford Research Institute Problem Solver (STRIPS) [50] are defined by their preconditions and effects which are typically conjunctions of literals. An action is applicable in a state if its preconditions are true in that state. Executing an applicable action changes the state according to its effects. The effects are specified by the add list and the delete list. Literals in the add list are made true in the next state and those in the delete list are made false. Action models can be translated to transition functions [195], both of them predict the next state for executing an action in a state.

## 2.2.2 Probabilistic Planning Domain Definition Language

The Planning Domain Definition Language (PDDL) [111] is a language commonly used to write STRIPS domains and problems. Probabilistic PDDL (PPDDL) is a syntactic extension of PDDL2.1 [52] used to describe probabilistic problems [195]. PPDDL supports actions with probabilistic effects. An example of a PPDDL action model in `Triangle Tireworld` [103] is shown in Figure 2.1. The symbolic action `move_vehicle` has three effects, two of which are certain and the other is probabilistic which occurs with a probability of 0.25. Effects in PPDDL are not mutually exclusive.

The PPDDL domain file specifies symbolic state predicates and symbolic actions. The PPDDL problem file specifies the objects (or constants), initial state, and **goal predicates** which are state predicates that must be satisfied in a state for it to be considered a goal state. The domain and problem files share the same state and action language and together, they specify a problem. Different problems of

```

domain triangle_tireworld_mdp {
  types { wp : object; };
  pvariables {
    ROAD(wp, wp) : { non-fluent, bool, default = false };
    vehicle_at(wp) : { state-fluent, bool, default = false };
    :
  };
  cpfs {
    vehicle_at'(?!l) =
      if (exists_{?from : wp} (move_vehicle(?from, ?l) ∧ vehicle_at(?from) ∧
                               road(?from, ?l) ∧ not_flattire))
      then true
      else if (exists_{?to : wp} (move_vehicle(?l, ?to) ∧ vehicle_at(?l) ∧
                               road(?l, ?to) ∧ not_flattire))
      then false
      else vehicle_at(?l);
    :
  };
  action-preconditions {
    forall_{?from: wp, ?to: wp} [move_vehicle(?from, ?to) => (vehicle_at(?from) ∧
                                                                road(?from, ?to) ∧ not_flattire)];
    :
  };
  reward = if (∼ goal_reward_received ∧
               exists_{?!l : wp} (vehicle_at(?l) ∧ GOAL_LOCATION(?l)))
  then 100
  else if (goal_reward_received) then 0
  else -1;
};

```

Figure 2.2: An excerpt of the domain file for Triangle Tireworld written in RDDDL. `:` denotes omitted parts of the domain file.

a domain can be constructed by changing the problem file (e.g., a different initial state, objects, or goals).

### 2.2.3 Relational Dynamic Influence Diagram Language

The Relational Dynamic Influence Diagram Language (RDDDL) [149] (pronounced “riddle”) is a planning language used in the last three International Probabilistic Planning Competitions (IPPCs) in 2011, [148], 2014 [62], and 2018. Semantically, RDDDL describes parameterised DBNs extended with an influence diagram. The domain file specifies object types, parameterised non-fluents, parameterised fluents,

conditional probability functions (CPFs), and a reward function. Fluents are state predicates with values that change with time while the values of non-fluents do not change. We refer to non-fluents, fluents, and state predicates interchangeably unless necessary to differentiate between them.

**Definition 6 (RDDDL Domain)**

A RDDDL domain  $D$  is the tuple  $(\mathcal{C}, \mathcal{P}, \mathcal{A}, \mathcal{T}, \mathcal{R})$  whose elements are defined in Definition 3.

The problem file defines the tuple  $(\mathbf{O}, s_0, H)$  where the initial state  $s_0$  contains the values of fluents and non-fluents. Together with  $D$ , they specify an RMDP for a problem. Multiple problems of  $D$  can be instantiated by varying  $\mathbf{O}$  and  $s_0$ .

Figure 2.2 shows an excerpt of the RDDDL domain file for `Triangle Tireworld` which specifies an object type (`wp`), a parameterised non-fluent (`ROAD(wp, wp)`), a parameterised fluent (`vehicle_at(wp)`), the CPF for `vehicle_at(wp)`, the parameterised preconditions for the action `move_vehicle(?from, ?to)`, and the parameterised reward function  $\mathcal{R}$ .<sup>5</sup>

**RDDL vs PPDDL.** RDDL is intended to model a class of problems that is difficult to model with PPDDL [149]. The key differences between RDDL and PPDDL are:

- RDDL models relations between fluents and can include **exogenous effects** (i.e., changes to the state not due to any action) while PPDDL models actions explicitly.
- RDDL models parallel effects while PPDDL models transitions with correlated effects.
- RDDL rewards are additive and sum over objects while PPDDL rewards are associated with state transitions and do not sum over objects.

## 2.3 Generative Models

A generative model predicts the next state and immediate reward for executing an action in a state. It consists of the transition function  $\mathcal{T}$  and the reward function  $\mathcal{R}$ .

---

<sup>5</sup>Here, we use the RDDDL syntax for fluents, non-fluents, and actions to match Figure 2.2.

We discuss model-based RL methods in Section 2.4.2 which solve learning problems using generative models that are approximations of the true models. Two types of generative models relevant to our work are discussed here.

### 2.3.1 Maximum Likelihood Model

The maximum likelihood model is a method for estimating the true model given some observations. We denote the maximum likelihood model by  $\mathcal{M}_{MLM}$ . It predicts the probability of a state transition as:

$$\widetilde{\mathcal{F}}(s'|s, a) = \frac{N(s, a, s')}{\sum_{s'} N(s, a, s')}, \quad (2.3)$$

where  $N(s, a, s')$  is the number of times  $s'$  was observed after executing the action  $a$  in the state  $s$ . In other words,  $\widetilde{\mathcal{F}}(s'|s, a)$  is the empirical frequency of observing  $s'$  after executing  $a$  in state  $s$ . Likewise, the predicted immediate reward is the empirical mean of the immediate rewards observed for executing  $a$  in  $s$ . If  $a$  has never been executed in  $s$ , then a prediction is not possible. For deterministic actions (i.e., actions with only one outcome), only one observation per state-action pair is required to predict correctly all the time. If the action is probabilistic, then more observations are required.

### 2.3.2 Dynamic Bayesian Network

A factored transition function can be represented by a Dynamic Bayesian Network (DBN) [35] which is a two-layer directed acyclic graph with nodes denoting actions and values of state predicates over consecutive time steps  $t$  and  $t + 1$ . An example of a DBN is illustrated in Figure 2.3. DBNs represent the conditional probability distribution of state predicates as defined by Equation 2.2 where the transition of a state predicate  $p$  depends on some state predicates  $\mathbf{P}^-$  and an action. Directed edges join  $\mathbf{P}^-$  and the action at time step  $t$  to  $p$  at  $t + 1$ .

A parameterised DBN is a DBN with nodes representing symbolic state predicates and actions. It models the transitions of symbolic state predicates and can be applied to multiple propositional groundings by substituting the variables with objects. That is, a DBN is constructed from the grounding of a parameterised DBN.



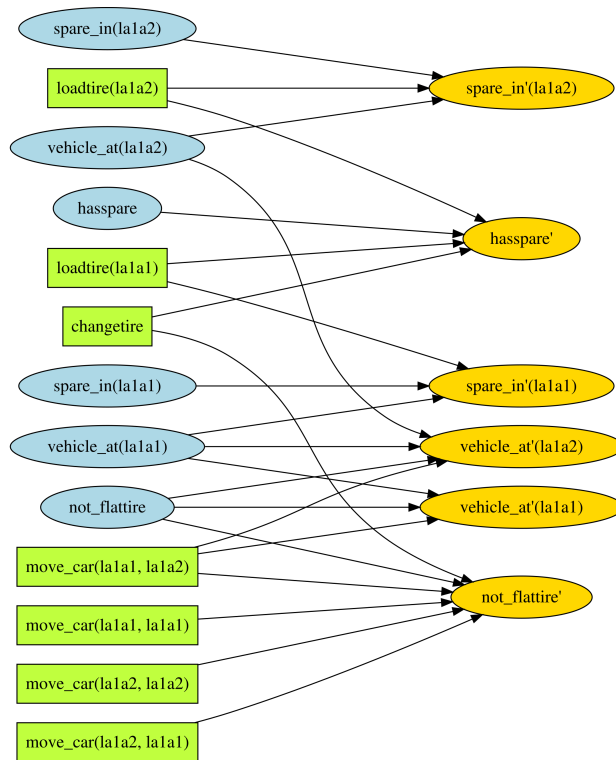


Figure 2.3: A portion of the DBN for a problem of `Triangle Tireworld`. Green rectangles denote actions at time step  $t$ , blue (orange) ovals denote state predicates at time step  $t$  ( $t + 1$ ).

A parameterised transition function  $\mathcal{T}$  for an RMDP (see Definition 3) can be represented by a parameterised DBN. This representation can be written in RDDDL (see Section 2.2.3). We denote a model which uses a parameterised DBN to represent  $\mathcal{T}$  by  $\mathcal{M}_{DBN}$ .

### 2.3.3 Model Learning

Transition functions, or action models, are normally hand-coded by human experts but this can be difficult in domains where there are complex dynamics. This is known as the knowledge engineering problem [33]. We shall refer to the learning of transition functions as model learning. We want to learn models which can be used to predict the next states for unseen state-action pairs and for any problem of a domain. The maximum likelihood model relies on a set of observations which consist of states  $s \in \mathcal{S}$  and actions  $a \in \mathcal{A}$ . Since  $\mathcal{S}$  and  $\mathcal{A}$  are specific to a problem,  $\mathcal{M}_{MLM}$  cannot make predictions for other problems. On the other hand, a **relational model** is a model which can make predictions in any problem of the same domain. This opens

up the possibility of learning a relational model from observations obtained in a problem, and then use the learned model to make predictions in other problems. In other words, the model provides valuable information on solving other problems. A parameterised DBN is a relational model as it can be ground to a DBN with the set of objects  $\mathbf{O}$  in any problem in order to make predictions in that problem.

There is work which addresses the automatic inference of models from training data. They differ from one another in the assumptions made, representations used, and the type of training data required. We review relevant model learning algorithms and select the most suitable one for our work. Since we use RDDDL to describe problems, a suitable model learner should learn transition functions which are represented by or can be translated to DBNs. Also, we are interested in decision making under uncertainty. This excludes model learners which learn deterministic action models such as [29, 194, 200] from consideration.

[126] learns propositional rules by searching for dependencies among state variables over time. They perform a best-first search over the space of possible dependencies using a heuristic that guides the search towards frequently occurring preconditions and frequently co-occurring pairs of preconditions and effects. However, propositional rules are not relational models. [134] learns relational probabilistic action models using three phases of greedy search with search operators. [134] was extended in [135] to include deictic references (i.e., actions can change the properties of objects beside those in their terms) and concepts which provide background information for deictic references and preconditions. [115] uses voted perceptrons to learn action models under noise and partial observability; each classifier predicts if a state predicate changes due to an action. One classifier is required for each term of an action.

[163] and [39] learn the structures of DBNs representing the transition functions of factored MDPs online. However, similar to [126], the DBNs are propositional. [108] learns parameterised DBNs from state transitions which are first transformed to relational representations for generalisation, and then transformed to propositional representations. No noise or partial observability are allowed in the training data. Inductive Logic Programming (ILP) [116] is used to generate a minimal set of rules to explain the effects in the propositional transitions. Exogenous effects can also

be learned and are represented by rules which do not involve any action. [142] proposes a framework which learns a RDDDL domain. ILP is used to learn rules from observations across multiple problem instances. Rules which are too general or are not applicable in another problem are removed.

Some work requires or utilises prior knowledge such as approximate action models [59, 75] (e.g., [59] requires action models that can have missing but not extraneous state predicates in preconditions and effects), known terms in actions [114, 194], or successful plans [188]. We avoid the need for expert knowledge or domain-specific knowledge in our work. Minimally, we assume that the terms in actions are known. For example, we know that the symbolic action `move_vehicle` in `Triangle Tireworld` is of the form `move_vehicle(WP1, WP2)`.

Following the above discussion, we use the model learner from Martínez et al. [108] as it meets all of our requirements. It learns a relational model with probabilistic and exogenous effects which can be translated to RDDDL.<sup>6</sup> We require model learners that can learn exogenous effects as they are present in our domains of interest (see Section 2.8.2). Martínez et al. [108] extend the propositional rules learner, LFIT (learning from interpretation transition), from Ribeiro and Inoue [143]. We denote Martínez et al. [108]’s model learner by RLFIT (relational LFIT).

**RLFIT.** We briefly describe RLFIT and refer readers to [108] for full details. There are two steps to RLFIT:

1. Generate candidate propositional rules with LFIT [143] to cover the training data.
2. Select the best subset of candidates which maximises a score function with a heuristic search.

LFIT is an ILP method which learns a set of minimal probabilistic rules that models all effects in the training data. LFIT uses a top-down approach to generate rules by specialisation from the most general rules. Next, the propositional rules are translated to relational rules and a score function is used to evaluate each rule. The

---

<sup>6</sup>The model learner from Rao and Jiang [142] seems most suitable for our use but the code is not publicly available.

score function is:

$$Score(\mathbf{R}, \Xi) = E_{\Xi \in \Xi}[\log(Pr(\Xi|\mathbf{R}))] - \rho \frac{Pen(\mathbf{R})}{Conf(\Xi, \epsilon)}, \quad (2.4)$$

where  $\mathbf{R}$  is a set of rules,  $\Xi$  is a set of observations or training data,  $\Xi$  is an observation  $(s, a, r, s')$ ,  $Pr(\Xi|\mathbf{R})$  is the likelihood that an observation  $\Xi$  is covered by  $\mathbf{R}$ ,  $\rho > 0$  is a hyperparameter,  $Pen(\mathbf{R})$  is the penalty term equal to the total number of literals in the body of every rule in  $\mathbf{R}$ , and  $Conf(\Xi, \epsilon)$  is the confidence obtained from the Hoeffding’s inequality—the upper bound on the probability that an estimate has an error not more than  $\epsilon$  is  $Conf(\Xi, \epsilon) \leq 1 - e^{-2\epsilon^2|\Xi|}$ . The score function is modified from [135] to include the confidence term so that the penalty is increased for specific rules which cover few training data (i.e., prefer general rules over specific rules).

**Learning preconditions.** By comparing  $s$  and  $s'$  in an observation  $(s, a, r, s')$ , a subset of the effects of  $a$  can be determined. On the other hand, learning preconditions is a difficult problem which requires prohibitively large training data depending on the number of literals in the preconditions [186]. We assume that when an in-applicable action is executed, the state remains unchanged and feedback is given to indicate that the execution has failed. Failed executions yield state transitions which are uninformative as they do not reveal insights on which part of the preconditions is missing or incorrect. If the action succeeds, the state in which it is executed in is a superset of its preconditions. Given sufficient training data where the action succeeds, its preconditions can be determined. Unfortunately, such training data are hard to come by as actions fail to execute most of the time when preconditions are unknown. Therefore, although RLFIT can learn preconditions, we assume in our experiments that the preconditions are known while the conditional probability functions are unknown.

### 2.3.4 Evaluate Correctness of Approximate Models

RLFIT and other aforementioned work in Section 2.3.3 are heuristic in nature. Thus, the learned models are not guaranteed to be correct or complete.

**True Model**Precondition: `vehicle_at(WP1) ∧ ROAD(WP1, WP2) ∧ not_flattire`Effect (0.75): `vehicle_at(WP2) ∧ ¬vehicle_at(WP1)`Effect (0.25): `vehicle_at(WP2) ∧ ¬vehicle_at(WP1) ∧ ¬not_flattire`**Learned Model #1**Precondition: `vehicle_at(WP1) ∧ ROAD(WP1, WP2) ∧ not_flattire`Effect (1): `vehicle_at(WP2) ∧ ¬vehicle_at(WP1)`**Learned Model #2**Precondition: `vehicle_at(WP1) ∧ ROAD(WP1, WP2) ∧ not_flattire`Effect (0.5): `vehicle_at(WP2) ∧ ¬vehicle_at(WP1)`Effect (0.5): `vehicle_at(WP2) ∧ ¬vehicle_at(WP1) ∧ ¬not_flattire`**Learned Model #3**Precondition: `vehicle_at(WP1) ∧ ROAD(WP1, WP2) ∧ not_flattire`Effect (0.25): `vehicle_at(WP2) ∧ ¬vehicle_at(WP1)`Effect (0.75): `vehicle_at(WP2) ∧ ¬vehicle_at(WP1) ∧ ¬not_flattire`

Figure 2.4: Action models for `move_vehicle(WP1, WP2)`. Numeric values in parentheses represent the probabilities of effects.

**Definition 7 (Approximate Models)**

*A model is incomplete if there is at least one probabilistic outcome of an action which is not included. A model is incorrect if there is at least one outcome of an action which is not true; this includes the probability of the outcome. An approximate model is a model which is incomplete, incorrect, or both.*

Model learners are evaluated by their computation time or complexity [126, 200], the correctness of the learned models [108, 135], or the performance of planning with the learned models [75, 188]. The correctness of an approximate model  $\tilde{\mathcal{M}}$  can be defined as the **average variational distance** between  $\tilde{\mathcal{M}}$  and the true model  $\mathcal{M}$  [135]:

$$VD(\mathcal{M}, \tilde{\mathcal{M}}) = \frac{1}{|\Xi^-|} \sum_{\Xi \in \Xi^-} |\mathcal{M}(\Xi) - \tilde{\mathcal{M}}(\Xi)|, \quad (2.5)$$

where  $\Xi^-$  is a set of observations used as test data (i.e., data that is not in the training data) and  $\mathcal{M}(\Xi)$  is the probability of making the observation  $\Xi$  as predicted by  $\mathcal{M}$ .

Another alternative metric is the **absolute distance** between two sets of rules. Given the true model  $r_1$  and an approximate model  $r_2$ , the absolute distance is given

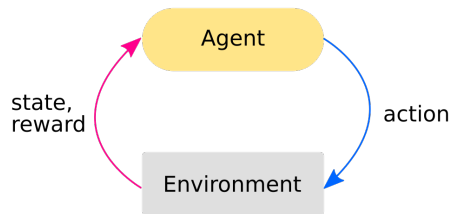


Figure 2.5: The RL framework.

as:

$$d(r_1, r_2) = d^-(r_1^p, r_2^p) + d^-(r_2^p, r_1^p) + d^-(r_1^e, r_2^e) + d^-(r_2^e, r_1^e), \quad (2.6)$$

where the superscripts  $p$  and  $e$  refer to the preconditions and effects of a rule, respectively, and  $d^-(r_1^p, r_2^p)$  is the number of literals that are in  $r_1^p$  but not in  $r_2^p$ . Absolute distance is more intuitive to understand than variational distance and does not depend on the distribution of the true model; however, variational distance favours similar distributions and is well-defined when the learned model predicts the probability of an actual observation as zero [134]. In our experiments, we use variational distance to measure the correctness of the learned models since it is also used by Martínez et al. [108] to evaluate RLFIT.

As an example to illustrate the difference between the average variational distance and absolute distance, Figure 2.4 shows the true action model and three different learned action models for the symbolic action `move_vehicle(WP1, WP2)` in `Triangle Tireworld`. Model #1 does not have the probabilistic effect of getting a flat tire while model #2 does. Both models have the same variational distance of 0.25 but different absolute distances. The absolute distance is 3 for model #1 and 0 for model #2. Models #2 and #3 have the same absolute distances of 0 but different variational distances of 0.25 and 0.5, respectively.

## 2.4 Reinforcement Learning

RL is the field of study for decision making in learning problems. Figure 2.5 illustrates the RL framework. An agent interacts with the environment by executing actions which could change the environment. The agent observes the new state and immediate reward which provides feedback on its action. If the agent is not able to observe some parts of the state (i.e., the values of some state predicates cannot

be observed), then the environment is partially observable. If there are errors in the observations (i.e., the values of some state predicates could be wrong), then the environment is noisy. We consider only fully observable environments with no noise. Such environments can be modelled as MDPs.

RL methods are categorised as model-based or model-free methods. The objective of an RL method is to find a policy which maximises the sum of expected discounted rewards or expected return.

**Definition 8 (Policy)**

A *deterministic policy*  $\pi : s \mapsto a$  maps a state  $s$  to an action  $a$  where  $s \in \mathbf{S}$  and  $a \in \mathbf{A}$ . A *stochastic policy* is a probability distribution over the actions  $\mathbf{A}$  conditioned on the state  $s$ .

The expected return for a policy  $\pi$  is computed using the value function [139];  $V^\pi(s)$  is the expected return starting from state  $s$  and then following  $\pi$ :

$$V^\pi(s) = \sum_{a \in \mathbf{A}} \pi(a|s) \left( \mathcal{R}(s, a) + \gamma \sum_{s' \in \mathbf{S}} \mathcal{T}(s'|s, a) V^\pi(s') \right), \quad (2.7)$$

where  $\pi(a|s)$  is the probability  $\pi$  selects  $a$  in  $s$ . The value function does not tell us which action should be selected. For this purpose, the action-value function or Q-function is used;  $Q^\pi(s, a)$  is the expected return starting from state  $s$ , executing action  $a$ , and then following  $\pi$ :

$$\begin{aligned} Q^\pi(s, a) &= \mathcal{R}(s, a) + \gamma \sum_{s' \in \mathbf{S}} \mathcal{T}(s'|s, a) V^\pi(s') \\ &= \mathcal{R}(s, a) + \gamma \sum_{s' \in \mathbf{S}} \mathcal{T}(s'|s, a) \sum_{a' \in \mathbf{A}} \pi(a'|s') Q^\pi(s', a'). \end{aligned} \quad (2.8)$$

**Definition 9 (Optimal Value Function)**

The *optimal value function*,  $V^*(s)$ , is the maximum value function over all policies:

$$V^*(s) = \max_{\pi} V^\pi(s). \quad (2.9)$$

**Definition 10 (Optimal Q-function)**

The optimal Q-function,  $Q^*(s, a)$ , is the maximum Q-function over all policies:

$$Q^*(s, a) = \max_{\pi} Q^{\pi}(s, a). \quad (2.10)$$

The optimal value function and Q-function are recursively related by the Bellman optimality equations [9]:

$$Q^*(s, a) = \mathcal{R}(s, a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{T}(s'|s, a) V^*(s'), \quad (2.11)$$

$$\begin{aligned} V^*(s) &= \max_a \mathcal{R}(s, a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{T}(s'|s, a) V^*(s') \\ &= \max_a Q^*(s, a). \end{aligned} \quad (2.12)$$

A policy can be generated from a Q-function. For example, the greedy policy selects an action with the maximal Q-value, or **greedy action**, with a random selection as a tiebreaker. The optimal policy  $\pi^*$  is a greedy policy generated from an optimal Q-function (note that there could be multiple optimal policies). A problem is solved if the optimal policy is known.

Balancing between exploration and exploitation is a perennial issue in RL. Exploration seeks meaningful observations to learn more about the environment while exploitation seeks to maximise reward with the known information. Exploration is often preferred initially and less so in later episodes. A greedy policy risks being too myopic, settling too quickly for actions which seem promising initially but are in fact **sub-optimal** (actions whose Q-values are not near-optimal or optimal). One policy which avoids this issue is the  $\epsilon$ -greedy policy which selects a random action with a probability of  $\epsilon$  and a greedy action otherwise. The softmax policy balances exploration and exploitation by selecting an action according to an exponential softmax distribution:

$$\pi(a|s) = \frac{e^{\frac{Q(s,a)}{\tau_{temp}}}}{\sum_{a' \in \mathcal{A}} e^{\frac{Q(s,a')}{\tau_{temp}}}}, \quad (2.13)$$

where  $e \approx 2.17828$  is the base of the natural logarithm and  $\tau_{temp} \in (0, 1]$  is a hyperparameter called “temperature”. When  $\tau_{temp}$  approaches zero, the softmax policy is almost the same as the greedy policy. Conversely, when  $\tau_{temp}$  increases, the



probability of selecting a non-greedy action increases.

### 2.4.1 Model-Free Reinforcement Learning

Model-free RL methods approximate the value function or Q-function directly and is sometimes called direct methods. They do not require nor utilise any model.

#### Temporal-Difference (TD) Learning

Temporal-difference (TD) learning updates estimates of the Q-values or values based on current estimates and new observations (i.e., it bootstraps). One of the most well-known TD method is Q-learning [189]. Q-learning learns a Q-function which converges to  $Q^*$  regardless of the policy being followed. In other words, Q-learning is an off-policy method which learns about the optimal policy while following an exploratory policy.<sup>7</sup> Given an observation  $(s_t, a_t, r_t, s_{t+1})$ , Q-learning updates the Q-function as follows:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \delta_t, \quad (2.14)$$

$$\delta_t = r_t + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t), \quad (2.15)$$

where  $\alpha \in (0, 1]$  is the learning rate and  $\delta_t$  is the TD error at time step  $t$ .  $\alpha$  controls how much the Q-value is updated based on the observation at time step  $t$  and is typically decayed over time steps.

In stochastic environments, Q-learning has an overestimation bias as it uses the maximum Q-value to estimate the maximum expected Q-value in Equation 2.15. This overestimation could slow the convergence of Q-learning. Double Q-learning [68] mitigates this issue by using two Q-functions,  $Q^B$  and  $Q^T$ , to estimate the maximum expected Q-value.  $Q^B$  is updated at every time step while  $Q^T$  is set to  $Q^B$  at the end of an episode. Double Q-learning updates  $Q^B$  with Equation 2.14 using the TD error:

$$\delta_t = r_t + \gamma Q_t^T(s_{t+1}, \pi_{Q^B}(s_{t+1})) - Q_t^B(s_t, a_t), \quad (2.16)$$

where  $\pi_{Q^B}$  is the greedy policy generated from  $Q^B$ .

---

<sup>7</sup>An on-policy method learns about the policy that is being followed.

**TD( $\lambda$ )**

TD( $\lambda$ ) [166] is a family of TD methods that combines TD learning with eligibility traces to speed up learning from temporally delayed reward signals (i.e., rewards which are given after a long sequence of actions). Eligibility traces address the temporal credit assignment by assigning credit for an immediate reward to most frequently visited states (frequency heuristic) and most recently visited states (recency heuristic). Another perspective to look at TD( $\lambda$ ) methods is that they combine Monte Carlo and TD methods, averaging over  $n$ -step returns with eligibility traces.  $\lambda \in [0, 1]$  is a hyperparameter that denotes the use and decay of eligibility traces. For example, TD(0) (where  $\lambda = 0$ ) is a one-step TD method which does not use eligibility traces. TD( $\lambda$ ) updates the Q-function as follows:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \delta_t e_t(s_t, a_t), \quad (2.17)$$

where  $e_t(s, a)$  is the eligibility trace for a state-action pair  $(s, a)$  at time step  $t$ . We use the replacing eligibility trace [159] which is updated at time step  $t$  as follows:

$$e_t(s, a) = \begin{cases} 1 & \text{if } s = s_t \text{ and } a = a_t \\ \gamma \lambda e_{t-1}(s, a) & \text{otherwise} \end{cases} \quad (2.18)$$

The eligibility trace is initialised to zero (i.e.,  $e_0(s, a) = 0$ ). We use eligibility traces for state-action pairs though it can also be used for states (i.e.,  $e_t(s)$  instead of  $e_t(s, a)$ ). Replacing traces can only be used with binary features. Accumulating traces [166] and dutch traces [182] are two other types of eligibility traces. A TD method can be augmented with eligibility traces to become a TD( $\lambda$ ) method. For example, Q( $\lambda$ ) [136, 189] is Q-learning augmented with eligibility traces by replacing Equation 2.14 with Equation 2.17.

**2.4.2 Model-Based Reinforcement Learning**

Model-based RL methods, or indirect methods, require a model of the problem. The model is typically an approximation of the true model comprising of  $\mathcal{T}$  and  $\mathcal{R}$  and is learned from observations. Model-based RL methods solve the estimated

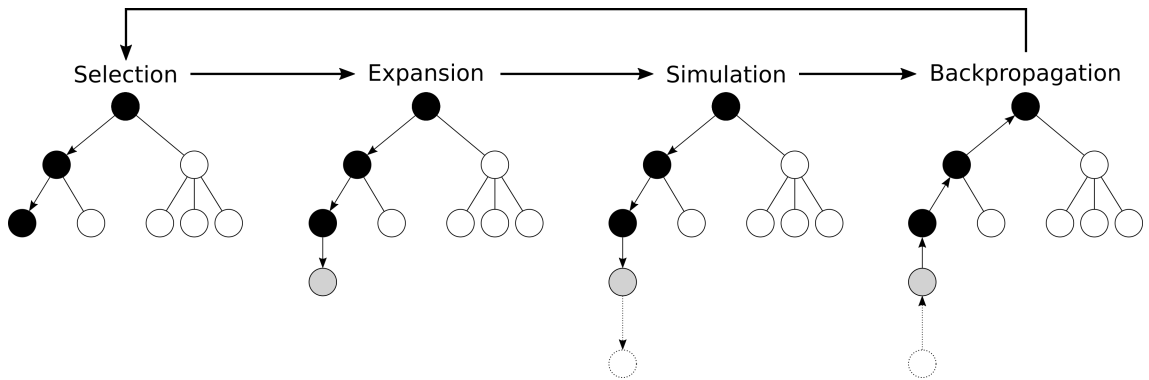


Figure 2.6: The four phases in MCTS: selection, expansion, simulation, and backpropagation.

MDP where  $\mathcal{T}$  and  $\mathcal{R}$  are learned; the estimated MDP is its internal representation of the underlying problem. These methods rely on the simulation lemma which establishes that if the learned model is a sufficiently accurate approximation of the true model, then the (near-)optimal policy of the estimated MDP is provably close to the (near-)optimal policy of the true MDP [84].

There are two main procedures which model-based RL methods perform in every time step. First, they use the learned model to select an action. This usually involves sampling from the model the immediate reward and next state for executing an action (i.e., the model is generative). Second, the agent executes an action and receives feedback from the environment. This feedback is used to update the learned model (i.e., online model learning) and possibly the uncertainty in the model.

## Monte Carlo Tree Search

Monte Carlo Tree Search or **MCTS** is a search algorithm which uses sampling methods to build a search tree. It is used with either the true model (in which case it is considered a planner) or a learned model. A local policy which covers a subset of the state space is generated from the search tree. A node in the search tree represents a state and an edge represents an action. The search tree is build up incrementally and the search can be terminated at any time (i.e., **MCTS** is an anytime algorithm). There are four phases in **MCTS** which are performed in this order: (1) selection, (2) expansion, (3) simulation, and (4) backpropagation.

Figure 2.6 illustrates the construction of the search tree in the four phases. A black circle represents a node which is selected by the **tree policy** during the

selection phase. The arrows represent the sequence of traversal. A white circle represents a node which has been added to the search tree but is not selected by the tree policy. A grey circle represents a node which is added during the expansion phase. During the simulation phase, a sequence of states are sampled; these states (in dotted borders) will not be added to the search tree. Lastly, backpropagation updates the statistics for the nodes visited (grey and black nodes). The arrows show the order of update starting from the leaf (grey) node to the root node. The cycle continues with the selection phase again until a termination condition is satisfied. Next, we describe each phase in details.

**Selection.** A root node is created for the current state. Starting from the root node, an action is selected using the tree policy. The next state is sampled using the generative model. This phase continues at the node for the next state until the node for the next state has not yet been added to the search tree. At the start, the root node is a leaf node.

The tree policy selects at random an action which has not yet been selected. If every action has been selected at least once, then the tree policy selects an action with the maximal value. The upper confidence bounds applied to trees (UCT) algorithm [88] is a variant of MCTS which uses a tree policy that selects an action  $a_t$  in a state  $s$  with consideration to the upper confidence bounds (UCB1) exploration bonus:

$$a_t = \arg \max_a \tilde{Q}^{UCT}(s, a) + \sqrt{\frac{2 \ln \sum_{a' \in \mathbf{A}} N(s, a')}{N(s, a)}}, \quad (2.19)$$

where  $\tilde{Q}^{UCT}(s, a)$  is the Q-value estimated by UCT and  $N(s, a)$  is the number of times  $a$  has been selected at the node for  $s$ . The first term represents exploitation (i.e., select the action with the maximal Q-value) and the second term represents the exploration bonus which encourages the selection of actions with small values of  $N(s, a)$ .

**Expansion.** The search tree is expanded by adding a new node for the next state from the selection phase. A directed edge joins the leaf node in the selection phase to this new node (which is the new leaf node). This edge represents the action selected by the tree policy.

**Simulation.** If the leaf node is not a terminal state, then a trajectory is sampled from the leaf node using a **rollout policy** until a terminal state is reached or the end of the time horizon is reached. **Policy rollout** [13] is the process of simulating state transitions to sample a trajectory. The rollout policy is typically a random policy which can be computed at very low cost. The sampled states in a policy rollout are not added to the search tree. The total discounted return for a sampled trajectory is:

$$V_{sim} = \sum_{t=t_{start}}^{t_{end}-1} \gamma^{t-t_{start}} r_t, \quad (2.20)$$

where  $t_{start}$  is the time step of the leaf node and  $t_{end}$  is the time step of the last simulated node.  $t_{end}$  is the time horizon  $H$  if the policy rollout simulates to the end of the time horizon.

**Backpropagation.** The nodes along the path are updated starting from the leaf node (added in the expansion phase) to the root node:

$$W(s, a) \leftarrow W(s, a) + \Delta W(s), \quad (2.21)$$

$$N(s, a) \leftarrow N(s, a) + 1, \quad (2.22)$$

$$\tilde{Q}^{UCT}(s, a) = \frac{W(s, a)}{N(s, a)}, \quad (2.23)$$

where  $W(s, a)$  is the cumulative value and  $\Delta W(s)$  is the discounted return starting from the node for  $s$  to the last node in the simulation phase.  $W(s, a)$  and  $N(s, a)$  are initialised to zeros. The estimated Q-value,  $\tilde{Q}^{UCT}(s, a)$ , is the average of the sampled returns. We define a general form for  $\Delta W(s)$  which considers the discount factor  $\gamma$  and non-zero action costs (i.e., negative immediate rewards for executing actions):

$$\Delta W(s) = \begin{cases} r + \gamma V_{sim} & \text{if node is a leaf node} \\ r + \gamma \Delta W(s') & \text{otherwise} \end{cases} \quad (2.24)$$

where  $r$  is the immediate reward for reaching the state  $s$  and the sampled trajectory is  $s \xrightarrow{a} s' \xrightarrow{a'} \dots$

After backpropagation, the selection phase is initiated again. The four phases

are repeated for a number of times where each iteration represents one trial. At the start of each trial, the tree policy is evaluated during the selection phase to select the most promising action. At the end of the trial, the Q-values are updated during backpropagation which improves the tree policy. Thus, MCTS interleaves policy evaluation and policy improvement. Using the tree policy, it expands the tree asymmetrically, focusing the search in the most promising regions of the state space.

Given sufficient trials, UCT converges to the limit and its estimated Q-values approach the optimal Q-values [88] but only if the generative model is the true model. The algorithm terminates based on a timeout which is a hyperparameter limiting the amount of time UCT runs for. A local policy is generated from the Q-values of actions at the root node (e.g., a greedy policy selects an action with the maximal Q-value). Alternatively, the policy can also select the action with the most number of visits (i.e.,  $N(s, a)$ ) [157]. Since most promising actions are selected during the selection phase, an action that has been selected most is the most promising action. This action might not have the maximal Q-value if actions are selected based on their Q-values and exploration bonuses (see Equation 2.19).

### 2.4.3 Measuring Performance

The objective of RL methods is to learn an optimal policy which maximises a criterion. Given a sequence of rewards  $r_0, \dots, r_{H-1}$  the agent receives in an episode with a time horizon  $H$ , the total reward is:

$$V = \sum_{t=0}^{H-1} r_t, \quad (2.25)$$

the average reward is:

$$V = \frac{1}{H} \sum_{t=0}^{H-1} r_t, \quad (2.26)$$

and the total discounted reward is:

$$V = \sum_{t=0}^{H-1} \gamma^t r_t. \quad (2.27)$$

In this dissertation, we will consider the maximisation of the total discounted reward. It is also important to consider the cost of learning the optimal policy. This is measured in three aspects: (1) sample complexity, (2) time complexity, and (3) space complexity.

**Sample complexity.** The sample complexity is the amount of observations needed to achieve near-optimal or optimal performance [81]. In other words, it is the number of time steps a policy selects a sub-optimal action. Often, an agent needs to explore in order to learn and in doing so, it executes some sub-optimal actions.

**Definition 11 (Sample Complexity)**

*The sample complexity of a policy  $\pi$  with respect to the accuracy  $\epsilon$  and confidence  $1 - \delta$  is the minimum number of time steps  $K$  such that for all  $t \geq K$ ,  $Pr[V^\pi(s_t) \geq V^*(s_t) - \epsilon] \geq 1 - \delta$  where  $\epsilon \in (0, 1)$  and  $\delta \in (0, 1)$ .*

An algorithm is near-optimal if it learns a policy  $\pi$  which achieves an expected return within  $\epsilon$  of the optimal return with a probability of no less than  $1 - \delta$ . The optimal return is the expected return from following an optimal policy. For episodic problems, since the time step resets to 0 at the start, the sample complexity is measured by the cumulative time steps over episodes.

**Time complexity.** The time complexity is the amount of computational work needed to run an algorithm. There are typically two main steps in an RL algorithm as illustrated in Figure 2.5: selecting an action to execute and learning from the observation. The time complexity per time step is the sum of the time complexity for these two steps. A model-free RL method which generates a policy from the Q-function takes  $O(|\mathbf{A}|)$  steps to select an action. This assumes that the time complexity of evaluating the Q-value of each action is  $O(1)$ . This can be as simple as retrieving the Q-value from a table. A model-based RL method is computationally more expensive if it performs a search to select an action.

**Space complexity.** Space complexity is the memory cost of an algorithm to store information required for its operation. For example, if the Q-function is represented

as a table with one element for each  $(s, a)$  pair, the space complexity is  $O(|\mathcal{S}||\mathcal{A}|)$  which is the number of elements in the table.

## 2.5 Function Approximation

A tabular representation of the Q-function is an exact representation. Each state is considered to be unique and different from the rest. However, in environments where there are structures (e.g., factored MDPs and RMDPs), states can be aggregated to an **abstract state** for the purpose of approximating the Q-values. A tabular representation does not offer generalisation over states since it represents states exactly. Furthermore, a tabular representation is impractical for problems with large state spaces. One solution is to approximate the Q-function by projecting the state space into a lower dimensional space using a set of features. A function approximation can be a neural network, decision tree, linear regression, Gaussian process, etc. We focus the discussion here on linear function approximation which is used in our work.

### 2.5.1 State Abstraction

A function approximation performs state abstraction when aggregating states to abstract states. When combined with an RL algorithm, this allows generalisation over states and even to unseen states. State abstraction also removes irrelevant information which is not essential for knowledge transfer between problems [185]. Li, Walsh, and Littman [100] propose a unifying framework for exact state abstraction and show that under certain conditions, some properties of the original problem are preserved in the abstract problem:

1. Model-irrelevance abstraction preserves the one-step model.
2.  $Q^\pi$ -irrelevance abstraction preserves the Q-function for all policies.
3.  $Q^*$ -irrelevance abstraction preserves the optimal Q-function.
4.  $a^*$ -irrelevance abstraction preserves the optimal action and its value.
5.  $\pi^*$ -irrelevance abstraction preserves the optimal action.

Abel, Hershkowitz, and Littman [1] propose approximate state abstractions which relaxes the conditions for aggregation. Applying the right state abstraction to a



given problem is a major challenge. Some work uses statistical learning to learn state abstraction and they differ in the criteria for aggregating states. For example, [26] aggregates states with the same reward and Q-values for each action whereas [110] aggregates states with the same optimal action and similar Q-values for the optimal action.

Analogous to state abstraction, temporally abstract actions or options are learned to achieve subtasks independently [7, 169]. This exploits the hierarchical nature of the problem which allows its decomposition to subtasks. Since subtasks are simpler, option-specific state abstraction is possible. Prior work [6, 37, 78] either operates in a propositional representation which does not allow generalisation over different problems or requires additional learning to generalise to a different problem.

### **Example 3 (Abstracting States with Features)**

*A feature can simply be a literal or a conjunction of literals. Let there be two features  $\phi_1 = \text{vehicle\_at}(la1a1)$  and  $\phi_2 = \neg\text{vehicle\_at}(la1a1) \wedge \text{vehicle\_at}(la1a2)$ . Following Example 1,  $\phi_1$  is true in  $s_0$  and false in  $s_1$  while  $\phi_2$  is false in  $s_0$  and true in  $s_1$ . In essence, states are mapped to these two features, abstracting away the other literals. Features are not restricted to literals and can be state variables, handcrafted predicates, or functions which return real values.*

## **2.5.2 Granularity**

A set of features partitions the state space into regions such that states in a region have the same approximated Q-values. Each region is an abstract state. If states with different optimal Q-values are mapped to the same abstract state, then the resulting policy could be unsound. Intuitively, this is because state abstraction removes information from the state which is crucial for making (near-)optimal decisions. The granularity of a function approximation is a measure of the size of the partitions of the state space (or the number of states mapped to an abstract state). The granularity is coarse if each feature covers, or is true in, a large region of the state space. The coverage of a feature is the region of the state space for which the feature evaluates to 1 (for binary features) or non-zero (for non-binary features) [57]. Features with low coverage give fine granularity in the function approximation, and

thus have better accuracy than features with high coverage. On the other hand, features with high coverage offer better generalisation as learning is done over a smaller number of abstract states.

### 2.5.3 Linear Function Approximation

In a linear function approximation, the Q-function is approximated by linear functions of features [168]:

$$\tilde{Q}(s, a) = \theta_0 + \theta_1 \phi_1(s, a) + \cdots + \theta_n \phi_n(s, a) = \boldsymbol{\theta}^T \boldsymbol{\Phi}(s, a), \quad (2.28)$$

where the set of features  $\boldsymbol{\Phi}(s, a)$  maps  $(s, a)$  to a vector of real numbers  $\mathbb{R}^{n+1}$  and  $\boldsymbol{\theta} \in \mathbb{R}^{n+1}$  is the vector of weights (or weight vector). The Q-function approximation  $\tilde{Q}$  is typically a non-linear function of literals. The Q-values of actions are decoupled by separating their features in  $\boldsymbol{\Phi}(s, a)$ :

$$\boldsymbol{\Phi}(s, a_1) = \begin{bmatrix} 1 \\ \phi_{1,1} \\ \phi_{1,2} \\ \vdots \\ \phi_{1,n} \\ \mathbf{0}_{(N-n-1) \times 1} \end{bmatrix}, \quad \boldsymbol{\Phi}(s, a_2) = \begin{bmatrix} \mathbf{0}_{(n+1) \times 1} \\ 1 \\ \phi_{2,1} \\ \phi_{2,2} \\ \vdots \\ \phi_{2,m} \\ \mathbf{0}_{(N-n-m-2) \times 1} \end{bmatrix}, \quad \dots,$$

where  $\phi_{j,i}$  is the  $i$ -th feature for  $a_j$ ,  $n$  is the number of features for  $a_1$ ,  $m$  is the number of features for  $a_2$ ,  $N$  is the cardinality of  $\boldsymbol{\Phi}(s, a)$ , and the constant value 1 is the bias term.  $\boldsymbol{\Phi}$  is a concatenation of the set of features for each action  $a \in \mathbf{A}$ :  $\boldsymbol{\Phi} := [\boldsymbol{\Phi}_{a_1}, \dots, \boldsymbol{\Phi}_{a_{|\mathbf{A}|}}]$  where  $\boldsymbol{\Phi}_{a_i}$  is the set of features for  $a_i$ .<sup>8</sup>  $\boldsymbol{\Phi}_{a_i}(s, a) := \mathbf{0}$  if  $a \neq a_i$ .

The learning objective is to find the set of features and weights such that  $\tilde{Q}(s, a)$  closely approximates  $Q^*(s, a)$ . The update rule for a weight component  $\theta \in \boldsymbol{\theta}$  is:

$$\theta \leftarrow \theta + \alpha \delta \frac{\phi(s, a)}{\|\boldsymbol{\Phi}(s, a)\|_1}, \quad (2.29)$$

<sup>8</sup>By omitting the arguments,  $\boldsymbol{\Phi}$  denotes the set of features rather than their values.

where  $\phi(s, a)$  is the value of the feature corresponding to  $\theta$  and the  $L1$  norm is the normalisation factor.

Linear function approximations applied to RL is well understood [190] and has some convergence guarantees under certain conditions [13, 168, 178]. A drawback of linear function approximations is that it is limited in representational capacity and cannot represent every possible value function or Q-function. As we will see later in the dissertation (Section 3.7), this drawback is not inherent in our domains of interest.

#### 2.5.4 Feature Discovery

Features can be binary (i.e., they map to 0 or 1) or non-binary, and can be hand-crafted or deduced automatically. A straightforward approach is to use literals as features since features partition states into abstract states and literals describe a state.

##### **Definition 12 (Base and Conjunctive Features)**

*A base feature is a literal and a conjunctive feature is a conjunction of base features. A feature is either a base feature or a conjunctive feature.*

Let  $\mathbf{P}^+$  denotes the set of positive literals for every state predicate and  $\neg\mathbf{P}^+$  denotes its negation (i.e., the set of negative literals). If  $\Phi$  is the set of literals  $\mathbf{P}^\# = \mathbf{P}^+ \cup \neg\mathbf{P}^+$ , then  $\tilde{Q}(s, a)$  is a linear function of the literals which could be a poor approximation of  $Q^*(s, a)$ . Non-linearity is introduced when conjunctive features are added to  $\Phi$ ;  $\tilde{Q}(s, a)$  is now a non-linear function of the literals but remains a linear function of the features. This results in a finer granularity of the function approximation since a conjunctive feature has a lower coverage than a base feature.

In many machine learning problems, automatically selecting features, or learning features, to represent the state is a key challenge. Recent efforts in feature discovery can be classified as model-free or model-based methods. Model-free methods [54, 55, 85, 129, 133, 191] select features based on an optimisation criterion such as the Bellman error or the TD error. Model-based methods [66, 77, 96, 105] use a model to prune unnecessary features. We use the model-free feature discovery algorithm

iFDD+ [55] which shall be described in Section 3.1.1. iFDD+ requires as input an initial set of features. We use a learned model to determine the set of literals which are necessary as (base) features for each action as discussed in Section 4.3.1. A comparison with model-free methods is out of the scope of our work. Instead, we focus on comparing model-based methods with our approach.

Jong and Stone [77] extend the maximum likelihood model with averagers, a class of function approximators, to generalise over and predict for unseen state-action pairs. The extended model is learned online and used in value iteration to compute an approximate value function. We approximate the Q-function with features while they do not deal with features. Instead, they approximate the model from which a value function is computed. Mahadevan and Maggioni [105] propose proto-value functions as the orthonormal basis set for approximating any value function. The proto-value functions are determined from the topology of the state space which is represented as an undirected graph. The topology can be learned online from observed state transitions. However, learning this graph can be computationally expensive and the graph does not apply for other problems of the same domain. Our method utilises the transition function represented by parameterised DBNs (see Section 4.3.1), and thus can be applied to any problem of the same domain without needing any further learning or observations if the model is available. In [124], the transition function is represented by logistic regression models which allows fast online model learning in high dimensional spaces. Features with a corresponding column of zeros in the weight matrix are deemed to be irrelevant. Similarly, Jung and Stone [80] use Gaussian processes for approximate policy evaluation; irrelevant features are features with an insignificantly small covariance. Our problems are written in RDDL which uses DBNs to model transition functions. A logistic regression model cannot fully capture the conditional dependencies in a DBN while Gaussian process is not applicable to our work since we deal with discrete states. Our work is most similar to [96] which extracts a minimal set of features by considering the conditional independencies in DBNs representing the learned transition function. This allows efficient learning of a policy defined over the features. A different approach is used in [66] where the necessary features are learned online rather than offline. It performs directed exploration to specific states in order to learn good models which

is then used to eliminate features and compute policies. The difference between this work and ours is that they eliminate features which can be ignored in every state for the purpose of learning policies while we consider features for Q-function approximation.

### 2.5.5 Neural Network

Neural networks, or artificial neural networks, consist of layers of nodes. Nodes, or artificial neurons, are connected to other nodes by directed edges which are weighted. The weights of edges are parameters of a neural network which shall be trained or learned. The input to a node in the input layer can be features such as literals (e.g., the input is 1 if the literal is true). The input to a node in subsequent layers, the hidden layers and the output layer, is the weighted sum of the outputs of nodes with directed edges to the node. The output of a node is determined by its input and an activation function. The activation function can be non-linear which introduces non-linearity to the function approximation. Feature discovery using algorithms such as *iFDD+* is not required as complex features are learned during training and are represented by the hidden layers.

**Comparison with linear function approximations.** While linear function approximations have a lower representational capacity than neural networks, they lead to a lower sample complexity since there are fewer parameters (or weights) to train. High capacity representations are better able to capture the fine granularity required to approximate the optimal value function or Q-function but have larger time and sample complexities [147]. One advantage of neural networks is that they do not require feature engineering. Instead, complex features are constructed in the hidden layers. Nevertheless, linear function approximations with feature discovery algorithms allow better control and understanding of the feature space. In our domains of interest (see Section 2.8), our empirical results show that linear function approximations are adequate in approximating the Q-functions.

A major disadvantage of neural networks is that they have poor interpretability—the ability to provide explanations in understandable terms to a human [40]—which makes the function approximation difficult to understand and the

learning process a black box. In contrast, symbolic representations such as decision trees and linear function approximations are easily interpretable [65]. One approach to interpret neural networks is to approximate them with decision trees [65]. The most clear and explicit explanations are logical decision rules which use knowledge related to the learning problem (e.g., state predicates) [199]. However, while this approach allows us to understand the learned policy, it does not shed light on the learning process such as the failure of convergence to a (near-)optimal policy. This aspect is crucial in our work; for example, we discuss the shortcomings of RRL in Section 3.4.1 which benefits from the human-interpretable representation of linear function approximations.

## 2.6 Relational Reinforcement Learning

The choice of a problem representation (see Section 2.1) is crucial in solving the problem effectively. While propositional representations like MDPs are commonly used, the size of the state space grows exponentially in the number of objects, making many RL and planning methods intractable in large scale problems. State abstraction techniques can be used to reduce the size of the state space but this might not necessarily lead to generalisation over all problems of a domain. Another approach is to use more expressive representations to represent the problem such as an RMDP. RRL combines the expressiveness of first-order logic with RL by utilising relational representations rather than propositional representations. Many RRL work use ILP to learn relational concepts which are inherent in relational problems from observations. One such concept is a relational function approximation.

**Transfer learning.** Solving large scale problems are challenging due to the extensive amount of exploration required before reaching some meaningful states where rewards are observed [192]. In transfer learning, the knowledge learned in a problem (**source problem**) is used to accelerate the learning process in another problem (**target problem**). RRL methods which learn a relational representation of the value function or Q-function enable transfer learning between problems of an arbitrary number of objects by transferring the value function or Q-function without the need of a mapping between problems [174].

In the remainder of this section, we review existing work on RRL. We also briefly cover planning methods for relational problems; any method which requires the true model is considered a planning method. For an in-depth discussion of RRL work, readers can refer to the survey papers [171, 180, 181, 190].

### 2.6.1 Model-Free RRL

Q-RRL [47] is the seminal RRL algorithm which uses ILP to approximate the Q-function with relational decision trees. The decision tree partitions the state space into regions or abstract states, each of which has a real-value representing the Q-value. This type of state abstraction is  $Q^*$ -irrelevance [100, 185]. In essence, Q-RRL performs state and action abstraction via a relational function approximation. An internal node is a query which is a conjunction of symbolic literals. Variables in a query are either substituted with objects in the terms of the action or a goal predicate or are existentially quantified; a query is true if it has at least one grounding which is true.

A decision tree is sensitive to the order of node splitting and is ill-suited in online RL where later observations yield new and often more meaningful information. The performance of RRL methods based on relational decision trees rely on the correct split of nodes [43]. Q-RRL deals with this issue by constructing a new decision tree after each episode from an ever increasing number of observations accumulated over episodes. This incurs high computational and memory costs. TG [44] overcomes this drawback by using an incremental decision tree learner in Q-RRL. Subsequent work extend or adopt the concepts of Q-RRL or TG and typically use relational decision trees to approximate the value function or Q-function.

RIB [43] uses instance based regression which has better learning stability than TG. It needs to maintain the observed instances and requires the specification of a domain-specific distance function between state-action pairs for nearest neighbour prediction for Q-values. Rodrigues, Gérard, and Rouveïrol [144] investigate the impact of different TD learning methods on RIB in online RL and found that the computation time can be reduced. Q-RRL is used with Gaussian processes for regression in place of a relational decision tree in [45]; Gaussian processes give the uncertainty of its estimated Q-values which can provide guided exploration. TRENDI

[42] combines TG and RIB to obtain their respective strengths. A relational decision tree is constructed incrementally using TG while RIB is used at each leaf node.

Some work looks at learning efficiently in problems with hierarchical structures. [30] learns relational options using Q-RRL to classify if a ground action is an action for the policy of a relational option. A hierarchical RRL algorithm [145] approximates the values of states conditioned on the task and its associated subtask with first-order rules where the task hierarchy is known.

In the face of concept drift, an unannounced change over time in a problem, the relational decision trees are restructured with four operations added to TG [141]. This requires statistics to be stored for not just leaf nodes but for every node. Similarly, [34] extends the UTree algorithm [110] to construct relational decision trees and perform online tree restructuring. We address transfer learning between different classes of problems and between different simulated environments which draws similarities with concept drift. Cross-domain transfer [172, 173] is a more general form of generalisation than ours (and other work mentioned here) but the required generality of the transferred knowledge causes loss of useful information.

Empirical results in [41, 45, 47] are limited to simplified domains, failing to demonstrate the applicability of Q-RRL and its extensions to approximate more complex value functions [192]. Value functions approximated with relational decision trees are piecewise constant which can be inappropriate for some relational problems [145]. Highly relational problems require complex patterns in their value functions and other approximators such as instance-based representations, kernels, or first-order features are more suitable as they have a lower granularity [181]. In addition, this work uses handcrafted features to facilitate learning. Our work does not rely on handcrafted features.

We use **first-order features**, or conjunctions of symbolic literals, as features in a linear function approximation (see Section 3.3). There are a small number of work which use such approximators rather than a relational decision tree. Walker [184] uses an ensemble of predictors to approximate the Q-function for each action. Each predictor is a set of weighted randomly generated conjunctive first-order features. Conjunctive features are randomly sampled from the feature space and added to a predictor if it covers a significant percentage of the training data which are tuples



$(s, a, Q(s, a))$ . Our work learns features online where conjunctive first-order features are added to reduce the TD errors (see Section 3.1.1). We use an ensemble of approximations to learn at different abstraction levels and from different reward signals (see Sections 3.5.2 and 4.4.2). Wu and Givan [192] use a beam search to learn first-order features which correlate well to the Bellman error of value functions. The weights of new features are approximated with a trajectory-based approximate value iteration approach given observed state trajectories. The first-order features are real-valued and consist of existential variables and at most one **free variable** per feature. The value of a feature with one free variable is the number of groundings that satisfy the feature normalised by the maximum value that the feature can take (e.g., the total number of objects in the problem). This has a higher representational capacity than binary first-order features considered in our work. Since features are binary, we treat existential variables and free variables as the same (and are collectively referred to as free variables in our work). In spite of this, there remains several differences between our work and [192]. First, we address online feature discovery and online RRL while they use supervised learning. Second, they consider problems which are described in PPDDL and give positive rewards for reaching the goal states and zero rewards otherwise. We use RDDDL to describe our problems which have additive rewards. Differences between PPDDL and RDDDL (see Section 2.2.3) meant that the types of problems they address are different from ours.

Some work requires extensive domain or background knowledge to define specific representations to facilitate efficient learning while we use only trivial domain knowledge to ground the Q-function approximation (see Section 3.4). Morales [113] defines abstract actions (r-actions) and abstract states (r-states) in a relational representation, then uses a modified Q-learning to learn policies based on the induced relational abstract state-action space. Some domain knowledge is required to define the r-states. In [147], the values and structure of a relational naive Bayes net which approximates the value function are learned. A distance metric is required to generalise over handcrafted non-binary first-order features. In [94], an agent learns in two representations: the problem space which is Markovian and the agent space which can be non-Markovian and consists of sensory measurements which are common in and hold the same semantics for all problems. Shaping rewards defined over

the agent space are learned and can be applied to accelerate learning in a target problem. However, specifying the agent space is a difficult design problem which requires considerable domain knowledge. In a later work [95], options in the agent space are learned and transferred. [156] is a case-based learning approach which requires state similarity to be quantified by the Euclidean distance and states are represented by hand-coded real-valued state variables. The Q-values are the sum of the Q-values of matching cases weighted by how similar these cases are to the state of interest. Instead of learning a relational abstraction of the value function, [179] constructs a relational abstraction of the underlying RMDP with the use of background knowledge, then solves this abstract MDP with a model-free RL method.

## 2.6.2 Model-Based RRL

Model-based RRL methods learn relational models and use planning techniques. MARLIE [31] learns a relational transition and reward function online. Both functions are represented by a set of relational probability trees. Expert knowledge on the dependence of random variables on other random variables is required to avoid learning the structure of the transition function. The learned model is used to provide better estimates of the Q-values by looking some steps ahead using a sampling method. In [130], high-level relations are learned from real-valued, multidimensional attributes of objects. The relations are represented by decision trees where a set of decision trees form an action model. The high number of permutations of object attributes can increase the sample complexity of RL algorithms. This is resolved by pruning irrelevant permutations.

REX [99] performs guided exploration to less visited state-action pairs, extending the work of [19, 84] to relational visit counts. REX uses the relational planner from [98] to sample actions based on the predicted beliefs over states. The beliefs are computed with approximate Bayesian inference using noisy indeterministic deictic (NID) rules learned by the model learner from [135]. [106] extends REX with active learning and model learning. When the planner fails to find a plan with an expected value larger than the minimum expected value, demonstration from a teacher who reveals the optimal action is requested. [106] uses the model learner from [108], which is capable of learning exogenous effects, in place of the model learner from

[135]. **REX-D** [107] extends **REX** with the option of requesting expert demonstrations on unknown parts of the model so that the model learner can improve the learned model.

It is well-known that model-based RL methods introduce an additional source of error due to its model errors which affects its asymptotic performance [20, 36, 51]. We adapt the approach in [157] to combine **UCT** with model-free **RRL** to provide better quality estimates of the Q-values (see Section 4.3.2) rather than to use planning methods to select actions directly. This reduces the dependence on learning accurate models which can be harder than learning Q-functions. We do not compare our work with the aforementioned model-based **RRL** methods as their main contributions are relational model learning while we use an existing model learner, **RLFIT** [108]. Instead, our contribution is the integration of model-based and model-free methods to reduce the sample complexity (see Section 4.4.2) and solve more complex classes of problems (see Section 5.4.3).

### 2.6.3 Planning Methods

Large scale planning problems can be solved efficiently by exploiting first-order representations such as **RMDPs** [180, 187]. Symbolic dynamic programming (**SDP**) solves first-order MDPs specified in situation calculus [16]. It performs goal regression to produce a symbolic description of the value function. While the representational capacity used in **SDP** exceeds that in **Q-RRL**, this comes at the cost of computationally expensive operations due to the exponential growth of logical formulas [171, 181]. Approximate methods [49, 150] overcome this issue by approximating the optimal value function rather than an exact representation. Another approach to make **SDP** more computationally efficient is to use model checking reduction on generalised first-order decision diagrams [79].

In [64], MDPs are solved with linear programming to learn a generalised, class-based value function which is a linear sum of local value functions for objects. A decision tree is learned to classify the classes of objects such that objects of the same class have similar value functions. A major assumption made is that the relations between objects do not change over time. This is false in many domains, thus limiting its applicability. [109] solves small scale MDPs with a planner and

obtains their value functions. ILP is then used to learn a relational decision tree which is a generalisation of the value functions. [160] maps actions in a plan to action operators, applies the action operators to the initial abstract state to obtain a trajectory of abstract states and action operators, and lastly converts the trajectory to a generalised plan with non-nested loops. However, this approach is limited to deterministic planning problems .

Most deep RL methods such as [112, 154] use convolutional neural networks (CNNs) to achieve generalisation by quantifying state similarities with the differences in pixels. However, such a measure of state similarity is not applicable to problems with state spaces which cannot be represented partially or entirely with images. We are interested in problems where states are represented by state predicates and this precludes many deep RL methods. Moreover, CNNs do not perform well in relational problems as they are ill-suited to represent relations between objects [171].

Neural networks of various architectures have been used to represent policies for relational problems. Garg, Bajpai, and Mausam [53] combine graph neural networks (GNNs) and deep RL to learn policies for RDDDL problems. GNNs represent relations between entities, and therefore are suitable for learning in first-order or object-oriented environments. Architectural inductive biases are required, making the architecture of the GNN domain-specific but this can be automatically determined from the RDDDL domain file. Instead of GNNs, an attention mechanism which has parallels with GNNs is used in [196]. The architectural inductive bias for each domain is defined by human experts. [71] evaluates three classes of deep neural network architectures. The training data is generated using imitation learning such that a network learns policies which imitate the action choices of planners. **ASNeTs** [177] is a neural network with an alternating sequence of action layers and proposition layers. The architecture is determined from the relations between actions and propositions (or state predicates) which are provided by the PPDDL domain and problem files. **ASNeTs** can be trained with imitation learning as well. Unlike our work, the aforementioned work require the true model. In [82], a densely connected neural network which functions as a generalised heuristic network is learned without requiring the true model. However, heuristic guides search in planning and does not

have the same role as Q-values which generate policies and have a direct impact on performance.

In Section 2.5.5, we compared linear function approximations, which is used in our work, with neural networks. We argued that RL methods based on neural networks as function approximators suffer from poor interpretability of the learned knowledge. In this dissertation, we investigate the types of generalised knowledge which can be learned (see Chapter 4) and the interpretability of these learned knowledge is crucial in discussing the soundness of our approaches. Indeed, Doshi-Velez and Kim [40] posit that interpretability is used to establish safety, reliability, robustness, and trust in a learning agent. Furthermore, interpretability is required in applications where errors can cause catastrophic results; interpretability makes potential failures easier to detect and helps in finding solutions to these failures [199].

## 2.7 Additional Related Work

We discussed related work in earlier parts of this chapter. Here, we discuss remaining related work which are divided into two parts.

### 2.7.1 Temporal Considerations

In Chapter 5, we propose a class of problems involving time-bounded goals (TGs), or goals which can only be achieved within a time window, and dynamic objects, or objects which are added to or removed from  $\mathcal{O}$  at any time step.

Boyan and Littman [17] propose the time-dependent MDP (TMDP) which considers stochastic time-dependent action durations that can be relative or absolute. Liu and Sukhatme [104] propose the time-varying MDP (TVMDP) where the transition function is time-varying. TMDP and TVMDP do not consider TGs. In this aspect, the time-varying SMDP (TV-SMDP) proposed by Duckworth, Lacerda, and Hawes [46] is the most similar to our work. TV-SMDP models problems with TGs specified in co-safe linear temporal logic (csLTL). Gaussian Process regression is used to learn the effects of latent environment variables on action durations, and UCT is used to propagate the predicted time in order to increase the probability of

satisfaction of csLTL within the time bound. Lacerda, Parker, and Hawes [97] consider a planning problem with a hard goal and multiple soft goals. The hard goal has a time bound which is defined in csLTL. The objective is to achieve the hard goal and maximise the utility for achieving the soft goals. These two approaches consider time bounds with end time only (i.e., goals with deadline) while we consider time bounds with start and end times (see Section 5.3.2).

The aforementioned work are planning algorithms. For the learning problem, Bradtke and Duff [18] adapt RL methods, TD(0), Q-Learning, real-time dynamic programming (RTDP), and adaptive RTDP, for SMDPs. On the other hand, Ornik and Topcu [127] propose a model-based RL method to solve TVMDPs. It learns a time-varying transition probability function that is maximally likely given a history of observations and a bound on the rate of change of the transition probabilities. This model is used to generate a policy which maximises the expected rewards (exploitation) and minimises the model uncertainty (exploration). Key differences between our work and this work are that we consider dynamic objects and initially unknown goals while they do not.

Srivastava et al. [161] propose open-universe partially observable MDPs (OUPOMDPs) to model problems where observations yield new objects which are analogous to dynamic objects. In an OUPOMDP, sensors and actuators are expressed in first-order logic. Sensors are responsible for observations that can consist of new objects while actions correspond to actuators that can act on new objects due to observations. Our work deals with dynamic objects which can be added to or removed from  $\mathbf{O}$  and its impact on online RRL (see Section 5.4) while [161] addresses problem representation for planning purposes.

### 2.7.2 Multi-Agent Coordination

Some problems require the coordination of multiple agents to achieve **joint goals** (i.e., goals that require multiple agents to achieve). A problem which involves more than one agent is a **multi-agent problem**. Our work only considers joint goals which require the temporally-coordinated execution of actions by some agents where each agent executes only one action. This allows us to decompose a multi-agent problem to single-agent problems. Our approach shall be presented in Section 5.7.

Schillinger, Bürger, and Dimarogonas [152] address the cooperation of multiple robots to achieve a global goal by executing their options concurrently while we are interested in multiple agents coordinating by executing their respective actions at the same time. Their work decomposes a global goal specified in syntactically co-safe Linear Temporal Logic (scLTL) into subgoals, then auctions options which achieve the subgoals. [125] auctions goals with time bounds which can be overlapping. Goals are known initially or at any time. Each robot maintains a simple temporal network (STN) which keeps track of its commitments to goals.<sup>9</sup> A bid is computed by inserting the goal with its time bound into the STN while maintaining temporal consistency and minimising the makespan (the time the last robot finishes its final task). We have two different approaches to coordinate multiple agents which we discuss in Sections 5.7.2 and 5.7.3. In our first approach, we use a temporal planner (e.g., OPTIC [12]) to allocate TGs; depending on the choice of a temporal planner, it is likely to use STNs to maintain temporal consistency. In our second approach, we use an auction algorithm where UCT estimates the cost of a bid given the current knowledge learned. [83] uses UCT to allocate goals with time bounds, possibly overlapping, to robots. It formulates a multi-agent goal allocation problem as a search tree. However, [83] does not consider multi-agent coordination as each goal requires only one robot to achieve. [198] coordinates robots to avoid conflicts and achieve synergy such that each robot can complete its individual goal. An iterative interdependent planning procedure generates a plan for each robot with increasing consideration to the plans of other robots over each iteration; this coordinates agents throughout the entire plan. While we only consider coordination at a time step to achieve a joint goal, [198] does not consider joint goals nor coordinated actions.

The aforementioned work address the planning problem and require the true model while our work learns policies for the single-agent problems without the need of the true model. In this aspect, work which combine hierarchical decomposition, planning, and RL [61, 70, 91, 146, 193] are more similar to our work. This work uses planners to decompose tasks into subtasks, then uses RL to learn policies for the subtasks. However, they address single-agent planning problems and use

---

<sup>9</sup>A STN is a graph which represents temporal constraints. Nodes represent events and edges represent temporal constraints between two events.

propositional representations for RL.

## 2.8 Benchmark Domains

We describe the domains and problems which are used in our empirical experiments. All domains and problems are written in RDDL. In each case, we specify the set of types  $\mathcal{C}$ , the set of symbolic state predicates  $\mathcal{P}$ , the set of symbolic actions  $\mathcal{A}$ , the parameterised reward function  $\mathcal{R}$ , and the time horizon  $H$ .

### 2.8.1 IPPC Benchmark Domains

We consider benchmark domains used in recent IPPCs [62, 148]. These domains have problems which are numbered from 1 to 10; a larger number denotes a problem with a larger scale. `Domain#` denotes a problem numbered `#` for `Domain`.

#### Academic Advising

In `Academic Advising` [63], a student has to pass some required courses. The set of types is  $\mathcal{C} = \{course\}$ . The symbolic state predicates  $\mathcal{P}$  are <sup>10</sup>:

- `passed(COURSE)`: a course `COURSE` is passed;
- `taken(COURSE)`: a course `COURSE` has been taken before;
- `PREREQ(COURSE1, COURSE2)`: the course `COURSE1` is a prerequisite of the course `COURSE2`;
- `PROGRAM_REQUIREMENT(COURSE)`: a course `COURSE` needs to be passed (i.e., it is a goal).

The symbolic action `takeCourse(COURSE)` takes the course `COURSE`. The passing rate of a course depends on the number of prerequisites the student has passed:

$$\text{Pass Probability} = \begin{cases} 1 & \text{if no prerequisites} \\ \frac{\text{Num. of prerequisites passed}}{\text{Num. of prerequisites}} & \text{otherwise} \end{cases} \quad (2.30)$$

This is a minor modification of the original formulation which sets the probability to 0.8 if there is no prerequisites and has an additional term of +1 in the denominator

<sup>10</sup>Names of fluents (non-fluents) are in lowercase (uppercase) letters.



of the fraction. Our formulation is such that the passing probability is 1 if all prerequisites (if any) of a course is passed. The immediate reward is additive over the following components:

- $-1$  for taking a course or  $-3$  for retaking a failed course;
- $-5$  if any required course has not been passed;
- $5$  for passing a required course.

**Solving the problem.** A student should take a course if all of its prerequisites are passed and the course is a required course, a prerequisite of a required course, a prerequisite of a prerequisite of a required course, and so on and so forth.<sup>11</sup>

We use `Academic Advising 3 (AA3)` and `Academic Advising 5 (AA5)` and the size of their state-action spaces are  $2^{30} \times 16$  and  $2^{40} \times 21$ , respectively. The time horizon is 40 for AA3 and AA5. The objects in AA3 are 15 courses of which four are required courses. The objects in AA5 are 20 courses of which eight are required courses. There are no randomised problems for this domain.

## Recon

In Recon [148], an agent moves in a grid environment where there is a base, hazard, and objects. Tools can be damaged if the agent is at or adjacent to a hazard and this reduces the probability of getting a good reading from the damaged tool. The agent can repair a tool at the base. A goal is to take a good picture of an object which is only possible if the camera tool is not damaged and water and life has been detected on the object beforehand with the water tool and life tool, respectively. The RDDDL domain file is shown in Section D.1 of Appendix D. The set of types is  $\mathcal{C} = \{agent, tool, obj, wp\}$ . The symbolic state predicates  $\mathcal{P}$  are:

- `damaged(TOOL)`: the tool *TOOL* is damaged;
- `waterChecked(OBJ)`: water is checked for at object *OBJ*;
- `waterDetected(OBJ)`: water is detected at object *OBJ*;
- `lifeChecked(OBJ)`: life is checked for at object *OBJ*;
- `lifeChecked2(OBJ)`: life is checked for at object *OBJ* (true after

---

<sup>11</sup>This is a high-level textual description of a policy which solves a problem. It might not be the optimal policy which depends on various factors such as the discount factor  $\gamma$  and the metric that is maximised (e.g., total undiscounted reward).

- `lifeChecked(OBJ)` is true);
- `lifeDetected(OBJ)`: life is detected at object  $OBJ$ ;
- `pictureTaken(OBJ)`: picture of object  $OBJ$  is taken;
- `agentAt(AGENT, WP)`: the agent  $AGENT$  is at location  $WP$ ;
- `ADJACENT(WP1, WP2)`: the location  $WP_1$  is adjacent to the location  $WP_2$ ;
- `BASE(WP)`: the base is at location  $WP$ ;
- `HAZARD(WP)`: a hazard is at location  $WP$ ;
- `OBJECT_AT(OBJ, WP)`: the object  $OBJ$  is at location  $WP$ ;
- `CAMERA_TOOL(TOOL)`: the tool  $TOOL$  is a camera tool;
- `LIFE_TOOL(TOOL)`: the tool  $TOOL$  is a life tool;
- `WATER_TOOL(TOOL)`: the tool  $TOOL$  is a water tool.

The symbolic actions  $\mathcal{A}$  are:

- `move(AGENT, WP)`: the agent  $AGENT$  moves to the location  $WP$ ;
- `useToolOn(AGENT, TOOL, OBJ)`: the agent  $AGENT$  uses the tool  $TOOL$  on the object  $OBJ$ ;
- `repair(AGENT, TOOL)`: the agent  $AGENT$  repairs the tool  $TOOL$ .

We replaced the actions `up`, `down`, `left`, and `right` in the original formulation of the domain with `move(AGENT, WP)`. This gives different ground actions for moving to each grid position. We do away with the action abstraction in the original formulation where all `move(AGENT, WP)` actions are abstracted to only four actions as this can complicate learning. The immediate reward is additive over the following components:

- $-1$  for executing an action;
- $20$  for taking a good picture of an object or  $-20$  for taking a bad picture of an object.

**Solving the problem.** The agent should avoid hazards when moving to objects. This is sometimes not possible if objects are at or adjacent to hazards. If the agent needs to use a tool which is damaged, the agent should return to the base to repair it first. To take a good picture of an object, the agent should use the water tool, followed by the life tool, and lastly, the camera tool.

We use `Recon 3 (RC3)` and `Recon 6 (RC6)` and the size of their state-action spaces

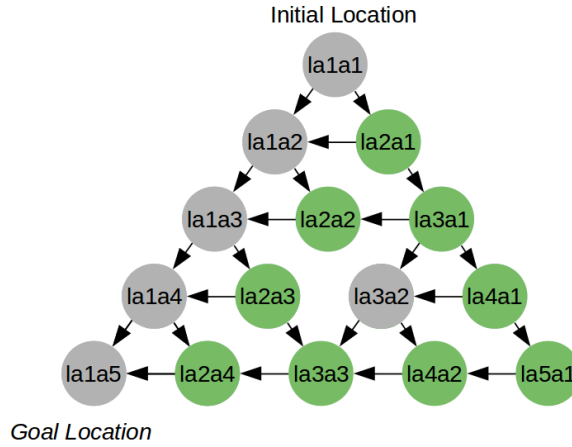


Figure 2.7: A state for the problem TT3 of the Triangle Tireworld domain. A green circle denotes a location with a spare tire and an arrow denotes the direction of traversal allowed.

are  $2^{42} \times 28$  and  $2^{55} \times 38$ , respectively. The time horizons are 40 for RC3 and 50 for RC6. The number of objects in RC3, with their types in parentheses, are: nine locations (*wp*) for a  $3 \times 3$  grid environment, five objects (*obj*), one agent (*agent*), and three tools (*tool*). The objects in RC6 are 16 locations (*wp*) for a  $4 \times 4$  grid environment, six objects (*obj*), one agent (*agent*), and three tools (*tool*). The problems are randomised by randomising the locations of objects ( $\text{OBJECT\_AT}(\text{OBJ}, \text{WP})$ ), hazards ( $\text{HAZARD}(\text{WP})$ ), and base ( $\text{BASE}(\text{WP})$ ).

### Triangle Tireworld

In Triangle Tireworld [103], a vehicle moves in a unidirectional environment to reach a goal location. The problem TT3 is illustrated in Figure 2.7. There are no spares along the shortest path from the initial location *la1a1* to the goal location *la1a5*. On the other hand, there is a spare in every location along the longest path. There is a probability of 0.5 of getting a flat tire when moving. The tire needs to be replaced with a spare tire before the vehicle can move; if there isn't one, a dead end is reached. The vehicle can load a spare tire if there is one at its current location. Due to the directed connectivity of locations, if the vehicle chooses a shorter path, the irreversible change to the state can cause an unavoidable dead end. The immediate reward is  $-1$  in every time step that the goal state is not reached, 100 for reaching the goal state, and  $-100$  for reaching a dead end. The set of types is  $\mathcal{C} = \{\text{wp}\}$ . The symbolic state predicates  $\mathcal{P}$  are:

- `vehicle_at(WP)`: the vehicle is at the location  $WP$ ;
- `spare_in(WP)`: a spare tire is at the location  $WP$ ;
- `not_flattire`: the vehicle does not have a flat tire;
- `hasspare`: the vehicle has a spare;
- `goal_reward_received`: the goal state is reached;
- `ROAD(WP1, WP2)`: the vehicle is allowed to move from the location  $WP_1$  to the location  $WP_2$ ;
- `GOAL_LOCATION(WP)`: the goal is for vehicle to be at the location  $WP$ .

The symbolic actions  $\mathcal{A}$  are:

- `move_vehicle(WP1, WP2)`: the vehicle moves from the location  $WP_1$  to the location  $WP_2$ ;
- `loadtire(WP)`: the vehicle loads a tire if there is one at the location  $WP$ ;
- `changetire`: the tire is changed if the vehicle has a spare tire.

**Solving the problem.** To reach the goal location with certainty, the vehicle should only move to a location with a spare tire and leads to at least one location with a spare tire also (unless the location is the goal location). The vehicle should load a spare tire when it doesn't have one and has the opportunity to do so.

We use `Triangle Tireworld 3 (TT3)` and `Triangle Tireworld 6 (TT6)` and the size of their state-action spaces are  $2^{33} \times 242$  and  $2^{59} \times 814$ , respectively. The time horizon is 40 for TT3 and TT6. The objects in TT3 are 15 locations (*location*). The objects in TT6 are 28 locations (*location*). There are no randomised problems for this domain.

## 2.8.2 Robotic Domains

We introduce three new robotic domains in this dissertation which shall be described next.

### Robot Fetch

`Robot Fetch` is a deterministic domain where a mobile robot is tasked with placing objects at their respective goal locations. The robot can move between any two locations directly and can only hold one item at any time. Each location can have

at most one item. Goals could be interdependent as an item needs to be removed from a location before another item can be placed there. The RDDL domain file is shown in Section D.2 of Appendix D. The set of types is  $\mathcal{C} = \{robot, wp, obj\}$ . The symbolic state predicates  $\mathcal{P}$  are:

- `robot_at(ROBOT, WP)`: the robot *ROBOT* is at the location *WP*;
- `localised(ROBOT)`: the robot *ROBOT* is localised;
- `emptyhand(ROBOT)`: the robot *ROBOT* is not holding any items;
- `holding(ROBOT, OBJ)`: the robot *ROBOT* is holding the item *OBJ*;
- `object_at(OBJ, WP)`: the item *OBJ* is at the location *WP*;
- `OBJECT_GOAL(OBJ, WP)`: the goal is to place the item *OBJ* at the location *WP*.

The symbolic actions  $\mathcal{A}$  are:

- `move(ROBOT, WP)`: the robot *ROBOT* moves to the location *WP*;
- `localise(ROBOT)`: the robot *ROBOT* localises;
- `pick_up(ROBOT, OBJ, WP)`: the robot *ROBOT* picks up the item *OBJ* from the location *WP*;
- `put_down(ROBOT, OBJ, WP)`: the robot *ROBOT* puts down the item *OBJ* at the location *WP*.

The immediate reward is additive over the following components:

- $-1$  for executing an action;
- $20$  for placing an item at its goal location;
- $-20$  for picking up an item from its goal location.

**Solving the problem.** An object should be moved to its goal location if no other objects are there. If this is not possible, then the robot should move any object to an empty location. It should not move an object from its goal location.

The small and large scale problems are denoted by **RF1** and **RF2** and the size of their state-action spaces are  $2^{28} \times 37$  and  $2^{84} \times 132$ , respectively. The time horizons are 30 for **RF1** and 40 for **RF2**. The objects in **RF1** are one robot (*robot*), five locations (*wp*), and three items (*obj*). The objects in **RF2** are one robot (*robot*), ten locations (*wp*), and six items (*obj*). The problems are randomised by randomising the initial locations of items (`object_at(OBJ, WP)`) and their goal locations

(OBJECT\_GOAL(*OBJ*, *WP*)).

## Robot Inspection

In **Robot Inspection**, a mobile robot needs to find objects by surveying a location where the objects are at before it can inspect them. The goals are to transmit information on inspected objects at the communication tower. The robot can move between any two locations directly and there is a probability of 0.08 that it is low on energy after moving. It needs to return to the docking station immediately to recharge, else it is stranded and a dead end is reached. Its camera can also lose calibration during inspection with a probability of 0.15 which reduces the success rate of surveying to 0.2 and inspection to 0.9 (the state is unaffected if these actions fail). The robot can calibrate its camera at the docking station which restores the success rate back to 1. The RDDL domain file is shown in Section D.3 of Appendix D. The set of types is  $\mathcal{C} = \{robot, wp, obj\}$ . The symbolic state predicates  $\mathcal{P}$  are:

- `robot_at(ROBOT, WP)`: the robot *ROBOT* is at the location *WP*;
- `docked(ROBOT)`: the robot *ROBOT* is docked at the docking station;
- `undocked(ROBOT)`: the robot *ROBOT* is undocked;
- `localised(ROBOT)`: the robot *ROBOT* is localised;
- `camera_calibrated(ROBOT)`: the camera of the robot *ROBOT* is calibrated;
- `object_found(OBJ)`: the object *OBJ* is found;
- `object_inspected(OBJ)`: the object *OBJ* is inspected;
- `object_info_received(OBJ)`: information on the object *OBJ* has been transmitted at the communication tower;
- `has_energy(ROBOT)`: the robot *ROBOT* has energy remaining;
- `low_energy(ROBOT)`: the robot *ROBOT* is low on energy;
- `reward_received(OBJ)`: reward for transmitting information on the object *OBJ* has been received (this is to ensure a one-time reward is received for each goal);
- `DOCK_AT(WP)`: the docking station is at the location *WP*;
- `OBJECT_AT(OBJ, WP)`: the object *OBJ* is at the location *WP*;
- `COMM_TOWER_AT(WP)`: the communication tower is at the location *WP*.

The symbolic actions  $\mathcal{A}$  are:

- $\text{move}(\text{ROBOT}, \text{WP})$ : the robot  $\text{ROBOT}$  moves to the location  $\text{WP}$ ;
- $\text{localise}(\text{ROBOT})$ : the robot  $\text{ROBOT}$  localises;
- $\text{dock}(\text{ROBOT})$ : the robot  $\text{ROBOT}$  docks;
- $\text{undock}(\text{ROBOT})$ : the robot  $\text{ROBOT}$  undocks;
- $\text{survey}(\text{ROBOT}, \text{WP})$ : the robot  $\text{ROBOT}$  surveys the location  $\text{WP}$ ;
- $\text{inspect\_object}(\text{ROBOT}, \text{OBJ})$ : the robot  $\text{ROBOT}$  inspects the object  $\text{OBJ}$ ;
- $\text{transmit\_info}(\text{ROBOT})$ : the robot  $\text{ROBOT}$  transmits information about any inspected object which has not had its information transmitted yet;
- $\text{calibrate\_camera}(\text{ROBOT})$ : the robot  $\text{ROBOT}$  calibrates its camera.

The immediate reward is additive over the following components:

- $-1$  for executing an action;
- $20$  for transmitting yet-to-received information on each inspected object at the communication tower.

For example, if  $N$  objects are inspected before transmitting them, the immediate reward is  $20N - 1$ .

**Solving the problem.** The robot can survey all locations, which does not cause the camera to lose calibration, before inspecting the objects since inspection has a high probability of success even with an uncalibrated camera. Once all objects are inspected, the robot should move to the communication tower to transmit. Alternatively, the robot can survey each location, inspect the objects, then transmit at the communication tower. If the camera loses calibration, the robot should return to the docking station to calibrate it if the camera is still required. Also, if the robot is low on energy, it should move to the docking station and dock immediately.

The small and large scale problems are denoted by **RI1** and **RI2** and the size of their state-action spaces are  $2^{23} \times 27$  and  $2^{29} \times 36$ , respectively. The time horizon is 40 for **RI1** and **RI2**. The objects in **RI1** are one robot (*robot*), five locations (*wp*), and three objects (*obj*). The objects in **RI2** are one robot (*robot*), ten locations (*wp*), and six objects (*obj*). The problems are randomised by randomising the locations of objects ( $\text{OBJECT\_AT}(\text{OBJ}, \text{WP})$ ), docking station ( $\text{DOCK\_AT}(\text{OBJ}, \text{WP})$ ),

**Plan Execution**

- 1 find\_person(p1)
- 2 move(wp5, wp4)
- 3 talk(p1)
- 4 move(wp4, wp2)
- 5 pick\_up(o1, wp2)
- 6 move(wp2, wp5)
- 7 put\_down(o1, wp5)

**Description / Object Type / Objects**

- ★ TIAGo / robot / r1
- ★ Person / person / p1, p2, p3
- ★ Item / obj / o1, o2, o3, ...
- ☆ Location / wp / wp1, wp2, wp3, ...

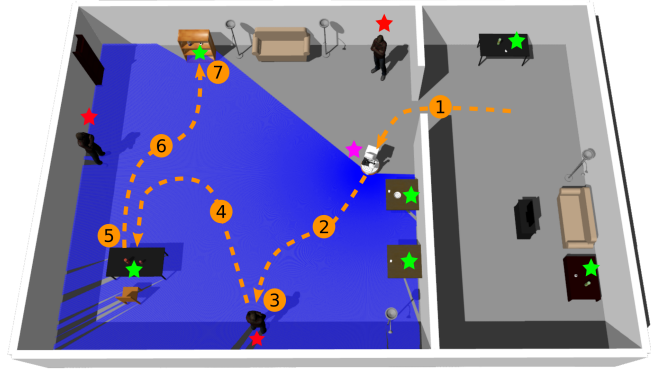


Figure 2.8: An environment in Gazebo for the Service Robot domain. A TIAGo robot fulfils a task from the person  $p1$  who requested assistance by bringing an item  $o1$  to the location  $wp5$ .

and communication tower ( $\text{COMM\_TOWER\_AT}(\text{OBJ}, \text{WP})$ ).

## Service Robot

In Service Robot, a TIAGo robot has to assist some people. A person can request assistance which is a probabilistic exogenous event. The robot does not initially know where the people are located and has to find them. It then navigates to the person and talks to the person. If the person has requested assistance, the robot will receive some tasks (or goals) of two possible types: (1) bring an item to a location, or (2) deliver an item to a person. The robot can move between any two locations directly. Figure 2.8 illustrates a simulated environment in Gazebo [89] for Service Robot. We describe this simulated environment in details in Section 5.5.1. The RDDDL domain file is shown in Section D.4 of Appendix D. The set of types is  $\mathcal{C} = \{\text{robot}, \text{wp}, \text{obj}, \text{person}\}$ . The symbolic state predicates  $\mathcal{P}$  are:

- $\text{robot\_at}(\text{ROBOT}, \text{WP})$ : the robot  $\text{ROBOT}$  is at the location  $\text{WP}$ ;
- $\text{localised}(\text{ROBOT})$ : the robot  $\text{ROBOT}$  is localised;
- $\text{emptyhand}(\text{ROBOT})$ : the robot  $\text{ROBOT}$  is not holding any item;
- $\text{holding}(\text{ROBOT}, \text{OBJ})$ : the robot  $\text{ROBOT}$  is holding the item  $\text{OBJ}$ ;
- $\text{object\_at}(\text{OBJ}, \text{WP})$ : the item  $\text{OBJ}$  is at the location  $\text{WP}$ ;
- $\text{object\_with}(\text{OBJ}, \text{PERSON})$ : the item  $\text{OBJ}$  is with the person  $\text{PERSON}$ ;
- $\text{goal\_object\_at}(\text{OBJ}, \text{WP})$ : the goal is for the item  $\text{OBJ}$  to be at the location  $\text{WP}$ ;
- $\text{goal\_object\_with}(\text{OBJ}, \text{PERSON})$ : the goal is for the item  $\text{OBJ}$  to be with



the person  $PERSON$ ;

- $person\_at(PERSON, WP)$ : the person  $PERSON$  is at the location  $WP$ ;
- $need\_assistance(PERSON)$ : the person  $PERSON$  needs assistance;
- $needed\_assistance(PERSON)$ : the person  $PERSON$  needed assistance (a person only needs assistance once);
- $reward\_received(OBJ)$ : the goal involving the item  $OBJ$  has been achieved (each item can be involved in at most one goal).

We use non-fluents to represent locations of tables and people and the tasks (goals) each person can give:

- $PROB\_NEED\_ASSISTANCE(PERSON)$ : the probability of the person  $PERSON$  needing assistance;
- $PERSON\_GOAL\_OBJECT\_AT(PERSON, OBJ, WP)$ : the person  $PERSON$  will instruct the robot to place the item  $OBJ$  at the location  $WP$ ;
- $PERSON\_GOAL\_OBJECT\_WITH(PERSON_1, OBJ, PERSON_2)$ : the person  $PERSON_1$  will instruct the robot to bring the item  $OBJ$  to the person  $PERSON_2$  where  $PERSON_1$  and  $PERSON_2$  can refer to the same person;
- $PERSON\_IS\_AT(PERSON, WP)$ : the person  $PERSON$  is at location  $WP$  (this is different from  $person\_at(PERSON, WP)$  which represents the knowledge of the robot);
- $TABLE\_AT(WP)$ : there is a table at location  $WP$ .

Except for  $TABLE\_AT(WP)$ , these non-fluents represent facts which are not made known to the robot while the fluents represent knowledge of the robot. For example, the goals a person will give is defined by the non-fluents and are unknown to the robot until it talks to the person. The symbolic actions  $\mathcal{A}$  are:

- $move(ROBOT, WP_1, WP_2)$ : the robot  $ROBOT$  moves from the location  $WP_1$  to the location  $WP_2$ ;
- $localise(ROBOT)$ : the robot  $ROBOT$  localises;
- $find\_person(ROBOT, PERSON)$ : the robot  $ROBOT$  explores the room to find the person  $PERSON$ ;
- $talk\_to\_person(ROBOT, PERSON)$ : the robot  $ROBOT$  talks to the person  $PERSON$ ;
- $pick\_up(ROBOT, OBJ)$ : the robot  $ROBOT$  picks up the item  $OBJ$ ;

- `put_down(ROBOT, OBJ)`: the robot *ROBOT* puts down the item *OBJ*;
- `take(ROBOT, OBJ, PERSON)`: the robot *ROBOT* takes the item *OBJ* from the person *PERSON*;
- `give(ROBOT, OBJ, PERSON)`: the robot *ROBOT* gives the item *OBJ* to the person *PERSON*;
- `noop`: do nothing.

The immediate reward is additive over the following components:

- $-1$  for executing an action;
- $20$  for completing a task or goal.

**Solving the problem.** If a person needs assistance, the robot should find the person, go to the person, and talk to the person. Otherwise, the robot should do nothing (i.e., `noop`) if it has no tasks to complete and is localised. If there are tasks to be completed, the robot should pick up the associated object from its current location and bring it to the specified location or person. If this person has not been found, then the robot needs to find this person.

We introduce three problems, **SR1**, **SR2**, and **SR3**, and the size of their state-action spaces are  $2^{59} \times 41$ ,  $2^{222} \times 156$ , and  $2^{222} \times 156$ , respectively. The time horizons are 20 for **SR1**, 40 for **SR2**, and 60 for **SR3**. The objects in **SR1** are three items (*obj*), one person (*person*), and five locations (*wp*). The objects in **SR2** and **SR3** are six items (*obj*), three people (*person*), and ten locations (*wp*). Each person who requires assistance gives two tasks to the robot after the robot talks to them. In **SR1**, *p1* needs assistance with a probability of 0.5. In **SR2**, *p1*, *p2*, and *p3* need assistance with a probability of 0.5, 0.3, and 0.0, respectively. In **SR3**, *p1*, *p2*, and *p3* need assistance with a probability of 0.5, 0.3, and 0.3, respectively. The scales of **SR2** and **SR3** are equal but the latter has two more goals (from *p3*) than the former. The problems are randomised by randomising the initial locations of items and the types of goals:

- `PERSON_GOAL_OBJECT_AT(p1, OBJ, WP)`: a goal from *p1* where *OBJ* and *WP* are randomised object instances;
- `PERSON_GOAL_OBJECT_WITH(p1, OBJ, p1)`: a goal from *p1* where *OBJ* is a randomised object instance;

- $\text{PERSON\_GOAL\_OBJECT\_AT}(p2, OBJ, WP)$ : a goal from  $p2$  where  $OBJ$  and  $WP$  are randomised object instances;
- $\text{PERSON\_GOAL\_OBJECT\_AT}(p2, OBJ, WP)$ : a goal from  $p2$  where  $OBJ$  and  $WP$  are randomised object instances;
- $\text{PERSON\_GOAL\_OBJECT\_WITH}(p3, OBJ, p1)$ : a goal from  $p3$  where  $OBJ$  is a randomised object instance;
- $\text{PERSON\_GOAL\_OBJECT\_WITH}(p3, OBJ, p2)$ : a goal from  $p3$  where  $OBJ$  is a randomised object instance.

The goals are randomised such that no item is involved in more than one goal. The goals from  $p3$  involve another person.

### 2.8.3 Properties of Domains

We discuss the differences between the six domains. In **Triangle Tireworld**, there is only one goal. In **Robot Inspection**, multiple goals can be achieved at once; the immediate reward can vary from  $-1$  to  $20N-1$  where  $N$  is the number of objects of type *obj*. In the remaining domains, only one goal can be achieved at a time. In **Service Robot**, goals are initially unknown and there are two types of goals (i.e.,  $\text{goal\_object\_at}(OBJ, WP)$  and  $\text{goal\_object\_with}(OBJ, PERSON)$ ). **Academic Advising** and **Robot Fetch** have goals which are interdependent. In the former, achieving some goals can increase the probability of success in achieving other goals. In the latter, there is an order to achieve the goals due to the constraint that each location can only hold one object. Furthermore, an achieved goal can be unachieved by picking up an object from its goal location. In **Triangle Tireworld** and **Recon**, actions can lead to irreversible changes to the state. In **Triangle Tireworld**, the connectivity of locations are unidirectional and spares are finite resources. In **Recon**, if a bad picture of an object is taken, the goal of taking a good picture of it can never be achieved. Remaining goals can still be achieved but the goal state cannot be reached. Lastly, there are dead ends in **Triangle Tireworld** and **Robot Inspection**.

## 2.9 Summary

Going forward, the next three chapters describe our work. We summarise how the background introduced in this chapter fits in with our work. Chapter 3 introduces our RRL method (Section 2.6) which learns a relational linear function approximation (Section 2.5.3) for the Q-function (Section 2.4.1), or first-order approximation, to solve problems represented by RMDPs (Section 2.1.3). We propose a model-based method to reduce the granularity (Section 2.5.2) of the first-order approximation; it requires a generative model such as the maximum likelihood model (Section 2.3.1).

Chapter 4 introduces different forms of generalised knowledge. We use an existing model learner to learn generative models in the form of parameterised DBNs (Section 2.3.2) written in RDDL (Section 2.2.3). We combine UCT (Section 2.4.2), which uses this generative model, with the first-order approximation. We propose a novel method to learn from observed dead ends (Section 2.1.6) in order to avoid them in the future.

Chapter 5 introduces a new class of problems which is an extension of an RMDP (Section 2.1.3) and a SMDP (Section 2.1.4). We extend the work from Chapters 3 and 4 to solve this more complex class of problems. In all three chapters, we analyse the time and space complexities (Section 2.4.3) of some of our methods and evaluate the sample complexity of our RRL method with empirical experiments for the six domains introduced in Section 2.8.

# Chapter 3

## First-Order Approximation

In this chapter, we present our online RRL method which learns an approximation of the Q-function with first-order features. This approximation, or first-order approximation, maps the state-action space of every problem of a domain to the same abstract space where states are represented by first-order features. The objective is to learn a first-order approximation which generates a policy that solves any problem of the same domain.

This chapter is organised as follows. First, we present an overview of our online RRL method in Section 3.1, bringing together the concepts from subsequent sections and showing how they fit into our RRL method. We extend an online feature discovery algorithm, iFDD+ [54], which incrementally adds conjunctive features to reduce the approximation errors of the Q-function approximation. In Section 3.3, we examine the generalisation property of first-order approximation. First-order features must be grounded to generate a policy. The efficient grounding of first-order approximation and the limitations of RRL are discussed in Section 3.4. Next, in Section 3.5, we present two methods to overcome the limitations. A first-order approximation allows transfer learning between problems regardless of the objects, initial states, and goal states. We introduce three different modes of transfer learning in Section 3.6. Lastly, the chapter concludes with empirical results for six domains in Section 3.7. Parts of the work in this chapter have been published before in [120–122].

**Algorithm 1:** Online RRL with ensemble of Q-function approximations

---

```

1 Function LQ-RRL( $P, \tilde{\mathbf{Q}}, \mathbf{w}, \Phi^c, \boldsymbol{\eta}, CK$ ):
  Input: Problem  $P = (\mathcal{C}, \mathcal{P}, \mathcal{A}, \mathcal{O}, \mathcal{T}, \mathcal{R}, s_0, H, \gamma)$ ,
           Ensemble of Approximations  $\tilde{\mathbf{Q}} = \{\tilde{Q}_1, \dots, \tilde{Q}_n\}$ ,
           Weights for ensemble  $\mathbf{w} = \{w_1, \dots, w_n\}$ ,
           Candidate features for ensemble  $\Phi^c = \{\Phi_1^c, \dots, \Phi_n^c\}$ ,
           Relevances of candidate features for ensemble  $\boldsymbol{\eta} = \{\boldsymbol{\eta}_1, \dots, \boldsymbol{\eta}_n\}$ ,
           Contextual knowledge  $CK$ 
2    $\tilde{Q}^\pi(s, a) := \sum_{\tilde{Q}_i \in \tilde{\mathbf{Q}}} w_i \tilde{Q}_i(s, a) := \sum_{\tilde{Q}_i \in \tilde{\mathbf{Q}}} w_i \boldsymbol{\theta}_i^T \Phi_i(s, a)$ 
3   for  $t = 0$  to  $H - 1$  and  $s_t$  is not terminal state do
4      $a_t = \pi(s_t)$ 
5      $s_{t+1}, r_t \leftarrow$  Execute action  $a_t$ 
6     for  $\tilde{Q}_i \in \tilde{\mathbf{Q}}$  do
7        $(\tilde{Q}_i, \Phi_i^c, \boldsymbol{\eta}_i) \leftarrow$  Update_Approximation( $\tilde{Q}_i, \mathbf{A}, \mathcal{O}, \Phi_i^c, \boldsymbol{\eta}_i, CK,$ 
8          $s_t, a_t, r_t, s_{t+1}$ )
9   return  $(\tilde{\mathbf{Q}}, \Phi^c, \boldsymbol{\eta})$ 

```

---

### 3.1 Online Relational Reinforcement Learning

Relational reinforcement learning (RRL) combines RL and relational learning to learn efficiently in relational problems. Typically, the states, actions, Q-functions, or policies are represented by first-order or relational representations which facilitate generalisation over states, actions, and goals. Our approach to RRL is to learn a first-order approximation of the Q-function which is a generalised Q-function.

#### Definition 13 (Generalised Q-function)

A Q-function is applicable in a problem  $P$  of a domain  $D$  if it maps every state and action pair  $(s, a)$  in the state-action space  $\mathbf{S}_P \times \mathbf{A}_P$  of  $P$  to a real value. A Q-function is a generalised Q-function for  $D$  if it is applicable in every problem  $P \in \mathbf{P}_D$  where  $\mathbf{P}_D$  is the set of all problems for  $D$ .

In a Q-function approximation, the state is abstracted with features. If a feature cannot be evaluated (i.e., mapped to a real value) in at least one state for a problem  $P$ , then the Q-function approximation is not applicable in  $P$ . A generalised Q-function generates a generalised policy.

#### Definition 14 (Generalised Policy)

A generalised policy  $\pi$  for a domain  $D$  is a policy which maps every state  $s \in \mathbf{S}_P$

to an action  $a \in \mathbf{A}_P$  for every problem  $P \in \mathbf{P}_D$ . A policy is generated from a Q-function such that the action selection in a state depends on the Q-values of actions in the state.

Our RRL method, denoted by LQ-RRL (RRL with linear function approximation for Q-function), learns a generalised Q-function. Our objective is to learn a generalised policy which maximises the sum of expected rewards in many (or all) problems of a domain, even in new problems different from the ones the policy is learned in. We consider problems with discrete states and actions. LQ-RRL is outlined in Algorithm 1. We provide an overview here while details on the various concepts will be deferred to later sections.

The inputs to Algorithm 1 are a problem  $P$  represented by an RMDP (the transition function  $\mathcal{T}$  is unknown), an ensemble of Q-function approximation(s) ( $\tilde{\mathcal{Q}}$ ) with at least one Q-function approximation, sets of weights ( $\mathbf{w}$ ) and candidate features ( $\Phi^c$ ) with their relevances ( $\boldsymbol{\eta}$ ) for each Q-function approximation in the ensemble, and contextual knowledge ( $CK$ ). Candidate features with their relevances are for online feature discovery. We discuss ground approximation of the Q-function in Section 3.2, first-order approximation of the Q-function in Section 3.3, ensemble of Q-function approximations in Section 3.5.2, online feature discovery in Section 3.1.1, and contextual knowledge in Section 3.4.2.

A Q-function approximation  $\tilde{Q}_i \in \tilde{\mathcal{Q}}$  can be initialised from scratch where its set of features  $\Phi_i$  is a set of base features (see Definition 12), its weights  $\boldsymbol{\theta}_i = \mathbf{0}$ , candidate features  $\Phi_i^c = \emptyset$ , and relevance of candidate features  $\boldsymbol{\eta}_i = \emptyset$ . Otherwise,  $\tilde{Q}_i$  can be learned in and transferred from a previous problem. The set of base features for a first-order approximation is determined with Algorithm 5 (see Section 3.3.1).

We use a linear function approximation to approximate the Q-function. The Q-value  $\tilde{Q}^\pi(s, a)$  estimated by an ensemble of approximations is determined from its constituent approximations (line 2). This is given by Equations 3.8 and 3.9 in Section 3.5.2 and Equation 4.7 in Section 4.4.2. The vector of real values,  $\Phi(s, a)$ , is returned by Algorithm 6 (see Section 3.4).

Lines 3 to 7 show the planning-execution-learning cycle for  $H$  time steps. To select an action using a policy  $\pi$  generated from  $\tilde{Q}^\pi$  (line 4), the Q-value of every action must be evaluated;  $\Phi_i(s_t, a)$  must be computed for every  $a \in \mathbf{A}$  and for every

**Algorithm 2:** Online feature discovery (iFDD+)

---

```

1 Function Online_Feature_Discovery( $\Phi_{active}, \mathbf{A}, \Phi^c, \boldsymbol{\eta}$ ):
  Input: Set of active features  $\Phi_{active}$ ,
           Set of actions  $\mathbf{A}$ ,
           Candidate features  $\Phi^c$ ,
           Relevances of candidate features  $\boldsymbol{\eta}$ ,
2  Generate a set of active candidate features  $\Phi_{active}^c$  from  $\Phi_{active}$ 
3  for  $\phi_i \in \Phi_{active}^c$  do
4     $\eta_i \leftarrow$  Update relevance of  $\phi_i$ 
5    Add (update)  $\phi_i$  to (in)  $\Phi^c$  and  $\eta_i$  to (in)  $\boldsymbol{\eta}$ 
6    if  $\eta_i > \xi$  then
7      for  $a \in \mathbf{A}$  do
8        if  $\phi_i$  comprises of base features of  $a$  then
9          Add  $\phi_i$  to  $\Phi_a$  and its weight to  $\boldsymbol{\theta}$ 
10         Remove  $\phi_i$  from  $\Phi^c$  and  $\eta_i$  from  $\boldsymbol{\eta}$ 
11 return ( $\Phi^c, \boldsymbol{\eta}$ )

```

---

$\tilde{Q}_i \in \tilde{Q}$ . Thus, Algorithm 6 is called  $|\mathbf{A}||\tilde{Q}|$  times in each time step. The action returned by  $\pi$ ,  $a_t$ , is executed in the current state  $s_t$  (line 5). The effects of executing  $a_t$  on the environment are observed in the form of a tuple  $(s_{t+1}, r_t)$  where  $s_{t+1}$  is the next state and  $r_t$  is the immediate reward. The observation for time step  $t$  is then  $\Xi = (s_t, a_t, s_{t+1}, r_t)$ .  $\tilde{Q}$  is updated by performing a step update for each Q-function approximation using a TD learning or TD( $\lambda$ ) method (line 7, see Algorithm 4). This is discussed in details in Section 2.4.1. Each Q-function approximation is not learned independently because the observation depends on the policy generated from  $\tilde{Q}^\pi$ . Algorithm 1 terminates after  $H$  time steps or if the terminal state is reached (line 3), and returns  $\tilde{Q}$ ,  $\Phi^c$ , and  $\boldsymbol{\eta}$  (line 8). In transfer learning, these outputs are used as inputs to another problem. We discuss transfer learning in Section 3.6.

### 3.1.1 Online Feature Discovery

Using a literal as a feature (such a feature is a base feature, see Definition 12) is often inadequate in approximating the optimal Q-function. Conjunctive features, which are conjunctions of literals, are added to  $\Phi$  to reduce the approximation errors. This is because a conjunction of literals has a lower coverage and gives a finer granularity to a Q-function approximation; a conjunctive feature encodes information that a base feature (i.e., a literal) cannot. For example, in Recon, a conjunctive feature



$\phi = \text{agentAt}(a1, wp1) \wedge \text{OBJECT\_AT}(o1, wp1)$  represents the relation that the agent  $a1$  is at the same location as the object  $o1$ . Since the agent needs to be at the same location as an object to take a good picture of it (which is the goal in **Recon**), a positive weight will be learned for  $\phi$ . The generated policy will then direct the agent to move to  $wp1$ . Adding conjunctive features to  $\Phi$  also introduces nonlinearities (of the literals) to the linear function approximation.

Since the state spaces of our domains are represented by literals, we consider binary features (i.e., true or false). Suppose that the set of literals  $\mathbf{P}^\# = \mathbf{P}^+ \cup \neg\mathbf{P}^+$  forms a set of base features. Then, the number of base features is  $2|\mathbf{P}|$  and the number of possible features is  $2^{2|\mathbf{P}|}$ . Clearly, this is not tractable as the number of features scale exponentially with the number of state predicates. Furthermore, some conjunctive features are unnecessary for the purpose of approximating the Q-function. Thus, there is a need to determine conjunctive features which are necessary and add them to  $\Phi$ . For this purpose, we implemented and extended the incremental Feature Dependency Discovery (**iFDD+**) from Geramifard, Dann, and How [54] as it has a low time complexity, scales well to large scale problems, and outperforms the state-of-the-art batch expansion method from [129]. Our work, excluding our extension for **iFDD+**, is not restricted to any particular feature discovery algorithm as long as they use features which are conjunctions of literals.

**iFDD+** is a model-free, online feature discovery algorithm which incrementally adds new features to  $\Phi$  when approximation errors persist in regions of the state space. We briefly describe **iFDD+** here. Readers can refer to [54, 55] for more details. Algorithm 2 shows the steps for **iFDD+**. The inputs are a set of features in  $\Phi$  which are active ( $\Phi_{active}$ ), the set of actions ( $\mathbf{A}$ ), a set of candidate features ( $\Phi^c$ ), and the relevances of the candidate features ( $\eta$ ).  $\Phi^c$  and  $\eta$  are initially the empty set. Algorithm 2 is called in line 5 of Algorithm 4 which updates the Q-function approximation in every time step.

A candidate feature is formed by the conjunction of two features (its **parent features**) in  $\Phi$ . A feature is active in a state  $s$  if it is true in  $s$  and is not a parent of any feature which is true in  $s$ . This is the sparse summary of active features introduced by Geramifard et al. [55]. We elaborate on this in a later part of this section. Given a set of active features, candidate features which could reduce the

approximation error are identified. They are the conjunction of every pair of active features in  $\Phi$  (line 2 of Algorithm 2). The relevance of every active candidate feature is updated (line 4) as follows:

$$\eta = \frac{\left| \sum_{i=0, \phi(s_i, a_i)=1}^t \delta_i \right|}{\sqrt{\sum_{i=0, \phi(s_i, a_i)=1}^t 1}}, \quad (3.1)$$

where  $\eta$  is the relevance of a candidate feature  $\phi$ ,  $(s_i, a_i)$  is the state-action pair in the observation at time step  $i$ ,  $t$  is the current time step, and  $\delta_i$  is the TD error at time step  $i$ . Equation 3.1 is determined from the conditions for a guaranteed rate of convergence in the error bound of the Q-function approximation. We refer readers to [56] for further details.

The relevances of candidate features are tracked over time steps. Newly generated candidate features and their relevances are added to  $\Phi^c$  and  $\eta$ , respectively, while relevances of existing candidate features are updated (line 5). A candidate feature is added to  $\Phi_a$ , the set of features for the action  $a$ , and its weight is added to  $\theta$  (line 9) if the following two conditions are satisfied: (1) the relevance of the candidate feature is greater than  $\xi$  (line 6), and (2) the candidate feature comprises of parent features which are features of  $a$  (line 8). Candidate features which are added to  $\Phi$  no longer need to be tracked and are removed from  $\Phi^c$  along with their relevances from  $\eta$  (line 10). The updated  $\Phi^c$  and  $\eta$  are returned (line 11).

iFDD+ greedily considers only features of the largest conjunction sets such that all the base features which evaluate to true are included, either by itself or as part of a conjunctive feature. Parent features are ignored because the conjunctive feature covers both of them in the abstract state. For example, consider the following three features:  $\phi_i$ ,  $\phi_j$ , and  $\phi_k = \phi_i \wedge \phi_j$ . If both  $\phi_i$  and  $\phi_j$  are true in a state  $s$ , then  $\phi_k$  must also be true. Due to the sparse summary, only  $\phi_k$  is considered active (i.e.,  $\phi_k(s, a) = 1$ ). Since  $\phi_i$  and  $\phi_j$  are parents of  $\phi_k$ , they are inactive (i.e.,  $\phi_i(s, a) = \phi_j(s, a) = 0$ ). The sparse summary reduces the time complexity as the weights of lesser features are updated and the relevances of lesser candidate features are updated. Furthermore, it separates the ambiguity as a conjunctive feature and its parent features will never map to 1 for the same state. This allows the update (or learning) of their weights to be separated. Following the same example, although  $\phi_i$

and  $\phi_j$  are true, only  $\phi_k$  is active and updated.

Adding features to  $\Phi$  reduces the granularity of a Q-function approximation  $\tilde{Q}$  as they have smaller coverage than their parent features. Since  $\Phi^c$  consists of candidate features which are the conjunction of any two features in  $\Phi$ , this in turn introduces new candidate features to  $\Phi^c$ . For each feature added to  $\Phi$ , its weight is initialised as the sum of the weights of its parent features and is added to  $\theta$  such that  $\tilde{Q}(s, a)$  remains unchanged [55]. This is in part due to the sparse summary. However, we note that this is not true and  $\tilde{Q}(s, a)$  could change if a feature  $\phi$  is a parent of more than one feature added to  $\Phi$ . When this happens, the weight of  $\phi$  is included more than once.

#### Example 4 (Change in Q-values due to Feature Discovery)

Consider an action  $a$  with three features,  $\phi_1$ ,  $\phi_2$ , and  $\phi_3$  with the respective weights  $\theta_1$ ,  $\theta_2$ , and  $\theta_3$ . In a particular state  $s$ , all three features are active and  $\tilde{Q}(s, a) = \theta_1 + \theta_2 + \theta_3$ . Suppose that a conjunctive feature  $\phi_4 = \phi_1 \wedge \phi_2$  is added to  $\Phi$  and its weight is  $\theta_4 = \theta_1 + \theta_2$ . Due to sparse summary, the active features are  $\phi_3$  and  $\phi_4$ . The Q-value after adding  $\phi_4$  is  $\tilde{Q}(s, a) = \theta_3 + \theta_4 = \theta_3 + (\theta_1 + \theta_2)$  which is the same as before. Now consider another scenario where  $\phi_4$  and a second conjunctive feature  $\phi_5 = \phi_1 \wedge \phi_3$  are added. The weight of  $\phi_5$  is  $\theta_5 = \theta_1 + \theta_3$ . The active features are now  $\phi_4$  and  $\phi_5$ . The Q-value is  $\tilde{Q}(s, a) = \theta_4 + \theta_5 = (\theta_1 + \theta_2) + (\theta_1 + \theta_3)$ . The Q-value has changed by a magnitude of  $\theta_1$  which is the weight of  $\phi_1$ .

The consequence of the change in Q-function is learning instability as the TD error could increase rather than decrease after a weight update. We remedy this by initialising the weight of a conjunctive feature  $\phi_k = \phi_i \wedge \phi_j$  as follows:

$$\theta_k = \frac{\theta_i}{N_i} + \frac{\theta_j}{N_j}, \quad (3.2)$$

where  $\theta_i$  ( $\theta_j$ ) is the weight of  $\phi_i$  ( $\phi_j$ ), and  $N_i$  ( $N_j$ ) is the number of conjunctive features added to  $\Phi$  which include  $\phi_i$  ( $\phi_j$ ) as a parent feature. If  $N_i = 1$  and  $N_j = 1$ , then the weight is initialised to the sum of the weights of its parent features as proposed by Geramifard et al. [55].

**Algorithm 3:** Adaptive online feature discovery ( $\tau$ -iFDD+)**1 Function**

Adaptive\_Online\_Feature\_Discovery( $\Phi_{active}^c, \mathbf{A}, \Phi^c, \eta, a, \xi, N_\Phi$ ):

**Input:** Active Candidate features  $\Phi_{active}^c$ ,

Set of actions  $\mathbf{A}$ ,

Candidate features  $\Phi^c$ ,

Relevances of candidate features  $\eta$ ,

Action  $a$ ,

A set of discovery thresholds for each action  $\xi$ ,

Maximum number of features that can be added  $N_\Phi$

2 Generate a set of active candidate features  $\Phi_{active}^c$  from  $\Phi_{active}$

3 **for**  $\phi_i \in \Phi_{active}^c$  **do**

4      $\eta_i \leftarrow$  Update relevance of  $\phi_i$

5     Add (update)  $\phi_i$  to (in)  $\Phi^c$  and  $\eta_i$  to (in)  $\eta$

6  $\bar{\Phi} \leftarrow$  Group  $\Phi_{active}^c$  and sort with descending relevance

7  $count = 0$

8 **for**  $\phi \in \bar{\Phi}$  **do**

9     **if**  $\eta$  of  $\phi > \xi(a)$  **then**

10          $count \leftarrow count + |\phi|$

11         **if**  $count < N_\Phi$  **then**

12             **for**  $a' \in \mathbf{A}$  **do**

13                 **for**  $\phi \in \phi$  **do**

14                     **if**  $\phi$  comprises of base features of  $a'$  **then**

15                         Add  $\phi$  to  $\Phi_{a'}$  and its weight to  $\theta$

16                         Remove  $\phi$  from  $\Phi^c$  and  $\eta_i$  from  $\eta$

17             **else**

18                  $\xi \leftarrow \eta + \epsilon$

19                 **for**  $a' \in \mathbf{A}$  **do**

20                     **if**  $\xi(a') < \xi$  **then**  $\xi(a') = \xi$

21                 **break**

22     **else**

23         **break**

24 **return** ( $\Phi^c, \eta$ )

**3.1.2 Adaptive online feature discovery**

$\xi$  is the only hyperparameter of iFDD+ and implicitly controls the rate of adding new features to  $\Phi$ . If  $\xi$  is too large, then features are added to  $\Phi$  belatedly which increases the sample complexity. If  $\xi$  is too small, many unnecessary features are added which increases the sample and time complexities. This is because adding a feature to  $\Phi$  increases the number of possible candidate features exponentially; the

relevance of every candidate feature is tracked and updated which could be computationally expensive. As noted by Geramifard et al. [55],  $\xi$  is domain-dependent and requires expert knowledge to set appropriately. This is non-trivial. For example, the domains we consider in Section 2.8 have a variety of reward functions. Furthermore, the additive nature of the reward function in RDDDL means that the range of immediate rewards is dependent on the number of objects (e.g., **Robot Inspection**, see Section 2.8.2). Hence,  $\xi$  needs to be set considering the domain and the number of objects in the problem. This potentially involves some experiments to tune  $\xi$ .

This issue motivates our adaptive variant of **iFDD+** which progressively increments  $\xi$  and places a hard constraint on the number of features which can be added to  $\Phi$  at each time step. We term our adaptive variant as  $\tau$ -**iFDD+** which is outlined in Algorithm 3. We focus on the extensions made to **iFDD+**. The additional inputs required are the action ( $a$ ) for which the update is for, a set of discovery thresholds for each action ( $\xi$ ), and the maximum number of features which can be added per time step ( $N_\Phi$ ). First, the relevances of active candidate features ( $\Phi_{active}^c$ ) are updated (lines 2 to 5). Then, features in  $\Phi_{active}^c$  are grouped together if they have the same relevance (line 6). The groups of features ( $\bar{\Phi}$ ) are then sorted with descending relevances (line 6). Each group of features  $\phi \in \bar{\Phi}$  is considered for addition to  $\Phi$  (lines 8 to 23). If the relevance of  $\phi$  exceeds the discovery threshold of  $a$  (line 9), then *count* is incremented with the number of features in  $\phi$  (line 10). If *count* is less than  $N_\Phi$ , then  $\phi$  is added to  $\Phi$ , and  $\Phi^c$  and  $\eta$  are updated accordingly (lines 11 to 16). Otherwise,  $\phi$  is not added as doing so will exceed  $N_\Phi$ , and the discovery threshold for each action is set to the relevance of  $\phi$  plus a small arbitrary amount  $\epsilon$  (lines 17 to 21). This ensures that candidate features which are not added can only be added in subsequent time steps if their relevances increase again. Since candidate features are ordered with descending relevances, remaining groups of features in  $\bar{\Phi}$  have relevances below the discovery threshold and will not be added. The discovery threshold is maintained for each action rather than a single value for all actions as in **iFDD+**. It is increased whenever the number of candidate features to be added exceeds  $N_\Phi$ . Intuitively, this implies that more observations are required to determine which candidate features are necessary.

The maximum number of features that can be added per time step is dependent

**Algorithm 4:** Update Q-function approximation

---

```

1 Function Update_Approximation( $\tilde{Q}, \mathbf{A}, \mathbf{O}, \Phi^c, \boldsymbol{\eta}, CK, s_t, a_t, r_t, s_{t+1}$ ):
  Input: Q-function approximation  $\tilde{Q}$ ,
           Set of actions  $\mathbf{A}$ ,
           Set of objects  $\mathbf{O}$ ,
           Candidate features  $\Phi^c$ ,
           Relevances of candidate features  $\boldsymbol{\eta}$ ,
           Contextual knowledge  $CK$ ,
           Current state  $s_t$ ,
           Action  $a_t$ ,
           Immediate reward  $r_t$ ,
           Next state  $s_{t+1}$ 
2    $(\Phi(s_t, a_t), \Phi_{active}) \leftarrow$  Evaluate_Features( $\Phi, \mathbf{O}, CK, s_t, a_t$ )
3   Compute TD error  $\delta_t$ 
4   Update weight vector  $\boldsymbol{\theta}$ 
5    $(\Phi^c, \boldsymbol{\eta}) \leftarrow$  Online_Feature_Discovery( $\Phi_{active}, \mathbf{A}, \Phi^c, \boldsymbol{\eta}$ )
6   return  $(\tilde{Q}, \Phi^c, \boldsymbol{\eta})$ 

```

---

on the domain and problem. For example,  $N_\Phi$  should have a large value for problems with a large number of state predicates. Since a state is described with literals, it makes sense that more features are required to represent abstract states. It is desired to have a domain-independent hyperparameter to avoid tedious tuning. Therefore, we set  $N_\Phi = \max\left(1, \frac{\tau}{100} \max_{a \in \mathbf{A}} |\Phi_a|\right)$  where  $\Phi_a$  is the set of base features for  $a$  and  $\tau$  is a hyperparameter. In other words, the maximum number of features that can be added per time step is  $\tau\%$  of the maximum number of base features among the actions. Effectively, we replace  $\xi$  with  $\tau$  which is more intuitive to set as it is independent of the reward function and explicitly limits the expansion rate of  $\Phi$ . In  $\tau$ -iFDD+,  $\xi$  is set to an arbitrary small value (i.e., 0.1) and is incrementally increased when required.

### 3.1.3 Update Q-function Approximation

A Q-function approximation is updated in each time step using Algorithm 4. Given an observation  $\Xi = (s_t, a_t, r_t, s_{t+1})$ ,  $\Phi$  is evaluated for  $(s_t, a_t)$  (line 2, this calls Algorithm 6) by considering contextual knowledge if in use. We discuss contextual knowledge in Section 3.4.2.  $\Phi(s_t, a_t)$  is a vector of real values where the  $i + 1$ -th element is the value of the  $i$ -th feature in  $\Phi$  (the first element in  $\Phi(s_t, a_t)$  is the bias term 1). A feature has a value of 1 if it is active; otherwise, it has a value of 0.  $\Phi_{active}$

is the set of active features. The TD error is computed (line 3) and the weights are updated (line 4). We use Double Q-learning with replacing eligibility trace for each feature (see Section 2.4.1). Newly added features have eligibility traces initialised to 0. The TD error is computed with Equation 2.16 and the weights are updated based on Equation 2.29. Features are incrementally added to  $\Phi$  (line 5). This is done with any feature discovery algorithm which deals with conjunctions of literals as features such as Algorithms 2 or 3.

## 3.2 Ground Approximation

LQ-RRL (Algorithm 1) requires at least one linear function approximation of the Q-function as an input. One possible Q-function approximation uses binary features which are represented by literals (see Definition 12).

### Definition 15 (Ground Approximation)

*A ground approximation  $\tilde{Q}^{gnd}$  is a linear function approximation of the Q-function where  $\Phi$  is a concatenation of the set of features  $\Phi_a$  for each action  $a \in \mathbf{A}$  and  $\Phi_a$  consists of literals or conjunction of literals.*

We use the set of literals  $\mathbf{P}^\#$  as the set of base features for every action. We term this selection of base features as model-free feature selection (MFFS) as it does not exploit any information from a model to prune unnecessary literals as base features. We present a model-based feature selection in Section 4.3.1. Since literals are ground over the set of objects  $\mathbf{O}$ , problems with different sets of objects will have different sets of literals.

### Example 5 (Dependency of Ground Approximation on Objects)

*In **Recon** (see Section 2.8.1), an agent moves in a grid environment, avoiding hazards to take good pictures of every object of type *obj* using its camera tool. In a particular problem of **Recon** where the grid size is  $N \times N$ , there are  $N^2$  objects of type *wp*. The agent can be in  $N^2$  locations which are represented by  $N^2$  ground state predicates of `agentAt(AGENT, WP)`. Since the ground approximation uses literals in  $\mathbf{P}^\#$  to construct features,  $\Phi$  is defined over  $\mathbf{O}$  and  $|\Phi|$  scales with  $|\mathbf{O}|$ .*

Since features are literals or conjunctions of literals, a ground approximation is

not applicable in a problem  $P$  if a feature  $\phi$  includes a literal which is not a literal in  $P$  since  $\phi$  cannot be evaluated to be true or false in the state space of  $P$ . If  $\mathbf{O}$  changes, then the ground approximation changes because: (1) base features are literals which are define over  $\mathbf{O}$ , and (2)  $\Phi$  is a concatenation of features for each ground action in  $\mathbf{A}$  which is define over  $\mathbf{O}$ .

Suppose that  $\tilde{Q}^{gnd}$  is learned in a problem  $P$  with the state-action space  $\mathbf{S}_P \times \mathbf{A}_P$ .  $\tilde{Q}^{gnd}$  is applicable in any problem with a state-action space  $\mathbf{S}^- \times \mathbf{A}^-$  if  $\mathbf{S}^- \subseteq \mathbf{S}_P$  and  $\mathbf{A}^- \subseteq \mathbf{A}_P$ . This poses three limitations of a ground approximation. First, transfer learning from large to small scale problems is not possible. For example, the ground approximation described in Example 5 is not applicable in another problem with a smaller grid size. This is of a lesser concern since it is often desired to perform transfer learning from small to large scale problems instead (see Section 3.6). Second, non-fluents are invariants of a state. If two problems have different non-fluents, they will have non-overlapping state spaces. This implies that a ground approximation learned in one problem will not be applicable in another problem with different non-fluents. Third, even if non-fluents are identical, transfer learning from small to large scale problems will not perform well. This is because the ground approximation does not consider the extended state-action space of the large scale problem. Given these limitations, we propose a first-order approximation which is independent of the objects in  $\mathbf{O}$  and ground non-fluents.

**Are negative literals necessary as features?** The inclusion of negative literals as base features might seem unnecessary due to the sparse summary of features in iFDD+. Consider two base features,  $\phi_i = \mathbf{p}_i$  and  $\phi_j = \mathbf{p}_j$  (the terms of the literals are excluded for brevity), a conjunctive feature,  $\phi_k = \phi_i \wedge \phi_j$ , and three states,  $s_1 = \mathbf{p}_i \wedge \neg \mathbf{p}_j \wedge \dots$ ,  $s_2 = \neg \mathbf{p}_i \wedge \mathbf{p}_j \wedge \dots$ , and  $s_3 = \mathbf{p}_i \wedge \mathbf{p}_j \wedge \dots$ . Due to the sparse summary, only one of the three features is active in each of the states:  $\phi_i(s_1, \cdot) = 1$ ,  $\phi_j(s_2, \cdot) = 1$ , and  $\phi_k(s_3, \cdot) = 1$ . In a state  $s_4 = \neg \mathbf{p}_i \wedge \neg \mathbf{p}_j \wedge \dots$ ,  $\phi_i$ ,  $\phi_j$ , and  $\phi_k$  maps to 0. The states  $s_1$ ,  $s_2$ ,  $s_3$ , and  $s_4$  represent all possible combinations of  $\mathbf{p}_i$  and  $\mathbf{p}_j$ . The features map each of the four states to unique vectors of real numbers even without using the negative literals,  $\neg \mathbf{p}_i$  and  $\neg \mathbf{p}_j$ , as base features. This implies that mapping a state to an abstract state without negative literals does not cause



a coarser granularity. This might seem attractive as the number of base features is reduced by half without the negative literals which gives lower computational and space complexities. However, in  $s_4$ , since none of the features are active, their weights are not updated. If  $s_4$  is a state of interest (e.g., a goal is achieved or a dead end is reached), then the exclusion of negative literals could lead to poor performance. Therefore, to be domain or problem agnostic, we include negative literals as base features.

### 3.3 Consistent Abstraction with First-Order Features

Given the limitations of the ground approximation, we propose a **first-order approximation** of the Q-function, denoted by  $\tilde{Q}^{fo}$  (we omit the superscript for brevity unless it is necessary to distinguish between  $\tilde{Q}^{fo}$  and  $\tilde{Q}^{gnd}$ ), which is independent of  $\mathcal{O}$ . Since states are aggregated to approximate the optimal Q-function (similar to other RRL methods such as Q-RRL [47]), our first-order approximation performs  $Q^*$ -irrelevance abstraction [100, 185]. We shall prove that the first-order approximation is a generalised Q-function under certain conditions in Section 3.3.2.

#### Definition 16 (First-Order Features and First-Order Approximation)

*A first-order feature for an action  $a$  is a lifted literal or a conjunction of lifted literals which contains bound and/or free variables. This is different from symbolic literals which make no distinction between bound and free variables. A first-order approximation is represented by a set of first-order features which is a concatenation of the set of features  $\Phi_{\hat{a}}$  for each symbolic action  $\hat{a} \in \mathcal{A}$ .<sup>1</sup>*

$\Phi$  and  $\theta$  are the concatenation of the set of features and weight vector, respectively, for each symbolic action rather than ground action. Otherwise, the first-order approximation will not be independent of  $\mathcal{O}$  because ground actions are defined over  $\mathcal{O}$ . Sample complexity is potentially reduced as actions in  $\mathbf{A}_{\hat{a}}$  (the set of ground actions for the symbolic action  $\hat{a}$ ) share the same weight components in  $\theta$ ; observations for an action  $a \in \mathbf{A}_{\hat{a}}$  are used to update the shared weights or Q-values

---

<sup>1</sup> $\hat{\cdot}$  denotes a lifted literal, a symbolic action, or a set of lifted literals.

---

**Algorithm 5:** Initialise a set of first-order features for a symbolic action
 

---

```

1  Function Get_First_Order_Features( $\mathbf{P}$ ,  $\mathbf{A}_{\hat{a}}$ ):
    Input: Set of state predicates  $\mathbf{P}$ ,
            Set of ground actions  $\mathbf{A}_{\hat{a}}$  for the symbolic action  $\hat{a}$ 
2   $\Phi_{\hat{a}} = \emptyset$ 
3  for  $a \in \mathbf{A}_{\hat{a}}$  do
4  |    $\Phi_a \leftarrow \text{Feature\_Selection}(\mathbf{P}, a)$ 
5  |    $\Phi_{\hat{a}} \leftarrow \Phi_{\hat{a}} \cup \text{Lift}(\Phi_a, \sigma_a)$ 
6  return Quantified( $\Phi_{\hat{a}}$ )

```

---

of every action in  $\mathbf{A}_{\hat{a}}$ . The first-order approximation requires two assumptions to perform well: (1) the problem is relational and (2) **deictic objects** (objects which are not in the terms of an action) of the same type can be treated as a homogeneous entity (i.e., a free variable).

### 3.3.1 Initialising First-Order Base Features

We present a method to generate first-order base features which is outlined in Algorithm 5 and illustrated with an example in Figure 3.1. This work was published in [122]. The inputs to Algorithm 5 are the set of state predicates  $\mathbf{P}$  and the set of ground actions for  $\hat{a}$  ( $\mathbf{A}_{\hat{a}}$ ), and the output is a set of first-order base features for  $\hat{a}$ . For each ground action  $a \in \mathbf{A}_{\hat{a}}$  (line 3), a set of ground features  $\Phi_a$  is initialised (`Feature_Selection` in line 4). We use `MFFS` for `Feature_Selection` (i.e., `Feature_Selection` returns the set of every state predicate and its negation). Each feature  $\phi \in \Phi_a$  is lifted in accordance with the grounding of  $a$ ,  $\sigma_a$  (`Lift` in line 5) where objects in the terms of  $a$  are substituted with their corresponding bound variables and objects which are not terms of  $a$  are not substituted. For example, a ground action  $\mathbf{a}(x, y)$  of the symbolic action  $\mathbf{a}(X, Y)$  has a grounding  $\sigma_a = \{X/x, Y/y\}$ . A feature is partially lifted if it contains deictic objects.  $\Phi_{\hat{a}}$  is the union of these partially and fully lifted features for each action  $a \in \mathbf{A}_{\hat{a}}$  (line 5). Remaining (deictic) objects in  $\Phi_{\hat{a}}$  are substituted with free variables to yield a set of first-order features (`Quantified` in line 6). Algorithm 5 is called once for each symbolic action.

#### Example 6 (First-Order Base Features)

Following Example 5, the  $N^2$  ground state predicates of `agentAt(AGENT, WP)`

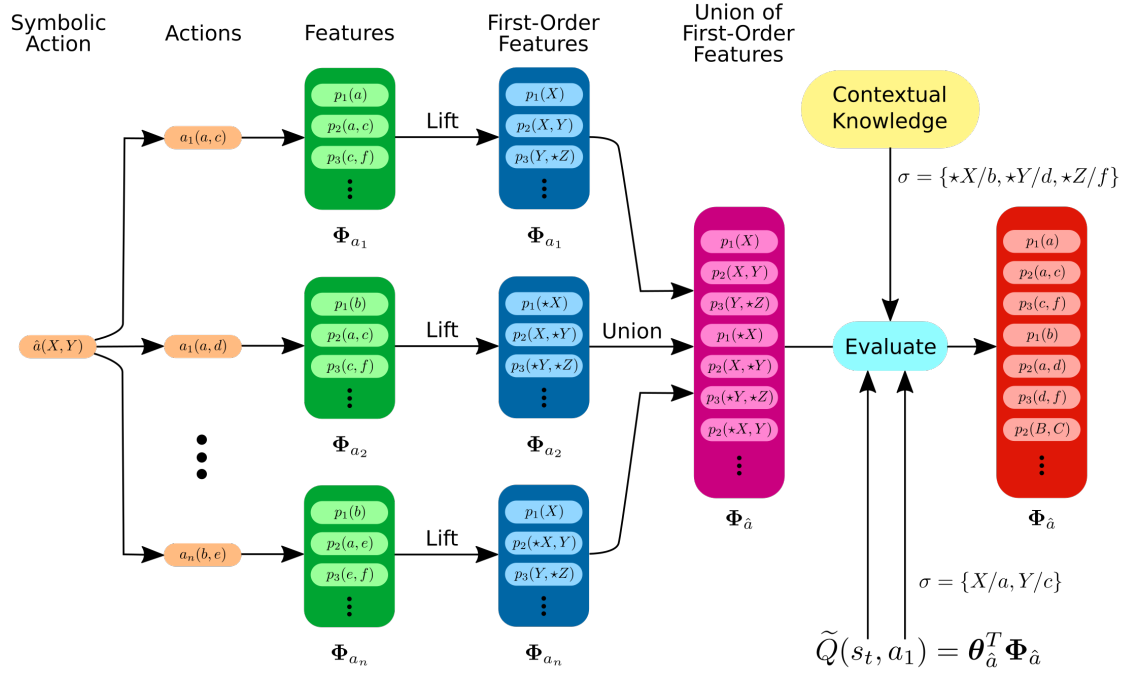


Figure 3.1: An illustration of initialising a set of first-order base features for a symbolic action and the grounding of first-order features. To evaluate first-order features, they are grounded with consideration to the state  $s_t$ , the objects in the action  $a_1$ , and contextual knowledge.

are mapped to only two first-order base features  $\text{agentAt}(\text{AGENT}, \text{WP})$  and  $\text{agentAt}(\text{AGENT}, * \text{WP})$  for the symbolic action  $\text{move}(\text{WP})$  as long as  $N > 1$ .<sup>2</sup> If  $N = 1$  (i.e., the grid is  $1 \times 1$ ), then there is only one object of type  $\text{wp}$  which is represented by the bound variable  $\text{WP}$  and the first-order base feature is only  $\text{agentAt}(\text{AGENT}, \text{WP})$ .

In *Academic Advising*, the first-order base features for  $\text{takeCourse}(\text{COURSE})$  are:

- $\text{PREREQ}(\text{COURSE}, * \text{COURSE})$
- $\text{PREREQ}(* \text{COURSE}, \text{COURSE})$
- $\text{PROGRAM\_REQUIREMENT}(\text{COURSE})$
- $\text{PROGRAM\_REQUIREMENT}(* \text{COURSE})$
- $\text{passed}(\text{COURSE})$
- $\text{passed}(* \text{COURSE})$
- $\text{taken}(\text{COURSE})$
- $\text{taken}(* \text{COURSE})$

<sup>2</sup> $* \mathcal{C}$  denotes a free variable where  $\mathcal{C}$  is the variable type.

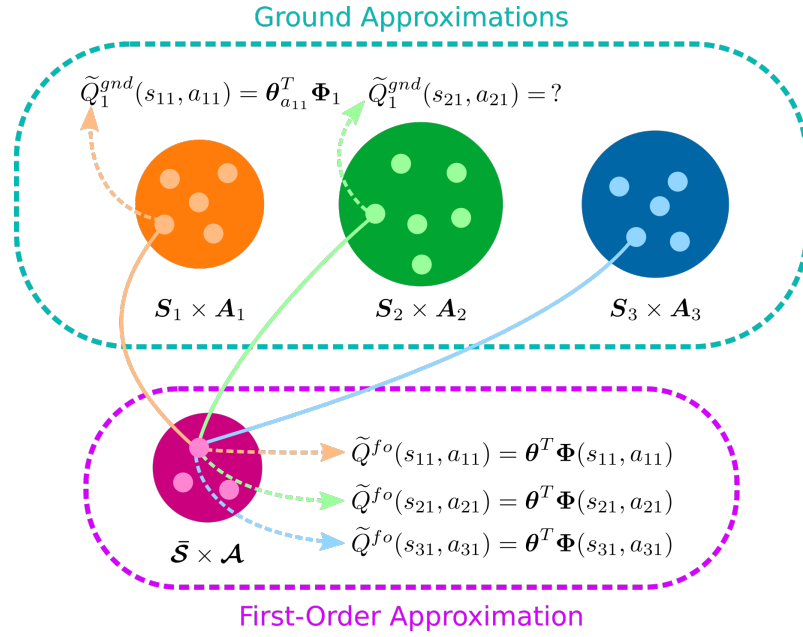


Figure 3.2: Illustration of the relation between the ground approximation and the first-order approximation. Three ground approximations are learned in different problems with non-overlapping state-action spaces and are abstracted to a first-order approximation. A state-action space is represented by a large circle with dark shade. A state-action pair is represented by a small circle with light shade.

Example 6 illustrates that, except for trivially small scale problems, first-order base features are independent of  $\mathbf{O}$  and the number of first-order base features is independent of  $|\mathbf{O}|$ . Therefore, a first-order approximation scales well to large scale problems.

In a ground approximation, every action has the same set of base features if **MFFS** is used. However, in a first-order approximation, actions could have different first-order base features even if **MFFS** is used. This is because the literals are lifted differently in accordance to the grounding of an action. Thus, only ground actions of the same symbolic action share the same features. Having the same set of features and sharing the same set of features are two different concepts. For example, consider the trivial case where there are only two actions and a feature  $\phi$ . If both actions share the same feature, then  $\Phi = [\phi]$ .<sup>3</sup> If both actions have the same feature, then  $\Phi = [\phi \ \phi]$

Figure 3.2 illustrates the relation between three ground approximations which are learned in different problems and a first-order approximation. The ground approximation learned in the first problem,  $\tilde{Q}_1^{gnd}$ , maps a state-action pair  $(s_{11}, a_{11})$  in

<sup>3</sup>For brevity, we omit the bias term here.

$\mathbf{S}_1 \times \mathbf{A}_1$  to a real value. However, it cannot map  $(s_{21}, a_{21})$  in  $\mathbf{S}_2 \times \mathbf{A}_2$  for the second problem to a real value because  $s_{21} \notin \mathbf{S}_1$  and  $a_{21} \notin \mathbf{A}_2$ . A first-order approximation  $\tilde{Q}^{fo}$  maps the state-action spaces of the three problems to the same abstract state-action space  $\bar{\mathbf{S}} \times \bar{\mathbf{A}}$ . Thus, it maps every state-action pair in the state-action spaces of the three problems to real values.

Since `Feature_Selection` (line 4 in Algorithm 5) returns the set of state predicates and their negations which is also the set of base features for a ground approximation, Algorithm 5 can be seen as a mapping from a ground approximation to a first-order approximation—the ground approximation is an abstraction of the original RMDP and the first-order approximation is an abstraction of the ground approximation. This implies that the first-order approximation has a coarser granularity than the ground approximation which we shall discuss in Section 3.4.1. We can represent the abstraction of a ground approximation to a first-order approximation as follows:

$$\Phi_{\hat{a}} = \bigcup_{a \in \mathbf{A}_{\hat{a}}} \bigcup_{\phi \in \Phi_a} \mathcal{L}(\phi, a), \quad (3.3)$$

where  $\Phi_a$  is the set of features in a ground approximation for the action  $a$  and  $\mathcal{L}$  is a function which maps a feature  $\phi$  for  $a$  to a first-order feature. If  $\Phi_a$  is the set of base features, then Equation 3.3 returns the set of first-order base features for  $\hat{a}$ .  $\mathcal{L}$  substitutes objects in  $\phi$  which are in the terms of  $a$  with bound variables and objects which are not in the terms of  $a$  with free variables.

#### Example 7 (Mapping to a First-Order Feature with $\mathcal{L}$ )

For a feature  $\phi = \mathbf{p}(x, z)$  and an action  $a = \mathbf{a}(x, y)$  for the symbolic action  $\mathbf{a}(X, Y)$ ,  $\mathcal{L}$  performs a substitution  $\{x/X, z/\star Z\}$  such that  $\mathcal{L}(\phi, a) = \mathbf{p}(X, \star Z)$  where  $X$  ( $\star Z$ ) is a bound (free) variable of the same type as the object  $x$  ( $z$ ).

A first-order feature  $\phi$  needs to be grounded to evaluate the value of  $\phi(s, a)$ . Bound variables are ground with objects in the terms of the action  $a$  while free variables can be ground to any object of the same type. We discuss the grounding of first-order features and its complications in Section 3.4.

#### Example 8 (Substituting Bound and Free Variables)

In *Recon*, the first-order feature  $\phi = \text{OBJECT\_AT}(\text{OBJ}, \star \text{WP})$  for the action  $a = \text{useToolOn}(a1, w1, o1)$  is ground to  $\text{OBJECT\_AT}(o1, \star \text{WP})$  using the grounding of  $a$ .

The remaining free variable,  $\star WP$ , can be substituted with any object of the type  $wp$ . Semantically,  $\phi$  evaluates to true if  $o1$  is at any location which is always the case. Later, in Section 3.4.1, we discuss the complication of this.

### 3.3.2 Consistent Abstraction and Abstract-Equivalent Problems

We prove that the first-order approximation is a generalised Q-function under certain conditions. A first-order approximation performs state and action abstraction, mapping the state space to an abstract state space and the action space to an abstract action space.<sup>4</sup>

#### Definition 17 (Consistent Abstraction)

If an abstraction function  $\mathcal{F} : \mathbf{X}_P \rightarrow \bar{\mathbf{X}} \forall P \in \mathbf{P}_D$  where  $\mathbf{X}_P$  is a set of entities for a problem  $P$ , then  $\mathcal{F}$  is a consistent abstraction function and  $\bar{\mathbf{X}}$  is a consistent abstraction of  $\mathbf{X}_P \forall P \in \mathbf{P}_D$ .

A consistent abstraction maps the state (action) space of every problem of a domain to the same abstract state (abstract action) space. A Q-function approximation which is learned over this abstract space is a generalised Q-function. A state abstraction function is denoted by  $\mathcal{S} : \mathbf{S} \rightarrow \bar{\mathbf{S}}$  and an action abstraction function is denoted by  $\mathcal{A} : \mathbf{A} \rightarrow \bar{\mathbf{A}}$  where  $\bar{\mathbf{S}}$  and  $\bar{\mathbf{A}}$  are abstract state and action spaces, respectively.

#### Theorem 1 (Consistent Abstraction for Generalised Q-function)

A Q-function approximation  $\tilde{Q}$  for a consistent abstract state-action space  $\bar{\mathbf{S}} \times \bar{\mathbf{A}}$ , where  $\tilde{Q}(\bar{s}, \bar{a}) \mapsto \mathbb{R} \forall \bar{s} \in \bar{\mathbf{S}}, \forall \bar{a} \in \bar{\mathbf{A}}$ , is a generalised Q-function.

*Proof.* Following Definition 17,  $\forall P \in \mathbf{P}_D, \forall (s, a) \in (\mathbf{S}_P, \mathbf{A}_P) (\tilde{Q}(s, a) = \tilde{Q}(\mathcal{S}(s), \mathcal{A}(a)) = \tilde{Q}(\bar{s}, \bar{a}))$  where  $\mathcal{S}(s) \mapsto \bar{s}, \mathcal{A}(a) \mapsto \bar{a}, \bar{s} \in \bar{\mathbf{S}}$ , and  $\bar{a} \in \bar{\mathbf{A}}$ . Following Definition 13,  $\tilde{Q}$  is applicable in every problem of the domain  $D$  and is a generalised Q-function.  $\square$

#### Definition 18 (Feature Space for First-Order Approximation)

The set of first-order base features  $\hat{\mathbf{P}}$  is the concatenation of the set of first-order

<sup>4</sup>The abstract actions here are not the same as temporally abstract actions or options in [169].

base features for every symbolic action  $\hat{a} \in \mathcal{A}$ :  $\hat{\mathcal{P}} = [\hat{\mathcal{P}}_{\hat{a}_1}, \dots, \hat{\mathcal{P}}_{\hat{a}_{|\mathcal{A}|}}]$ . From time step  $t = 0$  onward,  $\Phi = \hat{\mathcal{P}}$  until conjunctive features are added to  $\Phi$ . The set of possible features or feature space for a first-order approximation is  $\wp(\hat{\mathcal{P}}) = [\wp(\hat{\mathcal{P}}_{\hat{a}_1}), \dots, \wp(\hat{\mathcal{P}}_{\hat{a}_{|\mathcal{A}|}})]$ .

In a first-order approximation, the abstract state space is represented by first-order features and the abstract action space is the symbolic action space. Formally,  $\mathcal{S} : \mathcal{S} \rightarrow \wp(\hat{\mathcal{P}})$  and  $\mathcal{A} : \mathcal{A} \rightarrow \mathcal{A}$ . In other words,  $\bar{\mathcal{S}} = \wp(\hat{\mathcal{P}})$  and  $\bar{\mathcal{A}} = \mathcal{A}$ . We prove that under certain conditions,  $\bar{\mathcal{S}}$  and  $\bar{\mathcal{A}}$  are consistent abstract state and action spaces, respectively. Then, following Theorem 1, the first-order approximation is a generalised Q-function.

### Theorem 2 (Consistent Abstract Action Space)

$\bar{\mathcal{A}} = \mathcal{A}$  is a consistent abstract action space for all problems of a domain.

*Proof.* Following Definition 6, every problem  $P \in \mathcal{P}_D$  of a domain  $D$  has the same set of symbolic actions  $\mathcal{A}$ . □

It follows from Theorem 2 that  $\mathcal{A}$  is a consistent abstraction function. Similarly,  $\mathcal{S}$  is a consistent abstraction function if  $\wp(\hat{\mathcal{P}})$  is a consistent abstract state space. Before we prove this, we need to introduce the following concepts.

### Definition 19 (Subsumption of Problems and Abstract-Equivalent Problems)

The relation between two problems can be described with the following two predicates:

$$\mathbf{R}_1(P_i, P_j) \Leftrightarrow \forall \mathcal{C} (|\mathcal{O}_{P_j}^{\mathcal{C}}| \geq |\mathcal{O}_{P_i}^{\mathcal{C}}|),$$

$$\mathbf{R}_2(P_i, P_j) \Leftrightarrow \exists \mathcal{C} (|\mathcal{O}_{P_i}^{\mathcal{C}}| = 1 \wedge |\mathcal{O}_{P_j}^{\mathcal{C}}| > 1),$$

where  $P_i$  and  $P_j$  are problems of a domain  $D$ ,  $\mathcal{C} \in \mathcal{C}$  is a type,  $\mathcal{C}$  is the set of types in  $D$ , and  $\mathcal{O}_P^{\mathcal{C}} \subset \mathcal{O}_P$  is the set of objects in the problem  $P$  with the type  $\mathcal{C}$ .  $\mathbf{R}_1(P_i, P_j)$  is true if for all types,  $P_j$  has at least as many objects of that type as  $P_i$ .  $\mathbf{R}_2(P_i, P_j)$  is true if there is a type which  $P_i$  has only one object of and  $P_j$  has more than one object of. If  $\mathbf{R}_1(P_i, P_j) \wedge \mathbf{R}_2(P_i, P_j)$  is true, then  $P_i$  is subsumed by  $P_j$ . Otherwise, if  $\mathbf{R}_1(P_i, P_j) \wedge \neg \mathbf{R}_2(P_i, P_j) \wedge \neg \mathbf{R}_2(P_j, P_i)$  is true, then  $P_i$  and  $P_j$  are abstract-equivalent

problems. In other words, two problems are abstract-equivalent if for every type, both problems have either one object of or more than one object of.

**Example 9 (Subsumption of Problems for Service Robot)**

In *Service Robot* (see Section 2.8.2), a mobile robot attends to people who need assistance and completes tasks for them such as bringing items to a specific person. The small scale problem *SR1* is not abstract-equivalent with the large scale problem *SR2* because the former has only one object of type person while the latter has three objects of type person. That is,  $R_1(\text{SR1}, \text{SR2}) \wedge R_2(\text{SR1}, \text{SR2})$  is true and *SR2* subsumes *SR1*. The above also holds for *SR3* since it has the same set of objects as *SR2*. Thus, *SR2* and *SR3* are abstract-equivalent and *SR3* subsumes *SR1*. We shall see in Example 10 what it means for a problem to subsume another.

We use the concept of abstract-equivalent problems to specify the conditions for consistent abstract state space performed by the first-order approximation.

**Theorem 3 (Consistent Abstract State Space)**

$\bar{\mathcal{S}} = \wp(\hat{\mathcal{P}})$  is a consistent abstract state space for abstract-equivalent problems.

Before proving Theorem 3, we need to introduce an abstraction function  $\mathcal{P}$ :

$$\mathcal{P}(p, \hat{a}) = \bigcup_{a \in A_{\hat{a}}} \mathcal{L}(p, a), \quad (3.4)$$

which maps a literal  $p$  to a set of first-order base features for a symbolic action  $\hat{a}$ . In other words,  $\mathcal{P}$  returns the union of the lifted literals resulting from lifting  $p$  with the grounding of each ground action of  $\hat{a}$ .

*Proof.* See Section A.1 of Appendix A. □

A key step in the proof shows that  $\mathcal{P}$  is a consistent abstraction function for abstract-equivalent problems. Following Theorems 2 and 3, the first-order approximation performs consistent state and action abstractions for abstract-equivalent problems. Thus, it is a generalised Q-function if only abstract-equivalent problems are considered.

**Example 10 (Inconsistent Abstraction in Service Robot)**

We show that  $\mathcal{P}$  is inconsistent in problems which are not abstract-equivalent. Fol-



lowing Example 9, **SR1** and **SR2** are not abstract-equivalent. We consider the actions  $\hat{a} = \text{find\_person}(\text{ROBOT}, \text{PERSON})$ ,  $a_1 = \text{find\_person}(r1, p1)$ , and  $a_2 = \text{find\_person}(r1, p2)$  (for **SR2** only), and the literal  $p = \text{goal\_object\_with}(o1, p1)$ .

The abstractions for **SR1** are:

- $\mathcal{L}(p, a_1) = \text{goal\_object\_with}(\star\text{OBJ}, \text{PERSON})$
- $\mathcal{P}(p, \hat{a}) = \text{goal\_object\_with}(\star\text{OBJ}, \text{PERSON})$

The abstractions for **SR2** are:

- $\mathcal{L}(p, a_1) = \text{goal\_object\_with}(\star\text{OBJ}, \text{PERSON})$
- $\mathcal{L}(p, a_2) = \text{goal\_object\_with}(\star\text{OBJ}, \star\text{PERSON})$
- $\mathcal{P}(p, \hat{a}) = \{\text{goal\_object\_with}(\star\text{OBJ}, \text{PERSON}), \text{goal\_object\_with}(\star\text{OBJ}, \star\text{PERSON})\}$  (due to the union of features in line 5 of Algorithm 5)

Since  $\mathcal{P}$  maps  $p$  to different sets of first-order base features in **SR1** and **SR2**, it is not a consistent abstraction. Observe also that the set of first-order base features in **SR2** subsumes that in **SR1**. Thus, **SR2** subsumes **SR1** from this viewpoint (see Definition 19 and Example 9). Intuitively, the first-order feature  $\text{goal\_object\_with}(\star\text{OBJ}, \star\text{PERSON})$  implies that there exists another person ( $\star\text{PERSON}$ ) which cannot be the case if there is only one person.

### 3.3.3 Augmenting First-Order Base Features

The first-order approximation is a generalised Q-function only in abstract-equivalent problems which might seem to limit its application. However, this is often not the case in practice. Of the domains we consider, only **Service Robot** has problems which are not abstract-equivalent (see Example 9). This is because we instantiated a trivially small scale problem, **SR1**, for this domain which has only one object of type *person*. Other than **SR1**, the other two problems (i.e., **SR2** and **SR3**) are abstract-equivalent. In the remaining domains, the problems considered are abstract-equivalent though problems which are not abstract-equivalent can be instantiated. For example, in **Recon**, a problem with only one object of type *obj* is not abstract-equivalent with **RC3** and **RC6**. However, this problem is trivially simple with only one goal. The same reasoning applies to the other domains. Nevertheless, we propose a solution such that transfer learning between problems which are not

abstract-equivalent is still possible with a first-order approximation.

**Theorem 4 (Subsumption of First-Order Base Features)**

If a problem  $P_i$  is subsumed by another problem  $P_j$ , then the set of first-order base features for  $P_i$  is a subset of that for  $P_j$ .

*Proof.* See Section A.1 of Appendix A. □

The subsumption of first-order base features is illustrated in Example 10. Consider two problems,  $P_{source}$  and  $P_{target}$ , where  $P_{source}$  is subsumed by  $P_{target}$  (i.e., they are not abstract-equivalent). Since  $P_{source}$  is of a smaller scale than  $P_{target}$ , transfer learning from  $P_{source}$  to  $P_{target}$  is of interest.

Theorem 4 implies that the feature space of  $P_{source}$  is a subset of the feature space of  $P_{target}$ . Thus,  $\tilde{Q}_{source}$  is applicable in  $P_{target}$ .<sup>5</sup> That is, every first-order feature approximating  $\tilde{Q}_{source}$  can be evaluated in  $P_{target}$ . However, this does not mean that  $\tilde{Q}_{source}$  can approximate the optimal Q-function for  $P_{target}$ . In fact, this is unlikely as first-order base features present in  $P_{target}$  but not  $P_{source}$  are not considered by  $\tilde{Q}_{source}$ . We can still utilise  $\tilde{Q}_{source}$  to solve  $P_{target}$  by augmenting the features and weights of  $\tilde{Q}_{source}$ . The augmented Q-function approximation  $\tilde{Q}_{target}$  has an initial set of features given as:

$$\Phi_{\hat{a}}^{target} = \Phi_{\hat{a}}^{source} \cup \left( \hat{\mathcal{P}}_{\hat{a}}^{target} - \hat{\mathcal{P}}_{\hat{a}}^{source} \right). \quad (3.5)$$

The weights of the features are the same as that in  $\tilde{Q}_{source}$  unless they are newly added features (i.e, the second term in parentheses on the right hand side of Equation 3.5); their weights are initialised to zeros.

We provide an intuition for why it is beneficial to consider  $\tilde{Q}_{source}$  to solve  $P_{target}$ . There is at least one type for which  $P_{source}$  has only one object of and  $P_{target}$  has multiple objects of. In spite of this,  $\tilde{Q}_{source}$  can serve as a good initialisation for  $\tilde{Q}_{target}$ . Given the additional objects in  $P_{target}$  and the augmented set of base features, the learning objective is reduced to updating  $\tilde{Q}_{target}$  to consider these additional objects, base features, and possibly new conjunctive features. This is simpler than learning from scratch. Our empirical results in Section 3.7.4 for **Service Robot** show that this is indeed the case.

---

<sup>5</sup>The subscript *source* (*target*) denotes an entity belongs to the source (target) problem.

### 3.3.4 Conjunctive First-Order Features

In a ground approximation, a conjunctive feature is true if all of its parent features are true. On the other hand, in a first-order approximation, a conjunctive feature might not be true even if all of its parent features are true. This is because free variables in the parent features might be substituted with objects different from those used to ground the conjunctive feature.

#### Example 11 (False Conjunctive Features with True Parent Features)

*In a particular state  $s$  for a problem of *Recon*, the agent  $a1$  is at  $wp1$  and an object of type  $obj$ ,  $o1$ , is at  $wp2$ . There are no other objects of type  $obj$ . Suppose that there are three features for a particular action:  $\phi_1 = \text{agentAt}(AGENT, \star WP)$ ,  $\phi_2 = \neg \text{agentAt}(AGENT, \star WP)$ , and  $\phi_3 = \text{OBJECT\_AT}(OBJ, \star WP)$ . These features can be grounded such that they are true in  $s$ :  $\phi_1 = \text{agentAt}(a1, wp1)$ ,  $\phi_2 = \neg \text{agentAt}(a1, wp2)$ , and  $\phi_3 = \text{OBJECT\_AT}(o1, wp2)$ . Consider two conjunctive features:  $\phi_{c1} = \phi_1 \wedge \phi_2 = \text{agentAt}(AGENT, \star WP) \wedge \neg \text{agentAt}(AGENT, \star WP)$ , and  $\phi_{c2} = \phi_1 \wedge \phi_3 = \text{agentAt}(AGENT, \star WP) \wedge \text{OBJECT\_AT}(OBJ, \star WP)$ .  $\phi_{c1}$  evaluates to false in all states.  $\phi_{c2}$  evaluates to false in  $s$  since neither  $\text{agentAt}(a1, wp1) \wedge \text{OBJECT\_AT}(o1, wp1)$  nor  $\text{agentAt}(a1, wp2) \wedge \text{OBJECT\_AT}(o1, wp2)$  is true in  $s$  (i.e., no grounding of  $\phi_{c2}$  is true in  $s$ ). Conversely,  $\phi_{c2}$  is true in other states where  $a1$  and  $o1$  are in the same location.*

Example 11 shows two types of conjunctive features which are false despite having parent features which are true. The first type,  $\phi_{c1}$  in the example, can be identified easily: no conjunctive feature should consist of a literal and its negation. This applies to a ground approximation as well. Such features will always evaluate to false since the conjunction of a literal and its negation is false. Adding such features to  $\Phi$  does not affect the performance other than increasing the space and time complexities. The second type,  $\phi_{c2}$  in the example, affects online feature discovery: in each time step, the set of active candidate features  $\Phi_{active}^c$  is the power set of the active features. Since a conjunction of active first-order features is not necessarily true, any candidate feature which is not true is removed from  $\Phi_{active}^c$ .

**Algorithm 6:** Evaluate a set of first-order features

---

```

1  Function Evaluate_Features( $\Phi, \mathbf{O}, CK, s, a$ ):
   |   Input: First-order features  $\Phi$ ,
   |             Set of objects  $\mathbf{O}$ ,
   |             Contextual knowledge  $CK$ ,
   |             State  $s$ ,
   |             Action  $a$ 
2   $\sigma_0 \leftarrow$  Get the set of possible substitutions from  $\mathbf{O}$  for  $\Phi$ 
3   $(\sigma_1, \dots, \sigma_n) \leftarrow$  Apply contextual grounding  $CK$ 
4   $\sigma \leftarrow$  Apply_Ordered_Substitution( $(\sigma_0, \sigma_1, \dots, \sigma_n)$ )
5   $\Phi(s, a) \leftarrow$  Ground  $\Phi$  with  $\sigma$  and evaluate in  $s$ 
6   $\Phi_{active} = \{\phi \mid \phi \in \Phi \wedge \phi(s, a) \neq 0\}$ 
7  return ( $\Phi(s, a), \Phi_{active}$ )

```

---

**Algorithm 7:** Apply an ordered set of substitutions

---

```

1  Function Apply_Ordered_Substitutions( $\bar{\sigma}$ ):
   |   Input: Ordered set of substitutions  $\bar{\sigma} := (\sigma_0, \dots, \sigma_n)$ 
2   $\sigma = \sigma_0$ 
3  for  $i = 1$  to  $n$  do
4  |   for  $(V, \mathbf{O}_i) \in \sigma_i$  do
5  | |   if  $\sigma$  does not have a substitution for  $V$  then
6  | | |   Update  $\sigma$  with  $\{V/\mathbf{O}_i\}$ 
7  | |   else
8  | | |    $\mathbf{O}_0 \leftarrow$  Get objects which  $\sigma$  substitutes  $V$  with
9  | | |   if  $\mathbf{O}_i \cap \mathbf{O}_0 \neq \emptyset$  then Update  $\sigma$  with  $\{V/\mathbf{O}_i \cap \mathbf{O}_0\}$ 
10 |   return  $\sigma$ 

```

---

### 3.4 Grounding First-Order Features

We have now established that the first-order approximation is a generalised Q-function in abstract-equivalent problems and described the initialisation of first-order base features and the generation of first-order candidate features. The next step is to compute the Q-values given by the first-order approximation and update the first-order approximation (with Algorithm 4). This requires the evaluation of the values of first-order features. A first-order feature  $\phi$  needs to be grounded before it can be evaluated in a state. Algorithm 6 grounds and evaluates a set of first-order features  $\Phi$  for an action  $a$  in the state  $s$ , and returns the set of active features and  $\Phi(s, a)$ . Bound variables are substituted with objects in the terms of  $a$  while free variables can be substituted with any object of the same type. Even though

every ground action of  $\mathbf{A}_a$  shares the same weights and features, they could still have different Q-values because the bound variables are substituted with different objects. The work described in this section was published in [122].

**Definition 20 (Set of Substitutions and Ordered Substitutions)**

A substitution  $\sigma$  for a set of free variables  $\mathbf{V} = \{V_1, \dots, V_n\}$  substitutes each free variable  $V \in \mathbf{V}$  with an object of the same type as  $V$ .<sup>6</sup> A set of substitutions for  $\mathbf{V}$  is  $\sigma = \{V_1/\mathbf{O}^{C_1}, \dots, V_n/\mathbf{O}^{C_n}\}$  where  $C_i$  is the type of  $V_i$  and  $\mathbf{O}^C \subset \mathbf{O}$  is the set of objects of the type  $C$ . A set of substitutions is given by an ordered set  $\bar{\sigma} = (\sigma_1, \dots, \sigma_m)$  as outlined in Algorithm 7.

First, Algorithm 6 considers the grounding of a free variable with every object in  $\mathbf{O}$  of the same type (line 2). Next, an ordered set of substitutions is determined from contextual knowledge (line 3). We discuss contextual knowledge in Section 3.4.2. The ordered set of substitutions is applied using Algorithm 7 which returns a set of substitutions (line 4). The first-order features are grounded and evaluated in the state  $s$  (line 5). A first-order feature  $\phi$  is true in a state  $s$  if there exists a grounding of  $\phi$  which is true in  $s$ . This is analogous to an existential quantification.

Algorithm 7 returns a set of substitution  $\sigma$  resulting from the application of an ordered set of substitutions  $\bar{\sigma}$ .  $\sigma$  is initialised to the first set of substitutions in  $\bar{\sigma}$  (line 2). Subsequent sets of substitutions in  $\bar{\sigma}$  are considered sequentially (lines 3 to 9). For each free variable  $V$  which a set of substitutions  $\sigma_i$  grounds (line 4), if  $\sigma$  does not have a substitution for  $V$  (line 5), then the grounding of  $V$  is added to  $\sigma$  (line 6). Otherwise, let  $\mathbf{O}_0$  denotes the set of objects which  $\sigma$  grounds  $V$  with (line 8) and  $\mathbf{O}_i$  be the set of objects  $\sigma_i$  grounds  $V$  with. If the intersection of these two sets of objects is the empty set, then the grounding of  $V$  due to  $\sigma_i$  is not added to  $\sigma$ ; thus, the order of substitutions matters. Otherwise, the grounding is added to  $\sigma$  (line 9)—objects which are not in  $\mathbf{O}_i$  but are in  $\mathbf{O}_0$  are not included and will not be used to ground  $V$ .

**Theorem 5 (Time Complexity of Algorithm 6)**

The time complexity for Algorithm 6 is  $O(\prod_{\phi \in \Phi_a} \prod_{V_i \in \mathbf{V}_\phi} |\mathbf{O}^{C_i}|)$  where  $\mathbf{V}_\phi$  is a set of free variables in a first-order feature  $\phi$ ,  $C_i$  is the type of  $V_i$ , and  $\mathbf{O}^{C_i} \subset \mathbf{O}$  is a set

<sup>6</sup>Here, we omit  $\star$  from the notation of a free variable for better readability.

of objects of type  $\mathcal{C}_i$  which  $V_i$  is substituted with.

*Proof.* The most computationally expensive procedure in Algorithm 6 is to ground  $\Phi$  (line 5). By definition, elements in  $\Phi(s, a)$  have a value of 0 if they are not of the action  $a$ . Thus, only  $\Phi_{\hat{a}} \subset \Phi$  needs to be grounded and evaluated. The number of substitutions for a first-order feature  $\phi$  and for  $\Phi_{\hat{a}}$  are:

$$|\sigma_{\phi}| = \prod_{V_i \in \mathbf{V}_{\phi}} |\mathcal{O}^{\mathcal{C}_i}|, \quad (3.6)$$

$$|\sigma_{\Phi_{\hat{a}}}| = \prod_{\phi \in \Phi_{\hat{a}}} |\sigma_{\phi}| = \prod_{\phi \in \Phi_{\hat{a}}} \prod_{V_i \in \mathbf{V}_{\phi}} |\mathcal{O}^{\mathcal{C}_i}|, \quad (3.7)$$

respectively, where  $\sigma_{\phi}$  ( $\sigma_{\Phi_{\hat{a}}}$ ) is a set of possible substitutions for  $\phi$  ( $\Phi_{\hat{a}}$ ).  $\phi$  evaluates to true if there exists one grounding of it that evaluates to true in the state  $s$ . In the worst case scenario, every possible substitution is computed. This happens when none of the groundings before the last grounding evaluates to true. It follows that the time complexity of Algorithm 6 is the upper bound on the number of substitutions made to evaluate each feature in  $\Phi_{\hat{a}}$  which is given by Equation 3.7.  $\square$

Observe that the time complexity of Algorithm 6 can only be practically reduced by reducing the cardinality of  $\mathcal{O}^{\mathcal{C}_i}$ . The set of objects  $\mathcal{O}$  is inherent in a problem while the cardinality of  $\Phi$  depends on the nature of the problem and feature discovery.

### Example 12 (Number of Substitutions)

Consider a first-order feature  $\phi = \text{OBJECT\_AT}(\star\text{OBJ}, \star\text{WP})$  for a particular action in *Recon*. The number of substitutions for  $\phi$  is  $|\sigma_{\phi}| = |\mathcal{O}^{\text{obj}}| \times |\mathcal{O}^{\text{wp}}|$ . In *RC6* which has 16 locations (*wp*), six objects (*obj*), one agent (*agent*), and three tools (*tool*),  $|\sigma_{\phi}| = 96$ .

## 3.4.1 Granularity and Pitfalls of First-Order Abstraction

The coarser the granularity of a Q-function approximation, the more states are partitioned into a region (or abstract state) where they are considered the same for the purpose of approximating the Q-values. When this is not the case, the performance deteriorates. A first-order approximation gives a coarser granularity

than a ground approximation since the former is an abstraction of the latter (see Section 3.3.1 and Equation 3.3). We can look at this from another perspective. A state is partitioned by features. Consider the approximation  $\tilde{Q}(s, a)$  where  $a$  is a ground action of  $\hat{a}$ . For the ground approximation, the maximum number of features is  $|\Phi_a| = 2^{2|\mathcal{P}|}$ . For the first-order approximation, the maximum number of features is  $|\Phi_{\hat{a}}| = 2^{2|\hat{\mathcal{P}}|}$ . Since  $|\hat{\mathcal{P}}|$  is less than  $|\mathcal{P}|$  in most problems, the first-order approximation uses fewer features to partition the state space.<sup>7</sup> Therefore, the size of the state partitions (or granularity) in a first-order approximation must be larger than a ground approximation. Note that it is incorrect to consider  $|\Phi|$  for the above analysis because only  $\Phi_a \subset \Phi$  (or  $\Phi_{\hat{a}} \subset \Phi$  for the first-order approximation) is used to partition the state space for the approximation of the Q-values of an action  $a$  (or symbolic action  $\hat{a}$  for the first-order approximation). The coarse granularity of a first-order approximation is compounded by the use of free variables. Since a free variable can be substituted with a set of objects, a first-order feature  $\phi$  is true if there exists a substitution which makes it true. This increases the coverage of  $\phi$ ; the inability of a Q-function approximation to differentiate a large number of states as they are mapped to the same abstract state could cause poor performance.

### Definition 21 (Grounding Ambiguity)

*The **grounding ambiguity** is an issue inherent in the first-order approximation when a first-order feature can be ground in more than one ways which results in a higher than desired coverage. This is crucial as the choice of a substitution to ground a set of first-order features determines the Q-values and active features which in turn affects weight update and online feature discovery.*

The grounding ambiguity can be mitigated by reducing the set of objects which a free variable can be substituted with (i.e.,  $\mathbf{O}^{C_i}$  in Equation 3.6). This also reduces the time complexity of Algorithm 6 (see Theorem 5). Next, we provide some examples to illustrate the pitfalls of using a first-order approximation. In Sections 3.4.2 and 3.5, we propose methods to resolve these issues.

### Example 13 (Interdependent Goals)

*In **Robot Fetch** (see Section 2.8.2), a mobile robot is tasked with placing objects of*

---

<sup>7</sup> $|\hat{\mathcal{P}}| = |\mathcal{P}|$  if there is only one object for each type.



Figure 3.3: A problem of *Robot Fetch* which requires the robot to start by placing any of the objects,  $o1$ ,  $o2$ , or  $o3$ , at the location  $wp1$  instead of their respective goal locations as none of their goal locations are empty.

*type obj* at their respective goal locations. Goals are interdependent because at most only one object can be at a location at any time. The first-order base features for the action  $\text{move}(\text{ROBOT}, WP)$  include:

- $\text{OBJECT\_GOAL}(\star OBJ, WP)$
- $\text{OBJECT\_GOAL}(\star OBJ, \star WP)$
- $\text{object\_at}(\star OBJ, \star WP)$
- $\text{object\_at}(\star OBJ, WP)$

The conjunction of any of these features will consider at most one object, represented by  $\star OBJ$ , and at most two locations, represented by  $WP$  and  $\star WP$ . There is no bound variable  $OBJ$  because  $\text{move}(\text{ROBOT}, WP)$  does not have a term of the type *obj*. Figure 3.3 illustrates a particular problem of *Robot Fetch*. In the initial state, the robot has to move any of the objects,  $o1$ ,  $o2$ , or  $o3$ , to  $wp1$  rather than to its goal location. None of the conjunctive features which can be formed by the base features can inform a policy to “move  $o1$  from  $wp2$  to  $wp1$  so that  $o3$  can be moved from  $wp4$  to  $wp2$ ” as this involves two objects and three locations. A ground approximation does not face this issue since it approximates the  $Q$ -values of every action and can generate a policy which memorises the sequence of actions to reach the goal state. However, this will not generalise to another problem which has a different set of goals or objects.

#### Example 14 (Counting Features)

In *Academic Advising* (see Section 2.8.1), a student has to pass some required courses. These are the goals in the problem. Goals are interdependent because some courses could be prerequisites for other courses. The



success of passing a course depends on the number of prerequisites passed. Suppose that a conjunctive feature  $\phi = \text{PREREQ}(\star\text{COURSE}, \text{COURSE}) \wedge \text{passed}(\star\text{COURSE}) \wedge \text{PROGRAM\_REQUIREMENT}(\text{COURSE})$  is added to  $\Phi$  for the action  $\text{takeCourse}(\text{COURSE})$ . Since  $\star\text{COURSE}$  can be ground to any object of type *course*,  $\phi$  evaluates to true if *COURSE* is a program that must be passed and there exists another course which is a prerequisite of *COURSE* that has not been passed. The value of  $\phi(s, a)$  does not change even if there are more courses which  $\star\text{COURSE}$  can be substituted with such that  $\phi$  evaluates to true. This is the drawback of using binary features in a first-order approximation. While a ground approximation also uses binary features, it has an implicit counting—since every ground literal of  $\text{passed}(\text{COURSE})$  are base features, if  $N$  courses are passed, then  $N$  number of these features evaluate to true. In contrast, only the first-order features  $\text{passed}(\text{COURSE})$  and/or  $\text{passed}(\star\text{COURSE})$  will evaluate to true.

Examples 13 and 14 illustrate the limitations in the representational capacity of a first-order approximation. A first-order feature has at most two variables for each type: a bound variable and a free variable. Thus, a first-order approximation abstracts states such that only relations involving at most two objects of each type are retained.

### Example 15 (At Most Two Objects)

In *Academic Advising*, there is no first-order feature which can represent the relation “a course *CS11* is a prerequisite of the course *CS51* which is in turn a prerequisite of *CS53*”. A ground approximation can represent this relation with the feature  $\phi_1 = \text{PREREQ}(\text{CS11}, \text{CS51}) \wedge \text{PREREQ}(\text{CS51}, \text{CS53})$ . Consider a first-order feature  $\phi_2 = \text{PREREQ}(\star\text{COURSE}, \text{COURSE}) \wedge \text{PREREQ}(\text{COURSE}, \star\text{COURSE})$ . For the action  $\text{takeCourse}(\text{CS51})$ ,  $\phi_2$  can be grounded to  $\text{PREREQ}(\text{CS11}, \text{CS51}) \wedge \text{PREREQ}(\text{CS51}, \text{CS11})$  or  $\text{PREREQ}(\text{CS53}, \text{CS51}) \wedge \text{PREREQ}(\text{CS51}, \text{CS53})$ , both of which are not equivalent to  $\phi_1$ .

These issues are not limited to our first-order approximation but to RRL methods in general as relational representations do not explicitly enumerate over every object and are independent of the number of objects in the problem. We note that RRL methods such as those based on Q-RRL [47] are tested on a simplified **Blocks World**

domain where the goal is to stack the blocks into one column in no specific order.<sup>8</sup> Because of the grounding ambiguity, these RRL methods cannot learn policies which stack the blocks in a specific order. They only consider blocks in the action or goal description while all other blocks are treated as generic blocks [43].

Furthermore, they utilise a symbolic state predicate `above(BLOCK1, BLOCK2)` which is true if `BLOCK1` is on top of `BLOCK2` or another block, `BLOCK3`, where `above(BLOCK3, BLOCK2)` is true (i.e., this is a recursive relation). Analogously, we can introduce a symbolic state predicate `PREREQ2(COURSE1, COURSE2)` to `Academic Advising` which is true if `PREREQ(COURSE1, COURSE2)` is true or there exists another course, `COURSE3`, where `PREREQ(COURSE1, COURSE3) ∧ PREREQ2(COURSE3, COURSE2)` is true. This state predicate can resolve the issue described in Example 15 as its recursive nature implicitly represents the relation between more than two objects (e.g., `PREREQ2(CS11, CS53)` is true because `PREREQ(CS11, CS51)` and `PREREQ(CS51, CS53)` are true).

The limited representational capacity of the first-order approximation coupled with shared weights among ground actions of a symbolic action could cause plateaus.

### Definition 22 (Plateau)

*Plateaus are regions of the state space where there are more than one action with the maximal Q-value (i.e., a greedy action) but not all of these greedy actions are optimal actions.*

Two straightforward remedies to plateaus are to include non-fluents as first-order features and to separate the weights for actions. Using non-fluents as features is not necessary in a ground approximation since the value of a ground non-fluent does not change. However, a lifted non-fluent is evaluated to either true or false depending on its grounding. Non-fluents are often crucial in determining which action of  $\mathbf{A}_a$  to select. The issue of plateaus persists if there are no first-order features which represent aspects of the state crucial for determining the optimal action(s).

### Example 16 (Plateau in Grid Environments)

*We consider a problem in Recon where the grid is  $3 \times 3$  and there is only one*

---

<sup>8</sup>In `Blocks World`, there is a table and some blocks. Three type of goals are considered by previous RRL work: (1) stack all blocks into one column in no particular order, (2) unstack all blocks (i.e., all blocks are on the table), and (3) put a specific block on a specific block.

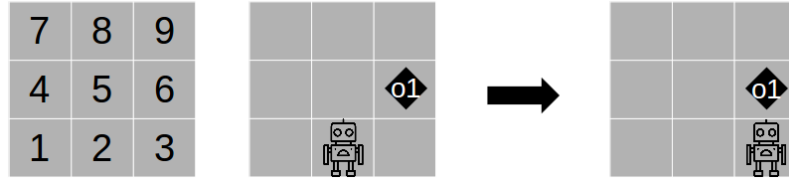


Figure 3.4: A problem of Recon where there is a plateau in state  $s_1$  for a first-order approximation. Left: the numberings represent the locations  $wp_1, wp_2, \dots, wp_9$ . Center: state  $s_1$ . Right: state  $s_2$ .

object,  $o1$ . The goal is to take a good picture of  $o1$ . This is illustrated in Figure 3.4. Let  $s_1$  be the state where the agent  $a1$  is at  $wp_2$  and needs to move to  $wp_3$ , then to  $wp_6$  where  $o1$  is at.  $\tilde{Q}(s_1, \text{move}(a1, wp_3))$  should have the maximal  $Q$ -value; however, all ground actions of  $\text{move}(\text{AGENT}, WP)$  have the same  $Q$ -value because first-order features can only represent situations where the agent is with the object or adjacent to the object. The latter is represented by a conjunctive feature  $\phi = \text{ADJACENT}(WP, \star WP) \wedge \text{agentAt}(\star WP) \wedge \text{OBJECT\_AT}(\star OBJ, WP)$ . Note that non-fluents are involved. There are no first-order features which represent “ $o1$  is  $X$  positions away from the agent” where  $X > 1$ . For a first-order approximation, such plateaus are common in problems with grid environments.

Next, the robot moves to  $wp_3$ ; this state is denoted by  $s_2$ .  $\tilde{Q}(s_2, \text{move}(a1, wp_6))$  should be larger than  $\tilde{Q}(s_2, \text{move}(a1, wp_2))$ . As mentioned above, the feature  $\phi$  resolves the plateau between  $\text{move}(a1, wp_2)$  and  $\text{move}(a1, wp_6)$ .

The inclusion of non-fluents does not entirely resolve the issue of plateaus. In the above example, if  $o1$  is more than one grid position away from the agent, there are no features, first-order or not, which represents this relation. We propose solutions to resolve plateaus in Section 3.5.

On another note, the first-order approximation requires all symmetric relations to be explicitly specified in the problem. Consider a grid environment where two locations,  $wp_1$  and  $wp_2$ , are adjacent to each other. There is no directionality in the adjacency and the agent can move between the two locations. If only the non-fluent  $\text{ADJACENT}(wp_1, wp_2)$  is specified in the problem, then the first-order feature  $\text{ADJACENT}(WP, \star WP)$  can be grounded to  $\text{ADJACENT}(wp_2, wp_1)$  which evaluates to false. Therefore, the non-fluent  $\text{ADJACENT}(wp_2, wp_1)$  must also be specified in the problem.

### 3.4.2 Contextual Knowledge

We propose to use contextual knowledge to mitigate the grounding ambiguity by reducing the set of objects a free variable is substituted with.  $\sigma_0$  is a set of substitutions where a free variable is substituted with any object of the same type. Algorithm 7 returns a set of substitutions for  $\Phi_{\hat{a}}$  ( $\sigma_{\Phi_{\hat{a}}}$ ) given an ordered set of substitutions  $\bar{\sigma}$  where the first set of substitutions is  $\sigma_0$  and the remaining sets are the **contextual grounding** or sets of substitutions due to contextual knowledge. Without contextual knowledge,  $\sigma_{\Phi_{\hat{a}}} = \sigma_0$ .

We propose three types of contextual knowledge: **ground context**, **goal context**, and **location context**. The contextual grounding for them are automatically generated. Goal context and location context require only some trivial domain knowledge to identify the relevant state predicates to consider. Goal (location) context is applicable in all problems with goals (locations) while ground context is inherent in all problems. There remains possibilities for other types of contextual knowledge for future work. Each type of contextual knowledge gives at least one substitution for one or more free variables. Multiple contextual knowledge can be combined by applying their contextual grounding in a sequential order using Algorithm 7. Semantically, first-order features grounded with contextual knowledge represent the current state (e.g., location of the agent) and objectives (e.g., achieve a particular goal). A first-order feature  $\phi$  is existentially quantified due to its free variables; applying a contextual knowledge imposes some conditions on its existential quantification which can be regarded as a conjunction of the conditions and  $\phi$ . This is illustrated in Example 21 later in this section when we present location context.

#### Ground Context

The generation of a set of first-order base features is given by Equation 3.3 where at least one literal is mapped to a first-order base feature. Ground context grounds a first-order feature by grounding its constituent base features back to the literals from which it was mapped from. Unlike goal context and location context, ground context substitutes free variables in each first-order feature independently. Ground context is not used to ground lifted non-fluents because it will only lead to grounded

non-fluents which are always true.

**Definition 23 (Ground Context)**

Let  $\mathcal{L}^{-1}$  be a function which returns the inverse of the substitution  $\mathcal{L}$  (see Equation 3.3) performs to map a feature to a first-order feature. Ground context grounds a first-order base feature  $\phi_i$  with the substitution  $\sigma_{\text{ground}} = \bigcup_{p \in \mathbf{P}^-} \mathcal{L}^{-1}(p, a)$  where  $\mathbf{P}^- \subset \mathbf{P}^\#$  is a set of literals which  $\mathcal{L}$  maps to  $\phi_i$  and a union of substitutions grounds a variable with the union of objects which each substitution grounds it with. A conjunctive first-order feature  $\phi_j$  is grounded by performing the above step for each of its constituent feature with the constraint that for each grounding of  $\phi_j$ , each free variable (which can be in more than one constituent feature) must be substituted with the same object. If this is not possible, then  $\phi_j$  cannot be grounded and evaluates to false.

**Example 17 (Determine Ground Context)**

We consider a trivial problem of *Academic Advising* where  $\mathbf{O} = \{CS11, CS12, CS13, CS21, CS22, CS23\}$ . We use *MFFS* for *Feature\_Selection* in Algorithm 5 where every literal is considered for mapping to first-order base features. For brevity, we only consider the ground positive literals of `passed(COURSE)`. For the action  $a_1 = \text{takeCourse}(CS11)$ ,  $\mathcal{L}$  maps `passed(CS12)`, ..., `passed(CS23)` to  $\phi = \text{passed}(\star\text{COURSE})$ . The ground context grounds  $\phi$  with a set of substitutions  $\sigma_{\text{ground}} = \bigcup_{O \in \mathbf{O}^-} \{\star\text{COURSE}/O\}$  where  $\mathbf{O}^- = \{CS12, CS13, CS21, CS22, CS23\}$ .  $\phi$  evaluates to true in a state if any of the literals, `passed(CS12)`, ..., `passed(CS23)`, it was mapped from is true in the state.

**Goal Context**

The goals  $\mathcal{G}$  in a problem are state predicates which represent the status of goals; the state predicate is true if the goal is achieved. We refer to these state predicates as **goal predicates**. In goal context, free variables are substituted with the objects in goal predicates.

**Definition 24 (Goal Context)**

The contextual grounding due to a goal  $g = \text{p}(X_1, \dots, X_n)$  is  $\sigma_{\text{goal}=g} =$

$\{\star X_1/x_1, \dots, \star X_n/x_n\}$ . The contextual grounding due to a set of goals  $\mathcal{G}$  is  $\sigma_{goal} = \bigcup_{g \in \mathcal{G}} \sigma_{goal=g}$ .

$\mathcal{G}$  can be trivially determined from the definition of terminal states or from the parameterised reward function described in RDDDL. We assume that they are known.

### Example 18 (Determine Goal Context for Recon)

In *Recon*, an immediate reward of 19 ( $-21$ ) is received for taking a good (bad) picture of an object. The state predicate `pictureTaken(OBJ)` represents the fact that the picture of the object `OBJ` has been taken. A terminal state is reached if the agent has taken pictures of all objects. The goal predicates are the ground literals of `pictureTaken(OBJ)` for every object of type `obj`. If a goal  $g$  is to take a good picture of an object `o1`, then  $\sigma_{goal=g} = \{\star OBJ/o1\}$ . Following Example 12, with goal context,  $|\sigma_\phi| = |\mathbf{O}^{obj}| \times |\mathbf{O}^{wp}| = 1 \times 16 = 16$  (reduction from 96).

### Example 19 (Determine Goal Context for Service Robot)

In *Service Robot*, the symbolic goal predicates are:

- `goal_object_with(OBJ, PERSON)`
- `goal_object_at(OBJ, WP)`

Suppose that the goals are to deliver items `o1` to the location `wp5` and `o2` to the person `p1`. The former is denoted by  $g_1 = \text{goal\_object\_at}(o1, wp5)$  and the latter by  $g_2 = \text{goal\_object\_with}(o2, p1)$ . The goal  $g_1$  involves the objects `o1` with type `obj` and `wp5` with type `wp`, and the corresponding contextual grounding is  $\sigma_{goal=g_1} = \{\star OBJ/o1, \star WP/wp5\}$ . Likewise,  $\sigma_{goal=g_2} = \{\star OBJ/o2, \star PERSON/p1\}$ .

### Example 20 (Determine Goal Context for Academic Advising)

In *Academic Advising*, the non-fluent `PROGRAM_REQUIREMENT(COURSE)` specifies a goal to pass `COURSE`. Thus, `PROGRAM_REQUIREMENT(COURSE)` is a goal predicate. If a goal  $g$  is to pass a course `C21`, then goal context gives a substitution  $\sigma_{goal=g} = \{\star COURSE/C21\}$ . If `C21` has a prerequisite `C11`, represented by the non-fluent `PREREQ(C11, C21)`, which is not a goal, then passing `C11` is a subgoal to achieve the goal  $g$ . However, goal context does not consider subgoals.

Each goal typically involves object(s) which is of the same type as other goals (e.g., *course* in *Academic Advising*). If every goal is considered, this might not mitigate the grounding ambiguity. In every domain we consider except for

**Academic Advising** and **Service Robot**, goal context will not reduce the set of objects to substitute a free variable with because the goals involve every object of a particular type. For example, in **Recon**, the goals are to take good pictures of every object of type *obj*, and in **Robot Fetch**, the goals are to place every object of type *obj* in their respective goal locations. Therefore, we limit the goals considered by goal context to only **active goals**. A goal is active if it is known and is not yet achieved.<sup>9</sup> We assume that achieved goals are no longer consequential for decision making and every active goal contributes to the expected return.

**Similarities with other work.** The Q-values after applying goal context for a goal represents the expected values of actions for achieving the goal. This is similar to the goal-associated Q-functions [183] or the universal value function approximators (UVFA) [151]. A goal-associated Q-function  $\tilde{Q}(s, a, g)$  approximates the Q-value of  $(s, a)$  in achieving a goal  $g$  while a UVFA  $V(s, g)$  is a goal-conditioned value function that approximates the value of a state  $s$  when trying to achieve a goal  $g$  from  $s$ . Since the number of goals often varies in problems of different scales, the number of goal-conditioned Q-functions and the number of UVFAs are equal to the number of goals and cannot be directly transferred to another problem. Veeriah, Oh, and Singh [183] and Schaul et al. [151] address this issue by generalising over goals. This requires the embedding of a goal into a real value vector. The embedding is straightforward for problems where the states are represented by images such as the Atari games [112] or real value vectors such as the angles and velocities of robot joints [4]. However, in relational problems, such embeddings are difficult and often require trial-and-error and extensive expert knowledge [53, 196]. Goal context is different from the additive rewards used in [150] where each goal is assumed to contribute uniformly and additively to the reward. In our work, a feature  $\phi(s, a) = 1$  if there exists a goal  $g$  with  $\sigma_{goal=g}$  which makes  $\phi$  true in the state  $s$ . If more than one such goal exists,  $\phi(s, a)$  remains at 1 and the Q-value is unaffected. **Q-RRL** [47] uses the objects in a goal predicate to ground free variables which is identical to our goal context. It considers only one goal predicate while goal context has no such limitation.

---

<sup>9</sup>In **Service Robot**, the goals are initially unknown to the robot.

## Location Context

For domains where agents move in an environment,  $\mathbf{P}$  contains state predicates which represent the location of the agent. Location context utilises these state predicates to identify the current location of the agent which is then used to substitute free variables. The intuition is that the locality of the agent is of interest, assuming that the agent can only interact with objects in its vicinity. Location context is particularly useful if there are many objects in  $\mathbf{O}$  which represent locations.

### Definition 25 (Location Context)

Given a symbolic state predicate  $\mathbf{p}(WP)$  which represents the location of an agent, location context gives a contextual grounding  $\sigma_{location} = \{\star WP/wp\}$  if  $\mathbf{p}(wp)$  is true (i.e., agent is at  $wp$ ).

### Example 21 (Determine Location Context for Recon)

In *Recon*, the agent  $a1$  uses tools on objects which are at the same location. If the agent is at  $wp1$  (i.e.,  $\mathbf{agentAt}(a1, wp1)$  is true), then the contextual grounding due to location context is  $\sigma_{location} = \{\star WP/wp1\}$ . The first-order feature  $\mathbf{OBJECT\_AT}(OBJ, \star WP)$  represents the location of  $OBJ$  while  $\mathbf{BASE}(WP)$  represents the location of the base. The first-order features  $\mathbf{OBJECT\_AT}(OBJ, \star WP)$  and  $\mathbf{BASE}(\star WP)$  are grounded to  $\mathbf{OBJECT\_AT}(OBJ, wp1)$  and  $\mathbf{BASE}(wp1)$ , respectively, using  $\sigma_{location}$ .  $\mathbf{OBJECT\_AT}(OBJ, \star WP)$  queries if the agent and the object are at the same location and  $\mathbf{BASE}(\star WP)$  queries if the agent and the base are at the same location. It is of no interest whether the object or the base is elsewhere other than the agent's current location. Due to location context, these first-order features are implicitly conjunctive (e.g.,  $\mathbf{OBJECT\_AT}(OBJ, \star WP) \wedge \mathbf{agentAt}(AGENT, \star WP)$ ). Following Example 12, with location context,  $|\sigma_\phi| = |\mathbf{O}^{obj}| \times |\mathbf{O}^{wp}| = 6 \times 1 = 6$  (reduction from 96).

## Combination of Contextual Knowledge

Different types of contextual knowledge can be combined. If contextual grounding from different types of contextual knowledge are in conflict, the order of applying the contextual grounding needs to be specified. We denote a contextual knowledge precedes another by the symbol  $\prec$  (e.g., *location*  $\prec$  *goal* denotes location context



precedes goal context). The ordered set of substitutions is then given to Algorithm 7 which resolves the conflicts and returns a set of substitutions.

**Example 22 (Goal Context and Location Context for Recon)**

From Example 18, the goal context gives  $\sigma_{goal=g} = \{\star OBJ/o1\}$ . From Example 21, the location context gives  $\sigma_{location} = \{\star WP/wp\}$ . When location context and goal context are combined,  $\sigma = \{\star OBJ/o1, \star WP/wp1\}$ . This queries if the agent is at the same location as the goal of interest. The order does not matter as there is no conflict between location context and goal context. Following Example 12,  $|\sigma_\phi| = |\mathcal{O}^{obj}| \times |\mathcal{O}^{wp}| = 1 \times 1 = 1$  (reduction from 96). In **Recon**, free variables are of three possible types: tool, obj, and wp. Goal context provides substitutions for  $\star OBJ$  and location context provides substitutions for  $\star WP$ .

**Example 23 (Conflict between Goal Context and Location Context in Robot Fetch)**

The non-fluent  $OBJECT\_GOAL(OBJ, WP)$  specifies a goal to put  $OBJ$  at  $WP$ . Thus,  $OBJECT\_GOAL(OBJ, WP)$  is a goal predicate. If a goal  $g$  is to put  $o1$  at  $wp1$ , then goal context gives a substitution  $\sigma_{goal=g} = \{\star OBJ/o1, \star WP/wp1\}$ . The state predicate  $robot\_at(ROBOT, WP)$  is true if the robot  $r1$  is at  $WP$ . Location context gives a substitution  $\sigma_{location} = \{\star WP/wp2\}$  if  $robot\_at(r1, wp2)$  is true. For goal  $\prec$  location,  $\bar{\sigma} = (\sigma_{goal=g}, \sigma_{location})$  and  $\sigma = \{\star OBJ/o1, \star WP/wp1\}$ . Location context is ignored as it gives  $\sigma = \{\star OBJ/o1, \star WP/\emptyset\}$ —the intersection  $\{wp1\} \cap \{wp2\} = \emptyset$ . For location  $\prec$  goal,  $\bar{\sigma} = (\sigma_{location}, \sigma_{goal=g})$  and  $\sigma = \{\star OBJ/o1, \star WP/wp2\}$ ; goal context only grounds  $\star OBJ$ . In **Robot Fetch**, free variables are of two possible types: obj and wp. Goal context provides substitutions for  $\star OBJ$  and  $\star WP$ , and location context provides substitutions for  $\star WP$ .

**Example 24 (Conflict between Goal Context and Location Context for Service Robot)**

In **Service Robot**, the state predicate  $robot\_at(ROBOT, WP)$  represents the location of the robot. If  $robot\_at(r1, wp4)$  is true, then location context gives  $\sigma_{location} = \{\star WP/wp4\}$ . Following Example 19, we can apply  $\sigma_{goal=g_2}$  and  $\sigma_{location}$  together without any conflict such that  $\bar{\sigma} = (\sigma_{location}, \sigma_{goal=g_2})$  and  $\sigma = \{\star WP/wp4, \star OBJ/o2, \star PERSON/p1\}$ . On the other hand,  $\sigma_{goal=g_1}$  and

$\sigma_{location}$  are conflicting as they ground  $\star WP$  to  $wp5$  and to  $wp4$ , respectively—the intersection  $\{wp5\} \cap \{wp4\} = \emptyset$ . For location  $\prec$  goal,  $\bar{\sigma} = (\sigma_{location}, \sigma_{goal=g1})$  and  $\sigma = \{\star OBJ/o1, \star WP/wp4\}$ . For goal  $\prec$  location,  $\bar{\sigma} = (\sigma_{goal=g1}, \sigma_{location})$  and  $\sigma = \{\star OBJ/o1, \star WP/wp5\}$ ; this uses only goal context. In **Service Robot**, free variables are of three possible types: *person*, *obj*, and *wp*. Goal context provides substitutions for  $\star OBJ$ ,  $\star PERSON$  (only for goals of the type `object_with(OBJ, PERSON)`), and  $\star WP$  (only for goals of the type `goal_object_at(OBJ, WP)`). Location context provides substitutions for  $\star WP$ .

## 3.5 Dealing with Plateaus

We have shown that the limited representational capacity of the first-order approximation can lead to plateaus. Here, we present two methods to resolve plateaus.

### 3.5.1 Model-Based Q-value Tree Expansion

The first method performs a search in the state space using a generative model to resolve plateaus. In a plateau, each greedy action leads to a possibly different state. The values of the next states can be used as a tiebreaker instead of a random selection; an action among the greedy actions is selected if its next state has the maximal value among the next states of other greedy actions. However, there could be plateaus at the next states too. To address this, we propose the Model-based Q-value Tree Expansion (MQTE) which is outlined in Algorithm 8. This work was published in [121]. MQTE estimates the multi-step Q-value of an action which serves as a tiebreaker to select an action among greedy actions in plateaus. The hyperparameter  $H_{MQTE}$  is the maximum number of time steps to look ahead. Since MQTE determines a most promising action to take among the greedy actions, it is used only if the policy wants to select a greedy action.

In the trivial case, there is only one greedy action. The action and its Q-value is returned (line 3). Otherwise, MQTE performs a breadth-first tree expansion starting from the root node which is a state node. A state node is created (line 5) with the following inputs:

1. the state  $s$ ,

2. the set of greedy actions in  $s$  ( $\mathbf{A}_{greedy}$ ),
3. the immediate reward for executing  $a$  to reach  $s$ ,
4. the value of the state ( $V$ ),
5. the probability of reaching  $s$  after executing  $a$ ,
6. the depth of the node in the search tree ( $d$ ), and
7. a boolean which is true if  $s$  is a terminal state ( $\top$  denotes true and  $\perp$  denotes false).

A queue is created with the root node  $n_{root}$  added (line 6). The state nodes in the queue are expanded in the order of first-in-first-out (lines 7 to 34). The expansion is terminated if the queue is empty (line 7), the depth of the search exceeds the maximum allowable depth (line 12), expanding the nodes in the queue will exceed the maximum number of expansions allowed (line 13), or the plateau at the root node has been resolved (line 16). The last condition is an early termination criterion which prevents unnecessary tree expansion. It requires a depth-first traversal of the search tree to evaluate the multi-step Q-values of greedy actions at the root node; this will be described later. An early termination reduces the computational cost of MQTE as the number of nodes expanded is exponential in the depth of the search tree. On the other hand, the computational cost of evaluating the Q-values is linear in the number of nodes and less expensive than model prediction during node expansion.

The generative model  $\mathcal{M}$  is used to predict the outcomes for executing each greedy action in the state for  $n_{state}$  (lines 19 to 22).  $outcomes$  is a set of tuples  $(s', r, Pr)$  where  $s'$  is the next state,  $r$  is the immediate reward, and  $Pr = \mathcal{T}(s'|s, a)$ . If  $\mathcal{M}$  fails to predict for any action, then  $n_{state}$  is not expanded (line 23) because the multi-step Q-values of every greedy action in  $n_{state}$  must be computed in order to resolve the plateau at  $n_{state}$ . The prediction could fail if  $\mathcal{M}$  is not the true model.

If the outcome for every greedy action can be predicted, then  $n_{state}$  is expanded. An action node is created (line 25). For each outcome, a state node  $n'_{state}$  for the next state  $s'$  is created (lines 28 and 31) and added to  $n_{action}$  as a child node (line 33). If  $s'$  is a terminal state, then expansion is no longer possible and  $n'_{state}$  is not added to the queue; a state node is created which has a value of 0 since there is no future return for terminal states. Otherwise, the greedy actions for  $s'$  is determined from  $\tilde{Q}$  (line 30) and  $n'_{state}$  is added to the queue (line 32).  $n_{action}$  is added as a

child node to  $n_{state}$  (line 34) after every outcome is added as its child node (line 33). This expands the search tree by one depth. Each expansion is a policy rollout for one time step. The root node is evaluated (line 35) with Algorithm 9 which returns the set of actions with the maximal multi-step Q-value.

Algorithm 9 computes the multi-step Q-value of every greedy action in a state by traversing the search tree which was expanded by Algorithm 8. If the state node is a leaf node, then the value of the state is the discounted expected return (line 4). The algorithm returns an empty set of greedy actions and the value of the state (line 5). If  $n_{state}$  is not a leaf node, then the multi-step Q-values of its greedy actions are evaluated (lines 7 to 15). The value of an action is evaluated (line 10) with Algorithm 10 which shall be described later. The set of actions with the maximal multi-step Q-value and the discounted expected return of the state are computed (lines 11 to 15). If this set of actions is a subset of the greedy actions in  $n_{state}$ , then MQTE has mitigated the plateau in  $n_{state}$ .

Algorithm 10 computes the multi-step Q-value of an action which is the sum of the expected returns for the next states resulting from the execution of the action weighed by the probability of the transition (line 5). The expected return of the next state is determined by Algorithm 9. Thus, Algorithms 9 and 10 perform a depth-first traversal of the search tree generated by Algorithm 8 until a leaf (state) node is reached. The value of a node is backpropagated to its parent: the value of an action node is the weighted sum of the values of its children (state) nodes and the value of a state node is the maximal value of its children (action) nodes.

Algorithm 8 returns a possibly reduced set of greedy actions and its multi-step Q-value (line 35). There could be more than one greedy actions returned. This is due to one or more of the following factors:

- There are multiple optimal actions (i.e., no plateau).
- $\mathcal{M}$  is unable to predict the next states and immediate rewards so tree expansion is not possible.
- The granularity of  $\tilde{Q}$  is too coarse to resolve plateaus even with multi-step estimates.
- $H_{MQTE}$  is too small for MQTE to expand to states which are not plateaus.

The policy selects a random action among the greedy actions. The performance

could improve if MQTE prunes at least one non-optimal action from the set of greedy actions.

We use the maximum likelihood model (see Section 2.3.1) for  $\mathcal{M}$  to make a prediction for  $(s_t, a_t)$  only if the number of observations for  $(s_t, a_t)$  exceeds a user-defined threshold,  $N_{known} > 0$ . In other words, *known* is true in line 20 of Algorithm 8 if the number of times  $a_t$  is executed in  $s_t$  exceeds  $N_{known}$ . Errors in predictions are compounded with each successive rollout (i.e., state node expansion) if  $\mathcal{M}$  is not the true model. We set  $H_{MQTE}$  with consideration to the computation time, the prediction errors, and the requirements of the problem. Due to the early termination criterion (line 16 in Algorithm 8), MQTE will not expand the search tree to the depth of  $H_{MQTE}$  if the plateau is resolved before then.

**Similarities with other work.** There are some similarities between MQTE and the model-based methods UCT [88], MVE [48], and STEVE [20]. MVE is a model-based RL method which uses a generative model to simulate the short-term horizon and Q-learning to estimate the long-term value. STEVE extends MVE by using an ensemble of generative models to predict the target values. MQTE acts as a tiebreaker for action selection in plateaus while MVE estimates higher-quality target values for Q-learning. MQTE performs a tree expansion where every greedy action is expanded while MVE considers only one action from a policy and UCT samples the state space with the UCB exploration bonus (see Section 2.4.2). Thus, they differ in the way the search tree is expanded and the purpose for which the search trees are utilised. MQTE is also less susceptible to approximation errors in  $\mathcal{M}$  or  $\tilde{Q}$ . If non-optimal actions are not eliminated by MQTE and are then selected by the policy, this can be regarded as exploration instead of exploitation. Exploration is preferred anyway if  $\mathcal{M}$  and/or  $\tilde{Q}$  have approximation errors in the current state or sampled next states.

### Example 25 (Resolving Plateaus in Grid Environments)

Following Example 16, we can use MQTE to resolve the plateau in the state  $s_1$  which is illustrated in Figure 3.4. In  $s_1$ , the agent  $a1$  is at  $wp2$  and needs to move to  $wp3$ , then to  $wp6$  where  $o1$  is at. Or, it can move to  $wp5$ , then to  $wp6$ . The optimal actions in  $s_1$  are  $\text{move}(a1, wp3)$  and  $\text{move}(a1, wp5)$ . However, there are three greedy actions:  $\mathbf{A}_{greedy} = \{\text{move}(a1, wp1), \text{move}(a1, wp3), \text{move}(a1, wp5)\}$ . MQTE performs a one step

rollout for each greedy action. The next states are  $s_2$ ,  $s_3$ , and  $s_4$ , where the agent is now at  $wp3$ ,  $wp5$ , and  $wp1$ , respectively. There is no plateau in  $s_2$  and  $s_3$  since  $o1$  is adjacent to the robot. The early termination criterion (line 16 in Algorithm 8) is not satisfied because there are two greedy actions in  $s_1$ :  $\text{move}(a1, wp3)$  and  $\text{move}(a1, wp5)$  (i.e.,  $\text{move}(a1, wp1)$  is eliminated). After an expansion to the depth of  $H_{MQTE}$ , Algorithm 8 returns these two actions for  $s_1$  which is a reduction from three greedy actions. Only one depth of expansion is required to resolve the plateau in  $s_1$ . If the agent is at  $wp1$  instead, then two depths of expansion are required. Thus,  $H_{MQTE}$  needs to be tuned with consideration to the nature of the problem.

### 3.5.2 Ensemble of Approximations

We now consider an ensemble of approximations which overcomes the limited representational capacity of the first-order approximation by including the ground approximation. An ensemble of ground and first-order approximations learns at different abstraction levels and combines the respective strength of each approximation—better generalisation due to the first-order approximation and finer granularity due to the ground approximation. This work was published in [120].

#### Definition 26 (Ensemble of Approximations)

*In an ensemble of Q-function approximations,  $\tilde{Q}$ , the policy is generated from a Q-function  $\tilde{Q}^\pi$  which returns a Q-value that is determined from the Q-values given by each approximation in the ensemble.*

In online RL, the approximations in an ensemble are learned interdependently; their features and weights are updated independently given observations which are dependent on the policy generated from  $\tilde{Q}^\pi$ . We propose two definitions of  $\tilde{Q}^\pi$ :

$$\tilde{Q}^\pi(s, a) = \frac{1}{2}\tilde{Q}^{gnd}(s, a) + \frac{1}{2}\tilde{Q}^{fo}(s, a), \quad (3.8)$$

$$\tilde{Q}^\pi(s, a) = \begin{cases} \tilde{Q}^{fo}(s, a) & \text{if } episode \leq SW \\ \tilde{Q}^{gnd}(s, a) & \text{otherwise} \end{cases} \quad (3.9)$$

where  $SW$  is a hyperparameter which sets the episode to switch the Q-function approximation which the policy is generated from. The policy generated from Equa-

tion 3.8 is denoted by  $\pi_{sum}$  and the policy generated from Equation 3.9 is denoted by  $\pi_{switch}$ .

Combining the Q-functions in Equation 3.8 resolves the plateaus due to a first-order approximation. While the Q-values from  $\tilde{Q}^{fo}$  are equal for multiple actions, those from  $\tilde{Q}^{gnd}$  are unlikely (because each ground action has its own weight components unlike in  $\tilde{Q}^{fo}$ ) and serve as a tiebreaker. On the other hand, we hypothesise that a first-order approximation will outperform a ground approximation initially due to better generalisation but has a worse asymptotic performance due to its coarser granularity. This motivates  $\pi_{switch}$ . A major limitation of the ensemble is that the ground approximation cannot be transferred and has to be learned from scratch. If a different problem is attempted in each episode, then the ensemble cannot be used to resolve plateaus.

## 3.6 Transfer Learning

Since a first-order approximation generates a generalised policy for abstract-equivalent problems, transfer learning is naturally achieved by directly transferring  $\tilde{Q}^{fo}$  from one problem to another of the same domain without any mapping between problems [174]. In large scale problems, meaningful observations are rare which makes learning difficult [192]. For example, consider a  $10 \times 10$  grid environment for a problem in *Recon*. The agent will wander around the environment most of the time, receiving an immediate reward of  $-1$ , until it stumbles on an object. Even then, the agent might not achieve the goal of taking a good picture of this object. This motivates transfer learning from small scale problems to large scale problems.

In an ensemble of ground and first-order approximations, we propose the following three modes of transfer learning:

1. Transfer  $\tilde{Q}^{fo}$  and keep it unchanged, use  $\pi_{switch}$ .
2. Transfer  $\tilde{Q}^{fo}$  and keep it unchanged, use  $\pi_{sum}$ .
3. Transfer  $\tilde{Q}^{fo}$  and update it online, use  $\pi_{sum}$ .

In (1), we transfer a learned  $\tilde{Q}^{fo}$  and use it as an exploratory policy to acquire meaningful observations to update  $\tilde{Q}^{gnd}$ , then switch to using  $\tilde{Q}^{gnd}$  to generate the policy when  $\tilde{Q}^{gnd}$  is sufficiently accurate. (2) is similar to (1) but  $\tilde{Q}^{fo}$  and  $\tilde{Q}^{gnd}$  are

considered by the policy in every episode. (3) differs from (1) and (2) in that  $\tilde{Q}^{fo}$  continues to be updated online in the current problem. In all of these modes,  $\tilde{Q}^{gnd}$  is learned online from scratch since transfer learning is typically not possible for a ground approximation.

These three modes have different pros and cons as we shall see in the empirical results discussed in Section 3.7.4. (1) could work well in domains where the first-order approximation has a poor asymptomatic performance by switching away from it. (2) and (3) could work well in other domains due to the combined strengths of ground and first-order approximations. While (3) has a higher computational cost than (2) due to the online update of two approximations, it is necessary if the source and target problems are rather different (e.g., they are not abstract-equivalent), necessitating the update of the transferred  $\tilde{Q}^{fo}$  in the target problems. We leave other possible modes of transfer learning for future work; we focus on these three modes which provide many interesting insights on learning at different abstraction levels (see Section 3.7.4).

A ground approximation is applicable in the source and target problems if they have the same set of objects and non-fluents are not used as base features. Since non-fluents are not the same in the source and target problems (else both problems are trivially similar), the ground approximation could generate a poor policy even if it is applicable in the target problem. This is because a ground approximation trained in the source problem generates a policy which considers the non-fluents in the source problem. In the target problem, where the non-fluents are different, it is plausible that the policy is poor.

**Example 26 (Non-Fluents as Base Features in a Ground Approximation)**

*In a particular problem  $P_{source}$  of **Recon**, there are two objects of type *obj*: *o1* and *o2*. The objects are located at *wp1* and *wp2* which are represented by the non-fluents `OBJECT_AT(o1, wp1)` and `OBJECT_AT(o2, wp2)`. Since the values of non-fluents do not change, it is not required to add them as base features in a ground approximation. Suppose that a *Q*-function approximation  $\tilde{Q}_{source}^{gnd}$  is learned in  $P_{source}$  and the policy guides the agent to move to the locations *wp1* and *wp2* to take pictures of *o1* and *o2*, respectively. In another problem,  $P_{target}$ , the set of objects are identical to  $P_{source}$  but the locations of *o1* and *o2* have changed to *wp3* and *wp4*, respectively. Since*



non-fluents are not features, the policy generated from  $\tilde{Q}_{source}^{gnd}$  guides the agent to  $wp1$  and  $wp2$  instead of  $wp3$  and  $wp4$ . Thus, transfer learning will perform poorly here.

Suppose that non-fluents are included as base features. Then,  $\tilde{Q}_{source}^{gnd}$  is not applicable in  $P_{target}$  since its base features,  $OBJECT\_AT(o1, wp1)$  and  $OBJECT\_AT(o2, wp2)$ , are not in the set of state predicates for  $P_{target}$ . If we ignore this issue and simply evaluate these two features (and any conjunctive features which include them) as false in  $P_{target}$ , the policy generated from  $\tilde{Q}_{source}^{gnd}$  will still not guide the agent to  $wp3$  and  $wp4$ .

## 3.7 Empirical Evaluation and Discussion

### 3.7.1 Experimental Setup

We describe the experimental setup for all empirical studies in this dissertation. The domains and problems are described in Section 2.8. Any experiments which deviate from this setup will be mentioned when presenting the results. We evaluate the performance by the total undiscounted reward received in each episode, the goal-directedness, and the computation time. The goal-directedness is measured by the cumulative number of goal states reached minus by the cumulative number of dead ends reached. Only `Triangle Tireworld` and `Robot Inspection` have dead ends. For the other domains, the goal-directedness is simply the cumulative number of goal states reached. Results are averaged over ten independent runs and the (one) standard deviations are represented by shaded areas in the figures. Due to the stochastic nature of the problems, line plots for the total undiscounted reward are averaged with a moving window of ten episodes to smooth the curves for better visual clarity and statistical comparisons.  $\theta$ ,  $\Phi$ ,  $\Phi^c$ , and  $\eta$  are updated across episodes while no information is exchanged between runs.

**Assumptions.** We assume that preconditions are known and an inapplicable action will never be selected for execution. This simplifies the problem and reduces the sample complexity but does not affect the analysis as comparisons are made across benchmarks which benefit from the same assumption.

**Algorithm 8:** Model-Based Q-value Tree Expansion (MQTE)

---

```

1  Function MQTE( $\mathcal{M}, \tilde{Q}, \mathbf{A}, s, H_{MQTE}, N_{max}, \gamma$ ):
   Input: Generative model  $\mathcal{M}$ ,
           Q-function approximation  $\tilde{Q}$ ,
           Set of actions  $\mathbf{A}$ ,
           State  $s$ ,
           Maximum depth to expand  $H_{MQTE}$ ,
           Maximum number of nodes to expand  $N_{max}$ ,
           Discount factor  $\gamma$ 
2   $\mathbf{A}_{greedy} = \arg \max_{a \in \mathbf{A}} \tilde{Q}(s, a)$ ,  $V = \max_{a \in \mathbf{A}} \tilde{Q}(s, a)$ 
3  if  $|\mathbf{A}_{greedy}| = 1$  then return ( $\mathbf{A}_{greedy}, V$ )
4   $d = 0$ ,  $N = 0$ 
5   $n_{root} \leftarrow \text{Create\_State\_Node}(s, \mathbf{A}_{greedy}, 0, V, 1, d, \perp)$ 
6   $queue = \{n_{root}\}$ 
7  while  $queue \neq \emptyset$  do
8       $n_{state} \leftarrow \text{Pop front of } queue$ 
9       $d_n \leftarrow \text{Get search depth of } n_{state}$ 
10     if  $d_n > d$  then
11          $d = d_n$ 
12         if  $d_n \geq H_{MQTE}$  then break
13         else if  $N + |queue| \geq N_{max}$  then break
14         else
15              $(\mathbf{A}_{greedy}, V) \leftarrow \text{Evaluate\_State\_Node}(n_{root}, \gamma)$ 
16             if  $|\mathbf{A}_{greedy}| = 1$  then return ( $\mathbf{A}_{greedy}, V$ )
17      $known = \top$ 
18      $all\_outcomes = \emptyset$ 
19     for  $a \in \text{Get\_Greedy\_Actions}(n_{state})$  do
20          $(outcomes, known) \leftarrow \mathcal{M}(s, a)$ 
21         if  $\neg known$  then break
22         Append  $(a, outcomes)$  to  $all\_outcomes$ 
23     if  $\neg known$  then continue
24     for  $(a, outcomes) \in all\_outcomes$  do
25          $n_{action} \leftarrow \text{Create\_Action\_Node}(a)$ 
26         for  $(s', r, Pr) \in outcomes$  do
27             if  $s'$  is terminal state then
28                  $n'_{state} \leftarrow \text{Create\_State\_Node}(s', \emptyset, r, 0, Pr, d_n + 1, \top)$ 
29             else
30                  $\mathbf{A}_{greedy} = \arg \max_{a \in \mathbf{A}} \tilde{Q}(s', a)$ ,  $V = \max_{a \in \mathbf{A}} \tilde{Q}(s', a)$ 
31                  $n'_{state} \leftarrow \text{Create\_State\_Node}(s', \mathbf{A}_{greedy}, r, V, Pr, d_n + 1, \perp)$ 
32                 Append  $n'_{state}$  to  $queue$ 
33             Add  $n'_{state}$  as child to  $n_{action}$ 
34         Add  $n_{action}$  as child to  $n_{state}$ 
35 return Evaluate_State_Node( $n_{root}, \gamma$ )

```

---

---

**Algorithm 9:** Evaluate state node for MQTE

---

```

1  Function Evaluate_State_Node( $n_{state}, \gamma$ ):
   Input: State node  $n_{state}$ , Discount factor  $\gamma$ 
2   $d \leftarrow$  Get_Depth( $n_{state}$ )
3  if  $n_{state}$  is terminal node or  $Get\_Children(n_{state}) = \emptyset$  then
4  |    $V \leftarrow \gamma^d \cdot Get\_Reward(n_{state}) + \gamma^{d+1} \cdot Get\_Value(n_{state})$ 
5  |   return ( $\emptyset, V$ )
6  else
7  |    $V_{max} = -\inf$ 
8  |    $\mathbf{A}_{greedy} = \emptyset$ 
9  |   for  $n_{child} \in Get\_Children(n_{state})$  do
10 |   |    $V \leftarrow$  Evaluate_Action_Node( $n_{child}, \gamma$ )
11 |   |   if  $V > V_{max}$  then
12 |   |   |    $\mathbf{A}_{greedy} = \{Get\_Action(n_{child})\}$ 
13 |   |   |    $V_{max} = V$ 
14 |   |   else if  $V = V_{max}$  then
15 |   |   |   Append action in  $n_{child}$  to  $\mathbf{A}_{greedy}$ 
16 |   return ( $\mathbf{A}_{greedy}, \gamma^d \cdot Get\_Reward(n_{state}) + \gamma^{d+1} V_{max}$ )

```

---



---

**Algorithm 10:** Evaluate value of action node for MQTE

---

```

1  Function Evaluate_Action_Node( $n_{action}, \gamma$ ):
   Input: Action node  $n_{action}$ , Discount factor  $\gamma$ 
2   $V = 0$ 
3  for  $n_{child} \in Get\_Children(n_{action})$  do
4  |   ( $\mathbf{A}_{greedy}, V_{child}$ )  $\leftarrow$  Evaluate_State_Node( $n_{child}, \gamma$ )
5  |    $V \leftarrow V + Get\_Transition\_Probability(n_{child}) \cdot V_{child}$ 
6  return  $V$ 

```

---

Hyperparameter	Value	Description
$\alpha$	0.3	Learning rate, linearly decayed over episodes to 0.05
$\epsilon$	1 (0.2 for transfer learning)	For $\epsilon$ -greedy policy, exponentially decayed over episodes to 0
$\gamma$	0.9	Discount factor
$\lambda$	0.7	Decay rate of eligibility traces
$\tau$	5	For $\tau$ -iFDD+
$\xi$	0.01 ( $\tau$ -iFDD+) or 3 (iFDD+)	Discovery threshold
$N_{known}$	3	Threshold for maximum likelihood model to make a prediction
$N_{max}$	200	Maximum number of nodes that can be expanded in MQTE
$H_{MQTE}$	3	Maximum look-ahead horizon for MQTE
$H$	20 to 60	Time horizon, depends on the problem (see Section 2.8)
Random seed	1 to 10	Results are aggregated over ten problems which use a different random seed each

Table 3.1: Values of hyperparameters used in experiments.

**Hyperparameters.** We use the  $\epsilon$ -greedy policy. The values of the hyperparameters are shown in Table 3.1. The values of  $\epsilon$ ,  $\alpha$ ,  $\gamma$ , and  $\lambda$  are typical for online RL.

**Transfer learning setup.** The source problems are small scale problems, AA3, RC3, TT3, RF1, RI1, and SR1, and the target problems are large scale problems, AA5, RC6, TT6, RF2, RI2, SR2, and SR3. The knowledge is learned from scratch in a source problem and then transferred to a target problem. Each independent run involves a pair of source and target problems and each run uses a different randomised problem except for Academic Advising and Triangle Tireworld which do not have randomised problems. The transferred knowledge continues to be updated (learned) in the target problems unless specified otherwise. An example of a transferred knowledge is the first-order approximation. Other types of knowledge shall be introduced in Chapter 4. The hyperparameters and the number of episodes are the same in both source and target problems except that exploration is reduced in the latter with  $\epsilon = 0.2$ . Crucially, for transfer of first-order approximations, the contextual knowledge must be the same in the source and target problems. The contextual knowledge used for each domain is listed in Table 3.3. This is determined with an ablation study which is discussed in Section 3.7.2.

**Simulated environment and hardware.** We use RDDLSim [149] to simulate the environment. RDDLSim receives the action executed by the agent, and returns the next state and immediate reward based on the preconditions, CPFs, and reward function defined in the RDDDL domain. Experiments are conducted on a machine with an 8 core Intel Xeon E5-2660 v3 2.60 GHz processor and 32 gigabytes of memory. Each experiment uses only one core and at most 8 gigabytes of memory.

**Abstract-equivalent problems.** Except for SR1, all problems considered are abstract-equivalent as we do not consider trivially small scale problems (e.g., only one object of a type). The only type which has one object is *agent* in Recon and *robot* in Robot Inspection and Robot Fetch. Problems which have more than one object of these types will not be abstract-equivalent but such problems are multi-agent problems. Transfer learning from SR1 to SR2 or SR3 requires the augmentation

of first-order base features which is described in Section 3.3.3.

**Benchmarks with other work.** The survey papers [174] and [176] for transfer learning in RL did not compare different work but evaluated each work on its own merit. This is due to the difficulty in comparing different work [174] as each work:

- utilises different assumptions (e.g., allowed dissimilarity in tasks),
- transfers different forms of knowledge (e.g., policies [172], models, options [30, 95], Q-functions [64, 141, 156]),
- addresses different classes of problems (e.g., discrete domains, continuous domains, relational domains), and
- allows different learners (e.g., TD learning, linear programming, policy search).

We discussed and compared existing model-free RRL methods with LQ-RRL in Section 2.6.1. It is difficult to use them as benchmarks because they use:

- different problem representations (Prolog [30, 42–45, 47], etc. [94, 95, 113]),
- supervised learning instead of RL [184, 192],
- continuous state variables [192], and
- background knowledge (expert guidance [41], handcrafted features [47, 156], distance metric [147], task hierarchy [145], etc. [94, 113, 179]).

Q-RRL and its various extensions solve problems described in Prolog. We note that much of this work is tested on the `Blocks World` domain (and almost never on any other domains) which is described in Prolog. Nevertheless, we evaluated LQ-RRL against two RRL methods, TG [44] and RIB [45], on `Blocks World`. This is discussed in Section C.1.2.

### 3.7.2 Ablation Study for Contextual Knowledge

Due to a multitude of hyperparameters, we first perform an ablation study to investigate the efficacy of free variables, contextual knowledge (see Section 3.4.2), and first-order approximation. We use these results to determine the most suitable combination of contextual knowledge for each domain which will then be used in subsequent experiments. The baseline in our ablation study is the ground approximation. We use  $\tau$ -iFDD+ where  $\tau = 5$ . The choice of  $\tau$  is determined after a sensitivity analysis. This will be elaborated on at the end of this section.

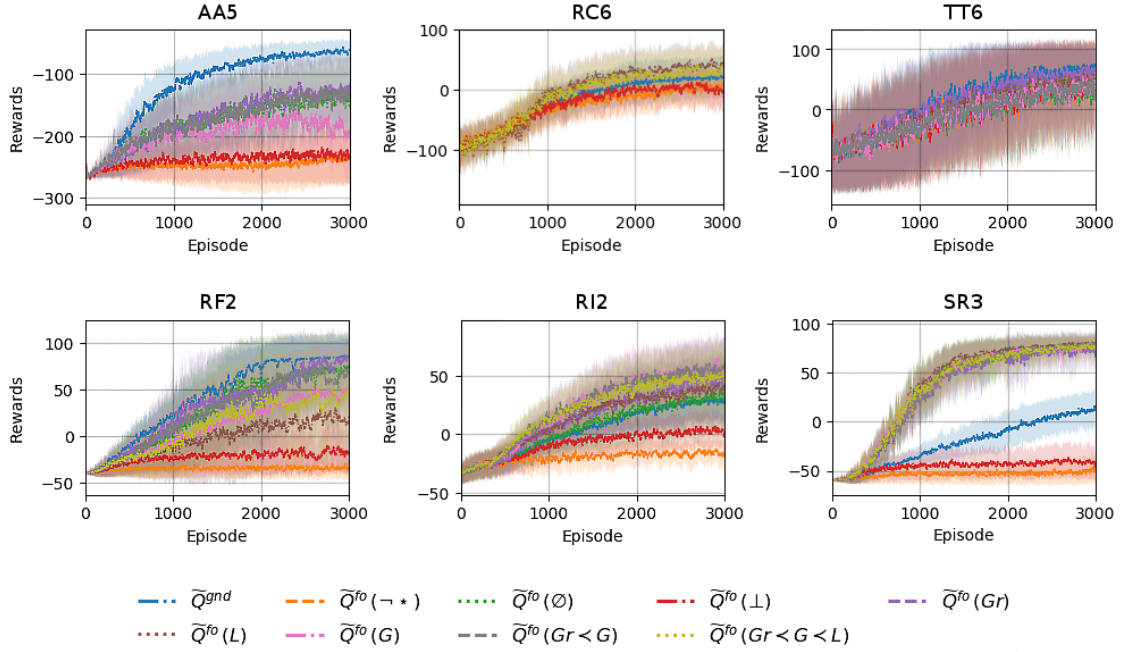


Figure 3.5: Ablation study involving large scale problems from six domains. The sample complexity is indicated by the total undiscounted rewards received in each episode.

Figures 3.5, 3.6, and 3.7 show the results for the ablation study. Table 3.2 summarises the results, showing the ranking of each contextual knowledge and their combinations. Table 3.3 shows the most suitable contextual knowledge for each domain, taking into account its computation time.

We compare the following approaches to deal with free variables:

- Features with free variables are not added to  $\Phi$  (denoted by  $\tilde{Q}^{fo}(\neg\star)$ ).
- No contextual knowledge is used ( $\tilde{Q}^{fo}(\emptyset)$ ).
- A free variable is always substituted with the same object which is selected arbitrary ( $\tilde{Q}^{fo}(\perp)$ ).
- Use contextual knowledge.

**Utility of free variables.** We examine the performance of first-order approximation where only features which do not contain free variables are used. This is denoted by  $\tilde{Q}^{fo}(\neg\star)$ . It has the worst performance in all problems except TT6 though it still performs worse than the first-order approximation with ground context. Therefore, free variables are important in order to approximate the optimal Q-function. The grounding of first-order features with free variables is the most expensive computation associated with a first-order approximation. However, Figure 3.7 shows that

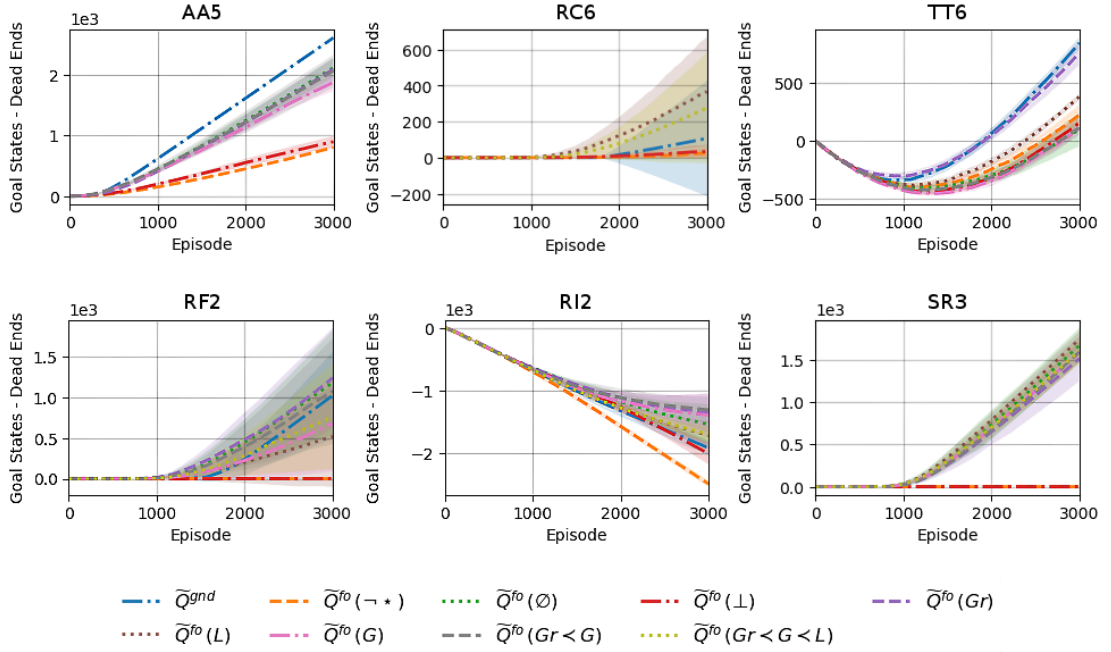


Figure 3.6: Ablation study involving large scale problems from six domains. The goal-directedness is measured by the cumulative number of goal states reached minus the cumulative number of dead ends reached.

the computation time of  $\tilde{Q}^{fo}(\neg\star)$  is relatively high in AA5, RC6, RF2, and RI2. This is because in most of the episodes, the goal state is not reached (see Figure 3.6); there is no further computation for an episode after the goal state is reached.

**Utility of contextual knowledge.** We have established that free variables are necessary. Next, we examine the utility of contextual knowledge for grounding first-order features with free variables. We introduced three types of contextual knowledge in Section 3.4.2: ground context (**Gr**), location context (**L**), and goal context (**G**). We tested these types of contextual knowledge and their various combinations. Their performance in terms of the total undiscounted rewards (see Figure 3.5) are largely similar in RC6, TT6, and SR3 though their performance in terms of goal-directedness differ. If different types or combinations of contextual knowledge give similar performance, then perhaps contextual knowledge is not useful. We investigate this with two treatments for free variables: (1) each free variable is grounded to the same arbitrary object ( $\tilde{Q}^{fo}(\perp)$ ) and (2) no contextual knowledge is used ( $\tilde{Q}^{fo}(\emptyset)$ ).  $\tilde{Q}^{fo}(\perp)$  fares slightly better than  $\tilde{Q}^{fo}(\neg\star)$  but otherwise performs poorly in all problems. On the other hand,  $\tilde{Q}^{fo}(\emptyset)$  performs comparably with the best contextual knowledge in AA5, RF2, and SR3. However, the computation time



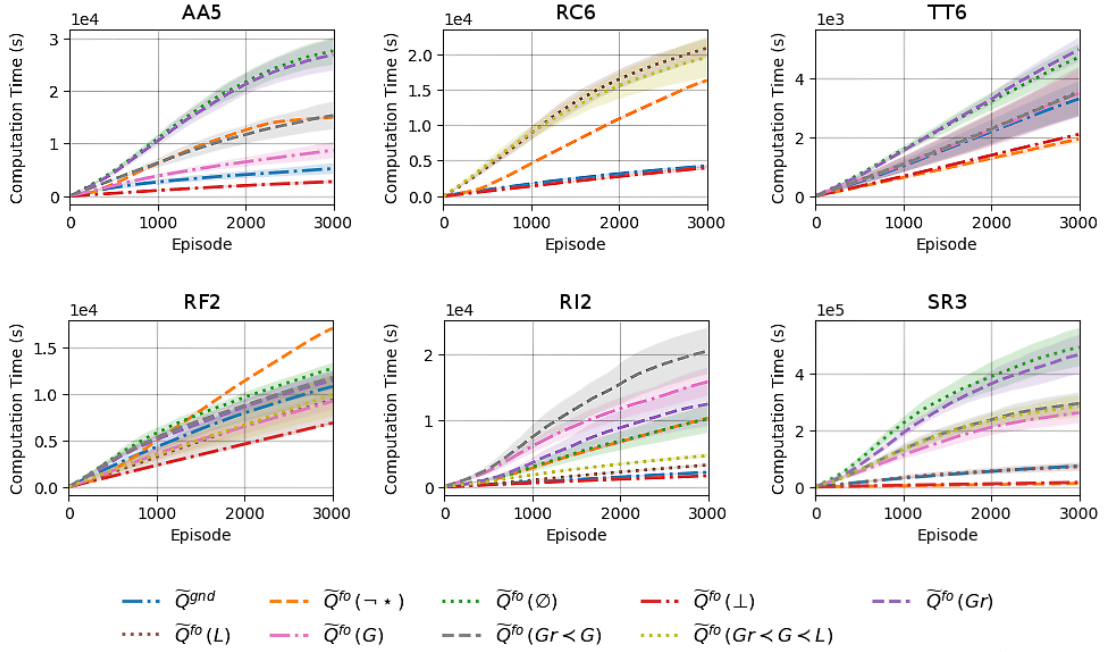


Figure 3.7: Ablation study involving large scale problems from six domains. The computational cost is measured by the cumulative computation time taken in seconds.

of  $\tilde{Q}^{fo}(\emptyset)$  is prohibitively large. In **Recon**, only results where location context is used are available. Due to the grid environment in **Recon**, there is a large number of locations (e.g., 25 locations in **RC6**). The computation time of Algorithm 6 to substitute the free variable  $\star WP$  without location context is prohibitively high.

In general, a first-order approximation has the lowest sample complexity and computation time when ground context, goal context, and location context (if applicable) are used together ( $\tilde{Q}^{fo}(\text{Gr} \prec \text{G} \prec \text{L})$ ). Ground context precedes the other two contextual knowledge as it is applicable in all problems. Also, the number of objects it substitutes a free variable with is typically larger than the other two contextual knowledge; location context substitutes with one object and goal context substitutes with a number of objects equal to the number of active goals. In an ordered set of substitutions, a substitution is not applied if it results in an empty set for the grounding of a free variable (see line 9 in Algorithm 7). Thus, it is preferable for ground context to be considered first as it substitutes free variables with a larger set of objects.

We now compare the performance of different contextual knowledge. In **AA5** and **TT6**, ground context performs best but has the highest computation time. Inter-

Domain	Gr	L	G	Gr $\prec$ G	Gr $\prec$ G $\prec$ L
Academic Advising	$\times$ , 1	NA	$\times$ , 3	$\times$ , 1	NA
Recon	—	$\checkmark$ , 1	—	—	$\checkmark$ , 2
Triangle Tireworld	$\times$ , 1	$\times$ , 2	$\times$ , 3	$\times$ , 3	NA
Robot Fetch	$\times$ , 1	$\times$ , 4	$\times$ , 3	$\times$ , 2	NA
Robot Inspection	$\checkmark$ , 4	$\checkmark$ , 5	$\checkmark$ , 1	$\checkmark$ , 1	$\checkmark$ , 3
Service Robot	$\checkmark$ , 2	$\checkmark$ , 1	$\checkmark$ , 2	$\checkmark$ , 2	$\checkmark$ , 2

Table 3.2: The ranking of contextual knowledge based on the total rewards with goal-directedness as a tiebreaker. The number indicates the rank with 1 being the best, the symbol  $\checkmark$  ( $\times$ ) indicates that the first-order approximation with contextual knowledge performs better (worse) than the ground approximation, NA indicates that the contextual knowledge is not applicable, and — indicates that the result is not available due to high computational costs. **Gr** denotes ground context, **L** denotes location context, and **G** denotes goal context.

Domain	Contextual Knowledge
Academic Advising	G
Recon	Gr $\prec$ G $\prec$ L
Triangle Tireworld	G
Robot Fetch	Gr $\prec$ G
Robot Inspection	Gr $\prec$ G $\prec$ L
Service Robot	Gr $\prec$ G $\prec$ L

Table 3.3: The contextual knowledge used for each domain in all experiments unless stated otherwise. **Gr** denotes ground context, **G** denotes goal context, and **L** denotes location context.

esting, ground context has a comparable computation time as  $\tilde{Q}^{fo}(\emptyset)$  yet gives a significant improvement in performance. The same performance but with a lower computation time is achieved in **AA5** by combining ground context with goal context (**Gr  $\prec$  G**). In **Triangle Tireworld** an immediate reward of  $-100$  is received when a dead end is reached. This causes large fluctuations in Figure 3.5 making it difficult to ascertain the best contextual knowledge. Nevertheless, it can be observed in Figure 3.6 that location context performs slightly worse than ground context but has a lower computation time (the plot for **L** is obscured by **G** and **Gr  $\prec$  G** in Figure 3.7). Goal context and location context are in conflict in **Triangle Tireworld** and cannot be used together as they substitute the same free variable  $\star WP$  and no other free variables. They are also in conflict in **Service Robot** but can be used together (see

Example 24). We tested  $\text{Gr} \prec \text{G} \prec \text{L}$  and  $\text{Gr} \prec \text{L} \prec \text{G}$  and found the former to have a slightly better performance. We omit the results for the latter for better clarity in the figures. In other domains where location context and goal context are not in conflict,  $\text{Gr} \prec \text{G} \prec \text{L}$  and  $\text{Gr} \prec \text{L} \prec \text{G}$  are equivalent.

**Comparing ground and first-order approximations.** First-order approximation significantly outperforms ground approximation in **RI2** and **SR3**. Although the first-order approximation only achieves marginally higher rewards than ground approximation in **RC6**, it clearly outperforms ground approximation in goal-directedness. In **RF2**, the ground approximation outperforms first-order approximation in rewards received but is worse in goal-directedness. This indicates that with a first-order approximation, more goal states are reached but with a longer sequence of actions (executing an action gives an immediate reward of  $-1$ ). In **AA5**, the ground approximation significantly outperforms the first-order approximation. The results for **AA5** and **RF2** demonstrate the pitfalls of using a first-order approximation. First-order approximation assumes independent goals and goal context considers only active goals. Both assumptions are violated in **Academic Advising** and **Robot Fetch**. These are discussed in Examples 13 and 14.

Plateaus are another cause of performance deterioration in a first-order approximation. **Academic Advising**, **Robot Fetch**, **Recon**, and **Triangle Tireworld** have plateaus. Plateaus in **Academic Advising** and **Robot Fetch** are due to the limited representative capacity of the first-order approximation as described in Example 15. Plateaus in **Recon** is discussed in Example 16. In **Triangle Tireworld**, if the vehicle moves along a shorter path, there are locations farther down the path where there are no spare tires and dead ends are unavoidable. The shorter a path is, the higher the probability of reaching a dead end which gives an immediate reward of  $-100$ . Similar to the scenario described in Example 15, there is no first-order feature which represents the availability of spare tires in locations other than the vehicle’s current location and adjacent locations. Thus, the shorter path and the longer path are equally attractive (i.e., a plateau) and the generated policy cannot avoid dead ends farther down the path. We present results for resolving plateaus in Section 3.7.3 and a method to deal with dead ends in Section 4.4.1.

In the remainder of our experiments, the first-order approximation uses the most suitable contextual knowledge in terms of sample complexity and computation time for each domain. This is listed in Table 3.3.

**Sensitivity analysis.** A sensitivity analysis for the hyperparameters  $\xi$  and  $\tau$  is discussed in Section C.1.3. We concluded that it is difficult to determine the optimal value of  $\xi$  for `iFDD+` due to its sensitivity. Conversely, it is relatively trivial to set the value of  $\tau$  for  $\tau$ -`iFDD+` as it does not impact performance as much. Thus, we use  $\tau$ -`iFDD+` with  $\tau = 5$  in subsequent experiments as it performs well in all problems and does not add too much features.

### 3.7.3 Resolving Plateaus

We investigate three methods to resolve plateaus due to a first-order approximation: (1) separate or decoupled weights for each action (denoted by `DW`), (2) `MQTE` (see Section 3.5.1), and (3) an ensemble of ground and first-order approximations (see Section 3.5.2). For `DW`, weights of first-order features are not shared among ground actions of the same symbolic action,  $\mathbf{A}_{\hat{a}}$ . This does not allow transfer learning as ground actions are defined over the set of objects (see Section 3.2). However, since plateaus in a first-order approximation are directly caused by the shared weights between actions in  $\mathbf{A}_{\hat{a}}$ , having separate weights is a straightforward and simple solution. For `MQTE`, we set  $H_{MQTE} = 3$  and use the maximum likelihood model which is learned online from scratch. For the ensemble, we use ground approximation and first-order approximation.

As discussed in Sections 3.4.1 and 3.7.2, there are plateaus in `Recon`, `Academic Advising`, `Triangle Tireworld`, and `Robot Fetch`. Therefore, we focus our analysis on these four domains. The results are shown in Figure 3.8. The baselines are the ground approximation ( $\tilde{Q}^{gnd}$ ) and the first-order approximation without using any of the three methods to resolve plateaus ( $\tilde{Q}^{fo}$ ). A summary of the results is outlined in Table 3.4 while statistics for the usage of `MQTE` are shown in Table 3.5.

In `AA5`, `MQTE` performs marginally better than the baseline  $\tilde{Q}^{fo}$ ; we shall see why this is so later. `DW` performs better than `MQTE`. The best performance is the ensemble

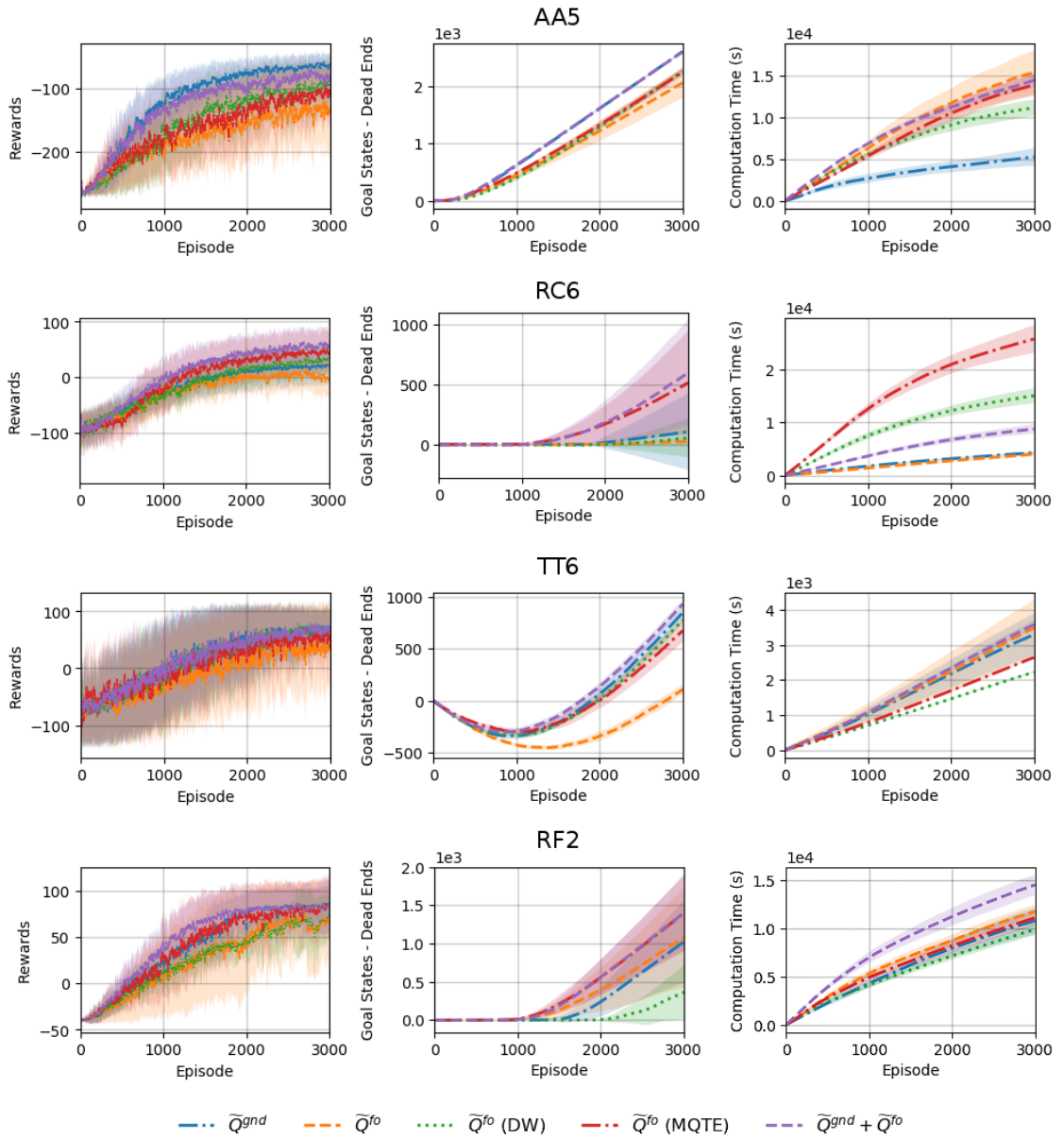


Figure 3.8: Performance of different methods for resolving plateaus in the first-order approximation. The methods are (1) using decoupled weights ( $\tilde{Q}^{fo}$  (DW)), (2) using MQTE ( $\tilde{Q}^{fo}$  (MQTE)), and (3) using an ensemble of ground and first-order approximations ( $\tilde{Q}^{gnd} + \tilde{Q}^{fo}$ ). The baselines are ground approximation ( $\tilde{Q}^{gnd}$ ) where plateaus are unlikely and first-order approximation ( $\tilde{Q}^{fo}$ ) where plateaus are not resolved.

Domain	Decoupled Weights (DW)	MQTE	Ensemble
Academic Advising	2	3	1
Recon	×	✓, 2	✓, 1
Triangle Tireworld	2	3	✓, 1
Robot Fetch	×	✓, 2	✓, 1

Table 3.4: Summary of the results for using different methods to resolve plateaus in the first-order approximation. The methods are (A) using decoupled weights, (B) using MQTE, and (C) using an ensemble of ground and first-order approximations. The symbols ✓ indicates that the method outperforms the ground approximation, × indicates that it worsens performance (relative to not using it at all), and a number indicates its ranking among all methods if it improves performance.

122

Problem	One action	Lesser actions	No change	Fail to predict	% Effectiveness	% Real Effectiveness
AA5	$3396.3 \pm 1266.2$	$75.6 \pm 96.6$	$971.5 \pm 569.1$	$22936.8 \pm 4277.9$	$77.2 \pm 11.9$	$13.1 \pm 5.5$
RC6	$4057.8 \pm 2016.5$	$1267.4 \pm 1178.6$	$3689.7 \pm 2010.4$	$9004.5 \pm 2111.4$	$60.4 \pm 17.3$	$30.1 \pm 10.3$
TT6	$3715.3 \pm 121.4$	$0.0 \pm 0.0$	$3.4 \pm 2.2$	$915.9 \pm 47.6$	$99.9 \pm 0.01$	$80.2 \pm 0.9$
RF2	$4746.7 \pm 3426.4$	$1283.0 \pm 1497.4$	$10475.1 \pm 3535.8$	$4120.4 \pm 858.9$	$36.8 \pm 20.1$	$29.6 \pm 16.2$

Table 3.5: Mean and the one standard deviation of six metrics accumulated in 3000 episodes and aggregated over ten problems: “One action” is the number of times MQTE has reduced the number of greedy actions to one, “Lesser actions” is the number of times MQTE has reduced the number of greedy actions but not to one, “No change” is the number of times MQTE has not reduced the number of greedy actions, “Fail to predict” is the number of times MQTE has not reduced the number of greedy actions due to the inability of the generative model to predict the outcomes, “% Effectiveness” is the percentage of times MQTE has reduced the number of greedy actions notwithstanding the failure of the model to predict, and “% Real Effectiveness” is the percentage of times MQTE has reduced the number of greedy actions. The first four metrics are mutually exclusive.

which fares slightly worse than the baseline  $\tilde{Q}^{gnd}$ . Therefore, the inclusion of  $\tilde{Q}^{fo}$  in the ensemble deteriorates performance. However,  $\tilde{Q}^{fo}$  is a generalised Q-function which allows transfer learning while  $\tilde{Q}^{gnd}$  does not. The computation time of the three methods to mitigate plateaus is comparable with the baseline  $\tilde{Q}^{fo}$ .

In **RC6**, **MQTE** and the ensemble outperform the baselines. The ensemble has the best performance suggesting that the combination of ground and first-order approximations achieves their respective strengths of generalisation and fine granularity. The computation time of the ensemble is also lower than **MQTE** and **DW**. **MQTE** has the highest cost because plateaus are prevalent in grid environments. Thus, **MQTE** expands the search tree to a depth of  $H_{MQTE} = 3$  in most of the states visited. It is worth noting that the baselines and **DW** perform poorly in terms of the number of goal states reached. The results for **RF2** are similar to those for **RC6**. A key difference is that the computation time for **MQTE** is comparable with the baselines because there is no grid environment in **Robot Fetch** and **MQTE** does not need to expand the search tree to many depths. Instead, the computation time of the ensemble is the highest in **RF2** as two approximations are learned.

In **TT6**, the three methods perform comparably with the baseline  $\tilde{Q}^{gnd}$  and have similar computation time. They also outperform the baseline  $\tilde{Q}^{fo}$ . This shows that the three methods are able to effectively resolve the plateaus. While the first-order approximation now performs comparably with a ground approximation, the former is more attractive as it enables transfer learning.

Table 3.4 summarises the discussion above. Clearly, the ensemble is the best method to resolve plateaus while **MQTE** is promising in **Recon** and **Robot Fetch**. The use of decoupled weights has mixed results. For transfer learning, the maximum likelihood model used here for **MQTE**, the ground approximation in the ensemble, and the first-order approximation with decoupled weights cannot be transferred. In Section 2.3.3, we discussed relational models which make predictions in all problems of a domain in contrast to the maximum likelihood model. This makes **MQTE** a more attractive solution to plateaus than the ensemble.

### Statistics for MQTE

Table 3.5 shows the statistics for the usage of MQTE which provide further insights to its performance. In our experiments, we assume that a state is a plateau if there is more than one greedy actions though this might not be the case if these actions are optimal (see Definition 22). If the states are not plateaus, then MQTE will not be able to reduce the number of greedy actions. In other words, the values for “No change” in Table 3.5 could be overestimated and the results here represent a lower bound on the performance. MQTE has successfully reduced the number of greedy actions in 13.1%, 30.1%, 80.2%, and 29.6% of the plateaus encountered in AA5, RC6, TT6, and RF2, respectively. We denote this metric by “% real effectiveness”. If we discount the number of times MQTE has failed to reduce the number of greedy actions due to the failure of the generative model in making a prediction, then the effectiveness increases to 77.2%, 60.4%, 99.9%, and 36.8%, respectively. We denote this metric by “% effectiveness”. The effectiveness is a significant increase over the real effectiveness which implies that MQTE is heavily reliant on the generative model which is expected of a model-based method. If the model is available beforehand instead of learning it online, then we can expect the effectiveness to be a more accurate indicator of the expected performance of MQTE.

In AA5, the real effectiveness is only 13.1% and this results in a marginal improvement in the sample complexity due to MQTE as shown in Figure 3.8. In comparison, the effectiveness is 77.2%. The reason for the high failure of model prediction is because there are 21 ground actions of `takeCourse(COURSE)` in AA5 and they have no preconditions. This means that the maximum likelihood model needs more observations to make a prediction (recall that a state-action pair needs to be observed at least  $N_{known} = 3$  times before a prediction can be made for it). In RF2, the real effectiveness is rather low at 29.6% but the reduction in sample complexity due to MQTE is significant. MQTE has the highest effectiveness and real effectiveness in TT6. This is because the number of applicable actions in the states of TT6 is small relative to the other problems. Furthermore, the transition function for `Triangle Tireworld` is simpler; actions lead to a small number of states (usually one or two). This is reflected by the high real effectiveness of 80.2%.

In RC6, the number of times MQTE has reduced the number of greedy actions but



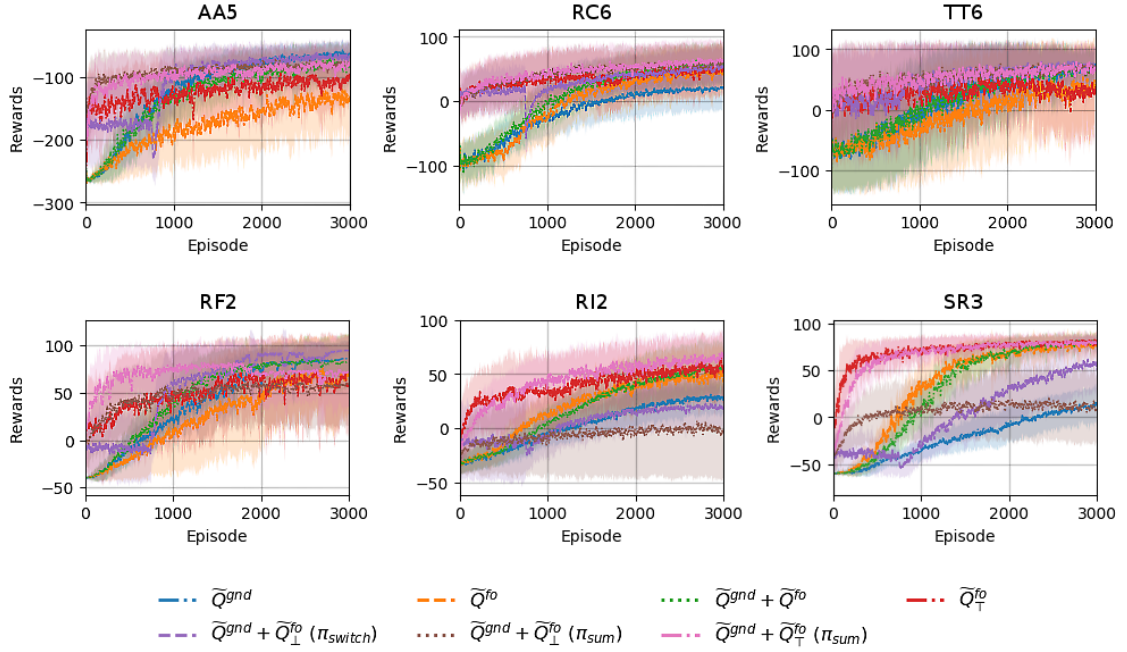


Figure 3.9: Results for transfer learning.  $\tilde{Q}^{gnd}$  ( $\tilde{Q}^{fo}$ ) represents the ground (first-order) approximation, and  $\tilde{Q}^{gnd} + \tilde{Q}^{fo}$  represents an ensemble. The subscripts  $\perp$  and  $\top$  denote that  $\tilde{Q}^{fo}$  is learned in and transferred from a small scale problem.  $\perp$  denotes that  $\tilde{Q}^{fo}$  is not updated in the large scale problem while  $\top$  denotes that  $\tilde{Q}^{fo}$  continues to be updated in the large scale problem.

not to one is relatively large (i.e., the column denoted by “Lesser actions”). This is typical of problems with grid environments as there can be more than one optimal path to reach a goal. In states where the goal is too far from the agent (i.e., the number of grid positions is larger than  $H_{MQTE}$ ), MQTE will not be able to resolve the plateau. The sequence of achieving the goals could be consequential. Suppose that two active goals are located at more than  $H_{MQTE}$  grid positions apart. When the agent moves to one of them and achieves the goal, it is now too far from the last active goal for MQTE to resolve the plateau. Had there been a third active goal in between the two goals, then MQTE can resolve the plateau and direct the agent to move towards the third goal.

### 3.7.4 Transfer Learning

We evaluate the generalisation property of the first-order approximation across problems of different scales. Figure 3.9 shows the results. The baselines are the ground approximation ( $\tilde{Q}^{gnd}$ ), first-order approximation without transfer learning ( $\tilde{Q}^{fo}$ ), and an ensemble of ground and first-order approximations ( $\tilde{Q}^{gnd} + \tilde{Q}^{fo}$ ) without

transfer learning.

We compared four modes of transfer learning. First,  $\tilde{Q}_\top^{fo}$  denotes the transferred first-order approximation continues to be updated online in the large scale problem. The next three modes use an ensemble where the first-order approximation is transferred from a small scale problem and the ground approximation is learned online from scratch. They are described in Section 3.6. Second,  $\tilde{Q}^{gnd} + \tilde{Q}_\perp^{fo}(\pi_{switch})$  denotes an ensemble is used with the policy  $\pi_{switch}$  and the first-order approximation is kept unchanged in the large scale problem. The policy  $\pi_{switch}$  is an  $\epsilon$ -greedy policy generated from the first-order approximation from episodes 1 to 750 (i.e.,  $SW = 750$ ), then generated from the ground approximation thereafter. We set  $SW = 750$  arbitrary and did not tune it. The third mode is similar to the second mode but uses the policy  $\pi_{sum}$  which is an  $\epsilon$ -greedy policy generated from the weighted sum of the Q-values from the ensemble. This is denoted by  $\tilde{Q}^{gnd} + \tilde{Q}_\perp^{fo}(\pi_{sum})$ . The fourth mode is similar to the third mode but the transferred first-order approximation continues to be updated online in the large scale problem. This is denoted by  $\tilde{Q}^{gnd} + \tilde{Q}_\top^{fo}(\pi_{sum})$ .

In all problems, transfer learning provides a jump start in the initial performance.  $\tilde{Q}_\top^{fo}$  outperforms  $\tilde{Q}^{fo}$  in all problems and  $\tilde{Q}^{gnd}$  in RC6, RI2, and SR3. In the remaining problems, AA5, TT6, and RF2,  $\tilde{Q}_\top^{fo}$  has poorer asymptotic performance than  $\tilde{Q}^{gnd}$  but this is not due to transfer learning. As discussed in Section 3.7.2, the ground approximation outperforms the first-order approximation in these domains.  $\tilde{Q}^{gnd} + \tilde{Q}_\perp^{fo}(\pi_{switch})$  mitigates this issue by considering the first-order approximation for the first 750 episodes only. Thus,  $\tilde{Q}^{gnd} + \tilde{Q}_\perp^{fo}(\pi_{switch})$  achieves a jump start in performance and comparable asymptotic performance to the baselines in AA5, TT6, and RF2. This demonstrates that even in domains where a first-order approximation does not perform well, it can provide guided exploration initially until the approximation errors of ground approximation, which is learned from scratch, is reduced. In AA5, RC6, RI2, and SR3, the total undiscounted reward received decreased rapidly at episode 751 which suggests that  $SW$  should be increased. Conversely, in RF2, the performance increases drastically at episode 751 which suggests that  $SW$  should be decreased.

$\tilde{Q}^{gnd} + \tilde{Q}_\perp^{fo}(\pi_{sum})$  uses  $\pi_{sum}$  which avoided the sharp decrease in the total undiscounted reward received. It has the best performance in AA5, RC6, and TT6.

$\tilde{Q}^{gnd} + \tilde{Q}_{\perp}^{fo} (\pi_{switch})$  outperforms  $\tilde{Q}^{gnd} + \tilde{Q}_{\perp}^{fo} (\pi_{sum})$  asymptotically in the other three problems. This is because the first-order approximation needs to be updated in the large scale problems else it deteriorates the asymptotic performance.  $\tilde{Q}^{gnd} + \tilde{Q}_{\perp}^{fo} (\pi_{switch})$  has a better asymptotic performance as it switches away from the first-order approximation. This is further demonstrated by the significant improvement in performance of  $\tilde{Q}^{gnd} + \tilde{Q}_{\top}^{fo} (\pi_{sum})$  over  $\tilde{Q}^{gnd} + \tilde{Q}_{\perp}^{fo} (\pi_{sum})$  in **RF2**, **RI2**, and **SR3**.  $\tilde{Q}^{gnd} + \tilde{Q}_{\top}^{fo} (\pi_{sum})$  also has the best performance in all problems except **SR3** but is the most computationally expensive among the four modes of transfer learning as it updates two approximations online.

It is imperative that the first-order approximation continues to be updated in the target problem if the source problem and target problem are not abstract-equivalent. The first-order approximation transferred from **SR1** is augmented with features as described in Section 3.3.3. These additional base features have initial weights of zeros and can form new candidate features. Further learning is required in **SR3** to update their weights and add any required candidate features to  $\Phi$ . Another reason for the poor performance of  $\tilde{Q}^{gnd} + \tilde{Q}_{\perp}^{fo} (\pi_{sum})$  in **RI2** and **SR3** is because of the poor performance of the ground approximation in these domains. This is evident in Figures 3.5 and 3.9; in Figure 3.9,  $\tilde{Q}_{\top}^{fo}$  marginally outperforms  $\tilde{Q}^{gnd} + \tilde{Q}_{\top}^{fo} (\pi_{sum})$  in **SR3**.

## 3.8 Summary

In this chapter, we presented our online RRL method, **LQ-RRL**, which learns a first-order approximation of the Q-function. In the first-order approximation, the state space is abstracted with lifted literals instead of literals. We showed that the abstract state-action space which the first-order approximation is defined over is a consistent abstract space for abstract-equivalent problems. This is important as it allows generalisation of learned knowledge over problems regardless of their objects, initial states, and goals, and enables transfer learning by directly transferring the first-order approximation.

The Q-function approximation is updated online in two steps: (1) Double Q-learning is used to update the weights and (2) an online feature discovery algorithm,

iFDD+, is used to incrementally add conjunctive features. We extended iFDD+ by replacing a problem-specific hyperparameter with a domain-independent one making it easy to tune without expert knowledge or additional training data.

Free variables in first-order features are crucial in giving a finer granularity in the Q-function approximation but caused ambiguity in how they should be ground. We ground them by considering three types of contextual knowledge which are general in nature and can be applied in most problems. We analysed the properties of a first-order approximation and the limitations in its representational capacity. In addition to contextual knowledge, we proposed two methods to resolve them: MQTE and an ensemble of ground and first-order approximations.

We evaluated LQ-RRL empirically and showed that it reduces sample complexity and keeps the computation time tractable. An ablation study is conducted to determine the best contextual knowledge for each domain. The study showed that contextual knowledge reduces the granularity of first-order approximation and plays a significant role in improving the performance of LQ-RRL. The first-order approximation is able to achieve generalisation across problems of different scales and transfer learning is possible even between non-abstract-equivalent problems. The pitfalls of using a first-order approximation are also resolved; the first-order approximation performs comparably with the ground approximation or better even in domains where the assumptions of the first-order approximation are violated.

# Chapter 4

## Generalised Knowledge for RRL

In Chapter 3, we presented LQ-RRL, an online RRL method which learns a first-order approximation of the Q-function. LQ-RRL uses mostly model-free techniques except for MQTE and learns from only the extrinsic reward signal. The first-order approximation is a form of generalised knowledge that can be used to solve other problems of the same domain. In this chapter, we present methods to learn generalised knowledge from information other than the extrinsic reward signal and utilise them to reduce the sample complexity.

This chapter is organised as follows. First, we present the types of generalised knowledge considered and an overview of the extensions to LQ-RRL in Section 4.1. In subsequent sections, we elaborate on the extensions. In Section 4.2, we discuss model learning with an existing model learner. The learned models are used by model-based methods to prune unnecessary features (Section 4.3.1) and provide better quality estimates of the Q-values (Section 4.3.2). For the latter, we examine various approaches to combine model-based and model-free RL methods to obtain their combined strengths: lower sample complexity and good asymptotic performance. Dead ends in some domains pose a major challenge to learning and planning. When dead ends are reached, a model-free RL method typically treats the observation as any other observations while a model-based RL method relies on the learned model to predict dead ends. In Section 4.4.1, we introduce a learning algorithm to learn effectively from observed dead ends and avoid them in subsequent episodes. This knowledge is generalised as a first-order representation, thereby allowing transfer learning. Another form of knowledge which can guide the policy is

**Algorithm 11:** Generalised-knowledge-assisted online RRL

---

```

1 Function GK-RRL( $P, \mathcal{M}, \Xi, \chi, \tilde{\mathcal{Q}}, \mathbf{w}, \Phi^c, \eta$ ):
  Input: Problem  $P = (\mathcal{C}, \mathcal{P}, \mathcal{A}, \mathcal{O}, \mathcal{T}, \mathcal{R}, s_0, H, \gamma)$ ,
           Generative model  $\mathcal{M}$ ,
           Experience buffer  $\Xi$ ,
           Failure buffer  $\chi$ ,
           Ensemble of Approximations  $\tilde{\mathcal{Q}} = \{\tilde{Q}_1, \dots, \tilde{Q}_n\}$ ,
           Weights for ensemble  $\mathbf{w} = \{w_1, \dots, w_n\}$ ,
           Candidate features for ensemble  $\Phi^c = \{\Phi_1^c, \dots, \Phi_n^c\}$ ,
           Relevances of candidate features for ensemble  $\eta = \{\eta_1, \dots, \eta_n\}$ 
2    $\tilde{Q}(s, a) := \sum_{\tilde{Q}_i \in \tilde{\mathcal{Q}}} w_i \tilde{Q}_i(s, a) := \sum_{\tilde{Q}_i \in \tilde{\mathcal{Q}}} w_i \theta_i^T \Phi_i(s, a)$ 
3   for  $\tilde{Q}_i \in \tilde{\mathcal{Q}}$  do
4      $(\tilde{Q}_i, \Phi_i^c, \eta_i) \leftarrow \text{Dyna}(P, \mathcal{M}, \tilde{Q}_i, \Phi_i^c, \eta_i, CK, H_{sim}, N_{sim})$ 
5   for  $t = 0$  to  $H - 1$  and  $s_t$  is not terminal state do
6      $\tilde{Q}^{UCT} \leftarrow \text{UCT}(\mathcal{M}, \tilde{\mathcal{Q}}, \chi, s_t)$ 
7      $\tilde{Q}^\pi \leftarrow \Lambda_{root} \tilde{\mathcal{Q}} + (1 - \Lambda_{root}) \tilde{Q}^{UCT}$ 
8      $a_t = \pi(s_t)$  where  $\pi$  is generated from  $\tilde{Q}^\pi$  and  $\chi$ 
9      $s_{t+1}, r_t \leftarrow \text{Execute action } a_t$ 
10    Add  $(s_t, a_t, r_t, s_{t+1})$  to  $\Gamma$  and  $\Xi$ 
11     $\mathbf{r}_{int} \leftarrow \text{Compute\_Intrinsic\_Rewards}(\Xi, \tilde{\mathcal{Q}})$ 
12    for  $\tilde{Q}_i \in \tilde{\mathcal{Q}}$  do
13       $r \leftarrow \text{Select\_Reward}(\{r_t, \mathbf{r}_{int}\})$  for  $\tilde{Q}_i$ 
14       $(\tilde{Q}_i, \Phi_i^c, \eta_i) \leftarrow \text{Update\_Approximation}(\tilde{Q}_i, \mathcal{A}, \mathcal{O}, \Phi_i^c, \eta_i, CK,$ 
15         $s_t, a_t, r, s_{t+1})$ 
16     $\mathcal{M} \leftarrow \text{Learn\_Model}(\Xi)$ 
17  if  $s_t$  is a dead end then
18     $\chi \leftarrow \text{LDE\_DT}(\Gamma, \mathcal{A}, \mathcal{M}, \chi)$ 
19  return  $(\mathcal{M}, \Xi, \chi, \tilde{\mathcal{Q}}, \Phi^c, \eta)$ 

```

---

intrinsic motivation. In Section 4.4.2, we propose a unifying framework to learn at different abstraction levels and from multiple reward signals consisting of the extrinsic reward and different types of intrinsic rewards. Lastly, we evaluate our work empirically in Section 4.5. Parts of the work in this chapter have been published before in [117–119, 123].

## 4.1 Types of Generalised Knowledge

It is well known in the literature that model-based RL methods have a lower sample complexity but poorer asymptotic performance than model-free RL methods

[20, 36, 51]. Model-based RL methods consider and utilise more information from observations (e.g., state transitions) than model-free RL methods which increases its learning efficiency. Often, the true model is not available. Instead, a model can be learned using supervised learning methods. If the model is able to make predictions in any problem of a domain (i.e., a relational model), then model-based RL methods can rapidly adapt to new problems. The downside is that model errors are an additional source of approximation error which can limit the asymptotic performance of model-based RL methods. This is especially true in our work since we do not perform online model learning. By combining model-based and model-free RL methods, we can improve both the initial and asymptomatic performance.

In addition to model-based methods, there are other means to improve the sample efficiency of RL methods such as more purposeful exploration [128] or to learn from failures. We extend LQ-RRL (Algorithm 1) to improve its sample efficiency. This is outlined in Algorithm 11; the text in blue are extensions or modifications to Algorithm 1. We refer to Algorithm 11 as **GK-RRL** or generalised-knowledge-assisted RRL. We provide an overview of the extensions introduced in **GK-RRL** while details shall be presented in the remainder of this chapter.

The additional inputs to **GK-RRL** are the generative model ( $\mathcal{M}$ ), experience buffer ( $\Xi$ ), and failure buffer ( $\chi$ ). Due to a multitude of hyperparameters, we omit them from the inputs.  $\Xi$  is a set of observations accumulated over episodes in a problem  $P$ .  $\chi$  is a set of state-action pairs which are observed to lead to dead ends.

The **Dyna** [167] family of algorithms uses a generative model to generate imagined observations which a Q-function approximation learns from. We can use **Dyna** to train one or more of the Q-function approximations in the ensemble (lines 3 to 4). We discuss our implementation of **Dyna** in Section 4.4.2 which is outlined in Algorithm 16. Given a generative model, planning methods such as the **UCT** algorithm [88] can be used to estimate the Q-values (line 6). The Q-values from the ensemble and **UCT** are combined in a mixing equation (line 7). We discuss the combination of **UCT** and model-free RRL in Section 4.3.2. The behaviour policy  $\pi$  considers the mixed Q-values and the failure buffer  $\chi$  to select an action (line 8). Actions which form a state-action pair with the state  $s_t$  that are in  $\chi$  will not be selected by the behaviour policy. This is because the action has been observed to lead to a dead

end after executing it in  $s_t$ . We discuss learning from dead ends in Section 4.4.1. The tree and rollout policies in UCT can also utilise  $\chi$  to select actions (line 6).

Observations are recorded in  $\Gamma$ , the trajectory observed in an episode, and the experience buffer  $\Xi$  (line 10).<sup>1</sup> A set of intrinsic rewards ( $\mathbf{r}_{int}$ ) are computed using statistics from  $\Xi$  and/or information from  $\tilde{Q}$  (line 11). The intrinsic rewards provide directed exploration as opposed to undirected exploration which can be inefficient. We discuss intrinsic rewards in Section 4.4.2. Q-function approximations in  $\tilde{Q}$  are either approximations for the extrinsic reward  $r_t$  or an intrinsic reward  $r_{int} \in \mathbf{r}_{int}$ . Each Q-function approximation is updated with its corresponding reward signal (lines 13 to 14) using Algorithm 4.

In addition to learning the ensemble of Q-function approximations, two other types of knowledge can also be learned concurrently. The first is the generative model (line 15). We discuss model learning in Section 4.2. The second is learning from dead ends. If the episode terminates with a dead end (see Definition 5), Algorithm 15 can be used to learn from dead ends effectively (lines 16 to 17). We discuss learning from dead ends in Section 4.4.1. GK-RRL additionally returns  $\mathcal{M}$ ,  $\Xi$ ,  $\chi$ , and the approximations for the intrinsic rewards (line 18). A new model can be learned offline from  $\Xi$  as the training data while the other outputs can be transferred to another problem of the same domain for transfer learning ( $\Xi$  can only be transferred to the same problem as it is not a generalised knowledge). That is, transfer learning is not limited to only the transfer of first-order approximations but also other types of generalised knowledge.

## 4.2 Model Predictions

Model-based methods, presented in subsequent sections, require generative models. A generative model predicts the next state and immediate reward for executing an action in a state. We assume that the reward function is known and the transition function is not known. This assumption does not make the learning problem much easier as the transition function is often much harder to learn [3, 190]. We

---

<sup>1</sup>A trajectory is the tuple  $(s_0, a_0, r_0, s_1, a_1, r_1, s_2, \dots, s_{H-1}, a_{H-1}, r_{H-1}, s_H)$ . For computation purposes, we represent the trajectory as an ordered set of observations  $\{(s_0, a_0, r_0, s_1), (s_1, a_1, r_1, s_2), \dots, (s_{H-1}, a_{H-1}, r_{H-1}, s_H)\}$ .



described two types of generative models in Section 2.3: the maximum likelihood model ( $\mathcal{M}_{MLM}$ ) and the parameterised DBN ( $\mathcal{M}_{DBN}$ ). In addition, we reviewed existing model learners in Section 2.3.3 and concluded that RLFIT [108] is the most suitable for our work.

Learning DBNs can be computationally expensive to perform online in every time step (line 15 in Algorithm 11). Furthermore, we observed that the learned models typically do not change significantly given an additional small number of observations (see Section 4.5.1). Therefore, we use RLFIT to learn  $\mathcal{M}_{DBN}$  offline from observations obtained in the source problems.  $\mathcal{M}_{DBN}$  is then transferred to target problems to accelerate learning.

We combine  $\mathcal{M}_{MLM}$ , which is learned online from scratch, and  $\mathcal{M}_{DBN}$  to make predictions. If the number of times  $(s, a)$  has been observed in  $\Xi$  exceeds a user-defined threshold  $N_{known}$  (this was first introduced in Section 3.5.1), then  $\mathcal{M}_{MLM}$  is used to make predictions since its predicted next states are always valid states. Otherwise,  $\mathcal{M}_{DBN}$  is used. In some domains, there are transitions of some state predicates which cannot be predicted. These state predicates are removed from the next state before adding the observation to  $\Xi$ . This is to prevent  $\mathcal{M}_{MLM}$  from predicting the transitions of these state predicates.

### Example 27 (Oracle Model and Unpredictable Transitions)

*In **Service Robot**, the locations of people and the goals given by people are not known a priori. Suppose that in a particular state  $s$ , the robot  $r1$  executes `find_person(r1,p1)` and found  $p1$  at  $wp4$ . The state predicate `person_at(p1,wp4)` transitions from false to true. If this observation is added to  $\Xi$ , then in the next episode (locations of people and the goals remain unchanged),  $\mathcal{M}_{MLM}$  predicts that executing `find_person(r1,p1)` in  $s$  will result in the transition of `person_at(p1,wp4)` from false to true.  $\mathcal{M}_{MLM}$  is akin to an **oracle model**: it knows  $p1$  is at  $wp4$  and what goals a person will give. To prevent a learned model from behaving like an oracle model, we remove predictions for the transitions of state predicates which should be unpredictable.*

**Why learn?** Probabilistic planning considers the uncertainties in actions and the environment to find an optimal plan to reach the goal state. A planner requires

**Algorithm 12:** Model-based Feature Selection (MBFS)

---

```

1  Function MBFS( $P, \nu$ ):
   |   Input: Problem  $P = (\mathcal{C}, \mathcal{P}, \mathcal{A}, \mathcal{O}, \mathcal{T}, \mathcal{R}, s_0, H, \gamma)$ ,
   |           Depth of connections  $\nu$ 
2  Construct graphs  $\mathbf{G}$  and sets of nodes  $\mathbf{N}$  from  $P$ 
3  for  $G : \mathbf{G}$  do
4  |   for  $n : \mathbf{N}$  do
5  |   |   if  $n$  and  $G$  have the same action then
6  |   |   |   Add  $n$  as parent nodes to  $G$ 
7  |    $\Phi = \emptyset$ 
8  |   for  $a : \mathbf{A}$  do
9  |   |    $\Phi_a \leftarrow \text{Get\_Neighbour\_of\_Action}(G, a, \nu)$ 
10 |   |    $\Phi_a \leftarrow \Phi_a \cup \neg\Phi_a$ 
11 |   |   Concatenate  $\Phi_a$  to  $\Phi$ 
12 |   return  $\Phi$ 

```

---

as input a generative model. If a model is known, then this raises the question of the need for RL when probabilistic planners such as [86, 87, 93] can often achieve superior performance without the need to learn. However, the soundness and optimality of a plan depends on the correctness of the model. Our empirical results in Section 4.5.6 show that the performance of probabilistic planners is poor when the models are approximate. In Sections 4.3.2 and 4.4.2, we present methods which utilise the learned approximate models to reduce the sample complexity of model-free RL methods.

## 4.3 Model-Based Learning and Planning

### 4.3.1 Model-Based Feature Selection

The value function or Q-function is approximated by projecting the state space into a lower dimensional space using a set of features. Sample complexity increases when unnecessary features are selected. We reproduce Equation 2.29 for the step update of a weight component  $\theta$  here for ease of reference:

$$\theta \leftarrow \theta + \alpha \delta \frac{\phi_f(s, a)}{\|\Phi(s, a)\|_1}.$$

If there is a large number of features in  $\Phi$ , then  $\theta$  receives a smaller step update

---

**Algorithm 13:** Get neighbours of an action

---

```

1  Function Get_Neighbour_of_Action( $\mathbf{G}, a, \nu$ ):
    Input: Set of graphs  $\mathbf{G}$ ,
            Action  $a$ ,
            Depth of connections  $\nu$ 
2  if  $\nu < 0$  then return  $\emptyset$ 
3   $\mathbf{p}^{CP} \leftarrow$  Get_Co_Parents( $\mathbf{G}, a$ )
4  if  $\nu = 0$  then return  $\mathbf{p}^{CP}$ 
5   $\mathbf{p}^C \leftarrow$  Get_Children( $\mathbf{G}, a$ )
6   $\mathbf{p}^{NB} = \mathbf{p}^{CP} \cup \mathbf{p}^C$ 
7  if  $\nu = 1$  then return  $\mathbf{p}^{NB}$ 
8  for  $p : \mathbf{p}^C$  do
9  |   $\mathbf{p}^{NB} = \mathbf{p}^{NB} \cup$  Get_Neighbour_of_State_Predicate( $\mathbf{G}, p, \nu - 1$ )
10 | return  $\mathbf{p}^{NB}$ 

```

---



---

**Algorithm 14:** Get neighbours of a state predicate

---

```

1  Function Get_Neighbour_of_State_Predicate( $\mathbf{G}, p, \nu$ ):
    Input: Set of graphs  $\mathbf{G}$ ,
            State predicate  $p$ ,
            Depth of connections  $\nu$ 
2  if  $\nu \leq 0$  then return  $\emptyset$ 
3   $\mathbf{p}^C \leftarrow$  Get_Children( $\mathbf{G}, p$ )
4   $\mathbf{p}^{NB} = \mathbf{p}^C$ 
5  if  $\nu = 1$  then return  $\mathbf{p}^{NB}$ 
6  for  $p' : \mathbf{p}^C$  do
7  |   $\mathbf{p}^{NB} \leftarrow \mathbf{p}^{NB} \cup$  Get_Neighbour_of_State_Predicate( $\mathbf{G}, p', \nu - 1$ )
8   $\mathbf{a}^{CP} \leftarrow$  Get_Co_Parents( $\mathbf{G}, p$ )
9  for  $a : \mathbf{a}^{CP}$  do
10 |   $\mathbf{p}^{NB} \leftarrow \mathbf{p}^{NB} \cup$  Get_Neighbour_of_Action( $\mathbf{G}, a, \nu - 1$ )
11 | return  $\mathbf{p}^{NB}$ 

```

---

due to normalisation. Hence, having too many unnecessary features can slow down learning. Intuitively, if there are many features, then it takes more observations to determine the importance (i.e., weight) of every feature. Conversely, omitting necessary features results in poor performance as information about the state which are important to approximate the Q-values are lost.

**Example 28 (Necessary and Unnecessary Features)**

*In Robot Inspection, a one-time immediate reward of 19 is given for transmitting information about an inspected object OBJ. Suppose that the robot r1 executes `transmit_info(r1)` in the state  $s_1 = \text{object\_inspected}(o1) \wedge$*

$\neg$ object\_info\_received( $o1$ )  $\wedge$  ... and receives an immediate reward of 19 for transmitting information about the object  $o1$ . The next state is  $s_1 = \text{object\_inspected}(o1) \wedge \text{object\_info\_received}(o1) \wedge \dots$ . If the robot executes `transmit_info( $r1$ )` again, it receives an immediate reward of  $-1$  (i.e., the cost of executing an action). Therefore, the value of `object_info_received( $o1$ )` can cause a drastic difference in the immediate reward. Thus, `object_info_received( $o1$ )` is a necessary feature for approximating the  $Q$ -values of `transmit_info( $r1$ )`. Because Equation 2.29 distributes the step update equally among weight components of active features, it takes several observations to learn that `object_info_received( $o1$ )` has a larger influence on the  $Q$ -values of `transmit_info( $r1$ )` than other features.

Similarly, in **Recon**, the agent receives a reward of 19 ( $-21$ ) for taking a good (bad) picture of an object  $o1$ . A good picture is taken if life has been detected on the object which is represented by the state predicate `lifeDetected( $o1$ )`. The state predicate `pictureTaken( $o1$ )` represents the fact that a picture of  $o1$  has been taken (a picture of each object can only be taken once). Since goals are independent, the  $Q$ -values of taking a picture of  $o1$  by executing the action `useToolOn( $a1, p1, o1$ )` does not depend on the state of other objects. Thus, the state predicates `lifeDetected( $OBJ$ )` and `pictureTaken( $OBJ$ )` for objects other than  $o1$  are unnecessary to approximate the  $Q$ -values of `useToolOn( $a1, p1, o1$ )`.

In Section 3.2, we introduced MFFS for `Feature_Selection` (line 4 in Algorithm 5) which returns the set of literals  $\mathbf{P}^\# = \mathbf{P}^+ \cup \neg\mathbf{P}^+$  as a set of base features for a ground approximation. For a first-order approximation, an additional step is performed to map  $\mathbf{P}^\#$  to first-order base features. We now propose a model-based feature selection or MBFS: `Feature_Selection` returns a set of literals  $\mathbf{P}^- \subset \mathbf{P}^\#$  after considering the structure of the transition function. This work was published in [119]. MBFS is motivated by the observation that the approximation of the  $Q$ -values of an action might not need to consider every literal because:

1. The transition of a state predicate depends on only a subset of  $\mathbf{P}$  in domains with factored transition functions.
2. The effects of an action typically change the values of only a small number of state predicates.

MBFS considers the structure of a DBN which represents the transition function

to determine the base features for each action. This is outlined in Algorithm 12. The inputs are the problem  $P$  and the hyperparameter  $\nu$  which is an integer and determines the extent to which literals are included as base features. A RDDDL domain file defines the preconditions of actions, conditional probability functions (CPFs), and a reward function; all of them are in parameterised forms. For a problem  $P$ , the parameterised forms are ground over the set of objects  $\mathbf{O}$  to propositional forms. We use the propositional forms to construct a set of directed graphs ( $\mathbf{G}$ ) and sets of nodes ( $\mathbf{N}$ ) automatically (line 2).

A node in a directed graph  $G \in \mathbf{G}$  or in a set of nodes  $\mathbf{n} \in \mathbf{N}$  is either a literal or an action. Each graph has only one child node which is a literal and the remaining nodes are parents to the child node with directed edges from the parents to the child. Graphs are constructed from the CPFs while sets of nodes are constructed from the preconditions and reward function. A graph  $G$  will have at most one node which is an action. Likewise, each set of nodes will have at most one action.

**Conditional probability function (CPF).** A CPF defines a conditional probability distribution for a state predicate  $p$ . The distribution can be piecewise with a set of mutually exclusive conditions for the transition of  $p$ . That is, an action causes the transition of  $p$  if the condition is satisfied. For each condition, a directed graph  $G$  is constructed: the child node in  $G$  is  $p$  and the parent nodes are the literals and action in the condition of the transition.

**Preconditions.** The preconditions of an action involve some literals which describe the conditions required for an action to be applicable in a state. The action and literals associated with the preconditions are grouped as a set of nodes  $\mathbf{n}$ .

**Reward function.** The reward function in RDDDL is arithmetic and additive. For each arithmetic equation in the summation, the action and literals involved are grouped as a set of nodes  $\mathbf{n}$ .

A set of nodes  $\mathbf{n} \in \mathbf{N}$  is added as parent nodes to a directed graph  $G \in \mathbf{G}$  if  $\mathbf{n}$  and  $G$  share the same action (lines 3 to 6 in Algorithm 12). Semantically, the preconditions of an action describe the condition under which the action affects a state and should be considered as a part of the conditional probability distribution.

On the other hand, the reward function describes the conditions under which executing an action produces a reward. Since the Q-value is the expected return of executing an action, information from the reward function should be represented in the graphs.

After the graphs  $\mathbf{G}$  are constructed, a set of base features for each action is determined by traversing  $\mathbf{G}$  (line 9). Algorithms 13 and 14 traverse the action node and state node of a graph, respectively. The subroutines used in the two algorithms are defined as follows:

- $\text{Get\_Co\_Parents}(\mathbf{G}, a) = \{p | \exists G \in \mathbf{G}, \exists p' \in \mathbf{P}(Arc(G, a, p') \wedge Arc(G, p, p'))\}$
- $\text{Get\_Children}(\mathbf{G}, a) = \{p | \exists G \in \mathbf{G} Arc(G, a, p)\}$
- $\text{Get\_Co\_Parents}(\mathbf{G}, p) = \{a | \exists G \in \mathbf{G}, \exists p' \in \mathbf{P}(Arc(G, p, p') \wedge Arc(G, a, p'))\}$
- $\text{Get\_Children}(\mathbf{G}, p) = \{p' | \exists G \in \mathbf{G} Arc(G, p, p')\}$

The relation  $Arc(G, p', p)$  is true if and only if there is a directed arc from  $p'$  (can be either an action or a literal) to  $p$  (a literal) in  $G$ .

Algorithm 13 returns the neighbours of an action  $a$  which are literals.  $\nu$  determines the extent to which literals are considered as neighbours of  $a$ ; the larger  $\nu$  is, the more neighbours  $a$  has. If  $\nu < 0$ , then there are no neighbours (line 2). If  $\nu = 0$ , then the neighbours are the co-parents of  $a$  (lines 3 and 4). If  $\nu = 1$ , then the children of  $a$  are neighbours also (lines 5 and 6). For larger values of  $\nu$ , the neighbours of the children of  $a$  are neighbours also (lines 8 and 9).

Algorithm 14 returns the neighbours of a literal  $p$  which are literals. If  $\nu \leq 0$ , then there are no neighbours (line 2). If  $\nu = 1$ , then the children of  $p$  are the neighbours (lines 3 to 5). For larger values of  $\nu$ , the neighbours of the children of  $p$  (lines 6 to 7) and the co-parents of  $p$  are the neighbours of  $p$  also. Thus, Algorithms 13 and 14 are called recursively with  $\nu$  decremented by a value of 1 each time. For a particular value of  $\nu$ , the set of literals returned by the algorithms is a subset of the set of literals returned for a larger value of  $\nu$ .

Q-values represent the expected return of executing an action which consists of the reward and the discounted future reward. Following this observation, a necessary feature is one that helps to predict the reward or the next state [132]. Therefore, children of actions are selected as base features as they represent parts of the state space where actions can affect directly or indirectly in the future.  $\nu$  represents the

number of steps in the future to consider. Co-parents, on the other hand, represent the conditions under which actions affect states. Hence, they are included as base features as well. We do not consider parents or ancestors of actions as they represent the past. Examples for MBFS can be found in Section B.1. Existing work related to MBFS was discussed in Section 2.5.4.

### Effect of Approximate Model

MBFS needs the information on the connectivity of state predicates and actions in a DBN rather than the nature of the transition (i.e., probability of transition and change in values of state predicates). Thus, MBFS is less susceptible to errors in approximate models. In the case where the model is initially unknown and is learned online, MFFS is used to initialise the base features. When the model is learned, the Q-function approximation is re-learned from scratch by using MBFS to initialise the set of base features, then replaying observations in the experience buffer  $\Xi$  [101]—an observation is replayed as if it has been observed again, the weight vector is updated and conjunctive features are added. Since eligibility traces are used, observations are replayed in the chronological order that they are acquired instead of a random order.

### Effect on Online Feature Discovery

In online feature discovery, conjunctive features are added to the set of features for an action  $a$  if their relevances exceed the discovery threshold  $\xi$ . These features are also added to the sets of features of other actions (see lines 7 to 10 in Algorithm 2 and lines 12 to 16 in Algorithm 3) if the candidate feature comprises of base features of the action. This condition is always true for a ground approximation which uses MFFS to initialise the base features since every action has the same set of base features (i.e.,  $P^\#$ ). In a first-order approximation, even with MFFS, the first-order base features can be different as they are lifted according to the terms of an action. With MBFS, the base features for each action are likely to be different in both ground and first-order approximations. The condition for adding conjunctive features ensures that base features which are eliminated by MBFS will not be added to the set of features as a constituent of a conjunctive feature.

## Effect on Ground Context

We introduced ground context in Section 3.4.2. From Definition 23, ground context substitutes free variables by performing the inverse of the mapping from literals to first-order features. With MFFS, every literal is mapped to first-order features. MBFS changes the mapping; thus, it influences ground context. In fact, MBFS eliminates irrelevant objects from consideration by ground context.

### Example 29 (Ground Context and MBFS)

For a particular problem  $P$  of *Recon* and  $\nu = 2$ , MBFS (Algorithm 12) returns four base features for  $a = \text{useToolOn}(a1, p1, o1)$ :  $\text{agentAt}(a1, wp2)$ ,  $\text{damaged}(p1)$ ,  $\text{lifeDetected}(o1)$ , and  $\text{pictureTaken}(o1)$  (see Examples 40 and 41 in Section B.1 for details). We omit their negation for brevity. They are mapped to the first-order base features  $\text{agentAt}(AGENT, \star WP)$ ,  $\text{damaged}(TOOL)$ ,  $\text{lifeDetected}(OBJ)$ , and  $\text{pictureTaken}(OBJ)$ . The only first-order base feature with free variables is  $\text{agentAt}(AGENT, \star WP)$  which is mapped from  $\text{agentAt}(a1, wp2)$ :  $\mathcal{L}(\text{agentAt}(a1, wp2)) = \text{agentAt}(AGENT, \star WP)$  (see Equation 3.3). From Definition 23,  $\mathbf{P}^- = \{\text{agentAt}(a1, wp2)\}$  and  $\mathcal{L}^{-1} = \{AGENT/a1, \star WP/wp2\}$ . It follows that the ground context is  $\sigma_{ground} = \{\star WP/wp2\}$  (the bound variable  $AGENT$  is not substituted with contextual knowledge). If MFFS is used instead of MBFS, then  $\mathbf{P}^- = \{\text{agentAt}(a1, wp1), \text{agentAt}(a1, wp2), \dots\}$  is the set of every ground predicate of  $\text{agentAt}(AGENT, WP)$  and  $\sigma_{ground} = \bigcup_{O \in \mathbf{O}^{wp}} \{\star WP/O\}$  where  $\mathbf{O}^{wp} = \{wp1, wp2, \dots\}$  is a set of objects of type  $wp$ . This illustrates that MBFS combines with ground context to eliminate irrelevant objects from grounding free variables.

## Soundness of Abstraction

As observed by McCallum [110], the state abstraction sufficient to represent the optimal policy is not necessarily sufficient to learn the optimal policy. We reproduced the toy problem (see Figure 4.1) and explanation from [76] for ease of reference. In the toy problem, there are four possible states and in each state, there are two possible actions. In the states  $s_1 = \neg X \wedge \neg Y$  and  $s_2 = \neg X \wedge Y$ , the optimal action is the solid arrow which gives a reward of 0. In the states  $s_3 = X \wedge \neg Y$  and  $s_4 = X \wedge Y$ ,



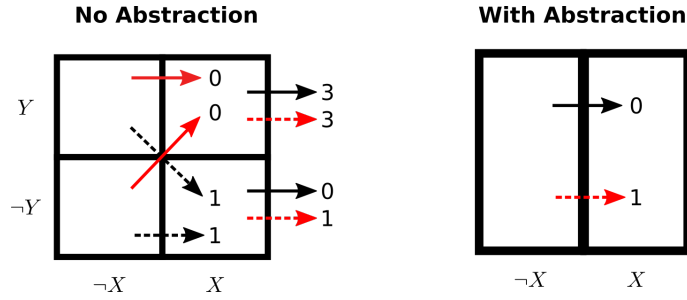


Figure 4.1: A toy problem with two state predicates:  $X$  and  $Y$ . The solid and dotted arrows denote two actions. A red arrow denotes the action is optimal. The numerical values are the immediate rewards for executing the actions. On the left, there is no abstraction. On the right,  $Y$  is abstracted away resulting in two abstract states.

the optimal action is the dotted arrow which gives a reward of 1 and 3, respectively. Thus, the optimal policy is to select the solid arrow when  $X$  is false and the dotted arrow when  $X$  is true.  $Y$  is irrelevant to represent the optimal policy. If  $Y$  is abstracted away, the four states are now represented by two abstract states:  $s_1 \mapsto \bar{s}_1$ ,  $s_2 \mapsto \bar{s}_1$ ,  $s_3 \mapsto \bar{s}_2$ ,  $s_4 \mapsto \bar{s}_2$ . With the abstraction, the optimal action in  $s_1$  and  $s_2$  is the dotted arrow instead of the solid arrow since both actions transition from  $\bar{s}_1$  to  $\bar{s}_2$ . This illustrates the perils of abstracting away state predicates. While a state predicate is unnecessary to represent the optimal policy, it might be required to learn the optimal policy. It is difficult to identify such state predicates.

Instead of abstracting away literals entirely, Jong and Stone [76] abstract away literals in options. Since options are partial policies for a subset of the state space, literals which are irrelevant for one option can still remain relevant for another. Similarly, MBFS determines unnecessary base features for each action; a literal removed from the set of base features for an action can still be a base feature for other actions. Nevertheless, MBFS could still eliminate literals which are required to learn the Q-values of actions, resulting in a poor performance. We investigate this in Section 4.5.2 where the empirical results for MBFS are presented.

### 4.3.2 UCT with Model-Free RRL

UCT, described in Section 2.4.2, is a sampling-based method and its optimality depends on the number of trials which can be prohibitively large. Furthermore, since we use learned models which can be approximate, this can worsen the quality of the estimated Q-values leading to poor performance. While UCT does not require

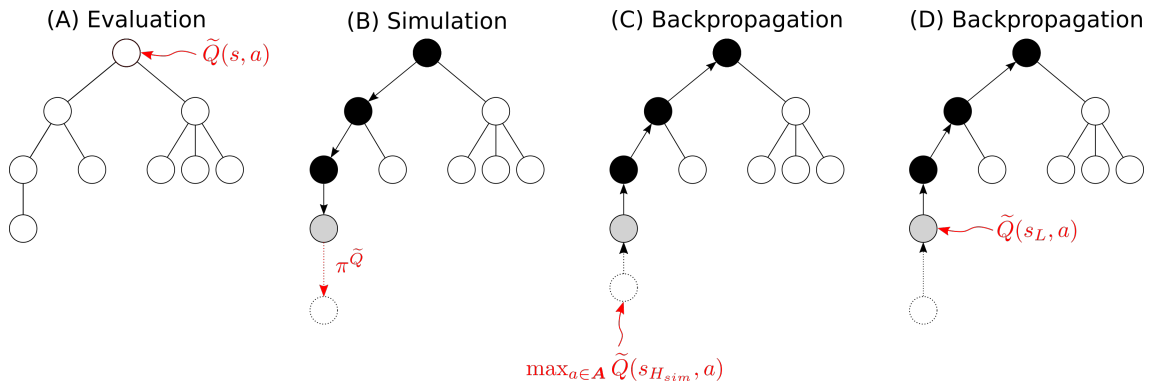


Figure 4.2: The four approaches in which the Q-values from a Q-function approximation,  $\tilde{Q}$ , can be combined with UCT. (A) The policy considers the Q-values of actions at the root node which is a mix of the Q-values estimated by UCT and by  $\tilde{Q}$ . (B) During the simulation phase, the rollout policy is generated from  $\tilde{Q}$ . (C) During backpropagation, the value of the last node in the rollout, which is not a terminal state, is estimated by  $\tilde{Q}$ . (D) During backpropagation, the Q-value of the leaf node is the mix of the return from the rollout and  $\tilde{Q}(s_L, a)$  where  $s_L$  is the state in the leaf node.

any domain-specific heuristic, it is straightforward to implement heuristic for the search. For example, Keller and Eyerich [86] initialise the Q-values based on a single-outcome determination of the MDP which considers only the most probable outcome of executing an action. This heuristic depends on the model and suffers from the aforementioned issue if the model is approximate. Instead, our implementation of UCT combines it with model-free RRL methods, utilising the Q-function approximation  $\tilde{Q}$  learned by model-free RRL methods which can be considered as a domain-independent heuristic.

Figure 4.2 illustrates the four approaches, labelled (A) to (D), in which UCT can be combined with  $\tilde{Q}$ . The four approaches can be used individually or together. (D) is proposed by Silver et al. [157] while (A) and (C) are similar concepts to (D). Although these four approaches are not novel, our contribution here is to combine them with model-free RRL methods and empirically evaluate them. Approximate models and unpredictable state transitions (see Example 27) degrade the performance of model-based methods while model-free RRL methods are unaffected since they do not rely on model predictions.

**(A) Mixing the Q-values at the root node.** The Q-values of actions in the root node is a mixed of the estimates from UCT and  $\tilde{Q}$ :

$$\tilde{Q}^\pi(s, a) = \Lambda_{root} \tilde{Q}(s, a) + (1 - \Lambda_{root}) \tilde{Q}^{UCT}(s, a), \quad (4.1)$$

where  $\Lambda_{root} \in [0, 1]$  is a mixing parameter and  $\tilde{Q}^{UCT}(s, a)$  is the Q-value estimated by UCT. The policy is generated from  $\tilde{Q}^\pi$  instead of  $\tilde{Q}^{UCT}$  or  $\tilde{Q}$ .

**(B) Informed policy rollout.** If the rollout policy is a random policy, this could yield an uninformative total discounted return for the rollout  $V_{sim}$  (see Equation 2.20), especially in large scale problems where it is rare to reach meaningful states through random exploration. If no meaningful states are reached during the simulation phase, then  $V_{sim}$  is simply the discounted sum of the cost of executing actions. Because of this, a large number of trials is required for UCT to converge. This issue can be resolved by using a policy generated from  $\tilde{Q}$  as the rollout policy instead. While this is computationally more expensive than a random policy due to the need to evaluate the features, a rollout is more likely to visit meaningful states and this reduces the need for many trials.

**(C) Limited horizon for policy rollout.** In the simulation phase, the rollout can be terminated after some steps rather than at terminal states. The total discounted return for the rollout is:

$$V_{sim} = \sum_{t=0}^{H_{rollout}-1} \gamma^t r_t + \gamma^{H_{rollout}} \max_{a \in \mathbf{A}} \tilde{Q}(s, a), \quad (4.2)$$

where  $H_{rollout}$  is the length of the rollout and  $s$  is the last state of the rollout. This equation replaces Equation 2.20. However, if the remaining time horizon is less than  $H_{rollout}$ , then Equation 2.20 is used instead. Limiting the length of the rollout can improve the quality of the Q-value estimates from UCT by allowing more trials before timeout. This is especially so if the rollout policy is expensive to compute. For example, a policy generated from a first-order approximation is much more expensive to compute than a random policy. Furthermore, decisions made further into the future will have lesser impact on the estimated value of an action than

immediate decisions as the uncertainty in the next states (i.e., a sampled state might not be the observed state) grows with the number of simulated steps [86]. Furthermore, due to model errors, the sampled next state might not even be a valid state. Again, this error is compounded with each simulated step.

**(D) Mixing the Q-values at the leaf node.** [157] sets the value of a leaf node to the weighted sum of the return from the simulation phase and the Q-value given by  $\tilde{Q}$ . Following this, the discounted return for the state  $s_L$  in the leaf node is computed as follows:

$$\Delta W(s_L) = \Lambda_{leaf} \tilde{Q}(s_L, a) + (1 - \Lambda_{leaf})(r + \gamma V_{sim}), \quad (4.3)$$

where  $\Lambda_{leaf} \in [0, 1]$  is a mixing parameter and  $a$  is the action selected in  $s_L$  during the expansion phase. This replaces the case for a leaf node in Equation 2.24 and affects backpropagation. Since Equation 4.3 changes the values used in backpropagation, it affects the tree policy. This can have a significant impact on how the search tree is expanded.

### Dead Ends and Action Costs

If the rollout reaches a dead end before the time horizon  $H$ , then Equation 2.20 underestimates the total discounted return for the rollout if no immediate reward (or penalty) is given for reaching a dead end and there are action costs.

#### Example 30 (Dead End in Rollout)

*In Robot Inspection, the action cost is  $-1$  and no penalty is given for reaching a dead end. In other words, the immediate reward for reaching a dead end is only the action cost. Consider two rollouts starting from the initial state at  $t_{start} = 0$ . The first rollout terminates at a dead end at  $t_{end} = 1$ . For simplicity, let  $\gamma = 1$ . Equation 2.20 gives  $V_{sim} = -1$ . The second rollout terminates at the end of the time horizon ( $t_{end} = H = 30$ ). In this rollout, a goal is achieved and an immediate reward of 19 is given. Equation 2.20 gives  $V_{sim} = -10$ . Clearly, the total return from the second rollout should be larger than the first rollout.*

The above example shows that even after reaching a dead end, the action costs

thereafter till the end of the time horizon should be considered. We modify Equation 2.20 as follows:

$$V_{sim} = \sum_{t=t_{start}}^{t_{end}-1} \gamma^{t-t_{start}} r_t + \sum_{t=t_{end}}^{H-1} \gamma^{t-t_{start}} r_{cost}, \quad (4.4)$$

where  $r_{cost} \leq 0$  is the action cost (assumed to be a constant for all actions). Following Example 30, Equation 4.4 gives  $V_{sim} = -30$  for the first rollout. Equation 4.2 is modified in the same manner.

### Search Tree Reuse

UCT can be computationally expensive as it requires many trials to converge. We can reduce the time complexity by reusing the search tree. At time step  $t$ , UCT expands a search tree with the state  $s_t$  as the root node. An action is executed in the environment (not simulated) which leads to the next state  $s_{t+1}$ . If  $s_{t+1}$  is one of the child nodes of the root node, then the search tree can be reuse by setting the root node to this child node at time step  $t + 1$ .

If  $s_{t+1}$  is not one of the child nodes of the root node, then a new search tree has to be built. This happens when the model is approximate or there are state transitions which cannot be predicted. For example, in **Service Robot**, the goals which a person could give cannot be predicted. Thus, when the robot talks to a person who needs assistance, the next state will not be in the search tree. A new search tree should be build as some goals are now made known to the robot.

If information which UCT depends on to expand the search tree has been updated, then the search tree should be updated accordingly or discarded and build from scratch. For example, if the model is updated, then the search tree could be invalidated as previously sampled states might turn out to be invalid states or outcomes of probabilistic actions are not previously sampled.<sup>2</sup> We do not perform online model learning in our experiments but instead augment the model prediction from  $\mathcal{M}_{DBN}$  with the prediction from  $\mathcal{M}_{MLM}$ . Furthermore, it is unlikely that an observation will significantly change the models. Thus, we can reuse the search tree from the previous step.

---

<sup>2</sup>The search tree is invalidated if its Q-value estimates would have been significantly different had a new search tree been built.

If we combine  $\tilde{Q}$  with UCT, when  $\tilde{Q}$  is updated, the search tree could be invalidated. Following the same argument as above, we assume that the change made to  $\tilde{Q}$  in each time step is not significantly large to the point that the search tree is invalidated. Thus, it is more beneficial to reuse the search tree and update the Q-values with the updated  $\tilde{Q}$  than to build a search tree from scratch as the computation time is limited by the timeout. For the aforementioned reasons, we only reuse the search tree from the previous step but not from the previous episode.

Lastly,  $\gamma$  has to be equal to 1 in UCT if we want to reuse the search tree. This is because the immediate rewards are discounted starting from the current state (see Equation 2.20). In the next state, since the time step has incremented by one, the Q-value estimates are no longer correct. Since all our domains have action costs, considering the undiscounted return is innocuous.  $\gamma$  remains unchanged elsewhere.

## 4.4 Policy Control

In this section, we discuss methods which influence the policy by considering auxiliary feedback signal to reduce the sample complexity and improve the performance.

### 4.4.1 Learning from Dead Ends

Dead ends are detrimental to learning as no further observations can be obtained in the episode and could pose potential damages to the agent or its environment. Dead ends are also an area of concern in planning to produce plans or policies which avoid dead ends with certainty [102], to analyse goal reachability, and to determine if problems are solvable [162]. Offline planners consider dead ends in order to guide search with the use of heuristics [32, 165] or by pruning the search space [22, 92]. This work uses the transition function to predict dead ends during search while our work does not require the transition function.<sup>3</sup> Instead, we present a novel family of algorithms to learn and avoid the situations leading to dead ends. Our work has been published in [117, 118, 123].

Online RL can be inefficient as it takes several observations of encountering a dead end to learn a policy which avoids the dead end. This is mainly due to two

---

<sup>3</sup>We assume that the preconditions of actions are known though this is not a hard requirement.

reasons: (1) the weights are updated incrementally with a learning rate  $\alpha$  that is typically small for learning stability, and (2) features are added incrementally. We assume that a goal-directed policy exists such that dead ends can be avoided with certainty. This is true in `Triangle Tireworld` and `Robot Inspection`, the two domains we considered which have dead ends.

**Definition 27 (Dead End Situation)**

A dead end situation  $\chi$  describes a situation where executing an action  $a$  in a state  $s$  has a non-zero probability of reaching a dead end.  $\chi$  is either  $(s, a)$  or  $(\bar{s}, \bar{a})$  where  $\bar{s}$  is the abstract state of  $s$  and  $\bar{a}$  is the symbolic action of  $a$ .

A naive method to learn from dead ends is to record every dead end situation in a failure buffer  $\chi$ . Given an observation  $(s, a, r, s')$  where  $s'$  is a dead end,  $\chi = (s, a)$  is added to  $\chi$ . A policy generated from a Q-function is augmented with an auxiliary rule: do not select an action  $a$  in a state  $s$  if  $(s, a)$  is in  $\chi$ .

**Example 31 (Dead End Situation)**

In `Triangle Tireworld`, a dead end is reached if the vehicle has a flat tire (`¬not_flattire`), it does not have a spare (`¬hasspare`), and its current location `WP` does not have a spare (`¬spare_in(WP)`). One possible dead end situation is  $\chi = (s, a)$  where:

$$s = \bigwedge (\text{GOAL\_LOCATION}(la1a5), \text{ROAD}(la1a1, la1a2), \dots, \neg \text{goal\_reward\_received}, \\ \neg \text{hasspare}, \neg \text{spare\_in}(la1a1), \neg \text{spare\_in}(la1a2), \dots, \\ \text{vehicle\_at}(la1a1), \neg \text{vehicle\_at}(la1a2), \dots, \text{not\_flattire}),$$

and  $a = \text{move\_vehicle}(la1a1, la1a2)$ . That is, the vehicle moves to `la1a2` and has a flat tire.

There are two issues with this naive method: (1) lack of generalisation because only previously observed dead ends can be avoided, and (2) high computational and space complexities in problems where there are many dead ends. The space complexity for storing  $\chi$  is  $O(|\chi|)$ . The time complexity for checking if a state-action pair is in  $\chi$  is  $O(|\chi|)$ ; this cost is incurred in every time step. It is desired

to have a generalised and compact representation such that  $|\chi|$  is reduced and an observed dead end situation can be generalised to unobserved dead end situations.

Recording a state-action pair  $(s, a)$  as  $\chi$  could be an over-specialised description of a dead end situation as some literals in  $s$  might be inconsequential. This is especially so for problems with factored transition functions where the transition of a state predicate depends on a subset of  $\mathbf{P}$ . In Example 31, the ground literals of `spare_in(WP)` for locations other than the vehicle's current location (`la1a1`) or the location it is moving to (`la1a2`) are inconsequential. Suppose that there are two dead end situations in  $\chi$ :  $\chi_i = (s_i, a)$  and  $\chi_j = (s_j, a)$  where  $s_i = p_1 \wedge \dots \wedge p_{m-1} \wedge p_m$  and  $s_j = p_1 \wedge \dots \wedge p_{m-1} \wedge \neg p_m$ . The difference between  $s_i$  and  $s_j$  is the value of  $p_m$ .  $s_i$  and  $s_j$  can be abstracted to the same abstract state,  $\bar{s} = p_1 \wedge \dots \wedge p_{m-1}$ , by eliminating  $p_m$ .  $\chi_i$  and  $\chi_j$  are replaced with  $\chi_k = (\bar{s}, a)$  in  $\chi$ . The condition for a state-action pair  $(s_t, a_t)$  to be in  $\chi$  is now as follows:

$$\exists(\bar{s}, a) \in \chi(a = a_t \wedge \bar{s} \subseteq s_t). \quad (4.5)$$

Instead of an exact match for states, an abstract state matches a state if the abstract state is a subset of the state.  $\chi_k$  covers or subsumes  $\chi_i$  and  $\chi_j$  since  $\bar{s} \subset s_i$  and  $\bar{s} \subset s_j$ . That is,  $\chi_k$  is a generalisation of  $\chi_i$  and  $\chi_j$ .

### Definition 28 (Coverage of Dead End Situation and Subsumption)

A dead end situation  $\chi = (\bar{s}, \bar{a})$  covers any state-action pair  $(s_t, a_t)$  if  $\bar{a} = a_t \wedge \bar{s} \subseteq s_t$  is true (see Equation 4.5). The set of state-action pairs covered by  $\chi$  is denoted by  $\text{coverage}(\chi)$ . A dead end situation  $\chi_i$  subsumes another dead end situation  $\chi_j$  if  $\text{coverage}(\chi_j) \subset \text{coverage}(\chi_i)$  is true; then  $\chi_i$  is said to be more general than  $\chi_j$ . In other words, the generality of a dead end situation is measured by the cardinality of its coverage.

The subsumption is only applicable when the states of two dead end situations differ by one literal. It can be done recursively (e.g.,  $\chi_k$  can be subsumed by another dead end situation). A literal which is eliminated due to subsumption is inconsequential in representing a dead end situation. Dead end situations which are subsumed by a (more general) dead end situation are removed from  $\chi$ . We refer to the aforementioned naive method with subsumption as LDE. It is domain-independent



and does not require expert knowledge.

Subsumption reduces the cardinality of  $\chi$  which in turn reduces the space complexity. Since subsumption is only possible between dead end situations involving the same action, the computational complexity of checking for possible subsumption is  $O\left(\binom{|\chi|}{2}\right)$ . However, subsumption is only attempted when a dead end situation is added to  $\chi$ . In practice, for every ten dead end situations which involves an action is added to  $\chi$ , we check for subsumption of its dead end situations. Subsumption does not generalise to unobserved dead end situations as a more general dead end situation can only be determined when all of its subsumed dead end situations are observed.

### First-Order Dead End Situations

To achieve generalisation to unobserved dead end situations, we map a state-action pair  $(s, a)$  to a first-order representation  $(\bar{s}, \hat{a})$  where  $s$  is a conjunction of literals  $P_s$ ,  $\hat{a}$  is the symbolic action of  $a$ , and  $\bar{s}$  is a conjunction of lifted literals given by:

$$\bar{s} = \bigwedge_{p \in P_s} \mathcal{L}(p, a). \quad (4.6)$$

Recall from Equation 3.3 that  $\mathcal{L}$  is a function which lifts a literal  $p$  by substituting objects in its terms with bound (free) variables if they are (are not) terms of  $a$ . If  $(s, a)$  leads to a dead end, then the first-order dead end situation  $\chi = (\bar{s}, \hat{a})$  is added to  $\chi$ . LDE-FO denotes this extension to LDE. A state-action pair  $(s_t, a_t)$  is in  $\chi$  if its first-order representation  $(\bar{s}_t, \hat{a}_t)$  is in  $\chi$ . Equation 4.5 is applied here where actions are replaced with their symbolic actions. LDE-FO assumes that there is a next state  $s_{t+1}$ , among the outcomes of  $a_t$ , which is a dead end. The soundness of LDE-FO shall be proved later. Intuitively, since it deals with first-order representations, LDE-FO can only be used in relational problems.

Although LDE-FO is computationally more expensive than LDE due to the additional step of converting a state-action pair to a first-order representation, it offers two benefits. First, a first-order dead end situation generalises to unobserved dead end situations. Second, first-order dead end situations generalise over problems in the same way that first-order approximation does;  $\chi$  can be transferred to any

problem of the same domain. LDE-FO makes the same assumptions as a first-order approximation (see Section 3.3). Grounding of first-order dead end situations is not required as LDE-FO finds matching state-action pairs using Equation 4.5 rather than evaluating the values of first-order features.

### Example 32 (First-Order Dead End Situation)

Following Example 31, the first-order dead end situation  $\chi = (\bar{s}, \hat{a})$  where:

$$\begin{aligned} \bar{s} = \bigwedge & (\text{GOAL\_LOCATION}(\star WP), \text{ROAD}(WP_1, WP_2), \text{ROAD}(WP_1, \star WP), \\ & \text{ROAD}(\star WP, WP_2), \text{ROAD}(\star WP, \star WP), \text{ROAD}(WP_2, \star WP), \\ & \neg \text{goal\_reward\_received}, \neg \text{has\_spare}, \neg \text{spare\_in}(WP_1), \\ & \neg \text{spare\_in}(\star WP), \text{spare\_in}(\star WP), \neg \text{spare\_in}(WP_2), \\ & \text{vehicle\_at}(WP_1), \neg \text{vehicle\_at}(\star WP), \\ & \neg \text{vehicle\_at}(WP_2), \text{not\_flattire}), \end{aligned}$$

and  $\hat{a} = \text{move\_vehicle}(WP_1, WP_2)$ . The coverage of  $\chi$  is measured by the number of states which map to abstract states that are subsets of  $\bar{s}$ . For ease of illustration, we eliminate the following lifted literals from  $\bar{s}$  since they are present in most of the states:  $\text{GOAL\_LOCATION}(\star WP)$  (unless the action is to move to the goal location),  $\text{ROAD}(\star WP, WP_2)$ ,  $\text{ROAD}(\star WP, \star WP)$ ,  $\text{ROAD}(WP_2, \star WP)$  (most locations lead to two other locations),  $\neg \text{goal\_reward\_received}$  (only false in the goal state),  $\text{spare\_in}(\star WP)$ ,  $\neg \text{spare\_in}(\star WP)$  (there will always be some locations with spares and without spares),  $\neg \text{vehicle\_at}(\star WP)$  (there will always be some locations where the vehicle is not at),  $\text{ROAD}(WP_1, WP_2)$ ,  $\text{vehicle\_at}(WP_1)$ ,  $\neg \text{vehicle\_at}(WP_2)$ ,  $\text{not\_flattire}$  (these will always be true if the preconditions of  $\text{move\_vehicle}(WP_1, WP_2)$  are satisfied). The simplified form of  $\bar{s}$  is:

$$\bigwedge (\neg \text{has\_spare}, \neg \text{spare\_in}(WP_1), \neg \text{spare\_in}(WP_2)).$$

This covers state-action pairs where the vehicle has no spare, its current location  $WP_1$  has no spare (which is inconsequential), and it moves to a location  $WP_2$  that

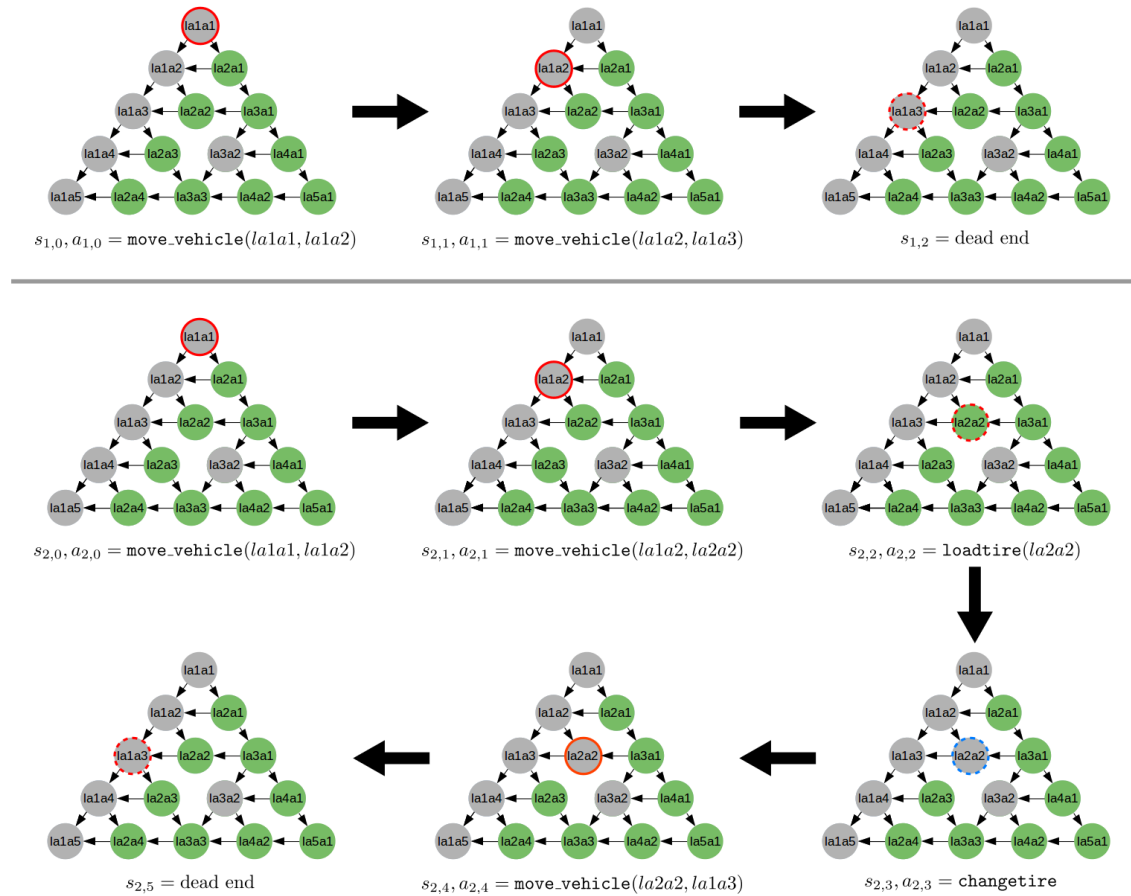


Figure 4.3: Trajectories in episodes 1 (top row) and 2 (middle row and bottom row) of the problem TT3 from Triangle Tireworld.  $s_{i,j}$  and  $a_{i,j}$  denote the state and action executed, respectively, at episode  $i$ , time step  $j$ . Each subfigure illustrates a state that an action is executed in. The next state is illustrated in the next subfigure. Locations are represented by circles. A green circle indicates that the location has a spare. A circle with a border indicates that the vehicle is at that location. A blue border indicates that the vehicle has a spare while a red border indicates otherwise. A dotted border indicates that the vehicle has a flat tire while a solid border indicates otherwise.

*has no spare. This demonstrates the generality of first-order dead end situations. The elimination of the lifted literals above is not done in practice by LDE-FO since it requires expert knowledge. However, the first-order representations of most states contain these lifted literals, and thus this elimination does not significantly increase the coverage of  $\chi$ .*

## Dead End Traps

There might be states where a dead end could be reached eventually regardless of which action is executed. These states are dead end traps.

---

**Algorithm 15:** Find dead end traps

---

```

1  Function LDE-DT( $\Gamma$ ,  $\mathbf{A}$ ,  $\mathcal{M}$ ,  $\chi$ ):
   |   Input: Trajectory  $\Gamma$ ,
   |           Set of actions  $\mathbf{A}$ ,
   |           Model  $\mathcal{M}$ ,
   |           Failure buffer  $\chi$ 
2  while  $\Gamma$  is not empty do
3  |    $(s, a, r, s') \leftarrow$  Pop from back of  $\Gamma$ 
4  |    $\mathbf{A}_{\text{applicable}} \leftarrow$  Get_Applicable_Actions( $s$ ,  $\mathbf{A}$ ,  $\mathcal{M}$ ,  $\chi$ )
5  |   if  $|\mathbf{A}_{\text{applicable}}| > 1$  then
6  |   |   Add  $(s, a)$  to  $\chi$ 
7  |   |   break
8  |   return  $\chi$ 

```

---

**Definition 29 (Dead End Trap)**

A dead end trap is a state where there is a non-zero probability of reaching a dead end trap or a dead end eventually regardless of which action is executed.

Definition 29 includes actions which do not change the state (e.g., inapplicable actions) since they lead back to the same state which is a dead end trap. Our dead end traps are semantically similar to traps in traditional planning. For example, Lipovetzky, Muise, and Geffner [102] define traps as conditional invariant formulas: if the formula is satisfied in a state  $s$ , it is satisfied in all states reachable from  $s$ . Then, a dead end trap is a trap which is mutually exclusive with the goal.

LDE and LDE-F0 avoid dead ends by influencing the policy to avoid actions which could lead to dead ends. However, this does not prevent dead ends in the presence of dead end traps. Suppose that a state  $s$  is a dead end trap. Given sufficient observations, the state-action pairs  $(s, a), \forall a \in \mathbf{A}$  will be added to  $\chi$  and no action remains for selection. We extend LDE and LDE-F0 to avoid dead end traps, they are denoted by LDE-DT and LDE-DT-F0, respectively. LDE-DT is outlined in Algorithm 15 which looks back at previous time steps to determine the situation which led to a dead end trap. Algorithm 15 is used only when a dead end is reached (line 17 in Algorithm 11). The inputs are a trajectory  $\Gamma$  which is a sequence of observations  $(s, a, r, s')$  from the initial state to the dead end, the set of actions  $\mathbf{A}$ , a model  $\mathcal{M}$  which specifies the preconditions of  $\mathbf{A}$ , and the failure buffer  $\chi$ . The observations in  $\Gamma$  are checked sequentially, starting from the most recent one (lines 2 to 7). For

each observation  $(s, a, r, s')$ , the set of applicable actions in  $s$  is determined (line 4). The model  $\mathcal{M}$  returns a set of actions with preconditions that are satisfied in  $s$ . If this model is not known, then every action is considered. Actions which form a state-action pair with  $s$  that is in  $\chi$  are also deemed to be inapplicable. If there is more than one applicable action (line 5),  $s$  might not be a dead end trap, and thus LDE-DT stops looking backwards and adds  $(s, a)$  (or  $(\bar{s}, \hat{a})$  for LDE-DT-F0) to  $\chi$  (line 6). Otherwise,  $s$  is a dead end trap and the preceding observation is considered next. This continues until the condition in line 5 is satisfied; situations which lead to dead end traps are added to  $\chi$ . If the condition is not satisfied, then the problem has an unavoidable dead end and  $\chi$  will not be updated. None of our domains have unavoidable dead ends.

### Example 33 (Looking Back in the Face of Dead End Traps)

We consider two particular episodes for the problem *TT3* which are illustrated in Figure 4.3. A dead end is reached in both episodes. In the first episode, which is illustrated in Figure 4.3 (top), the vehicle moves from *la1a1* to *la1a2* at time step  $t = 0$ . At  $t = 1$ , the vehicle moves to *la1a3*. It has a flat tire at  $t = 2$ . Since the vehicle did not have a spare and there are no spares at *la1a3*, a dead end is reached at  $t = 2$ . The state-action pair  $(s_{1,1}, \text{move\_vehicle}(la1a2, la1a3))$  is added to  $\chi$  where  $s_{1,1}$  denotes the state at episode 1, time step 1.

In the next episode, which is illustrated in Figure 4.3 (middle and bottom), the vehicle moves from *la1a1* to *la1a2* at  $t = 0$ . At  $t = 1$ , the policy considers the dead end situation encountered in the first episode and will not select  $\text{move\_vehicle}(la1a2, la1a3)$ . The only applicable action is to move to *la1a2*. At  $t = 1$ , the vehicle moves to *la1a2*. It then has a flat tire at  $t = 2$ . A spare is loaded at  $t = 2$  and the tire is changed at  $t = 3$ . At  $t = 4$ , the vehicle moves to *la1a3*. It has a flat tire at  $t = 5$  and a dead end  $(s_{2,5})$  is reached. The state-action pair  $(s_{2,4}, \text{move\_vehicle}(la2a2, la1a3))$  can be added to  $\chi$  but since there is only one applicable action,  $s_{2,4}$  is a dead end trap.

To avoid  $s_{2,4}$ , an alternative action has to be selected at  $t = 3$ . At  $t = 3$ , the vehicle has a flat tire and the only applicable action is `changetire`. Thus,  $s_{2,3}$  is also a dead end trap. LDE-DT continues to look backwards at the previous time steps. Similarly,  $s_{2,2}$  (only `loadtire(la2a2)` is ap-

plicable) and  $s_{2,1}$  (only `move_vehicle(la1a2, la2a2)` is applicable because `move_vehicle(la1a2, la1a3)` is ruled out by  $\chi$ ) are dead end traps. Lastly, at  $t = 0$ , there are two applicable actions. The dead end trap  $s_{2,1}$  can be avoided by executing the other action, `move_vehicle(la1a1, la2a1)`. The state-action pair  $(s_{2,1}, \text{move\_vehicle}(la1a1, la1a2))$  is added to  $\chi$ . In contrast, *LDE* will add  $(s_{2,4}, \text{move\_vehicle}(la2a2, la1a3))$  to  $\chi$ . This will not prevent a dead end since there are no other applicable actions in  $s_{2,4}$ .

### Soundness and Completeness of LDE

We have proposed a novel family of algorithms, LDE, LDE-DT, LDE-F0, and LDE-DT-F0, to learn from dead ends. Here, we examine their theoretical properties. LDE and its variants are sound if they do not erroneously determine that executing an action could lead to a dead end. For LDE-DT and LDE-DT-F0, this extends to dead end traps.

#### Theorem 6 (Soundness of LDE and its Variants)

*LDE, LDE-DT, LDE-F0, and LDE-DT-F0 are sound.*

*Proof.* See Section A.2 of Appendix A. □

Next, LDE and its variants are complete if every dead end situation is in  $\chi$ . The completeness does not ensure that dead ends are avoided since LDE and LDE-F0 do not prevent dead end traps. LDE and LDE-DT are complete if every dead end situation has been observed at least once.

### 4.4.2 Intrinsic Motivation

In RL, the agent receives a reward for executing an action. The objective is to maximise the total undiscounted reward received in an episode. This reward is considered as extrinsic as it is defined to guide the agent to achieve the goal(s). In challenging problems, the extrinsic reward is typically sparse; a meaningful reward is observed only after a long sequence of specific actions are executed. Undirected exploration performs poorly as it is rare to receive meaningful rewards. For example, in *Recon*, the agent receives an extrinsic reward of  $-1$  for each action executed, and  $19$  ( $-21$ ) for taking a good (bad) picture of an object. Taking a good picture requires

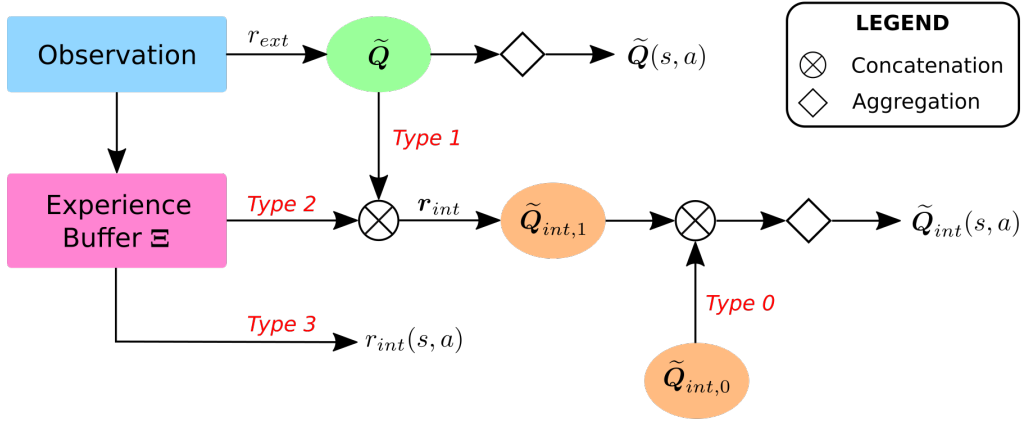


Figure 4.4: A framework for an ensemble of Q-function approximations which learns at different abstraction levels and from extrinsic ( $r_{ext}$ ) and intrinsic ( $r_{int}$ ) rewards. Multiple types of intrinsic rewards can be considered and are categorised into four types, depending on how they fit into the framework.

a specific sequence of actions to be executed. Thus, the agent usually observes an extrinsic reward of  $-1$  which is not informative for improving the policy. A solution is to use intrinsic motivation to provide guided or directed exploration with dense intrinsic rewards [128, 153, 158]. Directed exploration considers knowledge about the learning process (e.g., the learning progress or uncertainty) in contrast to undirected exploration. Other solutions for effective exploration include demonstrations from a teacher [11, 107].

The reward which an agent receives can be defined as  $r = r_{ext} + \beta r_{int}$  [21, 170] where  $r_{ext}$  ( $r_{int}$ ) is the extrinsic (intrinsic) reward and  $\beta$  is a coefficient representing the weight of the intrinsic reward.  $\beta$  can also be seen as the amount of exploration where a larger  $\beta$  results in more exploration. Since the preference for exploration reduces over time, the intrinsic reward typically changes over time and is non-stationary.

A function approximation can be trained with  $r$  as a reward signal such that it approximates the expected return for extrinsic and intrinsic rewards. This has two complications. First, non-stationary intrinsic rewards cause learning instability which can prevent convergence of the function approximation. Second, exploration cannot be reduced easily. For example, a greedy policy selects an action with the maximal Q-value but the Q-value is an estimate of the expected return for the extrinsic and intrinsic rewards. Thus, exploitation (i.e., to maximise extrinsic reward) and exploration cannot be separated. A solution to these two complications

is to learn separate function approximations on the extrinsic and intrinsic rewards [21, 170], hereafter denoted by **extrinsic approximation** and **intrinsic approximation**, respectively. The intrinsic approximation is learned in the same way as the extrinsic approximation—Double Q-learning with replacing eligibility traces and online feature discovery.

### Unifying Framework for Learning from Multiple Reward Signals

We presented an ensemble of approximations in Section 3.5.2 to learn at different abstraction levels (e.g, ground and first-order approximations). The ensemble can also be used to learn from multiple reward feedback such as extrinsic reward and different types of intrinsic rewards. Here, we extend on the concept of ensembles and propose a framework which unifies learning at different abstraction levels and from different feedback signals. The framework is illustrated in Figure 4.4. The experience buffer  $\Xi$  stores observations and keeps track of the statistics required to compute the intrinsic rewards. At each time step there is only one extrinsic reward ( $r_{ext}$ ) for executing an action while there can be several intrinsic rewards ( $\mathbf{r}_{int}$ ) of different types. We categorise the different types of intrinsic rewards depending on how they are computed or used with the other types of intrinsic rewards into four types: Type 0, Type 1, Type 2, and Type 3. Type 0 intrinsic rewards are provided as prior knowledge, Type 1 intrinsic rewards are derived from extrinsic approximation(s), and Type 2 and Type 3 intrinsic rewards are computed with statistics from the experience buffer. Unlike the other three types, Type 3 is represented in tabular form rather than a ground or first-order approximation.

Another reason for learning separate extrinsic and intrinsic approximations is because Type 1 intrinsic rewards are derived from the extrinsic approximations. The ensemble of extrinsic approximations, or extrinsic ensemble  $\tilde{\mathbf{Q}}$ , is discussed in Section 3.5.2. The intrinsic ensemble  $\tilde{\mathbf{Q}}_{int}$  is the concatenation of two intrinsic ensembles,  $\tilde{\mathbf{Q}}_{int,0}$  and  $\tilde{\mathbf{Q}}_{int,1}$ . Each intrinsic approximation in the intrinsic ensemble is either a ground approximation ( $\tilde{\mathbf{Q}}_{int}^{gd}$ ) or a first-order approximation ( $\tilde{\mathbf{Q}}_{int}^{fo}$ ). A policy  $\pi$  considers the Q-values:

$$\tilde{Q}^\pi(s, a) = \tilde{\mathbf{Q}}(s, a) + \beta(\tilde{\mathbf{Q}}_{int}(s, a) + r_{int}(s, a)), \quad (4.7)$$



where  $\tilde{Q}(s, a)$  and  $\tilde{Q}_{int}(s, a)$  are the aggregation of the Q-values from each approximation in their respective ensembles (see Definition 26), and  $r_{int}(s, a)$  is a Type 3 intrinsic reward. The amount of exploration can be controlled explicitly by varying  $\beta$  (e.g., reduce  $\beta$  to reduce exploration). There are three choices for the aggregation of an ensemble: (1) average, (2) sum, or (3) maximum. For the extrinsic ensemble,  $\tilde{Q}(s, a)$  is the average of the Q-values for each approximation in the ensemble as defined in Equation 3.8.

Variations in the framework is possible. For example, a Type 0 intrinsic reward can be used to initialise an extrinsic approximation instead of an intrinsic approximation  $\tilde{Q}_{int,0}$ . Also, an intrinsic approximation can be used to approximate the aggregation of multiple intrinsic rewards (e.g., the mean of  $\mathbf{r}_{int}$ ) rather than using one intrinsic approximation for each intrinsic reward  $r_{int} \in \mathbf{r}_{int}$ . These two variations are used in our experiments. Szita and Lőrincz [170] unify an optimistic initialisation of the Q-function with a model-based RL algorithm from [19]. This is not quite the same as our framework which is more general in the types of intrinsic rewards considered and focuses on the computational aspect of aggregating different types of intrinsic rewards.

There are many different types of intrinsic rewards introduced in prior work [5]. In the following sections, we describe a few of them and introduce some new types of intrinsic rewards. We also look at how these intrinsic rewards fit into our framework. The motivation is not to introduce new types of intrinsic rewards which outperform existing ones but to investigate the utility of our framework for learning from multiple reward signals and at different abstraction levels. In particular, we are interested in whether first-order approximations are appropriate for approximating intrinsic rewards, which could be non-stationary, to provide guided exploration.

### Model-Free Intrinsic Rewards

Model-free intrinsic rewards do not require a model for computation. We discuss four types of model-free intrinsic rewards, one of which is novel.

**State novelty (COUNT).** Curiosity-based intrinsic motivation uses counts to measure state novelty and encourages the agent to visit states which are rarely visited

or have never been visited. There are many variants of count-based methods for measuring state novelty [19, 84] or uncertainties in empirical transition and reward functions [164]. One major drawback of count-based methods is that states are rarely visited more than once in large scale problems; this can be addressed with relational visit counts [99] or pseudo-counts [8].

For discrete state and action spaces, a count-based intrinsic reward can be defined as  $r_{int}(s) = \frac{1}{N(s)}$ , where  $N(s)$  is the number of times that the state  $s$  has been visited. We use a state-action novelty  $r_{int}(s, a) = \frac{1}{N(s, a)}$  instead where  $N(s, a)$  is the number of times  $a$  has been executed in  $s$ . We denote this kind of intrinsic reward by **COUNT**. If it is computed on the fly with statistics from the experience buffer, **COUNT** is Type 3. Otherwise, it is Type 2 if approximated by a function approximation. Consider a simple example where a function approximation is used with Q-learning to approximate **COUNT**. Assume that all Q-values are zeros unless specified. In a particular time step,  $\tilde{Q}_{int}(s, a) = \alpha$  for  $N(s, a) = 1$ . In a subsequent time step when  $s$  is visited again,  $N(s, a) = 2$  and  $\tilde{Q}_{int}(s, a) = \alpha(1.5 - \alpha)$ . If  $\alpha < 0.5$ , then  $\tilde{Q}_{int}(s, a)$  increases when  $N(s, a)$  increases which runs contrary to the concept of state novelty. This phenomenon is due to the highly non-stationary nature of **COUNT** which makes it difficult to track and approximate. This is exacerbated when  $\alpha$ , which diminishes the step update of weights, is too small. Having a large enough  $\alpha$  resolves the issue. For example, if  $\alpha = 1$ , then  $\tilde{Q}_{int}(s, a) = 1$  when  $N(s, a) = 1$  and  $\tilde{Q}_{int}(s, a) = 0.5$  when  $N(s, a) = 2$ . However, a large  $\alpha$  can cause learning instability.

**TD error (TDE).** TD errors of an extrinsic approximation are measures of the learning progress and can be used as intrinsic rewards: at a time step  $t$ ,  $r_{int,t} = |\delta_t|$  where  $\delta_t$  is the TD error for the extrinsic approximation. We denote this kind of intrinsic reward by **TDE**. In our framework, this is a Type 1 intrinsic reward because it is derived from an extrinsic approximation. This kind of intrinsic reward is not new and has been used in [170]. The difference with our work is that **TDE** is based on the first-order approximation and is by itself approximated by another first-order approximation.

**Don't do what doesn't matters (DoWhaM).** In some domains, actions do not affect or change the state unless executed in specific states. For example, executing

an inapplicable action typically does not change the state. Some actions are rarely useful, affecting only a small number of states. We use DoWhaM [155], a kind of intrinsic reward which encourages the execution of rarely useful actions in states which they affect. DoWhaM is a curiosity-based method which prefers states with potentially novel observations over state novelty. It is a Type 2 intrinsic reward because it is computed with statistics from the experience buffer. The exploration bonus for an action  $a$  is given as:

$$B(a) = \frac{\kappa^{1 - \frac{N_{eff}(a)}{N(a)}} - 1}{\kappa - 1}, \quad (4.8)$$

where  $\kappa$  is a hyperparameter,  $N_{eff}(a)$  is the number of times  $a$  is executed and has changed the state, and  $N(a)$  is the number of times  $a$  has been executed. A small  $\kappa$  gives a uniform bonus on all actions while a large value rewards rarely effective actions. We set  $\kappa = 40$  as per [155].  $N_{eff}(a)$  and  $N(a)$  are computed with observations in the experience buffer; their counts are accumulated over episodes. The intrinsic reward for an observation  $(s, a, r_{ext}, s')$  is defined as [155]:

$$r_{int}(s, a, s') = \begin{cases} \frac{B(a)}{\sqrt{N(s')}} & \text{if } s \neq s' \\ 0 & \text{otherwise} \end{cases} \quad (4.9)$$

where  $N(s')$  is the number of times the state  $s'$  has been observed in the episode. [155] trains a neural network to approximate the intrinsic rewards while we use either the ground or first-order approximation.

**Trajectory to goal (T2G).** In some domains, there are common sequences of specific actions to achieve goals. This is the motivation for options [169]. We introduce a new kind of intrinsic reward, T2G, which gives intrinsic rewards for state-action pairs that are in a trajectory that leads from the initial state to a state where at least one goal is achieved. T2G is a Type 2 intrinsic reward since it uses observations in the experience buffer to compute the intrinsic reward for a state-action pair. T2G is potentially more stationary than TDE and COUNT because the trajectory changes less often. A trajectory is post-processed by removing subsequences where the state of the first observation in the subsequence and the next state of the last observation

in the subsequence are identical. This includes observations (i.e., subsequences of length one) where the state did not change.

To maintain tractability, we limit the number of trajectories for each goal, considering only the three shortest trajectories. The intrinsic reward assigned to a state-action pair is:

$$r_{int}(s, a) = \sum_{\forall \Gamma \in \Gamma_{T2G}, (s_t, a_t, r_t, s_{t+1}) \in \Gamma} \frac{i \cdot \mathbb{1}(s = s_t \wedge a = a_t)}{|\Gamma|} \quad (4.10)$$

where  $i$  is the  $i$ -th observation in the trajectory  $\Gamma$ ,  $|\Gamma|$  is the length of  $\Gamma$  (i.e., the number of observations),  $\Gamma_{T2G}$  is a set of post-processed trajectories which lead to states where at least one goal is achieved, and  $\mathbb{1}$  is the indicator function which equates to 1 if the enclosed condition is true.

Equation 4.10 assigns higher intrinsic rewards to state-action pairs if (1) they are closer to states where goals are achieved, and (2) they are in multiple trajectories. Intuitively, (1) guides the agent towards the goal rather than away from it, and (2) increases the incentive for visiting state-action pairs which are frequently observed to lead to goals.

### Model-Based Intrinsic Rewards

In contrast to model-free intrinsic rewards, model-based intrinsic rewards use a model to determine the intrinsic reward. They can be based on the surprise of observing a state after executing an action (e.g., [2]) or on the variance between predictions from an ensemble of models (e.g., [69]). These kinds of intrinsic rewards are suitable if a model-based method is used with online model learning; observations which can lead to a reduction in prediction error can improve the performance of model-based methods. Since we do not perform online model learning, we do not consider them.

Instead, we use the generative model to generate imagined observations which a Q-function approximation learns from. This follows the **Dyna** family of methods first introduced by Sutton [167]. Instead of sampling observations from the experience buffer at every time step, we use **Dyna** to generate a set of imagined observations at the start (lines 3 and 4 in Algorithm 11). This can be seen as an initialisation

---

**Algorithm 16:** Using Dyna to initialise a Q-function approximation

---

```

1 Function Dyna( $P, \mathcal{M}, \tilde{Q}, \Phi^c, \eta, CK, H_{sim}, N_{sim}$ ):
   Input: Problem  $P = (\mathcal{C}, \mathcal{P}, \mathcal{A}, \mathcal{O}, \mathcal{T}, \mathcal{R}, s_0, H, \gamma)$ ,
           Generative model  $\mathcal{M}$ ,
           Q-function approximation  $\tilde{Q}$ ,
           Candidate features  $\Phi^c$ ,
           Relevances of candidate features  $\eta$ ,
           Contextual knowledge  $CK$ ,
           Simulated horizon  $H_{sim}$ ,
           Number of rollouts  $N_{sim}$ 
2   for 1 to  $N_{sim}$  do
3     for  $t = 0$  to  $H_{sim} - 1$  and  $s_t$  is not terminal state do
4        $a_t = \pi(s_t)$ 
5        $s_{t+1}, r_t \leftarrow \text{Simulate}(\mathcal{M}, s_t, a_t)$ 
6        $(\tilde{Q}, \Phi^c, \eta) \leftarrow \text{Update\_Approximation}(\tilde{Q}, \mathcal{A}, \mathcal{O}, \Phi^c, \eta, CK, s_t,$ 
            $a_t, r_t, s_{t+1})$ 
7   return  $(\tilde{Q}, \Phi^c, \eta)$ 

```

---

of the Q-function approximation which in turn provides some information for the policy; an action which has a high initial Q-value is a promising action as predicted by the model. This kind of intrinsic reward is denoted by Dyna and is Type 0 as it is provided as prior knowledge.

Algorithm 16 outlines the procedure to initialise a Q-function approximation using Dyna. The inputs are a problem ( $P$ ), a generative model ( $\mathcal{M}$ ), a Q-function approximation ( $\tilde{Q}$ ), candidate features ( $\Phi^c$ ) with their relevances ( $\eta$ ), contextual knowledge ( $CK$ ), the simulated horizon ( $H_{sim}$ ), and the number of rollouts ( $N_{sim}$ ). Each rollout starts from the initial state  $s_0$  and is simulated for  $H_{sim}$  steps (lines 3 to 6). A policy selects an action  $a_t$  (line 4).  $\mathcal{M}$  predicts the next state and immediate reward (line 5) for executing  $a_t$  in the state  $s_t$ . Algorithm 4 updates  $\tilde{Q}$  based on the imagined observation (line 6). The time complexity of Algorithm 16 is  $O(N_{sim}H_{sim})$ . Though it can be computationally expensive, Algorithm 16 can be run offline to initialise  $\tilde{Q}$ .

$\tilde{Q}$  can be either the intrinsic or extrinsic approximation. If it is the intrinsic approximation, it will not be updated online since Algorithm 16 only runs at the start. If  $\tilde{Q}$  is the extrinsic approximation, it continues to be updated online based on real observations. In either case, Dyna provides directed exploration to regions

of the state space which the model predicts is of high value. Since we do not learn the model online, we utilise the generative model right from the start rather than in every step. The latter is preferred if the learned model has significant model errors as the rollout begins at the current state instead of the initial state and  $H_{sim}$  is typically restricted to a small value to avoid compounding the prediction error.

**Comparison with UCT.** A key difference between Dyna and UCT is that the former updates the Q-function approximation with imagined observations while the latter only influences the policy and does not affect the Q-function approximation. Thus, if the model is poor, Dyna can increase the approximation errors in the Q-function approximation and worsen performance. This can be mitigated by using Dyna to learn a separate intrinsic approximation; if the performance degrades significantly, it can be recovered by disregarding this intrinsic approximation when generating the policy.

### *Greedy- $\epsilon$ -Greedy Policy*

There are several undirected policies which are commonly used in RL such as the greedy,  $\epsilon$ -greedy, and softmax policies (see Section 2.4). Using undirected policies to select actions based on the Q-function given by Equation 4.7 has several potential drawbacks:

- Greedy policy can get stuck in a local minima if the intrinsic rewards are not sufficiently dense as exploration can be limited to a small region of the state space where intrinsic rewards are never observed.
- Softmax policy is sensitive to the magnitude of the Q-values which can be difficult to tune robustly across different domains. This is compounded when different types of intrinsic rewards are used.
- $\epsilon$ -greedy policy selects a random action with a probability  $\epsilon$  regardless of the intrinsic rewards, in effect performing undirected exploration rather than directed exploration.

Following these observations, we propose the *greedy- $\epsilon$ -greedy* policy which combines the greedy and  $\epsilon$ -greedy policies. The *greedy- $\epsilon$ -greedy* policy uses one of two modes to select actions:

1. Use a greedy policy to select an action  $a$  with the maximal  $\tilde{Q}^\pi(s, a)$  as given by Equation 4.7 if (A)  $\beta(\tilde{Q}_{int}(s, a) + r_{int}(s, a)) \neq 0$ , and (B) this mode has not selected  $a$  before in the state  $s$  for the current episode.
2. Otherwise, use  $\epsilon$ -greedy to select an action.

The first mode selects an action with the maximal  $\tilde{Q}^\pi(s, a)$  if the two conditions are satisfied. Condition (A) ensures that the action is greedy because of an intrinsic value. Otherwise, the *greedy- $\epsilon$ -greedy* policy might face the same issue as the greedy policy. Condition (B) prevents repetitive selection of the same action in a state which produces no change to the state. For example, consider the TDE intrinsic reward. The agent achieves a goal and observes  $(s, a, r_{ext}, s')$  where  $r_{ext} > 0$ . For simplicity, assume that the TD error is  $r_{ext}$  and  $r_{int} = |r_{ext}|$ .  $\tilde{Q}$  and  $\tilde{Q}_{int}$  are updated. Since actions typically change the values of a small number of state predicates, we can expect the active features in  $s$  and  $s'$  to have significant overlaps. Then,  $\tilde{Q}(s', a) > 0$  and  $\tilde{Q}_{int}(s', a) > 0$ . Without condition (B), the *greedy- $\epsilon$ -greedy* policy will select  $a$  again in  $s'$  and receive an observation  $(s', a, s'', 0)$ . Suppose that  $a$  does not affect  $s'$  and  $s'' = s'$ . The policy will then repeatedly select  $a$  until  $\tilde{Q}(s', a)$  and  $\tilde{Q}_{int}(s', a)$  decrease sufficiently.

The *greedy- $\epsilon$ -greedy* policy draws similarities with on-policy mixing in [67] which selects an action greedily with a probability of 0.5 and performs directed exploration otherwise. The purpose of on-policy mixing is to train the Q-function approximation with both on-policy and off-policy data which is different from our motivation.

## 4.5 Empirical Evaluation and Discussion

The experimental setup is as described in Section 3.7.1. The domains and problems are described in Section 2.8.

### 4.5.1 Correctness of Learned Models

We use RLIFT to learn the CPFs for Recon, Robot Fetch, Robot Inspection, and Service Robot. Academic Advising is excluded because the transition function for `passed(COURSE)` depends on arithmetic (see Equation 2.30) which RLIFT cannot learn. Triangle Tireworld is excluded as its transition function is rather simple

Domain	Number of Training Data per Symbolic Action		
	10	50	500
Recon	682.8 $\pm$ 461.9	5443.5 $\pm$ 3077.0	47623.4 $\pm$ 25635.2
Robot Fetch	216.7 $\pm$ 19.2	289.3 $\pm$ 38.5	447.5 $\pm$ 143.4
Robot Inspection	200.5 $\pm$ 31.2	450.8 $\pm$ 230.2	8856.7 $\pm$ 6616.1
Service Robot	258.2 $\pm$ 26.3	323.7 $\pm$ 27.9	462.7 $\pm$ 45.4

Table 4.1: Average computation time and one standard deviation in seconds for RLFIT to learn the models given training data of 10, 50, and 500 state transitions per symbolic action. The computation time is aggregated over ten independent runs.

with only three symbolic actions and can be easily learned; we are interested in dealing with approximate models.

**Training set.** We ran ten independent experimental runs for the small scale problems of `Recon` (RC3), `Robot Fetch` (RF1), `Robot Inspection` (RI1), and `Service Robot` (SR1), using the hyperparameters in Table 3.1 for 500 episodes. We randomly sampled 10 state transitions per symbolic action from a set of state transitions observed in each problem. This gives ten sets of training data, one from each run. We repeated this for 50 and 500 state transitions. In total, there are 30 sets of training data per domain and a model is learned from each set of training data. As RLFIT cannot handle states which are conjunctions of more than 20 literals due to intractability, we are restricted to collect the training data from small scale problems.

Table 4.1 shows the computation time for learning the models. The computation time depends on the number of symbolic actions ( $|\mathcal{A}|$ ), the number of training data, and the number of literals in a state ( $|\mathcal{P}|$ ). The computation time is relatively short for `Robot Fetch` and `Service Robot` but increases exponentially with the number of training data for `Recon` and `Robot Inspection`.

Figure 4.5 shows the average variational distance for each learned model. `Recon` has the lowest average variational distance among the four domains while the other domains have average variational distances of less than 0.5. In general, the average variational distances are comparable between the models learned from different sets of training data and different sizes of training data. It is plausible that RLFIT cannot learn the true model even with a large number of training data as the average



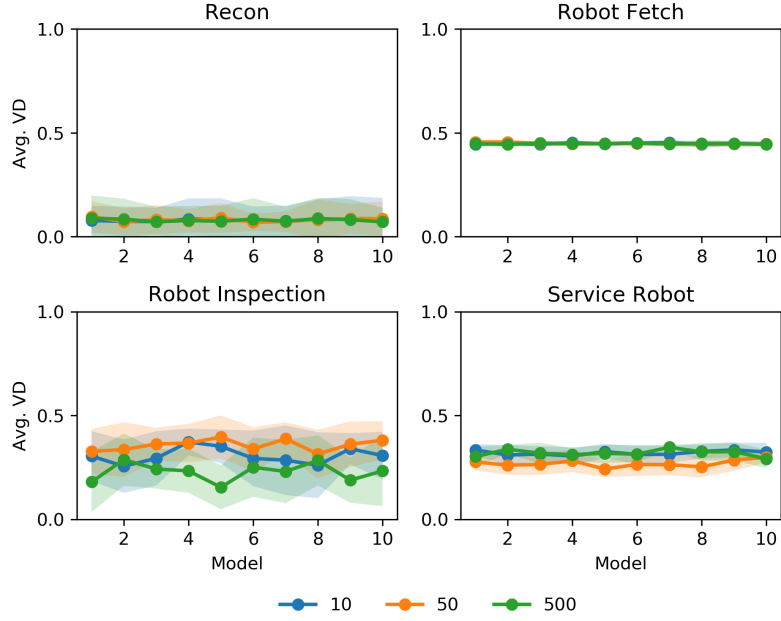


Figure 4.5: Average variational distance and one standard deviation for each of the ten models learned by RL-FIT given the training data of 10, 50, and 500 state transitions per symbolic action. The average variational distance is aggregated over ten independent runs.

variational distance did not decrease when the number of training data increases. Therefore, we opt to learn the models offline from training data obtained in the small scale problems, then use the learned models to solve the large scale problems. For the remaining experiments, we use the models learned from 500 randomly sampled state transitions per symbolic action unless stated otherwise.

#### 4.5.2 MBFS vs MFFS

We evaluate the performance of MBFS (see Section 4.3.1) by comparing it with MFFS. Since MBFS eliminates unnecessary base features, one of the evaluation metric is the cardinality of  $\Phi$ . Following the empirical results in Section C.1.3, we use  $\tau = 5$  for  $\tau$ -iFDD+ to limit the maximum number of candidate features which can be added to  $\Phi$  at each time step. However,  $\tau$  represents a percentage rather than an absolute number. That is, for the same value of  $\tau$ , the maximum number of features which can be added at each time step,  $N_\Phi$ , can vary for MBFS and MFFS (recall that  $N_\Phi = \max(1, \frac{\tau}{100} \max_{a \in \mathcal{A}} |\Phi_a|)$  where  $\Phi_a$  is the set of base features for  $a$ ). For a fair comparison,  $N_\Phi$  should be the same in both MBFS and MFFS. Thus, the experiments for MBFS use the same value of  $N_\Phi$  as MFFS for  $\tau = 5$ .

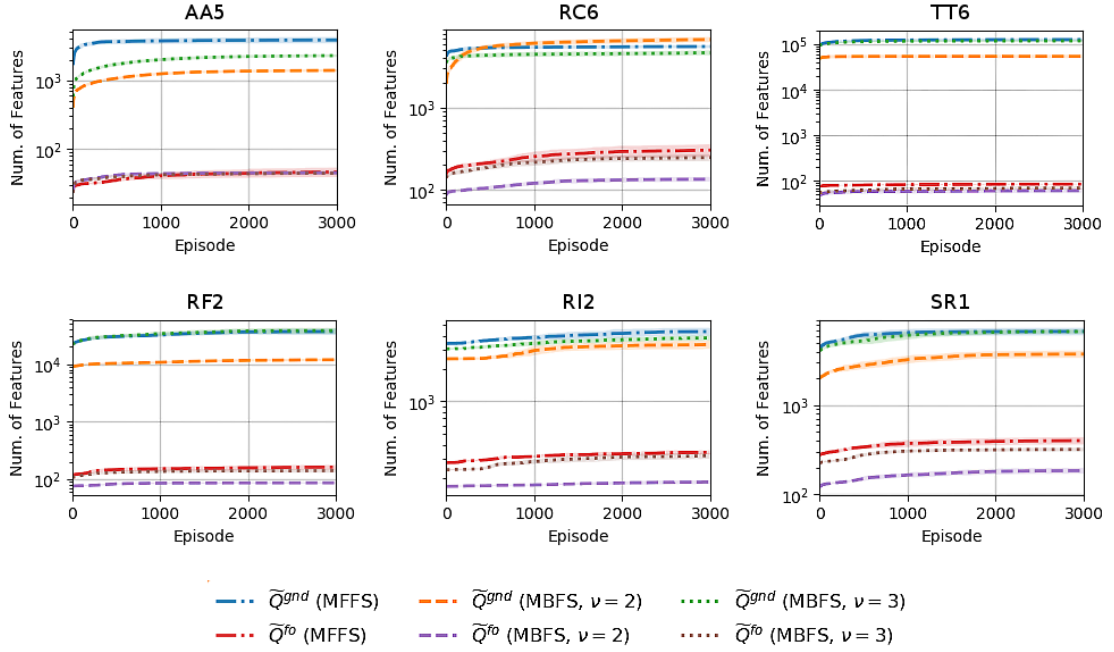


Figure 4.6: Comparing the number of features added to  $\Phi$  between MFFS and MBFS. The true model is used by MBFS to determine the base features.

**True model.** First, we assume that the true model is known for the sole purpose of using MBFS to determine the set of base features. Figure 4.6 shows the number of features in  $\Phi$ . MFFS generally has more features than MBFS. Increasing  $\nu$  from 2 to 3 results in more features. This is expected as the number of features is dependent on the number of base features from which candidate features are generated and added to  $\Phi$ .

Figure 4.7 shows the total rewards received in each episode. In general, the performance are comparable with MFFS which indicate that MBFS did not exclude necessary features and can achieve comparable performance with a smaller set of features. MBFS performs poorly in the following cases:

- ground approximation with  $\nu = 2$  and  $\nu = 3$  for AA5,
- ground approximation with  $\nu = 2$  for RF2, and
- first-order approximation with  $\nu = 2$  for RF2, RI1, and SR1.

The reason for the poor performance of MBFS in AA5 is because there is a sequence to achieve goals. Thus, there is a strong dependency between the first and last goals to be achieved. In *Academic Advising*, this is represented by the non-fluent  $\text{PREREQ}(\text{COURSE}_1, \text{COURSE}_2)$ . In AA5, for the goal  $\text{PROGRAM\_REQUIREMENT}(\text{CS53})$ , the non-fluents  $\text{PREREQ}(\text{CS12}, \text{CS24})$ ,

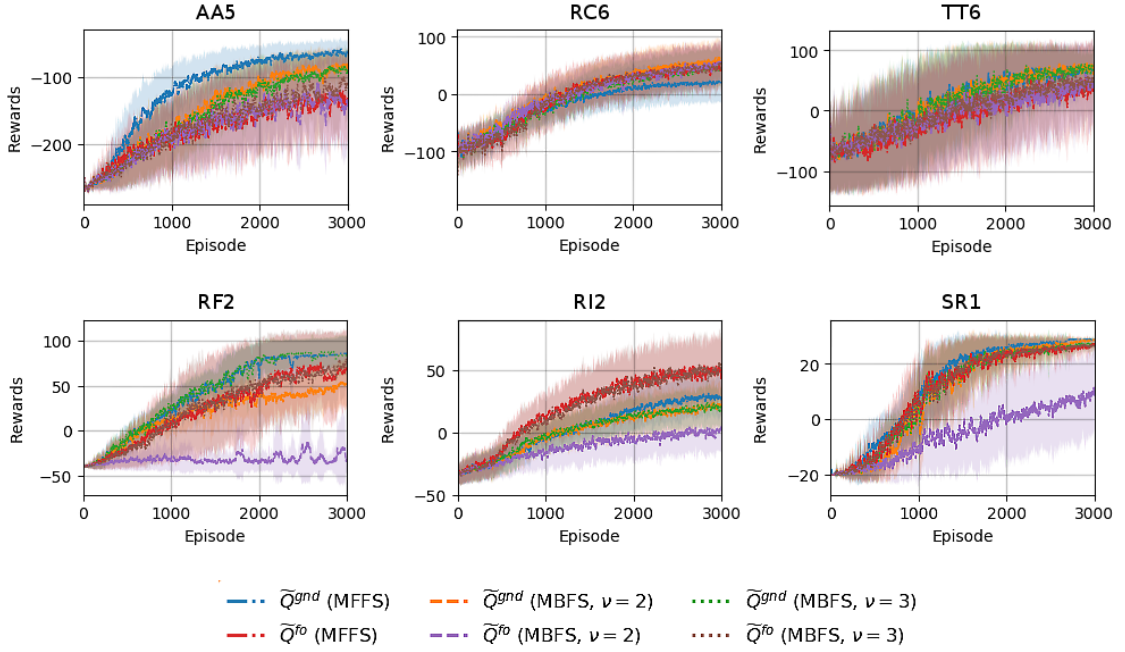


Figure 4.7: Comparing the total undiscounted rewards received in each episode between MFFS and MBFS. The true model is used by MBFS to determine the base features.

$\text{PREREQ}(CS24, CS34)$ , and  $\text{PREREQ}(CS34, CS53)$  specify that  $CS12$  should be passed first, followed by  $CS34$ , and lastly  $CS53$ . Therefore,  $\nu$  should be large enough to include the ground literals of  $\text{PREREQ}(COURSE_1, COURSE_2)$  as base features. In contrast, since MFFS uses every literal as base features, the ground approximation can easily memorise the sequence of courses to pass. When MBFS excludes some literals as base features, this is no longer possible. The reason for the poor performance of ground approximation with  $\nu = 2$  for RF2 is the same as AA5. In Robot Fetch, goals are interdependent because of the constraint that no two objects can be at the same location.

The poor performance of the first-order approximation with  $\nu = 2$  for RF2, RI2, SR1 is due to the increased abstraction in a first-order approximation. Since MBFS eliminates literals from the feature space, state abstraction is increased as states are now represented by fewer features. With the additional abstraction due to the first-order approximation, this can deteriorate performance. Figure 4.7 shows that this can be resolved by increasing  $\nu$  to 3. We discuss the tuning of  $\nu$  in Section C.2.1.

**Learned model.** Next, we evaluate the performance of MBFS when the learned model is used instead. If the model is poor and fails to represent some relations

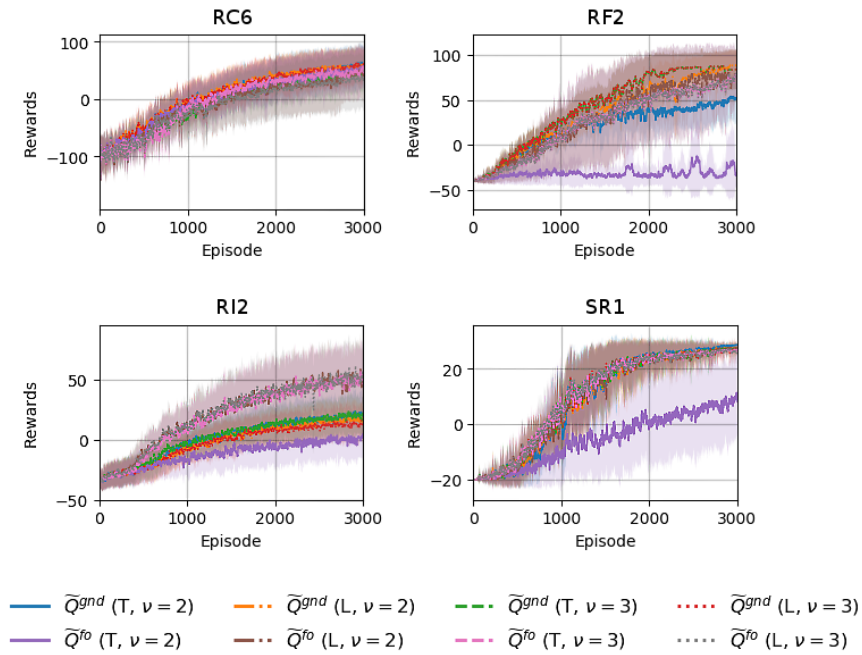


Figure 4.8: Comparing the total undiscounted rewards received in each episode between the true model (T) and the learned model (L) for MBFS.

between state predicates or between state predicates and actions, then MBFS could eliminate necessary base features. This incurs the same consequence as using too small values for  $\nu$ . The results are shown in Figure 4.8. We compare the performance between the true and learned models for the same type of Q-function approximation (i.e., ground approximation or first-order approximation) and for the same value of  $\nu$ . Results show that in general, using a learned model does not worsen the performance relative to using the true model. In fact, the performance with the learned model is better than the true model in RF2, RI2, and SR1 for  $\nu = 2$ . Some of the learned CPFs in these domains contain extraneous literals in their conditions governing the transition of state predicates. This can result in additional base features than with the true model. This counteracts the effects due to a too small value of  $\nu$ . While this result is positive, it is not necessarily reproducible given a different domain or learned model. Nevertheless, the results for the learned models are encouraging as they are at least comparable with the true model which demonstrates that MBFS is less sensitive to model errors as it only requires the relations between state predicates and actions rather than the nature of the relations (e.g., transition probability).

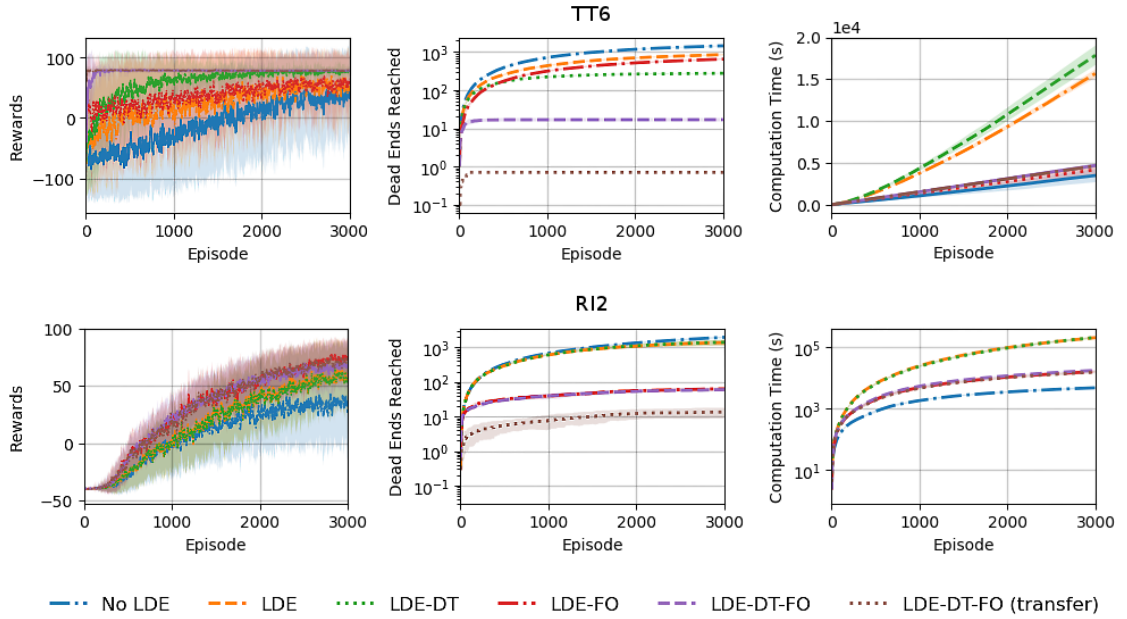


Figure 4.9: Performance of LDE and its variants for the problems TT6 (top) and RI2 (bottom). For LDE-DT-FO (transfer), first-order dead end situations which are learned in the small scale problems, TT3 and RI1, are transferred to the large scale problems, TT6 and RI2.

### 4.5.3 Learning from Dead Ends

We evaluate the performance of LDE and its variants. We consider the domains `Triangle Tireworld` and `Robot Inspection` which are the only two domains with dead ends. Figure 4.9 shows the results. Since there is no penalty for reaching a dead end in `Robot Inspection`, an immediate reward of  $-1$  is given for the remaining time steps if a dead end is reached. This is only for the analysis of the results. For LDE-DT-FO, we consider the case where first-order dead end situations are learned in the small scale problems, TT3 and RI1, and transferred to the large scale problems, TT6 and RI2. This is denoted by LDE-DT-FO (transfer) in the figure. Dead end situations can still be added if dead ends are encountered in the large scale problems. The first-order approximation is not transferred and is learned from scratch.

The performance is measured by the total rewards received and the number of dead ends reached. In both problems, the performance improved when learning from dead ends. In TT6, LDE-DT and LDE-FO outperform LDE. The former is due to the prevalence of dead end traps in `Triangle Tireworld` and the latter is due to the generalisation property of LDE-FO. The combination of the two, LDE-DT-FO, gives the best performance, requiring only a few episodes to learn the optimal policy.

When first-order dead end situations are transferred, the optimal policy is obtained almost immediately.

**Robot Inspection** does not have dead end traps. As such, learning from dead end traps does not improve the performance. Similar to **TT6**, learning first-order dead end situations improves performance. Although the transfer of dead end situations does not lead to an increase in rewards, it reduced the number of dead ends reached. Unlike **Triangle Tireworld** which gives an immediate reward of  $-100$  for reaching a dead end, **Robot Inspection** does not give a penalty for reaching a dead end (other than terminating the episode which prevents further accumulation of rewards). Thus, the difference in rewards between the variants of LDE for **Robot Inspection** is of a lesser magnitude than **Triangle Tireworld**.

The computation time for LDE and its variants are dependent on the cardinality of the failure buffer  $\chi$ . With a first-order representation, the cardinality of  $\chi$  is reduced which decreases the computation time. This is observed in Figure 4.9. The computation time of LDE-DT is higher than LDE due to the additional cost of checking for dead end traps. In **RI2**, the computation time for LDE and LDE-DT is prohibitively long. This is because there are many dead end situations in **Robot Inspection**; a dead end is reached if the robot is low on energy and does not move to the base immediately. This increases the cardinality of  $\chi$  and, subsequently, the time complexity of searching through  $\chi$  for a matching state-action pair. This highlights the scalability of first-order dead end situations.

#### 4.5.4 Intrinsic Motivation

The sensitivity analysis for  $\beta$  is discussed in Section C.2.2. Following the results of the sensitivity analysis, we use  $\beta = 0.1$  for subsequent experiments.

#### Comparing Policies

We evaluate the performance of our *greedy- $\epsilon$ -greedy* policy against the  *$\epsilon$ -greedy*, *greedy*, and *softmax* policies. For both *greedy- $\epsilon$ -greedy* and  *$\epsilon$ -greedy*,  $\epsilon$  is initialised to 1 and decayed exponentially to 0 over the episodes.  $\tau_{temp}$  (see Equation 2.13) is initialised and decayed in the exact same manner. We also tested  *$\epsilon$ -greedy* with lesser exploration by initialising  $\epsilon$  to 0.5. The baseline is the  *$\epsilon$ -greedy* policy with

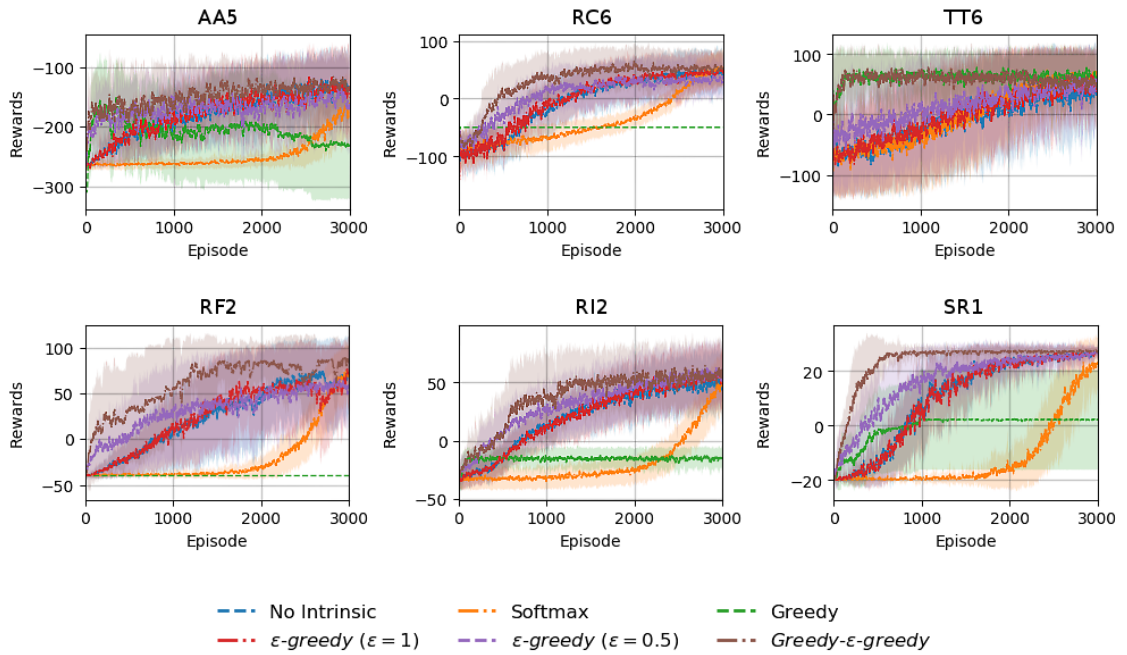


Figure 4.10: Comparing different policies when using intrinsic reward. The intrinsic reward used is TDE. Extrinsic and intrinsic approximations are first-order approximations.

no intrinsic rewards.

The results are shown in Figure 4.10. The intrinsic reward used is TDE. Both intrinsic and extrinsic approximations are first-order approximations. *greedy- $\epsilon$ -greedy* outperforms all policies in all problems except TT6 where it is tied with the greedy policy. The greedy policy has the worst performance with non-optimal asymptotic performance in all problems except TT6. This is because it is myopic in nature and does not sufficiently explore the state space even with intrinsic rewards. The  *$\epsilon$ -greedy* policy performs no better than the baseline as it ignores the intrinsic reward most of the time. By initialising  $\epsilon$  to 0.5 instead of 1, the performance of the  *$\epsilon$ -greedy* policy improves in all problems. The softmax policy tends to favour exploration more, then switches to exploitation belatedly which sees a sharp increase in the rewards at the end. Similar to the  *$\epsilon$ -greedy* policy, this can be resolved by reducing the initial value of  $\tau_{temp}$ . Nonetheless, *greedy- $\epsilon$ -greedy* policy clearly outperforms the other policies without the need to tune  $\epsilon$ . In subsequent experiments which involve intrinsic rewards, we use the *greedy- $\epsilon$ -greedy* policy.

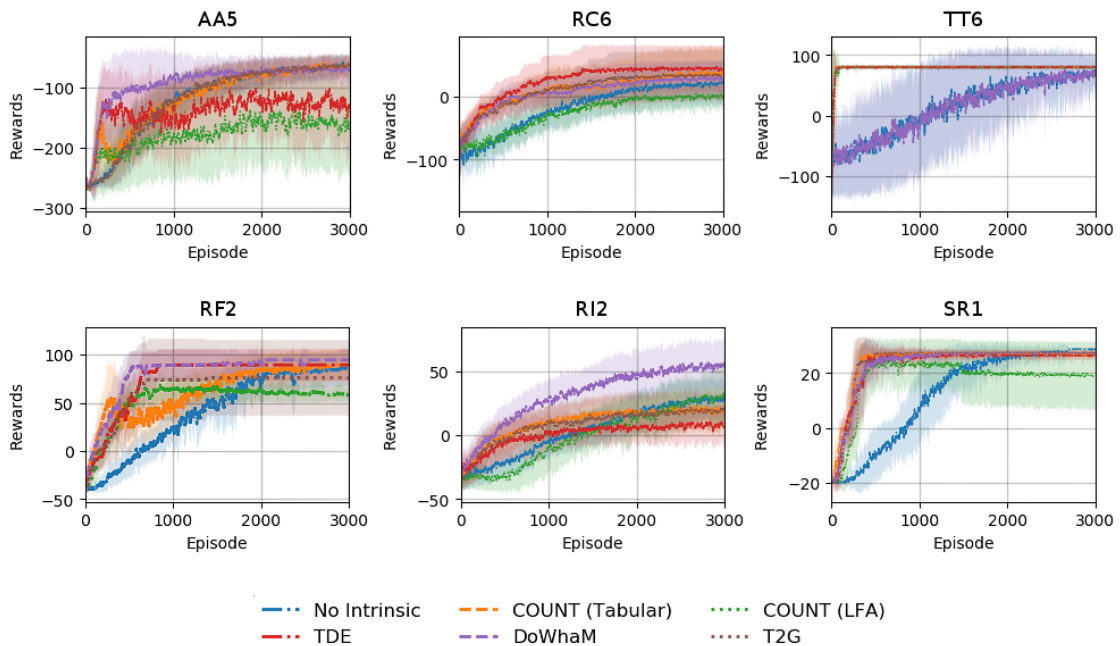


Figure 4.11: Comparing different types of intrinsic rewards. Extrinsic and intrinsic approximations are ground approximations. `COUNT` (Tabular) denotes the visit count intrinsic reward is approximated by a tabular representation while `COUNT` (LFA) denotes that it is approximated by a ground approximation.

### Comparing Model-Free Intrinsic Rewards

We evaluate the performance of the different kinds of model-free intrinsic rewards introduced in Section 4.4.2. Two sets of experiments are conducted. In the first set of experiments, all extrinsic and intrinsic approximations are ground approximations and the baseline is the extrinsic approximation without using any intrinsic reward. The second set is similar to the first set but all approximations are first-order approximations. We tested two types of representations for `COUNT`: tabular (denoted by Tabular) and linear function approximation (denoted by LFA).

First, we discuss the results for the ground approximation which are shown in Figure 4.11. `COUNT` (LFA) performs poorly and is outperformed by `COUNT` (Tabular) in all domains except `TT6`. This shows that the highly non-stationary nature of `COUNT` is not suited to be approximated by a linear function approximation (and possibly other types of function approximation). `COUNT` (LFA) performs well in `TT6` due to the nature of the greedy nature of the *greedy- $\epsilon$ -greedy* policy rather than the intrinsic rewards. As shown in Figure 4.10, the greedy policy performs well in `TT6`.



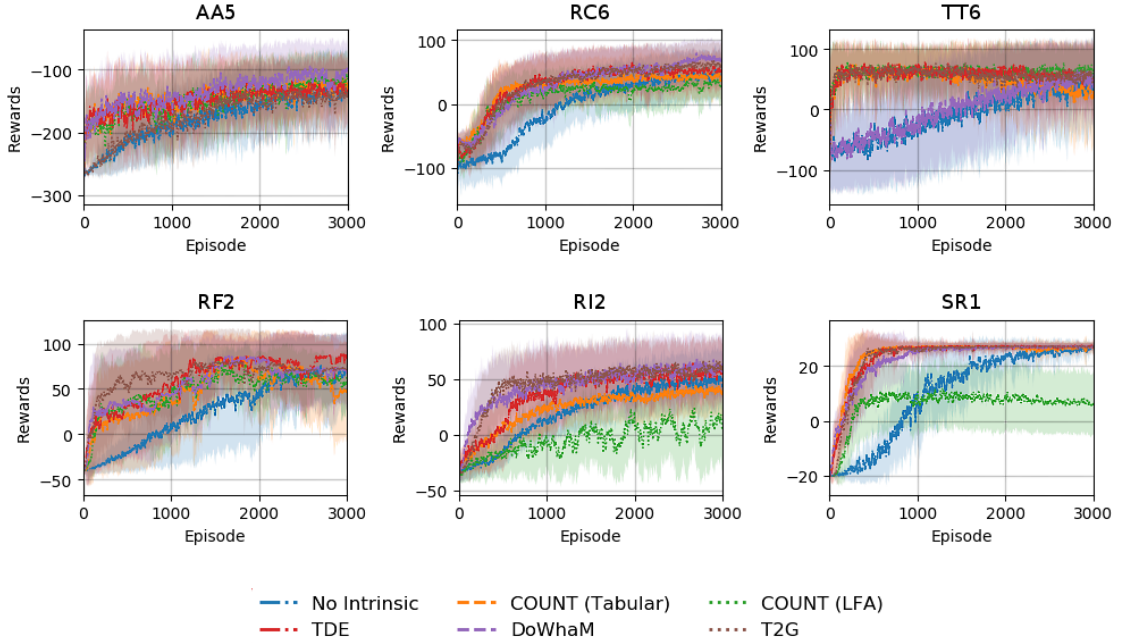


Figure 4.12: Comparing different types of intrinsic rewards. Extrinsic and intrinsic approximations are first-order approximations. COUNT (Tabular) denotes the visit count intrinsic reward is approximated by a tabular representation while COUNT (LFA) denotes that it is approximated by a first-order approximation.

DoWhaM performs well in most domains, especially in AA5 and RI2, but performs poorly in TT6. DoWhaM gives an intrinsic motivation to explore actions which rarely cause a change to the state and is well-suited in domains where many actions cause no change to the states (e.g., *Academic Advising* and *Robot Inspection*) and ill-suited in domains where every action causes some change to the states (e.g., *Triangle Tireworld*). DoWhaM gives no intrinsic reward in such cases as the exploration bonus is zero (see Equation 4.8).

TDE performs poorly in AA5 and RI2 though the initial performance is better than the baseline. This suggests that the extrinsic approximation is not able to converge, causing continual exploration towards states where the TD errors are significant. That is, the policy selects actions to minimise the approximation errors rather than to maximise the expected extrinsic returns. On the other hand, TDE performs well in the other four domains. As expected, since TDE is derived from the extrinsic approximation, its efficacy varies greatly across different domains and lacks robustness.

Since trajectories leading to goals typically change less frequently than the approximation errors in an extrinsic approximation or state novelty count, T2G is more

Domain	CT		CL		TDE		DoWhaM		T2G	
	GND	FO	GND	FO	GND	FO	GND	FO	GND	FO
Academic Advising	—	✓	×	✓	×	✓	✓	✓	—	—
Recon	✓	✓	×	×	✓	✓	✓	✓	✓	✓
Triangle Tireworld	✓	✓	✓	✓	✓	✓	—	—	✓	✓
Robot Fetch	✓	✓	×	✓	✓	✓	✓	✓	✓	✓
Robot Inspection	✓	✓	×	×	×	✓	✓	✓	✓	✓
Service Robot	✓	✓	×	×	✓	✓	✓	✓	✓	✓

Table 4.2: Summary of the performance of the different types of model-free intrinsic rewards. The abbreviations *GND* denotes ground approximation, *FO* denotes first-order approximation, *CT* denotes COUNT (Tabular), and *CL* denotes COUNT (LFA). ✓ indicates that the intrinsic reward improves performance relative to the baseline, × indicates that it worsens performance, and — indicates that it makes no difference.

stationary than TDE and COUNT. Therefore, T2G performs robustly across all domains. Though it is not the best performing intrinsic reward in some domains (e.g., RI2), it outperforms the baseline in all domains except AA5. The reason T2G is ineffective in AA5 is because T2G only considers at most three shortest trajectories to each goal. This makes T2G more stationary—if an optimal trajectory is found, then no other trajectories can be shorter. However, in **Academic Advising**, the prerequisites of a course must be passed to increase the passing probability. The shortest trajectories observed in **Academic Advising** are likely to have skipped the prerequisites. Since T2G does not consider the probability of a trajectory leading to a goal, it rewards actions for taking required courses without passing their prerequisites.

Next, we discuss the results for the first-order approximation which are shown in Figure 4.12. COUNT (LFA) performs well in AA5, TT6, and RF2. This fares better than the ground approximation which only performs well in TT6. This is possibly due to the generalisation property of the first-order approximation which deals better with non-stationary rewards. Each step update affects all ground actions of the same symbolic action which can be considered as multiple updates in each time step. Since updates are more frequent, the first-order approximation adapts better to non-stationary rewards. TDE, DoWhaM, and T2G perform well in all problems, outperforming the baseline in most cases. As was observed for the ground approximation, DoWhaM has no effect in TT6 and T2G has no effect in AA5 for the same reasons.

We summarise the above discussion in Table 4.2. In most cases, the different

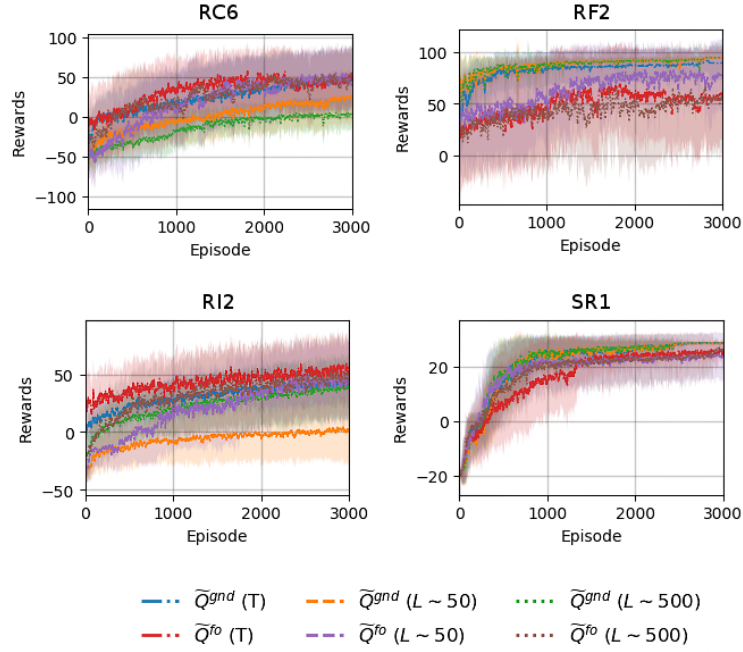


Figure 4.13: Dyna uses a model to generate imagined observations which an extrinsic ground ( $\tilde{Q}^{gnd}$ ) or first-order approximation ( $\tilde{Q}^{fo}$ ) learns from. The models are either the true models (T) or learned from a set of training data which contains either 50 ( $L \sim 50$ ) or 500 ( $L \sim 500$ ) state transitions per symbolic action.

kinds of intrinsic rewards have comparable performance. In other cases, one kind of intrinsic reward works better or worse than the others.

### Effect of Approximate Models on Dyna

The sensitivity analysis for the number of rollouts ( $N_{sim}$ ) used in Dyna is discussed in Section C.2.3. We use  $N_{sim} = 1000$  and  $H_{sim} = 40$  in all experiments unless stated otherwise. We examine the effect of approximate models on the performance of Dyna. Two sets of ten learned models are used; one set is learned from training data which consists of 50 state transitions per symbolic action and the other set from 500 state transitions per symbolic action. We denote the learned models by  $L \sim 50$  for the former and  $L \sim 500$  for the latter. Dyna uses a different learned model in each run. Here, we do not compare the performance between a ground approximation or a first-order approximation as this was discussed in Section 3.7.2.

Figure 4.13 shows the results for Dyna with the true and learned models. We expect the true model to have the best performance followed by  $L \sim 500$ . In general, this is true for all domains. We focus our discussion on cases where this is not true. For the first-order approximation,  $L \sim 50$  has the best performance

in RF2. Due to plateaus, the performance have fluctuations and large variances. Given this, it is conceivable that  $L \sim 50$  can outperform the true model. Next, in SR1, the first-order approximation with the true model performs slightly worse than the rest. The learned models predict the transitions of the state predicate `person_at(PERSON, WP)` albeit wrongly. For example, one particular model predicts that if the robot finds a person  $p1$ , then `person_at(p1, WP)` is true for all objects of  $WP$  where there is no table. The true model does not predict the transition of `person_at(PERSON, WP)` since the location of a person cannot be predicted. Even with the wrong predictions, the learned model can generate imagined observations where the person’s location is (wrongly) known, learning in more (imagined) states than the true model can generated.

We observed that learned models with similar average variational distances can give significantly different performance. For example, in RI2,  $L \sim 500$  outperforms  $L \sim 50$  even though they have very similar average variational distances (see Figure 4.5). As discussed in Section 2.3.4, the average variational distance is not a complete representation of the correctness of learned models. Therefore, it might not be a good predictor for the performance of planning or learning methods which rely on the learned models.

### Combinations of Intrinsic Rewards

We tested some combinations of model-based and model-free intrinsic rewards on four domains. The aggregation is the average of the Q-values from the approximations in the ensemble. We tested other aggregations, maximum and sum, and found that this makes little difference to the results. We combined the best performing model-free intrinsic rewards (see Table 4.2), `COUNT`, `TDE`, and `DoWhaM`, with `Dyna` which uses the learned model. All function approximations are first-order approximations. Figure 4.14 shows the results. The results for intrinsic rewards by themselves (i.e., no combination) are included as baselines. `DoWhaM` is omitted for better clarity as its performance is comparable with `TDE` (see Figure 4.12).

We tested three combinations of intrinsic rewards: (1) `TDE+Dyna`, (2) `TDE+COUNT+Dyna`, and (3) `TDE+DoWhaM+Dyna`. Combinations of intrinsic rewards have the best performance in all problems except SR1: (1) and (3) perform best in

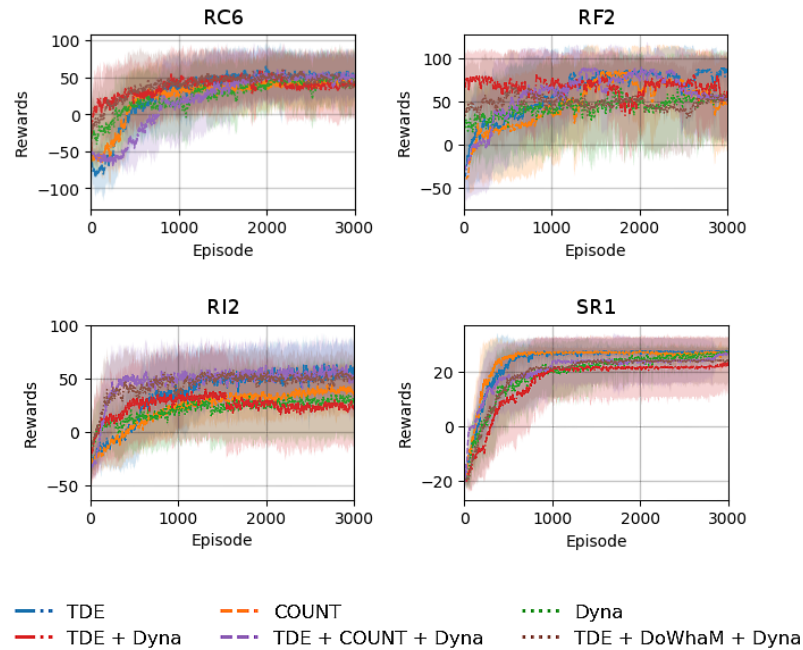


Figure 4.14: Comparing different combinations of intrinsic reward approximated with a first-order approximation. The extrinsic approximation is also a first-order approximation. COUNT denotes the visit count intrinsic approximated with a tabular representation.

RC6, (1) in RF2, and (2) and (3) in RI2. There is no combination which perform best in all problems. As TDE is derived from the extrinsic approximation which is initialised by Dyna, Dyna influences TDE. If the model is approximate and generates poor imagined observations, this can impact Dyna and also TDE if used with Dyna. In RI1, this is compensated by including COUNT in TDE+COUNT+Dyna or DoWhaM in TDE+DoWhaM+Dyna. In SR1, none of the combinations can improve the performance of Dyna. This suggests that unpredictable transitions is a limiting factor in using Dyna. Nevertheless, the asymptomatic performance is only slightly worse than the performance of the model-free intrinsic rewards.

### 4.5.5 Continual Learning

It is impractical to learn in the same problem for many episodes. Typically, the environment or the initial state changes. We evaluate the performance of ground approximation, first-order approximation, and intrinsic rewards in solving problems given fewer episodes than in previous experiments.

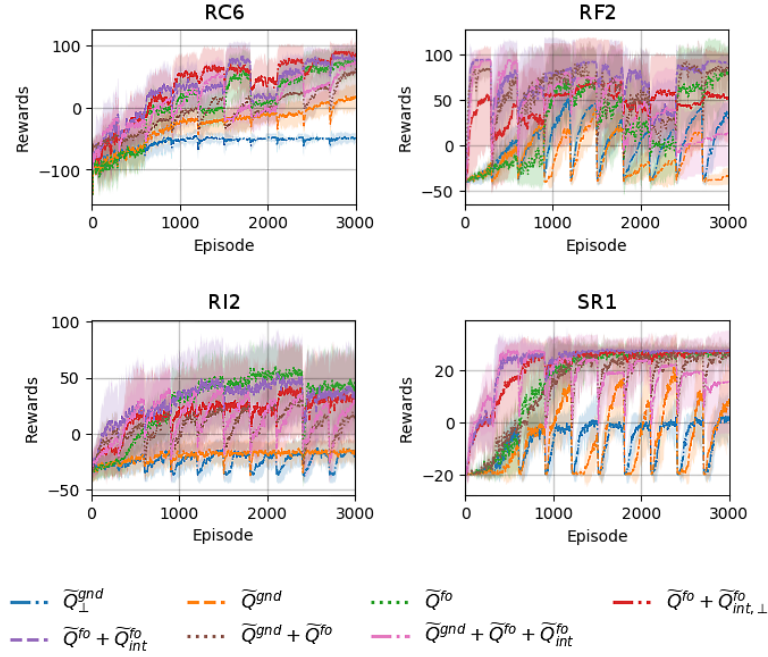


Figure 4.15: A different problem is attempted in every 300 episodes. For each problem (except for the first), Q-functions are transferred from the previous problem unless stated otherwise with the subscript  $\perp$ .

**Experimental setup.** In each run (out of ten independent runs), ten different problems of the same scale (i.e., the set of objects are the same) are attempted sequentially over 3000 episodes. In every 300 episodes, a different problem is attempted.<sup>4</sup> This is not simply learning policies to reach different goal states from any state as each problem has different non-fluents, and thus a different and non-overlapping state space. *Academic Advising* and *Triangle Tireworld* do not have randomised problems and are omitted. For transfer learning, Q-function approximations are transferred from the previous problem (if any) of the same run. Hyperparameters which are decayed over episodes (i.e.,  $\epsilon$  and  $\alpha$ ) are decayed over 3000 episodes rather than 300 episodes. For intrinsic reward, we use a first-order approximation  $\tilde{Q}_{int}^{fo}$  to approximate TDE which is the absolute TD error of the extrinsic first-order approximation  $\tilde{Q}^{fo}$ . The extrinsic and intrinsic first-order approximations use the same hyperparameters.

A ground approximation is applicable in problems of the same scale if and only if the set of objects are the same and non-fluents are excluded as features. Non-fluents

<sup>4</sup>We use the RDDDL parser from Keller and Eyerich [86] to parse the RDDDL domain and problem files. Due to the long computation time of parsing the files and initialising the data structures, it is infeasible to attempt a new problem in every episode for 3000 episodes.

can be excluded since ground non-fluents always evaluate to true in a problem. However, this is not the case if the source and target problems have different non-fluents. Nevertheless, we investigate if transfer learning with ground approximation is beneficial for problems of the same scale. Figure 4.15 shows the results. The performance of  $\tilde{Q}^{gnd}$  (with transfer) and  $\tilde{Q}_{\perp}^{gnd}$  (without transfer) are the worst in all problems. In RF2, it is better to learn  $\tilde{Q}^{gnd}$  from scratch than to transfer  $\tilde{Q}^{gnd}$  from the previous problem. Conversely, in the other problems, transferring  $\tilde{Q}^{gnd}$  performs better than learning  $\tilde{Q}^{gnd}$  from scratch.

As shown in Section 3.7.4, the ensemble of ground and first-order approximations ( $\tilde{Q}^{gnd} + \tilde{Q}^{fo}$ ) allows transfer learning from small scale to large scale problems. Here, the ensemble is trained in only 300 episodes and has not achieved optimal performance before it is transferred to the next problem. In spite of this, the ensemble is able to improve its asymptotic performance over ten different problems. However, using only the first-order approximation ( $\tilde{Q}^{fo}$ ) gives a better performance than the ensemble in RC6 and RI2. This implies that including the ground approximation actually worsens performance. This is because the ground approximation has to be learned from scratch. With only 300 episodes, the ground approximation remains a poor approximation. This is evident with the poor performance of the ground approximation in RC6 and RI2. On the other hand, the ground approximation performs relatively better in RF2 and SR1, and thus an ensemble gives comparable or better performance than the first-order approximation.

Next, we consider an ensemble of extrinsic and intrinsic first-order approximations. The intrinsic approximation is either learned from scratch ( $\tilde{Q}^{fo} + \tilde{Q}_{int,\perp}^{fo}$ ) or transferred from the previous problem ( $\tilde{Q}^{fo} + \tilde{Q}_{int}^{fo}$ ). In RC6, the performance between the two are comparable while in the other three domains, transferring the intrinsic approximation allows continual directed exploration across different problems which lead to improved performance. We observed that in RC6 and RF2, there are some randomised problems where  $\tilde{Q}^{fo} + \tilde{Q}_{int,\perp}^{fo}$  performs better and vice versa for some other randomised problems. If the intrinsic approximation is not transferred, then exploration is reduced. Exploitation is preferred over exploration in problems which are similar to previous problems; in such problems,  $\tilde{Q}^{fo} + \tilde{Q}_{int,\perp}^{fo}$  outperforms  $\tilde{Q}^{fo} + \tilde{Q}_{int}^{fo}$ .

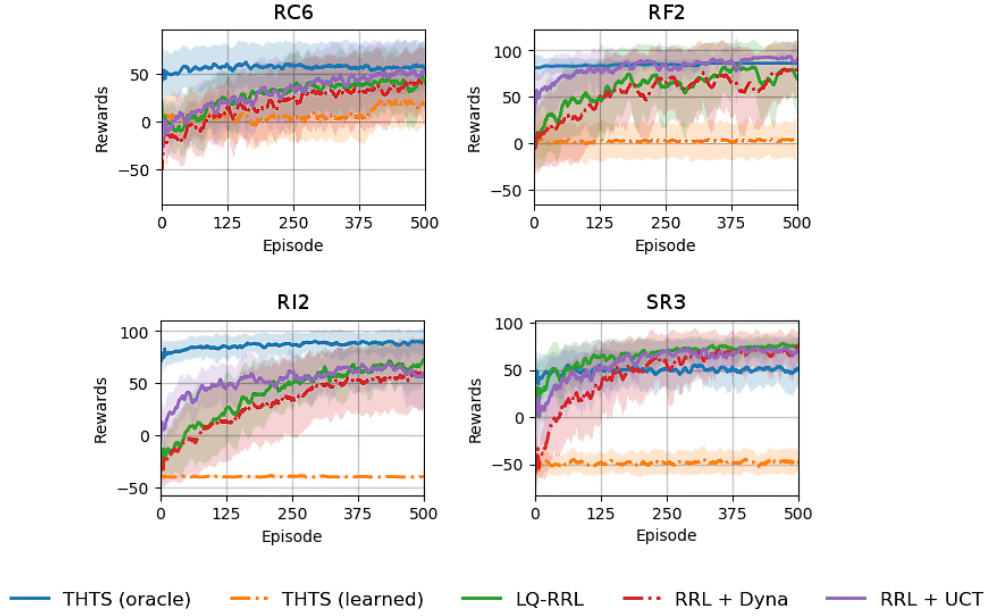


Figure 4.16: The total undiscounted rewards received in each episode for THTS, LQ-RRL, and combinations of model-based and model-free RL methods, RRL+Dyna and RRL+UCT.

Lastly, we consider an ensemble of first-order approximation, ground approximation, and intrinsic first-order approximation ( $\tilde{Q}^{gnd} + \tilde{Q}^{fo} + \tilde{Q}_{int}^{fo}$ ). As mentioned previously, the inclusion of a ground approximation worsens performance in RC6 and RI2. With the inclusion of  $\tilde{Q}_{int}^{fo}$ ,  $\tilde{Q}^{gnd} + \tilde{Q}^{fo} + \tilde{Q}_{int}^{fo}$  outperforms  $\tilde{Q}^{gnd} + \tilde{Q}^{fo}$  in these two problems. Interesting, the converse is true in RF2 and SR1;  $\tilde{Q}^{gnd} + \tilde{Q}^{fo} + \tilde{Q}_{int}^{fo}$  initially outperforms  $\tilde{Q}^{gnd} + \tilde{Q}^{fo}$  but performs worse in later episodes. This suggests that (directed) exploration is less preferred in later episodes. A possible solution is to decay  $\beta$ , the coefficient for intrinsic rewards, over episodes.

### 4.5.6 Combining Model-Based and Model-Free RL

We evaluate the performance of combinations of model-free and model-based RL methods. We consider the following two approaches: (1) transfer a first-order approximation, then train it with imagined observations using Dyna as outlined in Algorithm 16, and (2) transfer a first-order approximation, then combine it with UCT as described in Section 4.3.2. (1) and (2) are variants of GK-RRL (Algorithm 11) and use learned models. We denote (1) by RRL+Dyna and (2) by RRL+UCT. We compare their performance with baselines which are either model-based or model-free RL methods. For the model-based RL method, we use the trial-based heuristic tree



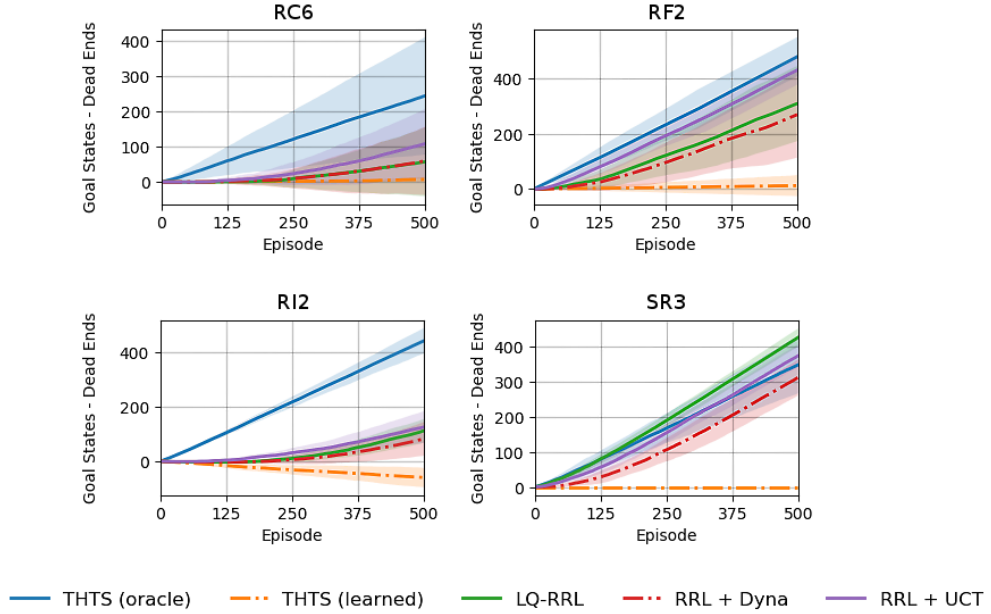


Figure 4.17: The goal-directedness for THTS, LQ-RRL, and combinations of model-based and model-free RL methods, RRL+Dyna and RRL+UCT.

search (THTS) [87] which is the state-of-the-art planner for finite horizon MDPs.<sup>5</sup> The model-free RL method is LQ-RRL (Algorithm 1). The lower baselines are THTS with learned models and LQ-RRL. The upper baseline is THTS with the oracle model (the oracle model and the true model is the same for all domains except **Service Robot**). We investigate if combinations of model-free and model-based RL methods can outperform the lower baselines.

**Experimental setup.** The first-order approximations are learned from scratch in 3000 episodes of the small scale problems (i.e., the source problems) using LQ-RRL and transferred to the large scale problems (i.e., the target problems). The target problems are limited to 500 episodes due to the high computational cost of UCT. Because of plateaus in **Recon** and **Robot Fetch**, the ensemble of ground and first-order approximations is used in the target problems. LDE-DT-F0 is used in all runs of RI2 except those involving THTS. The first-order dead end situations are learned in RI1 and transferred to RI2. The hyperparameters for **Dyna** are  $N_{sim} = 500$  and  $H_{sim} = 40$ . The hyperparameters for UCT are  $\Lambda_{root} = \Lambda_{leaf} = 0.5$ . For RC6, RI2, and RF2, the rollout policy is a greedy policy generated from the first-

<sup>5</sup>We use the version of THTS that was used in the IPPC 2014—partial Bellman updates are used for backpropagation and iterative deepening search is used to initialise the Q-values of nodes. We refer readers to [86, 87] for details.

order approximation. We limit the length of the rollout ( $H_{rollout}$ ) to 5 as it is computationally expensive to ground the first-order approximation (see Theorem 5) in order to generate the greedy policy. For **SR3**, the rollout policy is a random policy with  $H_{rollout} = H = 60$  (i.e., rollout till the end of the time horizon). Due to its large number of first-order base features, making it expensive to ground the first-order approximation, we find that this improves the performance by allowing for more trials before timeout.

**Termination condition for UCT.** The termination condition for UCT is 10 seconds for timeout and the number of trials must be at least thrice the number of applicable actions at the root node.<sup>6</sup>

Figures 4.16 and 4.17 show the results. In all domains, the performance of **THTS** when the oracle model is available is the best (except in **SR3**) but is the worst when only the learned models are available. This is expected. One possible reason that the performance is not the best in **SR3** even with the oracle model is the large state-action space in **SR3**. Next, we compare **RRL+Dyna**, **RRL+UCT**, and **LQ-RRL**. In all domains, **RRL+UCT** outperforms **RRL+Dyna** while **RRL+Dyna** performs marginally worse than **LQ-RRL**. This is because **Dyna** trains the transferred first-order approximation and increases its approximation errors due to model errors. In Section 4.5.4, **Dyna** initialised the Q-function approximation and this gives a jump start in the performance. The errors introduced by **Dyna** are reduced when the Q-function approximation is updated. Thus, if the model is approximate, then **Dyna** should only be used to initialise a Q-function approximation. Even though **UCT** uses the same approximate models as **Dyna**, it did not worsen the performance because **UCT** does not directly affect the Q-function approximation as discussed in Section 4.4.2.

In **RF2**, **RRL+UCT** significantly outperforms **LQ-RRL** which demonstrates the advantage of utilising an approximate model to reduce sample complexity. In the remaining domains, **RRL+UCT** marginally outperforms **LQ-RRL**. This is because **Robot Fetch** is a deterministic domain and it is relatively easy to learn its model.

---

<sup>6</sup>The timeout for **THTS** is one second which might give an advantage to **UCT**. However, the purpose of implementing our **UCT** is to combine with the first-order approximation rather than to outperform **THTS** which has incorporated caching and other efficient data structures for fast computations.

In the other domains, the errors in the learned models restrict the improvement in performance due to UCT. In RI2, RRL+UCT outperforms LQ-RRL in terms of the total rewards received. This is because model-based RL methods such as UCT can exploit the learned dead end situations (the models learned by RLFIT cannot predict dead ends accurately) more than model-free RL methods which look at only one step ahead. This demonstrates the utility of combining a relational model and first-order dead end situations, two types of generalised knowledge. It is also worth noting that in SR3, both LQ-RRL and RRL+UCT marginally outperform the upper baseline, THTS with the oracle model.

## 4.6 Summary

In this chapter, we presented GK-RRL which is an extension to LQ-RRL. GK-RRL utilises the following additional generalised knowledge: relational models, first-order approximations of intrinsic rewards, and first-order dead end situations. This generalised knowledge can be learned from observations and transferred to target problems to accelerate learning. GK-RRL incorporates UCT to provide better quality estimates for the Q-values. If the true model is not available or known, then model learning algorithms can be used to learn an approximate model. Due to model errors, this diminishes the efficacy of model-based RL methods. To mitigate this, we examined various approaches to combine a Q-function approximation with UCT. Empirical results showed that the combination of UCT and model-free RRL can improve performance but this is limited by the correctness of the learned model. We used RLFIT from [106] to learn models for four domains. While it is promising that the models can be learned from very few training data, results suggest that given additional training data, RLFIT is unable to improve on the correctness.

The learned model can be utilised in other ways. First, MBFS considers the structures of graphs representing transition functions to prune unnecessary base features. Empirical results showed that MBFS is able to achieve comparable performance with MFFS but with fewer features. We demonstrated that MBFS is less sensitive to the errors in approximate models as it only requires information on the connectivity of the graphs rather than the exact nature of the transition function. Next, Dyna

is used to initialise the Q-function approximations which can give a jump start in performance. This can be considered as a type of model-based intrinsic reward. Four kinds of model-free intrinsic rewards are also introduced, one of which is novel. We proposed a unifying framework which uses an ensemble of Q-function approximations to learn at different abstraction levels and from multiple reward signals consisting of the extrinsic reward and different types of intrinsic rewards. Empirical results showed that this approach of learning can improve performance.

Lastly, we introduced a novel family of algorithms to learn dead end situations or state-action pairs which were previously observed to lead to dead ends. A first-order variant of the algorithm allows transfer learning while another variant learns from dead end traps which improves the efficacy. Empirical results showed that learning from dead ends is effective for avoiding dead ends.

## Chapter 5

# Dynamic Objects, Time, and Coordination

In Chapters 3 and 4, we considered problems which ignore the complexities present in real world applications. Experiments were conducted in RDDLSim [149] which is a simplistic simulator that ignores physical elements in the real world. For example, robots operating in the real world have to deal with uncertainty in sensor measurements, physical constraints, and temporal considerations among others. In this chapter, we begin with a motivating problem for **Service Robot** in Section 5.1 which considers dynamic objects, action durations, and time bounds for goals. In Sections 5.2 and 5.3, we describe the formalism for representing these added complexities in a new class of problems. Next, we discuss the impact they have on the methods presented in the previous chapters and propose extensions for **GK-RRL** to solve this class of problems in Section 5.4. We explore the possibility of using this class of problems to address multi-agent coordination. In Section 5.7, we present a framework to decompose a multi-agent coordination problem into single-agent problems. Taking a step towards real world applications, we discuss transfer learning between not just different classes of problems but also different simulated environments in Section 5.5. The purpose is to train and test RL agents in different environments. We built a simulated environment in Gazebo [89] which is more representative of the real world than RDDLSim’s simulated environment. In Section 5.6, we discuss empirical experiments conducted in RDDLSim’s and Gazebo’s simulated environments. The objective is to evaluate the robustness of our methods

when transferring from simulation to the real world. We take a step towards this objective by demonstrating transfer learning from RDDLSim’s to Gazebo’s simulated environments. While real world experiments are lacking, the differences between the two simulated environments and the different classes of problems are significant. Parts of the work in this chapter have been published before in [24, 122].

## 5.1 Service Robot in the Real World

We described **Service Robot** in Section 2.8.2. Here, we consider three added layers of complexity which are typical of a robot operating in the real world. Firstly, a robot might not be fully aware of its environment; the existence of some objects are initially not known to the robot and are discovered later. Secondly, objects are deliberately ignored as they seem inconsequential but are later considered when they turned out to be important. For example, suppose that there are 100 people of whom only a few require assistance. It is impractical to consider every person but without knowing whom will require assistance, the robot considers a person only when the person needs assistance. The first and second points can be treated in the same manner. We formally describe what this entails in Section 5.2.

Lastly, executing an action takes time, the duration of which can be stochastic. If the transitions of states are time-dependent, then the duration it takes to execute an action has to be considered in synthesizing a plan or policy. For example, the tasks given by people need to be achieved within a limited time bound as a person does not wait indefinitely for the robot to fetch an object. We discuss temporal constructs in Section 5.3. Another reason to consider the transition of time is to coordinate the actions of the robot and another agent (either another robot or a person) at a specified time instance. We explore this possibility in Section 5.7.

While there are many types of complexities inherent in the real world which we did not address here, such as noisy sensors and partial observability, the ones we considered pose interesting challenges which have not been studied in the context of RRL.

## 5.2 Dynamic Object Relational Markov Decision Process

An RMDP consists of a set of objects  $\mathbf{O}$  which remains unchanged in every time step. We consider dynamic objects which can be added to or removed from  $\mathbf{O}$ .

### Definition 30 (Dynamic Object)

*A dynamic object is an object which is added to or removed from the set of objects  $\mathbf{O}$  at any time step.*

Dynamic objects are an artefact of the unknown or ignored objects in the environment or can also be due to the creation of new objects in an environment capable of doing so (e.g., an assembly plant). An MDP or RMDP cannot model problems with dynamic objects since  $\mathbf{O}$  changes dynamically which in turn changes the set of state predicates  $\mathbf{P}$ , the set of ground actions  $\mathbf{A}$ , and the state space  $\mathbf{S}$ . We adapt the dynamic object RMDP (DORMDP) proposed by Mausam and Weld [109] to model problems with dynamic objects.

### Definition 31 (Dynamic Object RMDP)

*A DORMDP is a tuple  $(\mathcal{C}, \mathcal{P}, \mathcal{A}, \mathbf{O}, \mathcal{T}, \mathcal{R}, s_0, H, \gamma)$  where the elements are the same as those in an RMDP (see Definition 3) except that objects can be added to or removed from  $\mathbf{O}$  due to the effects of actions or exogenous events. Since  $\mathbf{P}$  and  $\mathbf{A}$  are defined over  $\mathbf{O}$  and  $\mathbf{S}$  is defined over  $\mathbf{P}$ , they are updated when  $\mathbf{O}$  changes.*

The number of objects in  $\mathbf{O}$  is unbounded due to dynamic objects. Thus, the cardinality of  $\mathbf{P}$ ,  $\mathbf{A}$ , and  $\mathbf{S}$  are also unbounded. When a dynamic object is added, state predicates which represent its properties or its relations with other objects are added to  $\mathbf{P}$ . As a result, the state changes when dynamic objects are added. Actions which involve this dynamic object are also added to  $\mathbf{A}$ .

### 5.2.1 Dynamic Objects in Service Robot

In *Service Robot*, the following objects are initially not made known to the robot:

1. all objects of type *person*, and
2. objects of type *wp* if `PERSON_IS_AT(PERSON, WP)` is true.

An object *PERSON* of type *person* is known if *PERSON* needs assistance

(`need_assistance(PERSON)`) or the robot receives a task to deliver an item to *PERSON* (`goal_object_with(OBJ, PERSON)`). When objects of type *person* are known, they are added to  $\mathbf{O}$ . Also, state predicates and actions with at least one of these objects in its terms are added to  $\mathbf{P}$  and  $\mathbf{A}$ , respectively. The objects *PERSON* of type *person* and *WP* of type *wp* are known if `person_at(PERSON, WP)` is true after the robot executes the action `find_person(ROBOT, PERSON)`. `person_at(PERSON, WP)` represents the knowledge of the robot and is different from `PERSON_IS_AT(PERSON, WP)` which represents the fact that *PERSON* is at *WP*. When objects of type *WP* are known, they are added to  $\mathbf{O}$ , and the ground state predicates of `robot_at(ROBOT, WP)` and ground actions of `move(ROBOT, WP1, WP2)` associated with these objects are added to  $\mathbf{P}$  and  $\mathbf{A}$ , respectively.

Considering more dynamic objects might seem to make the problem harder to solve but this is not always the case. The fewer objects there are in  $\mathbf{O}$ , the fewer actions there are to select from because  $\mathbf{A}$  is defined over  $\mathbf{O}$ . Suppose that all objects are unknown except for the robot (*r1*) and its initial location. In an initial state  $s_0$  where the robot is at *wp1* (`robot_at(r1, wp1)`), we have  $\mathbf{O} = \{r1, wp1\}$  and  $\mathbf{A} = \{\text{localise}(r1)\}$ . In the next time step, the exogenous event `need_assistance(p1)` occurred. *p1* is added to  $\mathbf{O}$  and `find_person(r1, p1)` is added to  $\mathbf{A}$ . There are now two actions to select from. The robot executes `find_person(r1, p1)` and finds *p1* at *wp4*; *wp4* is added to  $\mathbf{O}$  and `move(r1, wp1, wp4)` is added to  $\mathbf{A}$ . Now, there are three actions to select from. The small cardinality of  $\mathbf{A}$  makes the problem trivially simple.

### 5.3 Temporal Considerations

In this section, we present temporal considerations for problems which are typical of acting in the real world where actions are executed over time and it is preferable to achieve goals as quickly as possible.



```

(:durative-action put_down
 :parameters (?r - robot ?o - obj ?loc - wp)
 :duration (= ?duration 60)
 :condition (and (over all (robot_at ?r ?loc))
                 (at start (holding ?r ?o)))
 :effect (and (at end (not (holding ?r ?o)))
              (at end (object_at ?o ?loc)))
 )

```

Figure 5.1: A PDDL2.1 action model for `put_down` in `Service Robot`.

### 5.3.1 Durative Actions

In the previous chapters, actions are assumed to be instantaneous; states change instantaneously upon execution according to the effects of actions (if any). For problems where the notion of time does not matter, this assumption is valid. If this is not the case, then the duration which an action executes for needs to be considered. PDDL2.1 [52] is a formal planning language extended from PDDL [111] (see Section 2.2.2) to include durative actions—actions with temporally annotated conditions (in place of preconditions) and effects. An example of a PDDL2.1 action model is shown in Figure 5.1. An annotated condition specifies if the condition must hold at the start of the execution, the end of the execution, or over the duration of the execution. The annotated effects specify if the effects take place at the start or the end of the execution. On the other hand, for instantaneous actions, their preconditions must hold at the start of the execution and their effects take place at the end of the execution.

We use RDDL [149] to model domains and problems. Since RDDL does not support temporal constructs, we consider a simpler form of durative actions which does not have temporally annotated conditions and effects but only a duration for execution; preconditions and effects function in the same way as those of instantaneous actions. Hereafter, we refer to the durative actions from PDDL2.1 as PDDL2.1 actions and our form of durative actions as durative actions. Unlike PDDL2.1 actions, durative actions can have probabilistic effects.

#### Definition 32 (Durative Actions)

*A durative action is an action which takes an amount of time to complete execution, starting in a state and finishing in another state (which could be the same state*

as before). The action duration  $T_{dur}$  is the amount of time it takes.  $T_{dur}$  can be deterministic or stochastic and drawn from a distribution which could be unknown. The action's preconditions must be satisfied at the start of its execution while its effects take place at the end of its execution.

One of the uses for temporally annotated conditions and effects is to model problems where actions can execute concurrently such as multi-agent problems as it is important to consider actions which can interfere with each other (e.g., a robot is opening a door while another robot is trying to closing it). While our work does not consider temporally annotated conditions like temporal planners such as [12, 28] do, we address probabilistic problems which PDDL2.1 cannot represent. A SMDP (see Section 2.1.4) can be used to model problems with durative actions.

### 5.3.2 Time-Bounded Goals

Goals can have urgency where it is desired to achieve the goals as quickly as possible. For example, a person is unlikely to wait indefinitely for a robot to deliver an item. A time-bounded goal (TG) is a goal which must be achieved within a time bound or time window.

#### Definition 33 (TGs and Actions)

A TG is a tuple  $\dot{g} = (g, a, T^+, T^-)$  where  $g$  is a goal predicate,  $a$  is an action,  $T^+$  and  $T^-$  are the start and end times, respectively, of the time bound  $[T^+, T^-]$ , and  $T^+ \leq T^-$ .<sup>1</sup>  $\dot{g}$  is achieved if  $g$  is made true within the time bound due to the execution of  $a$  which must begin at  $T^+$ . The action is an optional argument without which there is no constraint on the action or the start time of its execution to achieve the TG.  $\dot{g}$  is violated and can no longer be achieved if it is not achieved by  $T^-$ .

In the previous chapters, we dealt with goals with no time bounds nor specific actions to achieve them. Such a goal can be expressed as a TG:  $\dot{g} = (g, \emptyset, 0, \infty)$ . For simplicity, we refer to such goals and TGs collectively as TGs for the remainder of this chapter. TGs that include actions address situations where a specific action is required, possibly because of the nature (or hazards) of the environment or for the

---

<sup>1</sup>The accent  $\dot{x}$  denotes a time-dependent entity,  $T$  denotes the continuous time, and  $t$  denotes the discrete time step.

coordination with other agent(s). We discuss the coordination of multiple agents in Section 5.7.

### TGs in Service Robot

In *Service Robot*, two TGs are given when the robot talks to a person who needs assistance:  $\dot{g}_i = (g_i, \emptyset, T_i^+, T_i^-)$  and  $\dot{g}_j = (g_j, \emptyset, T_j^+, T_j^-)$ . An implicit order to achieve these two TGs is imposed by defining the time bounds for the two TGs as:

$$T_i^+ = T_j^+ = T, \quad (5.1)$$

$$T_i^- = T_i^+ + \Delta T, \quad (5.2)$$

$$T_j^- = T_j^+ + 2\Delta T, \quad (5.3)$$

where  $\Delta T$  is a real value which represents the amount of time given to achieve TGs and  $T$  represents the elapsed time when the robot is given the TGs (i.e., after talking to the person). No actions are involved in the TGs. The optimal behaviour is to achieve  $\dot{g}_i$  first as it has  $\Delta T$  remaining time to achieve it while  $\dot{g}_j$  has  $2\Delta T$  remaining time. The order to achieve the TGs,  $\dot{g}_i \prec \dot{g}_j$ , is implicit because there is no state predicate which represents this relation. The order is not a hard constraint if  $\Delta T$  is large enough such that  $\dot{g}_i$  can be achieved after  $\dot{g}_j$ . Another possible implicit order is  $\dot{g}_j \prec \dot{g}_i$  if the values of the time bounds are defined as  $T_i^- = T_i^+ + 2\Delta T$  and  $T_j^- = T_j^+ + \Delta T$ . To prevent a learning algorithm from memorising the order of TGs to achieve, the implicit order is randomised between the two possibilities in each episode.  $\Delta T$  must be set appropriately such that it is not trivially large (e.g.,  $\Delta T \approx \infty$ ) or unreasonably small (e.g.,  $\Delta T = 1$  second). Since TGs are active only upon talking to a person who needs assistance, an optimal behaviour is to attend to a person who needs assistance after all active TGs are achieved since there is no penalty for keeping a person waiting (other than the action cost of  $-1$ ).

#### Example 34 (Act First, Talk Later)

*Suppose that the robot talks to a person and receives two TGs:  $\dot{g}_1 = (g_1, \emptyset, T, T + \Delta T)$  and  $\dot{g}_2 = (g_2, \emptyset, T, T + 2\Delta T)$  at the elapsed time  $T$ . On average, the robot has  $\Delta T$  of time to achieve each TG. Subsequently, the robot talks to another person and receives*

two more TGs:  $\dot{g}_3 = (g_3, \emptyset, T + \epsilon, T + \epsilon + \Delta T)$  and  $\dot{g}_4 = (g_4, \emptyset, T + \epsilon, T + \epsilon + 2\Delta T)$  at the elapsed time  $T + \epsilon$ . On average, the robot has approximately  $\frac{\Delta T}{4}$  of time to achieve each TG, half of what it had before talking to the second person (for simplicity, we assume that  $\epsilon$  is negligible here).

### 5.3.3 Time-Dependent DORMDP

We propose the time-dependent dynamic object RMDP (TDORMDP) to model problems which have a relational structure, dynamic objects, durative actions, and TGs. A TDORMDP combines a DORMDP with a SMDP (see Section 2.1.4).

#### Definition 34 (Time-Dependent Dynamic Object RMDP)

A TDORMDP is a tuple  $(\mathcal{C}, \mathcal{P}, \dot{\mathbf{A}}, \mathbf{O}, \dot{\mathbf{G}}, \dot{\mathcal{T}}, \dot{\mathcal{R}}, \mathcal{D}, \dot{s}_0, H, \gamma)$  where the elements are the same as those in an DORMDP (see Definition 31) except that  $\dot{\mathbf{A}}$  is a set of symbolic durative actions,  $\dot{\mathbf{G}}$  is a set of TGs,  $\dot{\mathcal{T}}$  is the time-dependent parameterised transition function,  $\dot{\mathcal{R}}$  is the time-dependent parameterised reward function,  $\mathcal{D} : \mathbf{S} \times \mathbf{A} \times \mathbf{S} \rightarrow \mathbb{R}$  is the action duration distribution, and  $\dot{s}_0$  is the initial state augmented with the initial elapsed time  $T_0$ .

The set of durative actions,  $\dot{\mathbf{A}}$ , is defined over  $\dot{\mathbf{A}}$  and  $\mathbf{O}$ . The time-dependent transition function  $\dot{\mathcal{T}} : \dot{\mathbf{S}} \times \dot{\mathbf{A}} \times \dot{\mathbf{G}} \rightarrow \dot{\mathbf{S}}$  is the result of the grounding of  $\dot{\mathcal{T}}$ .  $\dot{\mathbf{S}}$  is a set of states  $\mathbf{S}$  augmented with the elapsed time. The transition of state predicates depends on the state, the elapsed time, action durations, and TGs. Similarly, the time-dependent reward function  $\dot{\mathcal{R}} : \dot{\mathbf{S}} \times \dot{\mathbf{A}} \times \dot{\mathbf{G}} \rightarrow \mathbb{R}$  is the result of the grounding of  $\dot{\mathcal{R}}$  and defines the immediate reward after executing a durative action in a state augmented with the elapsed time.

The elapsed time at the initial state,  $T_0$ , is typically initialised to 0. The elapsed time at time step  $t$ ,  $T_t$ , is incremented by the action duration  $T_{dur}$  at time step  $t$ :  $T_t = T_{t-1} + T_{dur}$ .  $\mathcal{D}$  is a strictly positive and continuous distribution for action durations and is defined over ground actions and ground states. We do not assume that  $\mathcal{D}$  has a first-order structure unlike the transition function and reward function as this is often violated. For example, the action duration for moving between locations depends on the length of the path and does not follow a first-order relation.  $\dot{\mathbf{G}}$  can be the empty set initially and TGs can be added to  $\dot{\mathbf{G}}$  at any time step. The

achievement or violation of TGs at a time step  $t$  depends on the state  $\dot{s}_t$ , the durative action  $\dot{a}_t$  executed at  $t$ , and the elapsed time  $T_{t+1}$ . Any change to the state,  $\mathbf{O}$ , and  $\dot{\mathcal{G}}$  takes into effect after the execution of an action and not before or during. That is, the changes occur at discrete time steps  $t$  rather than continuous time  $T$ .

## 5.4 Dealing with TDORMDP Problems

In Section 4.1, we presented **GK-RRL** (Algorithm 11). In this section, we discuss the impact of dynamic objects and TGs on **GK-RRL** and propose extensions to address some of them. We refer to the extension of **GK-RRL** to solve a TDORMDP problem as **GK-RRL+**. The input problem to **GK-RRL+** is a TDORMDP rather than an RMDP. Dynamic objects affect the ground approximation (see Section 3.2), contextual grounding (see Section 3.4.2), **MBFS** (see Section 4.3.1), and model-based methods, **MQTE** (see Section 3.5), **Dyna** (see Section 4.4.2), and **UCT** (see Section 4.3.2). On the other hand, TGs affect contextual grounding and all of the aforementioned model-based methods. Existing work related to TDORMDPs was discussed in Section 2.7.1.

### 5.4.1 Dynamic Base Features

A ground approximation uses literals as base features. Since new state predicates are added to  $\mathbf{P}$ , their literals have to be added as base features to  $\Phi$  which in turn generates new candidate features. Likewise, if objects are removed from  $\mathbf{O}$ , then state predicates and features which involve these objects are removed from  $\mathbf{P}$  and  $\Phi$ , respectively. This dynamically changing set of base features, or dynamic base features, seems analogous to online feature discovery but with some differences. In online feature discovery, conjunctive features are added to reduce the approximation errors and granularity of the function approximation. Here, base features are added because the state representation has changed. A Q-function maps state-action pairs to Q-values. If the representation of the state changes with time, then this mapping changes and the optimal Q-function is non-stationary. Thus, dynamic objects induce non-stationary in the optimal Q-function which could increase the sample complexity.

Since a first-order approximation is an abstraction of the ground approximation (see Section 3.3.1), the set of first-order base features  $\hat{\mathcal{P}}$  could also be affected when  $\mathcal{O}$  changes. However, it is to a lesser extent than the ground approximation as the mapping of literals to base features is one-to-one while the mapping of literals to first-order base features is typically many-to-one.

We can initialise  $\hat{\mathcal{P}}$  such that it does not change regardless of the change in  $\mathcal{O}$ . Suppose that  $\mathcal{O}$  changes at time step  $t$ , an MDP is constructed from a DORMDP or TDORMDP (in the same way an MDP is constructed from an RMDP) and represents a new problem. If this problem subsumes the previous problem at time step  $t - 1$ , then new first-order base features are added to  $\hat{\mathcal{P}}$ . Otherwise, if the two problems are abstract-equivalent, then  $\hat{\mathcal{P}}$  remains unchanged. We presented the subsumption of problems and abstract-equivalent problems in Section 3.3.2. A problem subsumes another problem if it has multiple objects of a type while the subsumed problem has only one object of the same type. If we initialise  $\hat{\mathcal{P}}$  by assuming that there are at least two objects of every type, then the addition of dynamic objects will not change  $\hat{\mathcal{P}}$ . If an object is removed, we do not need to remove any base features from  $\hat{\mathcal{P}}$  as the grounding of first-order features considers the updated  $\mathcal{O}$ . Thus, a first-order approximation is a more suitable for solving DORMDP and TDORMDP problems.

### Effect on Grounding First-Order Approximation

Following the above solution, dynamic objects will not affect  $\hat{\mathcal{P}}$ . However, dynamic objects still affect the first-order approximation. When they are added to  $\mathcal{O}$ , they are considered by contextual knowledge. Conversely, if objects are removed from  $\mathcal{O}$ , then they will no longer be considered for grounding free variables. In other words, dynamic objects affect the grounding of first-order features. While this seems innocuous since the optimal Q-values might change depending on what objects there are, it can worsen performance if this is not the case. We illustrate this with an example.

#### Example 35 (Effect of Dynamic Objects on Evaluation of Features)

*In Service Robot, the first-order base features for the symbolic action  $\hat{a} = \text{find\_person}(\text{ROBOT}, \text{PERSON})$  include:*

- $\phi_1 = \text{need\_assistance}(\text{PERSON})$
- $\phi_2 = \neg \text{need\_assistance}(\text{PERSON})$
- $\phi_3 = \text{need\_assistance}(\star \text{PERSON})$
- $\phi_4 = \neg \text{need\_assistance}(\star \text{PERSON})$

When there is only one object of type *PERSON*,  $\phi_3$  and  $\phi_4$  will not evaluate to true as there is no other object to substitute  $\star \text{PERSON}$  with. Suppose that there are two people,  $p1$  and  $p2$ , and only  $p1$  needs assistance. If  $p1$  and  $p2$  are in  $\mathbf{O}$ , then  $\phi_1$  and  $\phi_4$  are true for the action `find_person(r1, p1)`. If only  $p1$  is in  $\mathbf{O}$  (i.e.,  $p2$  has not been added), then only  $\phi_1$  is true. The existence of  $p2$  in  $\mathbf{O}$  could affect the *Q*-values of `find_person(r1, p1)` though it shouldn't.

Example 35 illustrates the dependence of the grounding of first-order features on the class of the problem. That is, the grounding can be different in RMDPs where there are no dynamic objects and in DORMDPs or TDORMDPs where there are dynamic objects. This affects transfer learning between different classes of problems which we shall investigate empirically in Section 5.6.

### Effect on MBFS

MBFS determines the set of base features from the connectivity of graphs constructed from the CPFs of every state predicate in  $\mathbf{P}$ . When state predicates are added to  $\mathbf{P}$  due to the addition of dynamic objects to  $\mathbf{O}$ , MBFS constructs graphs for these new state predicates and there could be additional base features. If the base features have changed, one option is to retrain the *Q*-function approximation from scratch by replaying observations from the experience buffer.

### 5.4.2 Representing Violated TGs

Goal context is determined from TGs as before. However, if a TG is violated, it is no longer an active goal and will not be considered by goal context. This could affect the grounding of first-order features and change the *Q*-values. Ideally, this changes the policy such that other active TGs are pursued. However, if the first-order features do not contain free variables which are grounded by goal context, then its grounding is unaffected by the violated TG.

**Example 36 (Violated TG and Goal Context)**

Goal context gives  $\sigma_{goal=g} = \{\star OBJ/o1, \dots, \star PERSON/p1\}$  for a TG with the goal predicate  $g = \text{goal\_object\_at}(o1, p1)$ . If the TG is violated in a state  $\dot{s}_i$ , then goal context will not consider  $g$  and the free variables  $\star OBJ$  and  $\star PERSON$  might not be substituted with  $o1$  and  $p1$ , respectively. This affects the Q-values of actions if the grounding of their first-order features change as a result. Otherwise, their Q-values can remain unaffected. Consider another state,  $\dot{s}_j$ , which is identical to  $\dot{s}_i$  except for the elapsed time. In  $\dot{s}_j$ , the TG is not yet violated. The Q-values  $\tilde{Q}(s_i, \text{give}(r1, o1, p1))$  and  $\tilde{Q}(s_j, \text{give}(r1, o1, p1))$  can be the same because  $\sigma_{goal=g}$  is not used to substitute  $\star OBJ$  and  $\star PERSON$  for  $\text{give}(r1, o1, p1)$  in all states. This is because  $o1$  and  $p1$  are objects in the terms of  $\text{give}(r1, o1, p1)$ ; the bound variables  $OBJ$  and  $PERSON$  rather than the free variables are substituted with  $o1$  and  $p1$ , respectively.

The reason why the Q-values could remain unchanged is because from the perspective of the Q-function approximation, the state has not changed. The Q-function approximation considers the state represented by features or first-order features. If there are no literals which represent the fact that a TG is violated, then only the change in contextual grounding due to a violated TG can affect the Q-values. If the Q-values of some actions remain unchanged, the policy might attempt to achieve a violated TG. In Example 36, the policy might select  $\text{give}(r1, o1, p1)$  in  $\dot{s}_i$  to achieve  $\text{goal\_object\_at}(o1, p1)$  even though the TG is violated. The state transition is unaffected in this case;  $o1$  is now with  $p1$  but the immediate reward is  $-1$  (i.e., the action cost). A ground approximation also suffers from the same issue and perhaps even more so since it does not depend on contextual grounding.

We discuss four possible solutions. The first solution is to use the elapsed time as a base feature. Since time is a continuous real value, the set of features is now a hybrid of binary and real-value features. This causes some complications. **iFDD+** is a feature discovery algorithm for binary features. We reproduce Equation 3.1 for the relevance of a candidate feature here for ease of reference:

$$\eta = \frac{\left| \sum_{i=0, \phi(s_i, a_i)=1}^t \delta_i \right|}{\sqrt{\sum_{i=0, \phi(s_i, a_i)=1}^t 1}}$$



The relevance of a candidate feature is the normalised, cumulative absolute of TD errors observed when the candidate feature is true. We modify the equation to consider real-valued features:

$$\eta = \frac{|\sum_{i=0}^t \phi(s_i, a_i) \delta_i|}{\sqrt{\sum_{i=0}^t \phi(s_i, a_i)}}. \quad (5.4)$$

Equation 5.4 is equivalent to Equation 3.1 for binary features. For real-valued features, we weight the TD error by the value of the feature which has to be normalised such that  $\phi(s, a) \in [0, 1]$ . The maximum duration of time has to be known for normalisation. If this value is large, then the base feature for the elapsed time is insignificantly small initially. Another possibility which averts this issue is to use the normalised remaining time before a TG is violated:  $\frac{T-t}{\Delta T}$ . Pardo et al. [131] show that by conditioning the value function on the remaining time, the policies learned consider the remaining time. However, they consider discrete time steps while we deal with continuous time which is a considerably harder problem.

In either case, using the elapsed time or remaining time as a feature is not suitable in a first-order approximation which abstracts actions to symbolic actions. This is because while symbolic actions have similar effects (assuming the problems are relational), they might not have similar action durations. For example, consider the symbolic action `move(ROBOT, WP1, WP2)` which has a duration dependent on the length of the path from `WP1` to `WP2`. The action duration affects the transition of elapsed time and the value of the feature for time. If the action durations cannot be generalised for symbolic actions, it is not possible to use time as a first-order feature.

The second solution is to introduce a symbolic state predicate which represents the fact that a TG is violated. For example, the symbolic state predicate `violated(OBJ)` can be added to `Service Robot` to represent the fact that the TG involving `OBJ` is violated. From Definition 3, problems of a domain have the same set of symbolic state predicates  $\mathcal{P}$ . Therefore, this solution creates a new variant domain of `Service Robot`. Transfer learning between domains, or inter-domain transfer, is not straightforward and typically requires mappings between state predicates and actions of both domains [172]. Furthermore, increasing the cardinality of

$\mathcal{P}$  leads to an increase in the size of the state space. We use an intermediary solution which utilises  $\text{reward\_received}(OBJ) \in \mathcal{P}$  to represent a TG that is achieved or will have been achieved had it not been violated. This is not quite the same as  $\text{violated}(OBJ)$  but has an advantage of not changing  $\mathcal{P}$  and the transition function remains time-invariant. The reward function is time-dependent as positive rewards are not given for achieving violated TGs. We discuss the significance of time-invariant transition functions in Section 5.4.3.

The third solution is to consider goal conditioned Q-functions or value functions. We discussed them in Section 3.4.2 and concluded that they are not suitable for relational problems. The last solution is to use a model-based method which performs policy rollouts to estimate Q-values. This not only considers whether TGs are violated in the current state but also in subsequent states and can provide better estimates of the Q-values. Thus, we chose this solution which shall be described in the next section.

### 5.4.3 Model-Based Methods

We presented the model-based methods **MQTE** in Section 3.5, **Dyna** in Section 4.4.2, and **UCT** in Section 4.3.2. These methods use a generative model to predict the next state and immediate reward. To solve TDORMDPs, the generative model must predict the next state with its elapsed time.

Our variant of **Dyna** generates imagined observations at the start rather than at every time step (see Algorithm 16), and thus there are no imagined observations which consist of dynamic objects since they are not in  $\mathcal{O}$  initially. Since **Dyna** simulates rollouts only in the initial state space (before dynamic objects are added), the Q-function approximation can generate a policy which is poor in regions of the state space where dynamic objects are added (or the expanded state space). However, we posit that this might not be the case as first-order approximations are less unaffected by dynamic objects and its generalisation property extends learning in the initial state space to the expanded state space.

## Predicting and Propagating the Transition of Elapsed Time

The elapsed time in a state is the elapsed time in the previous state plus the action duration in the previous step. The elapsed time is propagated during policy rollout in MQTE, Dyna, and UCT in order to reason about the remaining time to achieve TGs or to determine if TGs are active or violated. We assume that action durations are not known and have to be learned. In **Service Robot**, the action durations are stochastic. We assume that the action duration distribution  $\mathcal{D}$  is normally distributed. Given a set of observations, the generative model predicts an action duration as the observed mean plus two standard deviations which gives a 95% confidence. In our application (see Example 5.3.2), it is preferred to overestimate the duration as TGs are violated only when the elapsed time has exceeded the end time of its time bound ( $T^{-1}$ ).

### Search Tree Reuse in UCT

The search tree built in the previous time step by UCT can be reused if the current state (or observed state in the current time step) matches the state of a child node of the root node. This was discussed in Section 4.3.2. If dynamic objects are added to  $\mathbf{O}$ , then the state changes since  $\mathbf{P}$  changes and the search tree cannot be reused. Intuitively, the search tree should be build from scratch when new information about the state is known.

Since a state is augmented with the elapsed time, it is almost never the case that the predicated elapsed time is equal to the observed elapsed time because they are continuous values. Therefore, the condition to reuse the search tree is relaxed. If the predicted elapsed time is not more than the observed elapsed time and there are no TGs added to  $\mathcal{G}$  in the previous time step, then the search tree is reused. If the first condition is not satisfied, then the search tree was built on the premise of an underestimation of the elapsed time; TGs which were achieved in the search tree might actually be violated. Thus, the Q-values of actions estimated by UCT are no longer accurate. If the second condition is violated, the search tree has to be build from scratch to consider the new TGs.

## Time-Invariant Transition of State Predicates

We use a variant of TDORMDP where the transitions of state predicates in  $\dot{\mathcal{S}}$  do not depend on time. While UCT can deal with time-dependent transition functions if the generative model can predict the time-dependent transitions of state predicates, the model learner RLFIT [108] only considers discrete states, and thus cannot learn time-dependent transition functions.

### 5.4.4 Learning from Hindsight

If the Q-function approximation considers state representations which do not contain information on whether TGs are violated, as discussed in Section 5.4.2, then TD learning can update the Q-function approximation wrongly. Suppose that there is one TG  $\dot{g} = (g, \dot{a}, T^+, T^-)$ . In a state  $\dot{s}_i$  where the elapsed time  $T_i < T^-$ , the action  $\dot{a}$  is executed which achieves the goal predicate  $g$ . Since this is achieved before  $T^-$ ,  $\dot{g}$  is achieved and an immediate reward  $r > 0$  is given. Now, in another state  $\dot{s}_j$  which is identical to  $\dot{s}_i$  except that the elapsed time  $T_j > T^-$ ,  $\dot{a}$  is executed which achieves  $g$  but  $\dot{g}$  is already violated. The immediate reward is 0. In these two scenarios, the states  $\dot{s}_i$  and  $\dot{s}_j$  map to the same state  $s$  in the Q-function approximation. Given that  $\tilde{Q}(s, \dot{a})$  is initialised to zero, TD learning updates and increases  $\tilde{Q}(s, \dot{a})$  in the first scenario since  $r > 0$ . This encourages the policy to select  $\dot{a}$  in both  $\dot{s}_i$  and  $\dot{s}_j$ . Suppose that the second scenario is encountered in many episodes thereafter and  $\tilde{Q}(s, \dot{a}) \sim 0$ . This discourages the policy from selecting  $\dot{a}$  in  $\dot{s}_i$  and  $\dot{s}_j$  even though executing  $\dot{a}$  in  $\dot{s}_i$  achieves  $\dot{g}$ .

To resolve this, the observation in the second scenario is imagined to have achieved  $\dot{g}$ ; TD learning updates  $\tilde{Q}(s, \dot{a})$  with the immediate reward  $r$  instead of 0. This imagined observation is only used in TD learning and does not affect the evaluation metric (e.g., the total return). The imagination of achieving a violated TG is rather similar to the key concept in hindsight experience replay (HER) [4]. HER replays trajectories from the experience buffer, imagining that each trajectory achieves an arbitrary goal which is the last state in the trajectory. While HER is motivated by sample efficiency in problems with sparse rewards, the purpose of our approach is to prevent unsound updates in TD learning which can worsen performance.

### 5.4.5 Extensions to Online RRL

We summarise **GK-RRL+** which is the extension to **GK-RRL** (Algorithm 11). The modifications made, where line numbers refer to Algorithm 11, are as follows:

- Base features of Q-function approximations are initialised to account for dynamic objects (line 1) as discussed in Section 5.4.1. This includes both **MFFS** and **MBFS**.
- First-order features are grounded with  $\mathbf{O}$  which changes due to dynamic objects. This affects the computation of Q-values (line 2) and step update of a Q-function approximation (line 14).
- Model-based methods **MQTE** (line 8), **Dyna** (lines 3 to 4), and **UCT** (line 6) compute the transition of elapsed time as discussed in Section 5.4.3.
- The step update of a Q-function approximation (line 14) considers learning from hindsight as discussed in Section 5.4.4.

## 5.5 Simulation-to-Simulation Transfer

In simulation-to-real-world (Sim-to-Real) transfer, knowledge that is learned in a simulated environment is transferred to the real world to accelerate learning and improve performance in the real world. This utilises the abundance of simulations to reduce the number of real world experiments which are often more expensive. An important challenge in Sim-to-Real transfer is to overcome the gap between simulation and the real world which can cause policies learned in simulated environments to perform poorly in the real world.

Since we are not able to perform experiments in the real world due to limitations in resources, we perform simulation-to-simulation (Sim-to-Sim) transfer instead where knowledge is transferred from **RDDLsim**'s to **Gazebo**'s simulated environments. Notwithstanding the lesser challenge in Sim-to-Sim transfer, both Sim-to-Sim and Sim-to-Real transfer deal with the same challenge to knowledge transfer: discrepancies in the learning and test environments. Sim-to-Sim transfer can be regarded as a preliminary step towards Sim-to-Real transfer [60]. In the remainder of this section, we describe **Gazebo**'s simulated environment and its differences with **RDDLsim**'s simulated environment. Our work on Sim-to-Sim transfer was published

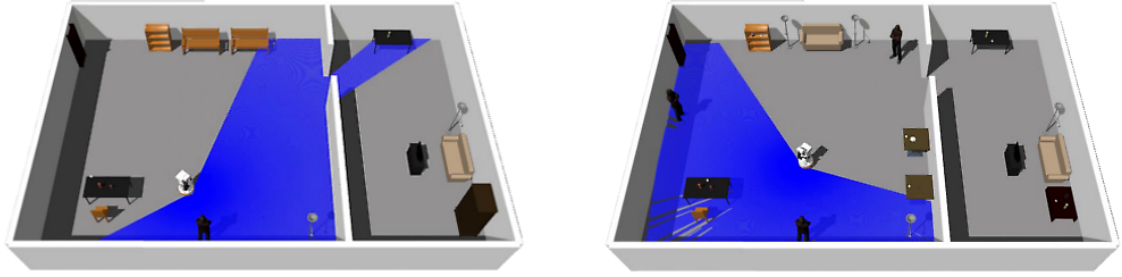


Figure 5.2: Simulated environments in Gazebo for the **Service Robot** domain. On the left is the environment for one person (SR1) and on the right is the environment for three people (SR2 and SR3).

in [122].

### 5.5.1 Simulated Environment in Gazebo

In the previous chapters, we used RDDLSim [149] to simulate the environment. RDDLSim returns the next state and immediate reward based on the preconditions, CPFs, and reward function defined in the RDDDL domain file. This simulated environment is computationally fast but lacks realism for robotic applications in the real world. A more realistic simulated environment can be build with Gazebo [89], a 3D simulator which utilises a physics engine. We built two 3D environments for **Service Robot** in Gazebo which are shown in Figure 5.2. Figure 5.3 shows snapshots of a particular experimental run where the robot achieves a goal. We use the Robot Operating System (ROS) [140], a set of software libraries and tools, to build the interface for the TIAGo robot. This interface includes software which perform sensor measurements (for localisation, navigation, and detecting people), motion planning (for navigation), manipulation planning (for grasping), etc. The ROSPlan framework [23, 25] is used to embed task planning into ROS. We implemented ROS packages to perform the following tasks:

- execute high-level actions (i.e., actions in  $\mathbf{A}$ );
- monitor the localisation accuracy of the robot;
- interface with GK-RRL+.

Our ROS packages are integrated with ROSPlan. We use existing ROS packages to execute some motor control actions such as navigation and grasping. The items are

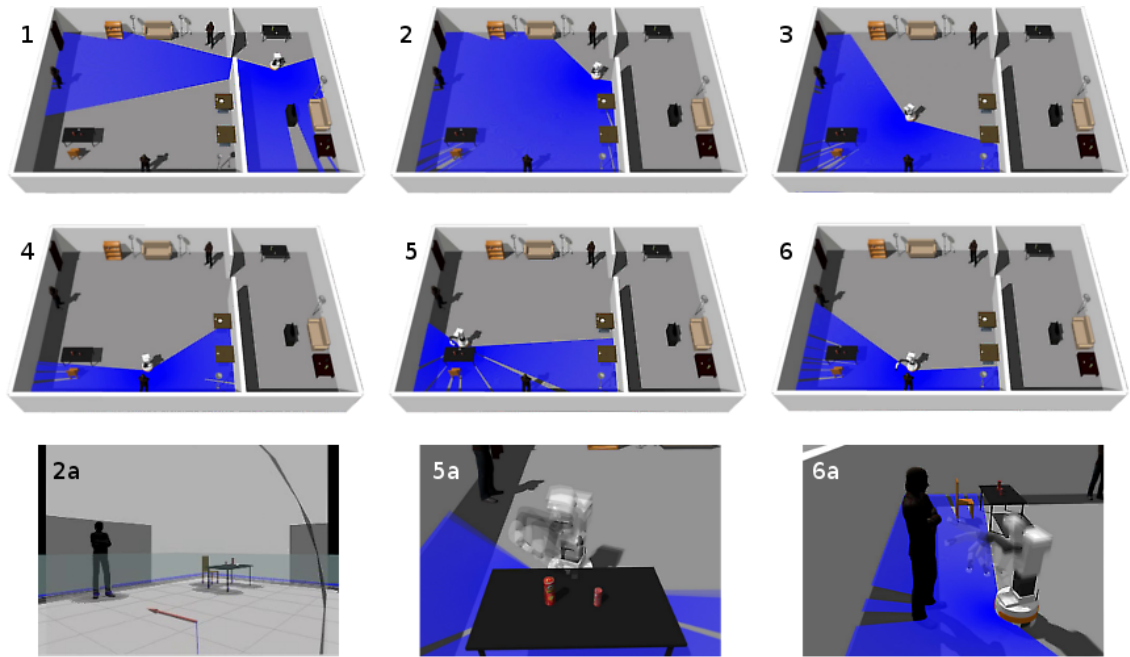


Figure 5.3: Snapshots of a robot executing some actions in the Gazebo environment. The blue rays emanating from the robot is its laser scan. (1): The robot localises. (2) and (2a): The robot finds a person using its camera where (2a) is the camera view. (3): The robot moves towards the person. (4): The robot talks to the person and receives some tasks. (5) and (5a): The robot picks up an item. (6) and (6a): The robot gives the item to the person and completes the task.

cubes with ArUco markers to make it easier for the robot to detect and grasp.<sup>2</sup>

### 5.5.2 Differences between Simulated Environments

There are some differences between RDDLSim’s and Gazebo’s simulated environments for **Service Robot**. Firstly, in RDDLSim, the robot loses localisation when moving with a probability of 0.1; in Gazebo, it is deemed to have lost localisation when the covariance of the pose estimation from adaptive Monte Carlo localisation exceeds user-defined thresholds (3 metres for position and 10 degrees for orientation). Secondly, in RDDLSim, the robot always succeeds in finding a person (`find_person(ROBOT, PERSON)`) and ends up in the same location as the person. In Gazebo, the robot follows an exploration path to find a person. It might find the intended person and/or other people it encounters along the way. Also, it might not be in the same location as the person since it could detect people from a distance with its camera. The action terminates when the robot reaches the end

<sup>2</sup>ArUco markers are binary square fiducial markers which provide points of references for computer vision algorithms to estimate the pose of objects.

of the exploration path or the intended person is found. Lastly, in RDDLSim, picking up and putting down an item always succeed while these actions might fail in Gazebo (the state remains unchanged when they fail).

Dynamic objects in **Service Robot** are defined in Section 5.2.1. In Gazebo, since a robot can find other people besides the intended person, dynamic objects of type *PERSON* can be added to  $\mathbf{O}$  (state predicates and actions are added to  $\mathbf{P}$  and  $\mathbf{A}$  accordingly) when the robot executes `find_person(ROBOT, PERSON)`. For example, *p2* can be added to  $\mathbf{O}$  when `find_person(r1, p1)` is executed. In RDDLSim, this will not happen since the robot will only find the intended person and no one else.

Our work (e.g., **GK-RRL+**) learns and plans at the task level rather than at the motor control level (e.g., making decisions on what actuation commands to give). At the task level, states are sufficiently represented by literals. On the other hand, sensory measurements (e.g., visual information from a robot’s camera) often need to be included as part of the state for motion control problems. Although Gazebo’s simulated environment is significantly richer than RDDLSim’s, for the purpose of decision making at the task level, this discrepancy does not directly affect Sim-to-Sim transfer. Research on Sim-to-Real transfer for motion control problems are concerned with discrepancies in physical quantities (e.g., friction, rigidity, mass) [27], visual information (e.g., lights, shadows, and textures) [74], and dynamics [73]. Our work is not concerned with and does not deal with these low-level discrepancies.

## 5.6 Empirical Evaluation and Discussion

We use two simulated environments for experiments: RDDLSim and Gazebo. Since RDDLSim does not consider time, we augment the observation from RDDLSim with the action duration which is the mean of the observed action durations in Gazebo injected with normally distributed noise (standard deviation is 20% of the mean).  $P_{source} \rightarrow P_{target}$  denotes that a first-order approximation learned in the source problem,  $P_{source}$ , is transferred to the target problem,  $P_{target}$ . The experimental setup is as described in Section 3.7.1. An episode can also be terminated if all TGs are either achieved or violated. If it terminates with at least one violated TG, then



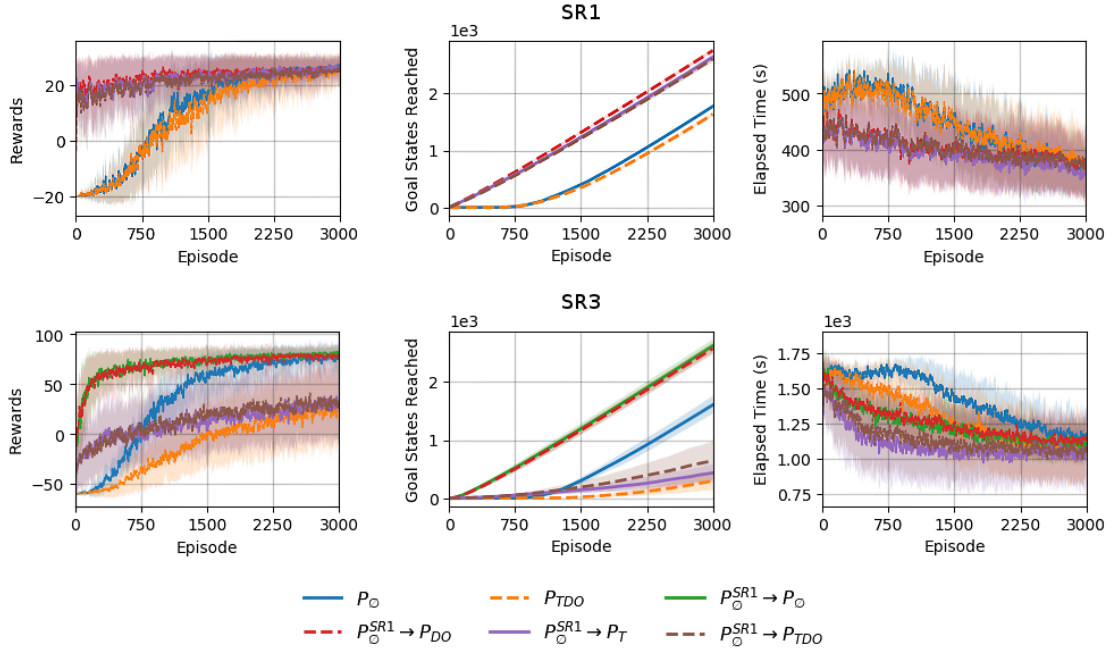


Figure 5.4: Results for learning from scratch and transfer learning between different classes of problems. The source problems are randomised RMDP problems of SR1 ( $P_{\emptyset}^{SR1}$ ). The target problems are randomised TDORMDP problems of SR1 (top row) and SR3 (bottom row), and can have dynamic objects ( $P_{DO}$ ), TGs ( $P_T$ ), or both ( $P_{TDO}$ ).

an immediate reward of  $-1$  is given for the remaining time steps. This is only for the presentation of results.

The effect of  $\Delta T$  on the performance when solving TDORMDP problems is examined and discussed in Section C.3.1. Following this, we use  $\Delta T = 300$  seconds for all subsequent experiments.

### 5.6.1 Dynamic Objects and TGs

We evaluate GK-RRL+ on problems with dynamic objects (denoted by  $P_{DO}$ ), problems with TGs ( $P_T$ ), and problems with dynamic objects and TGs ( $P_{TDO}$ ). As a baseline, problems without dynamic objects and TGs ( $P_{\emptyset}$ ) are also considered.  $P_{\emptyset}$  is represented by RMDPs while the rest are represented by TDORMDPs. We also investigate transfer learning from RMDPs to TDORMDPs where the source problems are  $P_{\emptyset}^{SR1}$  and the target problems are SR1 and SR3.<sup>3</sup>

<sup>3</sup>For a problem  $P$ , the subscript denotes if the problem has dynamic objects and/or TGs and the superscript denotes the problem instance.

**Learning from scratch.** First, we analyse the performance of learning first-order approximations from scratch. Figure 5.4 shows the results. For **SR1**, the performance in  $P_\emptyset$  and  $P_{TDO}$  are comparable with the former having a slightly better performance in terms of rewards and goal states reached. For **SR3**, the asymptotic performance in  $P_\emptyset$  is significantly better. This indicates that the presence of either dynamic objects, TGs, or both, can worsen the performance of first-order approximation. The elapsed time in  $P_\emptyset$  is longer than that in  $P_{TDO}$ . While a shorter elapsed time typically indicates a more optimal policy, this is not the case when TGs are involved. When TGs are violated and there remains no other TGs to be achieved, the episode terminates resulting in a shorter elapsed time.

**Transfer learning with model-free methods.** Next, we analyse the performance of transfer learning. We limit the discussion here to model-free methods and defer model-based methods to a later part of the section. Figure 5.4 shows the results. There are three classes of target problems:  $P_{DO}$ ,  $P_T$ , and  $P_{TDO}$ . This is to determine the individual impact of dynamic objects and TGs on the performance. For transfer learning from **SR1** to **SR1**, the source and target problems are different randomised problems. The result for  $P_\emptyset^{SR1} \rightarrow P_\emptyset^{SR3}$  was presented in Section 3.7.4 and is included here for ease of reference.

For **SR1**, the performance in the three classes of target problems are comparable. Transfer learning gives a jump start in performance, demonstrating that even if the first-order approximations are learned in  $P_\emptyset$ , they can still be utilised to reduce sample complexity in target problems involving dynamic objects and TGs. This holds true even for the large scale problem, **SR3**. Comparing the three types of problems for **SR3**, it is evident that TGs deteriorate the performance of first-order approximation; the performance in  $P_\emptyset^{SR1} \rightarrow P_\emptyset^{SR3}$  and  $P_\emptyset^{SR1} \rightarrow P_{DO}^{SR3}$  are comparable while the performance in  $P_\emptyset^{SR1} \rightarrow P_T^{SR3}$  and  $P_\emptyset^{SR1} \rightarrow P_{TDO}^{SR3}$  are comparable. We discussed the impact of dynamic objects on transfer learning between RMDP and TDORMDP in Section 5.4.1. Here, this is not evident. This is because the discrepancy between **SR1** and **SR3**, both of them are not abstract-equivalent problems, is greater than that due to dynamic objects. When the first-order approximation is updated in the target problem (i.e., **SR3**), it adapts to the target problem and dynamic objects

concurrently. In the next section, empirical results show that dynamic objects do affect transfer learning.

The first-order approximation is less affected by TGs in **SR1** than **SR3** because there are only two TGs in **SR1**. This gives two possible implicit orders of TGs which are randomised in each episode (see Section 5.3.2). The probability for a policy to choose the right order to achieve the TGs is 0.5. In **SR3**, there are six TGs which gives 720 possible implicit orders of TGs. It is much less likely for a policy to choose the right order to achieve the TGs by chance. If the policy learns that the robot should only talk to a person after active TGs are achieved, then there will be at most two active TGs at any time step (see Example 34). Since the first-order approximation is learned in  $P_{\emptyset}^{SR1}$  where there is only one person and goals have no time bounds, this behaviour was never learned. This behaviour was also not learned in  $P_{TDO}^{SR3}$  (i.e., learning from scratch) because there are no state predicates which represent the implicit order. Therefore, the Q-function approximation cannot generate a policy which achieves TGs in the implicit order.

In **Robot Fetch**, the symbolic state predicates `object_at(OBJ, WP)` and `OBJECT_GOAL(OBJ, WP)` imply an order to achieve goals. Similarly, in **Academic Advising**, the symbolic state predicate `PREREQ(COURSE1, COURSE2)` implies an order to achieve goals. We discussed the shortcoming of first-order approximation in dealing with interdependent goals in Section 3.4.1. Thus, the first-order approximation cannot deal with TGs which are interdependent due to the implicit order. While the ground approximation with its finer granularity can solve **Robot Fetch** and **Academic Advising**, as shown in Section 3.7.2, it does so by memorising the sequence of actions to reach the goal state (see Example 13). This will not work for the implicit order of TGs which is randomised in each episode. Furthermore, results in Section 3.7.2 show that the ground approximation performs poorly and does not scale well in **Service Robot**.

**Transfer learning with model-free and model-based methods.** We evaluate different methods for solving TDORMDP problems of **SR3**. Figure 5.5 shows the results. Unless stated otherwise, the experimental setup and hyperparameters are described in Section 4.5.6. The hyperparameters for **UCT** are  $\Lambda_{root} = \Lambda_{leaf} = 0.5$

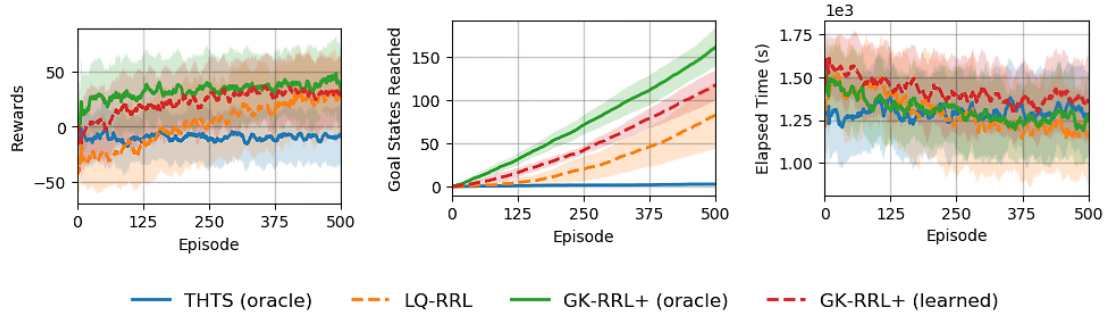


Figure 5.5: Performance of different methods in solving TDORMDP problems of SR3. The first-order approximation is learned in SR1 and transferred to SR3. If stated in the legend, either the oracle or learned model is used.

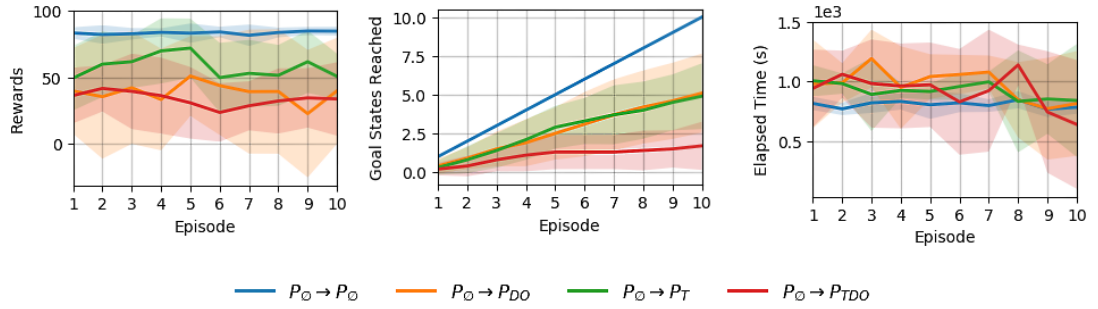


Figure 5.6: Result for Sim-to-Sim transfer.

and  $H_{rollout} = 5$ . The benchmarks are LQ-RRL and THTS [87]. GK-RRL+ uses the transferred first-order approximation and either the oracle model or learned model, LQ-RRL uses only the transferred first-order approximation, and THTS uses only the oracle model. The performance of THTS represents the best achievable performance by a state-of-the-art model-based method which does not consider the transition of time. It performs the worst as it attempts to achieve violated TGs, failing to reach the goal state most of the time. LQ-RRL performs initially worse than THTS but improves asymptotically. GK-RRL+ with the oracle model has the best performance followed by GK-RRL+ with the learned model. Their performance are significantly better than LQ-RRL and THTS. This demonstrates that the combination of model-based and model-free methods can solve TDORMDPs effectively.

### 5.6.2 Sim-to-Sim Transfer

In Sim-to-Sim transfer, knowledge learned in source problems, simulated in RDDLSim’s environment, are leveraged to solve target problems, simulated in Gazebo’s

environment. The source and target problems are different randomised problems of SR3 (20 problems in total). For brevity,  $P_{source} \rightarrow P_{target}$  denotes  $P_{source}^{SR3} \rightarrow P_{target}^{SR3}$  here.

**Zero-shot transfer.** The source problems do not have dynamic objects and TGs while there are four different classes of target problems:  $P_{\emptyset}$ ,  $P_{DO}$ ,  $P_T$ , and  $P_{TDO}$ . For each run out of ten independent runs, a first-order approximation is learned from scratch in 1000 episodes of the source problem where ten different problems are attempted sequentially; a new problem is attempted in every 100 episodes. In other words, each first-order approximation is learned over ten different problems rather than one problem. This tends to improve its generalisation such that it can generate policies which perform well in more problems. This approach draws similarities to [64, 109, 185] which generate training examples from multiple problems to learn generalised abstractions and shares the same motivation as domain randomisation [72, 175]. Each first-order approximation, paired with a different learned model (used by UCT), generates a greedy policy to solve ten target problems over ten episodes; a new problem is attempted in each episode. In essence, this is zero-shot transfer since a target problem is attempted in only one episode and with no prior observations of the target problem. The source and target problems are different randomised problems. UCT is used when TGs are involved (i.e.,  $P_{\emptyset} \rightarrow P_T$  and  $P_{\emptyset} \rightarrow P_{TDO}$ ).

**Experimental setup.** The hardware for running the experiments are different from previous experiments. We use a virtual machine with two Intel Core™ i7-930 2.80 GHz processors and 8 gigabytes of memory. The hyperparameters for UCT are  $\Lambda_{root} = \Lambda_{leaf} = 0.5$  and  $H_{rollout} = 5$ . The timeout for UCT is 30 seconds, an increase from 10 seconds for the experiments in Sections 4.5.6 and 5.6.1 because the hardware is now of considerably lower processing power.

Figure 5.6 shows the results for the four classes of target problems.<sup>4</sup> Firstly,  $P_{\emptyset} \rightarrow P_{\emptyset}$  performs well with near-optimal or optimal performance, solving every problem. This demonstrates zero-shot transfer between different simulated environments. Secondly,  $P_{\emptyset} \rightarrow P_{DO}$  performs poorer than  $P_{\emptyset} \rightarrow P_{\emptyset}$ . We also tried  $P_{DO} \rightarrow P_{DO}$  (results are not shown) and the performance is only marginally bet-

---

<sup>4</sup>Results are not averaged with a moving window.

ter. This is in contrast with the results in Section 5.6.1 which show that dynamic objects do not affect transfer learning. The reason is because the conditions under which dynamic objects of type *person* are added to  $\mathbf{O}$  are different in RDDLSim and Gazebo as discussed in Section 5.5.2. Thus, the combination of different simulated environments and classes of problems causes significant **domain shift** (i.e., the characteristics between the source and target problems are different) which affected zero-shot transfer.

Thirdly, the performance for  $P_\emptyset \rightarrow P_T$  in terms of total rewards is poorer than  $P_\emptyset \rightarrow P_\emptyset$  but better than  $P_\emptyset \rightarrow P_{DO}$ . The latter suggests that the domain shift due to TGs is less than that due to dynamic objects. The performance is sub-optimal because learned approximate models are used by UCT to deal with TGs. Both  $P_\emptyset \rightarrow P_T$  and  $P_\emptyset \rightarrow P_{DO}$  reached a comparable number of goal states, solving five out of ten problems on average. Lastly, we consider target problems with TGs and dynamic objects. Clearly, this class of problems pose the largest domain shift among the four classes we considered. As expected,  $P_\emptyset \rightarrow P_{TDO}$  has the worst performance and only a small number of goal states were reached.

The results for zero-shot transfer do not indicate the sample complexity of GK-RRL+ since no learning takes place in the target problems. Nevertheless, in spite of a domain shift, GK-RRL+ achieves a jump start in performance for all four classes of target problems. This is evident by comparing with the results for learning from scratch as shown in Figure 5.4 where the total rewards received initially is  $-60$  for SR3 (i.e., an immediate reward of  $-1$  is received in every time step).

## 5.7 Multi-Agent Coordination

So far, we have considered single-agent problems. In this section, we discuss how our work can be extended to address the coordination of multiple agents. First, we define coordinated actions. Next, we propose a framework which decomposes a multi-agent problem to single-agent problems, interleaving multi-agent coordination and single-agent decision making, learning, and acting. Each agent has a **single-agent module** (SAM) which is a collection of algorithms (e.g., GK-RRL+) and hardware (if any) responsible for decision making, learning, and acting. We refer to the

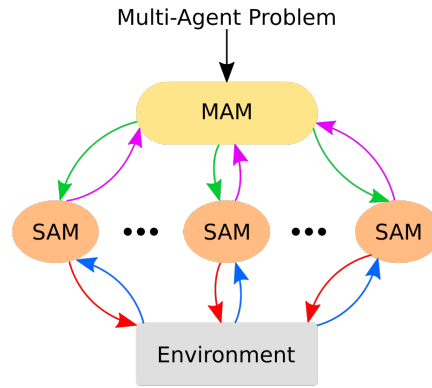


Figure 5.7: A multi-agent decomposition framework for multi-agent coordination which consists of a multi-agent module (MAM) and single-agent modules (SAMs). The green arrows denote information pertaining to the single-agent problems, the purple arrows denote feedback from the SAMs, the red arrows denote actions executed by the SAMs, and the blue arrows denote observations from the environment.

algorithm(s) responsible for multi-agent coordination to achieve goals and joint goals as the **multi-agent module** (MAM). Our multi-agent decomposition framework is illustrated in Figure 5.7. In later parts of this section, we present two variants of this framework. They differ primarily in their MAMs and the information exchanged between the MAM and the SAMs. Existing work related to multi-agent coordination was discussed in Section 2.7.2.

### 5.7.1 Coordinated Actions

Coordinated actions are actions which agents execute to produce an intended effect or to achieve a joint goal (i.e., a goal that is common among the agents). To coordinate, the joint goal is decomposed to a set of TGs, one for each agent involved. The TG is the tuple  $(g, a, T^+, T^-)$  where  $a$  is the durative or instantaneous action the agent needs to execute and  $g$  is the goal predicate which is part of the effects of  $a$ . The start time  $T^+$  of the TGs are equal if the coordination needs to start at the same time. Similarly, the end time  $T^-$  of the TGs are equal if the coordinated actions need to complete execution at the same time. We identify two types of coordinated actions.

#### Definition 35 (Weakly Coordinated Actions)

*Weakly coordinated actions are coordinated actions with no temporally annotated conditions and effects. Each agent involved has one coordinated action to execute.*

**Example 37 (Weakly Coordinated Actions to Achieve a Joint Goal)**

We consider a multi-agent variant of *Service Robot*. Suppose that two items,  $o_1$  and  $o_2$ , are to be brought to a person  $p_1$  at the same time by the time  $T + \Delta T$ . Since a robot can only hold at most one item, this requires two robots to achieve the task or joint goal. The joint goal is allocated to two robots,  $r_1$  and  $r_2$ . It is decomposed to two TGs:  $\dot{g}_1 = (\text{object\_with}(o_1, p_1), \text{give}(r_1, o_1, p_1), T', T + \Delta T)$  and  $\dot{g}_2 = (\text{object\_with}(o_2, p_1), \text{give}(r_2, o_2, p_1), T', T + \Delta T)$  where  $\dot{g}_1$  shall be achieved by  $r_1$  and  $\dot{g}_2$  by  $r_2$ , and  $T'$  is the elapsed time when both robots concurrently start to execute their respective actions.

Weakly coordinated actions, such as those described in Example 37, are coordinated actions which require only one TG for each agent involved. The coordination between agents is represented by the time bounds of the TGs. An immediate reward is given only when every agent achieves its respective TG within the time bound. Each agent learns and plans to achieve its TG assuming that the other agents will do their part. Based on this assumption, the agent's internal model (i.e., transition and reward functions) is independent of other agents and we can decompose the multi-agent problem to single-agent problems which are represented by TDORMDPs. An agent's internal model differs from the environment where state transitions and rewards depend on other agents.

**Definition 36 (Strongly Coordinated Actions)**

*Strongly coordinated actions are different from weakly coordinated actions in either one or both of the following aspects:*

- *Coordinated actions have temporally annotated conditions and effects; either the conditions, effects, or both, depend on other coordinated actions.*
- *Agents have a sequence of coordinated actions to execute.*

**Example 38 (Strongly Coordinated Actions to Achieve a Joint Goal)**

A task is to bring a large item  $o_1$  to a person  $p_1$ . Since a robot cannot move a large item alone, the task is a joint goal which requires two robots. Suppose that the robot  $r_1$  is at  $wp_1$ , the robot  $r_2$  is at  $wp_2$ ,  $p_1$  is at  $wp_4$ , and  $o_1$  is at  $wp_3$ . The sequence of actions which  $r_1$  has to execute are  $\text{move}(r_1, wp_1, wp_3)$ ,  $\text{pick\_up}(r_1, o_1)$ ,  $\text{move}(r_1, wp_3, wp_4)$ , and  $\text{give}(r_1, o_1, p_1)$ . The sequence of actions which  $r_2$  has



to execute are  $\text{move}(r2, wp2, wp3)$ ,  $\text{pick\_up}(r2, o1)$ ,  $\text{move}(r1, wp3, wp4)$ , and  $\text{give}(r2, o1, p1)$ .

The sequence of coordinated actions is (1)  $\text{pick\_up}(r1, o1)$  and  $\text{pick\_up}(r2, o1)$ , (2)  $\text{move}(r1, wp3, wp4)$  and  $\text{move}(r2, wp3, wp4)$ , and (3)  $\text{give}(r1, o1, p1)$  and  $\text{give}(r2, o1, p1)$ . A TG is constructed for each coordinated action and for each agent. Thus, the joint goal is decomposed to six TGs, three for each robot. (3) is represented as two TGs similar to the TGs in Example 37. (1) and (2) are represented in the same manner where the goal predicates are the effects of the actions. For (1), the TGs are  $(\text{holding}(r1, o1), \text{pick\_up}(r1, o1), T_1, T_2)$  and  $(\text{holding}(r2, o1), \text{pick\_up}(r2, o1), T_1, T_2)$  where  $T_1$  is the elapsed time which both robots start to execute the actions and  $T_2 - T_1$  is the duration allowed for the actions. A preceding TG must be achieved before the next TG can be attempted. If a preceding TG is violated, then the remaining TGs are violated as well.

The coordinated actions are strongly coordinated actions because each robot is required to execute a sequence of three coordinated actions. The coordinated actions have temporally annotated effects which are dependent on the other robot—both robots need to pick up and hold  $o1$  together; if one robot fails, then the other robot fails as well.

Learning to solve joint goals which require strongly coordinated actions present challenges that are beyond the scope of our work. In Example 38, the transitions of goal predicates  $\text{holding}(r1, o1)$  and  $\text{holding}(r2, o1)$  depend on both robots. Thus, the transition function is defined over the joint state-action space of the two robots. This problem cannot be represented by a TDORMDP which only models single-agent problems. In short, we only consider an extension of our work to address weakly coordinated actions which allow the decomposition of a multi-agent problem into multiple single-agent problems.

The full treatment of multi-agent problems is better addressed by work in multi-agent reinforcement learning (MARL) [197]. MARL algorithms suffer from poor scalability and have a high sample complexity as they learn in the joint state and action spaces which have sizes that scale exponentially with the number of agents. The joint state space considers the properties of every agent and its relation with the environment and objects in the environment. Each of the  $K$  agents has a state space

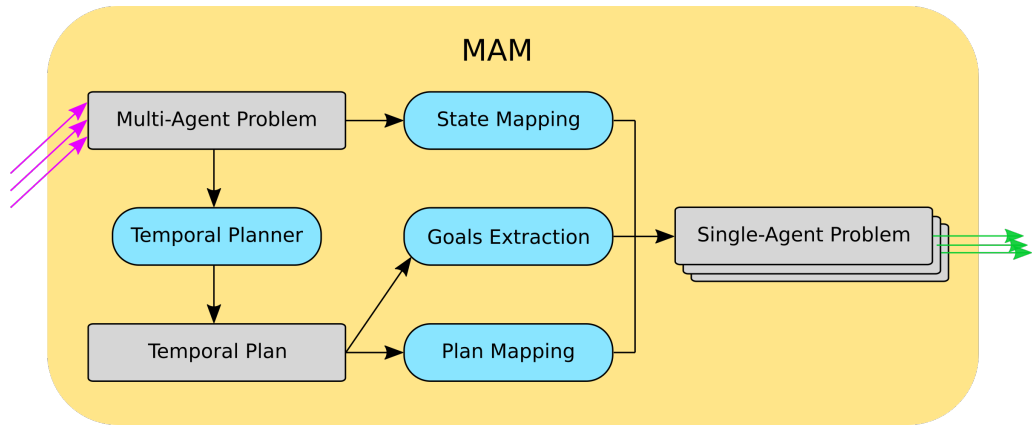


Figure 5.8: The MAM of a multi-agent decomposition framework which decomposes a multi-agent problem to a set of single-agent problems using information from a temporal planner. A purple arrow denotes feedback from a SAM and a green arrow denotes information sent to a SAM.

$\mathcal{S}_i$  and an action space  $\mathcal{A}_i$  where the subscript  $i$  denotes the  $i$ -th agent. The joint state space is  $\prod_{i=1}^K \mathcal{S}_i$  and the joint action space is  $\prod_{i=1}^K \mathcal{A}_i$ . Our work avoids the combinatorial exploration of the state-action space by considering the multi-agent problem as a set of single-agent problems. The limitation is that it only addresses weakly coordinated actions.

### 5.7.2 Decomposition with Temporal Planning

The first variant of our multi-agent decomposition framework uses a temporal planner such as [12, 28] to allocate TGs to agents (or SAMs). The MAM is illustrated in Figure 5.8. It uses a temporal planner to produce a temporal plan involving multiple agents to achieve goals and joint goals. Each action in a temporal plan is associated with a start time and an end time and concurrent actions are possible. Thus, a temporal planner can be used for multi-agent coordination. However, temporal planners do not typically consider uncertainty and probabilistic effects. To resolve this, our framework integrates temporal planning (in the MAM) and RL (in the SAMs). Both of them operate at different levels of abstraction for the problem. This decoupled approach allows us to use the most appropriate method to solve different aspects of the problem: temporal planning for coordination among multiple agents and RL for decision making under uncertainty. This work was published in [24]. In the paper, simulated experiments demonstrated the efficacy of this framework in coordinating some heterogeneous robots for the autonomous maintenance of an offshore energy

platform. Here, we present the main concepts and refer readers to the paper for details.

The framework performs four main steps:

1. It receives a multi-agent problem and uses a temporal planner to produce a temporal plan.
2. The multi-agent problem and temporal plan are mapped to single-agent problems and plans.
3. Each single-agent problem is solved independently.
4. The multi-agent problem is updated following feedback from SAMs. Go to Step 1.

Step 4 is only required if some TGs remain active after every SAM has stopped running. A SAM stops running if it has no active TGs (either achieved or violated), it has reached the end of the time horizon, or the elapsed time has exceeded the duration of the temporal plan. The multi-agent problem is expressed in PDDL2.1 and specifies the initial states for all agents and a set of goals and joint goals with no time bound. Next, we describe the four steps in details.

**Step 1: Temporal planning.** A temporal planner solves the multi-agent problem and produces a temporal plan such that the temporally annotated conditions of actions are satisfied over the duration they are executed and all goals and joint goals are achieved (if possible). The temporal planner requires the true model of the multi-agent problem which is written in PDDL2.1. We assume that the true model is known. Actions in the temporal plan which involve multiple agents are coordinated actions.

**Step 2: Mapping to single-agent problems.** As PDDL2.1 cannot describe probabilistic problems, the single-agent problem is modelled as a TDORMDP. The multi-agent problem and temporal plan are mapped to single-agent problems and plans. The transition functions for the single-agent problems are unknown. This step consists of three substeps which are illustrated in Figure 5.8.

**Step 2a: Goals extraction.** The temporal planner reasons with the temporal constraints of PDDL2.1 actions and produces a temporal plan to achieve goals and

joint goals. A goal requires one agent while a joint goal requires the coordination of multiple agents. The goals and joint goals with their respective actions are mapped to TGs which are allocated to the associated agents.

**Step 2b: Plan mapping.** The temporal plan is decomposed to a set of single-agent plans such that an agent’s plan consists of PDDL2.1 actions associated with the agent. Each PDDL2.1 action in the single-agent plan is then replaced with a sequence of durative actions (the durative actions are in  $\dot{\mathbf{A}}$  of a TDORMDP).

**Step 2c: State mapping.** The initial state for each agent is determined from the initial state of the multi-agent problem. This involves the mapping of literals between the two problems. The state space of a single-agent problem is reduced by including only **relevant objects** in  $\mathbf{O}$ . An object is relevant to an agent if it is associated with a literal that describes the agent’s initial state, is associated with a goal predicate that is allocated to the agent, or is in the term of an action in the agent’s single-agent plan. This reduces the state space but assumes that irrelevant objects are not required to achieve the agent’s TGs.<sup>5</sup>

**Step 3: Solve single-agent problem.** The initial state, set of TGs, and set of objects form a TDORMDP where  $\dot{\mathcal{T}}$  and  $\mathcal{D}$  are unknown; an approximate model of  $\dot{\mathcal{T}}$  is available. An agent’s SAM then executes its plan (from Step 2b) or a policy to achieve its allocated TGs. This is outlined in Algorithm 17 which shall be described later.

**Step 4: Update multi-agent problem and replan.** When a SAM terminates, it sends its current state and the set of active TGs and violated TGs to the MAM. The multi-agent problem is updated and a new temporal plan is produced. The cycle repeats by going back to Step 1. This continues until all goals are achieved. Note that the goals and joint goals considered here do not have time bounds; the

---

<sup>5</sup>This assumption is violated in certain situations. For example, an agent needs to enter a room to achieve a goal. Only one door is considered relevant. If this door is locked, the agent cannot achieve its goals as its policy will not consider another door. We can reduce the strictness of relevant objects by considering objects which are related to relevant objects (through some literals) as relevant objects as well.

time bounds are added by the MAM to coordinate agents. Thus, violated TGs can still be attempted by changing the time bounds when MAM replans.

**Assumptions.** We assume the following information are known: the mapping from PDDL2.1 actions to durative actions, the mapping from multi-agent joint state to single-agent states, and the multi-agent true model (i.e., PDDL2.1 domain). The temporal planner requires the true model to produce a temporal plan. While this is a significant assumption, in our work [24], the PDDL2.1 actions are abstract, deterministic actions which are easier to model than durative actions with probabilistic outcomes. Similar assumptions are also made in related work such as [61, 70, 91, 193].

### Reactive Plan Execution

The SAM uses Algorithm 17 to solve its single-agent problem. The inputs are similar to GK-RRL (Algorithm 11) with some exceptions; the problem  $P$  is now a TDORMDP instead of an RMDP and a single-agent plan ( $Plan$ ) and the maximum allowable elapsed time ( $T_{max}$ ) are additional inputs.  $T_{max}$  is the duration of the temporal plan from which  $Plan$  is mapped from. For brevity, we consider only a Q-function approximation rather than an ensemble though it is straightforward to extend the algorithm to learn an ensemble.

Algorithm 17 combines plan execution, learning, and plan adaptation. The Q-function approximation ( $\tilde{Q}$ ) can be transferred from a previous problem or initialised with Dyna (line 2). Dyna is outlined in Algorithm 16. The agent follows the plan  $Plan$  which specifies the action to select at each time step  $t$  (line 6). If an unexpected outcome occurs which renders the action in  $Plan$  to be inapplicable (line 7), the agent follows a policy  $\pi$  generated from  $\tilde{Q}$  (line 9). The plan will no longer be followed. The advantage of following a policy rather than a plan is the reduced computation time which makes it attractive for reactive plan adaptation in the face of uncertainty. When unexpected states are reached, a planner needs to replan. This can happen frequently if there are many agents and the environment is highly stochastic. Since the temporal planner coordinates multiple agents, a new temporal plan can affect the plans of multiple agents. For example, a robot moves to a location

**Algorithm 17:** Reactive plan execution**1 Function**


---

**Reactive\_Plan\_Execution**( $P, \mathcal{M}, \tilde{Q}, \Phi^c, \eta, CK, H_{sim}, N_{sim}, Plan, T_{max}$ ):

**Input:** Problem  $P = (\mathcal{C}, \mathcal{P}, \dot{\mathbf{A}}, \mathbf{O}, \dot{\mathbf{g}}, \dot{\mathbf{T}}, \dot{\mathbf{R}}, \mathcal{D}, \dot{\mathbf{s}}_0, H, \gamma)$ ,
Generative model  $\mathcal{M}$ ,Q-function approximation  $\tilde{Q}$ ,Candidate features  $\Phi^c$ ,Relevances of candidate features  $\eta$ ,Contextual knowledge  $CK$ ,Simulated horizon  $H_{sim}$ ,Number of rollouts  $N_{sim}$ ,Plan  $Plan$ ,Maximum allowable elapsed time  $T_{max}$ 2  $(\tilde{Q}, \Phi^c, \eta) \leftarrow \text{Dyna}(P, \mathcal{M}, \tilde{Q}, \Phi^c, \eta, CK, H_{sim}, N_{sim})$ 3  $T = 0, Follow = \top$ 4 **for**  $t = 0$  **to**  $H - 1$  **and**  $\dot{\mathbf{s}}_t$  *is not terminal state* **and**  $T < T_{max}$  **do**5     **if**  $Follow$  **then**6          $\dot{\mathbf{a}}_t \leftarrow \text{Follow\_Plan}(Plan)$ 7         **if**  $\dot{\mathbf{a}}_t$  *is not applicable in*  $\dot{\mathbf{s}}_t$  **then**8              $Follow = \perp$ 9         **if**  $Follow = \perp$  **then**  $\dot{\mathbf{a}}_t \leftarrow \pi(\dot{\mathbf{s}}_t)$ 10          $\dot{\mathbf{s}}_{t+1}, r_t, T_{dur} \leftarrow \text{Execute action } \dot{\mathbf{a}}_t$ 11          $T \leftarrow T + T_{dur}$ 12          $(\tilde{Q}, \Phi^c, \eta) \leftarrow \text{Update\_Approximation}(\tilde{Q}, \mathbf{A}, \mathbf{O}, \Phi^c, \eta, CK, \dot{\mathbf{s}}_t, \dot{\mathbf{a}}_t,$   
 $r_t, \dot{\mathbf{s}}_{t+1})$ 13          $\mathcal{M} \leftarrow \text{Learn\_Model}(\mathcal{M}, \dot{\mathbf{s}}_t, \dot{\mathbf{a}}_t, r_t, \dot{\mathbf{s}}_{t+1}, T_{dur})$ 14 **return**  $(\tilde{Q}, \Phi^c, \eta, \mathcal{M}, \dot{\mathbf{s}}_t)$ 


---

only to be instructed to move elsewhere, thereby achieving no meaningful progress in the entire process. Thus, replanning should be limited. Instead, the policy is used as a reactive plan adaptation to avoid affecting other agents.

The action is executed (line 10) and the resulting observation is used to update the elapsed time (line 11),  $\tilde{Q}$  (line 12, see Algorithm 4), and  $\mathcal{M}$  (line 13). The algorithm terminates (i.e., the SAM stops running) if there are no active TGs (either achieved or violated), it has reached the end of the time horizon, or the elapsed time has exceeded  $T_{max}$  (line 4). The last condition is necessary to avoid prolonged idling. For example, a robot is stuck and takes longer than expected to reach its destination. The execution of this action counts as one discrete time step but significant time has elapsed. Since the MAM replans only after all SAMs stop running, SAMs idle

---

**Algorithm 18:** Multi-agent coordination for weakly coordinated actions

---

```

1 Function Multi_Agent_Coordination(SAM):
2   Input: A set of SAMs, SAM
3    $\dot{\mathcal{G}}_{MAM} = \emptyset$ 
4   while at least one SAM  $\in$  SAM is running do
5     Run SAM concurrently
6     if a new joint goal  $\dot{g}_{joint}$  is received by a SAM then
7        $(\dot{g}_1, \dots, \dot{g}_N) \leftarrow \text{Decompose\_Joint\_Goal}(\dot{g}_{joint})$ 
8       Add  $(\dot{g}_1, \dots, \dot{g}_N)$  to  $\dot{\mathcal{G}}_{MAM}$ 
9       Remove violated/achieved sets of TGs from  $\dot{\mathcal{G}}_{MAM}$ 
10       $\dot{\mathcal{G}}_{MAM} \leftarrow \text{Time\_Bounded\_Goals\_Auction}(\mathbf{SAM}, \dot{\mathcal{G}}_{MAM})$ 

```

---

until the last SAM stopped running. Algorithm 17 returns the current state of the agent which is feed back to the MAM (line 14).

### 5.7.3 Auctioning TGs

There are some shortcomings of the framework presented in the previous section. It requires substantial expert knowledge to map between multi-agent problem and single-agent problems. Furthermore, the integration of the MAM and SAMs is not tightly coupled; TGs are only allocated at the start of an episode which ignores the possibility that TGs could be active at any time step. Therefore, we present a second variant of our multi-agent decomposition framework to address these shortcomings. One advantage of the first variant is that it solves a multi-agent problem written in PDDL2.1. This is a more general class of multi-agent problems than the one considered here which uses durative actions instead of PDDL2.1 actions. We present the concepts of the second variant, leaving empirical evaluations for future work.

In the second variant, Algorithm 18 is the MAM that coordinates the SAMs. The input is a set of SAMs (**SAM**), one for each agent.  $\dot{\mathcal{G}}_{MAM}$  is a set of TGs for the multi-agent problem (line 2). Algorithm 18 runs until all SAMs stopped running (line 3). Each SAM runs GK-RRL+ and stops running when GK-RRL+ terminates. All SAMs can run concurrently (line 4). At any time, a SAM can receive a joint goal (line 5) which is decomposed to a set of TGs, one for each agent required to achieve the joint goal (line 6). All TGs in this set differ only in their goal predicates and do not involve any action yet. This set of TGs is added to  $\dot{\mathcal{G}}_{MAM}$  (line 7). SAMs can

---

**Algorithm 19:** Sequential single-item auctions of TGs

---

```

1 Function Time_Bounded_Goals_Auction( $\mathbf{SAM}, \dot{\mathcal{G}}_{MAM}$ ):
   Input: A set of SAMs,  $\mathbf{SAM}$ ,
           Sets of TGs,  $\dot{\mathcal{G}}_{MAM}$ 
2 while  $\dot{\mathcal{G}}_{MAM} \neq \emptyset$  do
3    $Best = \emptyset$ 
4    $T_{min} = -\infty$ 
5   for  $(\dot{g}_1, \dots, \dot{g}_N) \in \dot{\mathcal{G}}_{MAM}$  do
6      $Bids \leftarrow \text{Auction}(\mathbf{SAM}, (\dot{g}_1, \dots, \dot{g}_N))$ 
7     for  $\mathbf{C} \in \text{Combination}(Bids)$  do
8        $T_{start} \leftarrow$  Get largest start time from bids in  $\mathbf{C}$ 
9        $T_{dur} \leftarrow$  Get largest action duration from bids in  $\mathbf{C}$ 
10      if  $T_{start} \geq T^+$  and  $(T_{start} + T_{dur}) \leq T^-$  and
11          $(T_{start} + T_{dur}) < T_{min}$  then
12          $Best = (\mathbf{C}, T_{start}, T_{dur})$ 
13          $T_{min} = T_{start} + T_{dur}$ 
13   if  $Best = \emptyset$  then
14     break
15   else
16      $(\mathbf{C}, T_{start}, T_{dur}) \leftarrow Best$ 
17     for  $Bid \in \mathbf{C}$  do
18        $(\mathbf{SAM}, \dot{g}, a, \cdot, \cdot) \leftarrow Bid$ 
19       Update  $\dot{g}$ : Set action to  $a$ ,  $T^+ = T_{start}$ ,  $T^- = T_{start} + T_{dur}$ 
20       Add  $\dot{g}$  to  $\mathbf{SAM}$ 's  $\dot{\mathcal{G}}$ 
21       Update  $\mathbf{SAM}$ 's simulated state
22       Remove  $(\dot{g}_1, \dots, \dot{g}_N)$  from  $\dot{\mathcal{G}}_{MAM}$ 
23 return  $\dot{\mathcal{G}}_{MAM}$ 

```

---

also receive TGs which are not joint goals. For simplicity, we use joint goals to refer to goals and joint goals in the remainder of this section. If any joint goal is violated or achieved, its set of TGs is removed from  $\dot{\mathcal{G}}_{MAM}$  (line 8). An auction algorithm allocates  $\dot{\mathcal{G}}_{MAM}$  to the SAMs (line 9). While the auction algorithm is running, the SAMs can continue as per normal (e.g., the agents continue executing their actions). As defined in Definition 34,  $\dot{\mathcal{G}}$  in a TDORMDP can change dynamically. Therefore, the methods presented in Section 5.4 are still applicable here.

The key component in Algorithm 18 is the auction algorithm which is outlined in Algorithm 19. It performs a sequential single-item auction which combines the advantages of single-round combinatorial auction and parallel single-item auctions [90], balancing between the intractable computational costs to compute bids in the



former and the poor optimality of allocations in the latter. The inputs to Algorithm 19 are a set of SAMs (**SAM**), one for each agent, and sets of TGs ( $\dot{\mathcal{G}}_{MAM}$ ), one set for each joint goal. Each set of TGs in  $\dot{\mathcal{G}}_{MAM}$  is auctioned sequentially (lines 2 to 22). SAMs bid for the TGs (line 6). A bid  $Bid \in \mathbf{Bids}$  is a tuple  $(SAM, \dot{g}, a, T_{start}, T_{dur})$  where  $SAM$  is the SAM which places the bid,  $\dot{g}$  is the TG it bids for,  $T_{start}$  is the elapsed time when the agent shall execute the action  $a$  to achieve  $\dot{g}$ , and  $T_{dur}$  is the action duration. A SAM can place at most one bid for each TG. To achieve a TG, the conditions  $T_{start} \geq T^+$  and  $T_{start} + T_{dur} \leq T^-$  must be satisfied. If this is not possible, then the SAM does not bid. We discuss the computation of a bid later. **Bids** are sets of bids  $\{\mathbf{bid}_1, \dots, \mathbf{bid}_N\}$  where  $\mathbf{bid}_i$  is a set of bids for  $\dot{g}_i$ . If one of the TGs has no bids (i.e.,  $\mathbf{bid}_i$  is the empty set), then the joint goal cannot be allocated; this is because either the TG cannot be achieved or the SAMs have not yet learned how to achieve the TG. Otherwise, all possible combinations of bids are considered (lines 7 to 12).

A combination of bids (**C**) consists of bids from  $N$  SAMs such that each SAM bids for exactly one TG in  $(\dot{g}_1, \dots, \dot{g}_N)$  and each TG has no more than one bid. The set of possible combinations of bids is given by the Cartesian product  $\mathbf{bid}_1 \times \dots \times \mathbf{bid}_N$  minus combinations with more than one bid from the same SAM (line 7). The start time for the coordination must be the largest of the start time from each bid (line 8). Similarly, the duration of the coordination must be the largest of the duration from each bid (line 9). The best combination is the combination which achieves the joint goal within its time bound and at the earliest time (lines 10 to 12).

If a best combination of bids is not found (line 13), then no joint goals can be achieved by the SAMs and the auction terminates (line 14). Otherwise (lines 15 to 22), the SAMs involved in the best combination of bids are allocated the TG it bids for (line 20). The TG is updated to take into consideration the action the SAM will execute and the time bound for coordination with the other agents (line 19). The current states of the SAMs are updated to **simulated states** which are the states the agents are predicted to be in after achieving their allocated TGs (line 21). This will affect their bids for the next TGs. In other words, an agent takes into account the TGs it has committed to when bidding for new TGs. The set of TGs which

are allocated are removed from  $\dot{\mathcal{G}}_{MAM}$  (line 22). Those which are not allocated are retained for future rounds of auctions (line 23). Perhaps by then, a SAM has learned how to achieve the TG?

We now discuss in detail how a SAM determines its bid for a TG,  $\dot{g}$ . The SAM finds a plan to achieve  $\dot{g}$  using GK-RRL+. The input TDORMDP problem  $P$  is augmented by adding  $\dot{g}$  to its set of TGs,  $\dot{\mathcal{G}}$ . GK-RRL+ uses the generative model to perform policy rollouts to find at least one trajectory which achieves  $\dot{g}$ . UCT is used for this purpose. A SAM does not bid if none of its sampled trajectories achieve  $\dot{g}$ . Otherwise, it bids using information from the trajectory that achieves  $\dot{g}$  at the earliest time. The last action  $a$  executed in this trajectory achieves  $\dot{g}$  since a rollout terminates when  $\dot{g}$  is achieved. The SAM submits a bid  $(SAM, \dot{g}, a, T_{start}, T_{dur})$  where  $T_{start}$  is the elapsed time when the action starts to execute and  $T_{dur}$  is the action duration. Each trial start from the current state of the SAM or from its simulated state if the SAM has previously committed to one or more TGs. That is, a SAM places priority on the TGs it has committed to and bids for the next TG under the premise that it will have achieve, or attempt to achieve, these TGs. Since GK-RRL+ learns from observations, the quality of the bids (e.g., a shorter time to achieve a TG) can improve over time.

**Assumptions.** We make the following assumptions in Algorithms 18 and 19:

1. The decomposition of a joint goal to a set of TGs is known.
2. TGs are not for strongly coordinated actions.
3. All TGs are equally important with the same immediate reward which is given to every robot which achieves it.
4. The end time of a new joint goal is later than the end time of existing joint goals.

If the set of joint goals is known beforehand and is finite, then the first assumption requires trivial expert knowledge. The necessity of the second assumption was discussed in Section 5.7.1. The third assumption allows us to sort the TGs with ascending end time of their time bounds (line 2 in Algorithm 19). If some joint goals have higher rewards than others, then it can be more optimal to sort them with descending rewards, descending number of agents required, or a weighted sum

of these factors. The fourth assumption is necessary because SAMs place bids based on rollouts from their simulated states. If new joint goals have end times earlier than previous joint goals, it is possible that no SAMs can bid for the TGs of new joint goals as they are violated in the SAMs' simulated states. One solution is to auction and reallocate previously allocated TGs.

**Example 39 (Multi-Agent Coordination in Service Robot)**

In a multi-agent variant of *Service Robot*, a robot  $r1$  talks to a person  $p1$  and receives a task at the elapsed time of 100 to bring the items  $o1$  and  $o2$  to  $p1$  at the same time by the elapsed time of 400. Since this task requires two robots, it is a joint goal  $\dot{g}_{joint} = (\text{object\_with}(o1, p1) \wedge \text{object\_with}(o2, p1), \emptyset, 100, 400)$ .  $\dot{g}_{joint}$  is decomposed to two TGs:  $\dot{g}_1 = (\text{object\_with}(o1, p1), \emptyset, 100, 400)$  and  $\dot{g}_2 = (\text{object\_with}(o2, p1), \emptyset, 100, 400)$  (line 6 in Algorithm 18).  $\dot{g}_1$  and  $\dot{g}_2$  have the same time bounds as  $\dot{g}_{joint}$  and do not yet involve any actions. The actions will be determined later. Given  $\dot{\mathcal{G}}_{MAM} = \{(\dot{g}_1, \dot{g}_2)\}$ , Algorithm 19 allocates  $\dot{g}_1$  and  $\dot{g}_2$  to two SAMs which can coordinate to achieve the joint goal (line 9 in Algorithm 18). The auction algorithm (Algorithm 19) considers the following bids from three robots,  $r1$ ,  $r2$ , and  $r3$ :

- $Bid_1 = (SAM_{r1}, \dot{g}_1, \text{give}(r1, o1, p1), 200, 50)$
- $Bid_2 = (SAM_{r1}, \dot{g}_2, \text{give}(r1, o2, p1), 300, 50)$
- $Bid_3 = (SAM_{r2}, \dot{g}_1, \text{give}(r2, o1, p1), 250, 50)$
- $Bid_4 = (SAM_{r3}, \dot{g}_2, \text{give}(r3, o2, p1), 300, 100)$

The combinations of bids are  $\mathbf{C}_1 = \{Bid_1, Bid_4\}$ ,  $\mathbf{C}_2 = \{Bid_2, Bid_3\}$ , and  $\mathbf{C}_3 = \{Bid_3, Bid_4\}$ . The largest start time and action duration for the combinations (lines 8 and 9 in Algorithm 19) are:

- $\mathbf{C}_1: T_{start} = 300, T_{dur} = 100$
- $\mathbf{C}_2: T_{start} = 300, T_{dur} = 50$
- $\mathbf{C}_3: T_{start} = 300, T_{dur} = 100$

$\mathbf{C}_2$  is the best bid as  $r1$  and  $r2$  can achieve the joint goal by the earliest time of  $T_{start} + T_{dur} = 350$ . The TGs are updated based on  $\mathbf{C}_2$ :

- $\dot{g}_1 = (\text{object\_with}(o1, p1), \text{give}(r2, o1, p1), 300, 350)$
- $\dot{g}_2 = (\text{object\_with}(o2, p1), \text{give}(r1, o2, p1), 300, 350)$

$\dot{g}_1$  is allocated to  $SAM_{r2}$  and  $\dot{g}_2$  to  $SAM_{r1}$  (line 18 in Algorithm 19). The time bound

is reduced to a smaller range from  $[100, 400]$  to  $[300, 350]$ . The former represents the inherent time bound of the joint goal while the latter represents the time bound determined to coordinate actions to achieve the joint goal.

**Further considerations.** We discuss some technicalities with Algorithms 18 and 19:

1. What if SAMs are learning from scratch?
2. What if a SAM learns to achieve a TG that failed to be allocated previously?
3. What if an agent is executing an action when the auction is called?
4. What if the agents are heterogeneous?

(1) If SAMs are learning from scratch, the agents do not initially know how to achieve any TG and the SAMs will not bid. Although the agent explores and learns from observations, without any TGs, there is no reward feedback from which the agent learns. This cycle continues until the agent receives a TG which is not a joint goal. This learning process is inefficient as potential learning opportunities are missed. Instead of deferring a joint goal to future rounds of auctions, TGs which have no bids are randomly allocated to SAMs (with the constraint that each SAM can have at most one TG of a joint goal allocated to it). Through exploration, a SAM might achieve the TG. This is highly unlikely since TGs have time bounds. Thus, hindsight learning (see Section 5.4.4) ensures that the agent learns even when the TG is violated.

(2) When a SAM learns to achieve a TG, it is possible that previously unallocated joint goals can now be allocated. The condition to call for auctioning (line 5 in Algorithm 18) can be modified to be called periodically at the expense of higher computational costs.

(3) Since agents execute actions asynchronously, it is possible that an auction is called while an agent is executing an action. If the agent has committed to a TG, its SAM computes bids based on its simulated state. Otherwise, it computes bids based on the predicted state at the end of the execution.

(4) Agents can be heterogeneous; they can have different sets of actions. For example, a robot which does not have a manipulator arm will not be capable of picking up or putting down items. Our framework handles heterogeneous agents in

the same manner as homogeneous agents. Following the same example, if an agent cannot pick up or put down items, then its SAM will not bid for TGs which require such actions. This is because none of its sampled trajectories during policy rollout can achieve the TGs.

**Theorem 7 (Time complexity of Algorithm 19)**

*The time complexity of Algorithm 19 is  $O(|\dot{\mathcal{G}}_{MAM}|^2 K^2 N_{trial} H_{rollout})$  where  $K$  is the number of agents.*

*Proof.* In each iteration, Algorithm 19 either allocates a joint goal in  $\dot{\mathcal{G}}_{MAM}$  to SAMs or terminates. The number of remaining joint goals decreases by one at the end of an iteration. A joint goal can be decomposed to at most  $K$  TGs since there are only  $K$  agents. For each joint goal, there are two steps involved: (1) the bid from each SAM, and (2) the determination of the best combination of bids (lines 5 to 12 in Algorithm 19). First, the cost of a SAM computing a bid is  $N_{trial} H_{rollout}$  which is the time complexity for simulating  $N_{trial}$  trials (i.e., UCT terminates after  $N_{trial}$  trials) up to a length of  $H_{rollout}$ . If every SAM bids for each TG, then there are  $K^2$  bids and the time complexity is  $O(K^2 N_{trial} H_{rollout})$ . Second, the best combination of bids is determined. The combinations of bids are the Cartesian product of the sets of bids minus combinations where more than one bid comes from the same SAM. The number of valid combinations is  $K(K - 1)$ . Since the number of joint goals decrease by one in each iteration, we have  $O((K^2 N_{trial} H_{rollout} + K(K - 1))(|\dot{\mathcal{G}}_{MAM}| + (|\dot{\mathcal{G}}_{MAM}| - 1) + \dots + 1)) = O(|\dot{\mathcal{G}}_{MAM}|^2 K^2 N_{trial} H_{rollout})$ .  $\square$

The time complexity of Algorithm 19 scales exponentially with the number of agents which is expected due to the combinatorial explosion when more agents are involved.

**Theorem 8 (Completeness of Algorithm 19)**

*If every joint goal, each decomposed to a set of TGs in  $\dot{\mathcal{G}}_{MAM}$ , can be allocated to the agents, Algorithm 19 is not guaranteed to find this allocation and can fail to allocate one or more joint goals. In other words, Algorithm 19 is incomplete.*

*Proof.* Algorithm 19 is incomplete because it uses sequential single-item auctions and allocates one joint goal at a time to the agent which can achieve it at the earliest time. To prove that it is incomplete, we show an example where it fails to

allocate every joint goal when it is possible. Suppose that there are two joint goals,  $\dot{g}_{joint1}$  and  $\dot{g}_{joint2}$ , and four agents,  $a1$ ,  $a2$ ,  $a3$ , and  $a4$ .  $a1$  and  $a2$  can coordinate to achieve either  $\dot{g}_{joint1}$  or  $\dot{g}_{joint2}$  but not both, and  $a3$  and  $a4$  can coordinate to achieve  $\dot{g}_{joint1}$ . Thus,  $\dot{g}_{joint1}$  should be allocated to  $a3$  and  $a4$  and  $\dot{g}_{joint2}$  to  $a1$  and  $a2$ .

If the combination of bids from  $a1$  and  $a2$  for  $\dot{g}_{joint1}$  has the earliest end time than the other combinations of bids, Algorithm 19 allocates  $\dot{g}_{joint1}$  to  $a1$  and  $a2$ . In the next round of auction, Algorithm 19 auctions  $\dot{g}_{joint2}$  but receives no bids because  $a1$  and  $a2$  have been allocated  $\dot{g}_{joint1}$  and can no longer achieve  $\dot{g}_{joint2}$  while  $a3$  and  $a4$  cannot achieve  $\dot{g}_{joint2}$ . Therefore,  $\dot{g}_{joint2}$  is not allocated. This example shows that Algorithm 19 is incomplete.  $\square$

Theorem 8 is corroborated by [90, 125] which state that sequential single-item auctions do not guarantee optimality and are incomplete.

## 5.8 Summary

In this chapter, we considered some aspects of a problem which are inherent in the real world. First, not all objects in the environment are known to the agent. Or, the agent might want to ignore objects until they are relevant to solve the task at hand if there are many objects. We use DORMDP [109] to model problems with dynamic objects. Second, the real world is temporospatial. We proposed TDORMDP, an extension of DORMDP to include temporal constructs. Actions are durative, states are augmented with the elapsed time, and the transition function and reward function can be time-dependent. Furthermore, goals can have time bounds within which they are to be achieved.

The two classes of problems, DORMDP and TDORMDP, are richer and more complex than RMDPs. This has certain implications for GK-RRL. Potential solutions were deliberated and implemented—GK-RRL+ is an extension of GK-RRL to solve DORMDP and TDORMDP classes of problems.

TGs provide a natural extension of GK-RRL+ to solve multi-agent coordination where the TGs are coordinated actions which must be executed at a specified time instance to achieve a joint goal. We presented a multi-agent decomposition framework which decomposes a multi-agent problem into single-agent problems. While

this is much more tractable than learning and planning in the joint multi-agent state-action space, our framework is limited to weakly coordinated actions. We proposed two variants of this framework. The first variant uses a temporal planner to plan for and coordinate multiple agents. The second variant uses an auction algorithm. Both methods have their pros and cons; the choice of which variant to use depends on the nature of the multi-agent problem and what prior information is available.

In our experiments, we investigated transfer learning between different classes of problems and simulated environments. In previous chapters, experiments use a simplistic simulated environment. Here, we built and use a simulated environment in Gazebo which is significantly more realistic and complex. Empirical results showed that **GK-RRL+** is able to adapt to and generalise over different classes of problems. We also investigated the possibility of Sim-to-Sim and zero-shot transfer where the source problems use RDDLSim’s simulated environment and the target problems use Gazebo’s simulated environment. Empirical results showed that **GK-RRL+** is able to achieve better performance than learning from scratch. However, we note that although the two simulated environments are drastically different, from the perspective of **GK-RRL+** which learns and makes decisions at the task level, some of the discrepancies are not relevant for decision making.

# Chapter 6

## Discussion

In this chapter, we discuss the broader impact of our work, some reflections on the decisions we made, things we have tried and did not work, and some views on the future of RRL.

**Past and present research trends.** The first RRL method is proposed by Džeroski, De Raedt, and Driessens [47] which proved to be highly influential as much work on RRL was published in the following years such as [30, 44, 99, 141, 184, 192], to name a few. In more recent years, research interest is growing in the field of deep RL where the majority of the work rely on the generalisation of convolutional neural networks. However, this form of generalisation does not consider relations between objects; thus, it does not address relational problems. Recent work has proposed promising solutions which combine the strengths of deep RL and relational learning [53, 177, 196]. Due to its high representational capacity, deep neural networks can approximate highly complex value functions while the generalisation due to relational learning reduces the high sample complexity typically associated with deep neural networks. However, this work requires the true model to induce domain-specific architectural bias in the neural networks.

**Our research direction.** In this dissertation, we have been interested in the learning problem, where the true model is not known, rather than the planning problem. This dissertation looks back at the more traditional RRL approaches instead of deep RL, and explores the potentials of RRL that remain unanswered. We combined some existing methods (e.g., Double Q-learning [68] and online feature



discovery [54]) with RRL that has never been attempted before. We also investigated the limitations of RRL methods. The significance of our finding shows that there are types of relational problems which RRL cannot solve. We proposed two solutions, and further augment our RRL method, LQ-RRL, by learning generalised knowledge, or knowledge that can be transferred to any problem of the same domain. This extended variant is denoted by GK-RRL. In essence, our work moves beyond the usual objective of RRL methods, which is approximating the value function or Q-function relationally, and entails learning non-stationary intrinsic rewards and dead end situations. In addition, we considered both model-free and model-based approaches and investigated empirically if models learned from observations can be useful despite model errors. The common theme behind our work is to explore the types of generalised knowledge which can be learned and how they can be exploited to reduce the sample complexity. This is an important step towards real world applications of RRL methods where dynamically changing environments and expensive data collection necessitate efficient learning and generalisation to new situations.

**Domains and problems.** A common benchmark in RRL is the **Blocks World** domain which we find to be trivially simple: actions are deterministic, there is a clear relational structure, and only one goal is considered. More challenging benchmark domains for RRL are required to test the limitations of RRL methods. The different inputs required by different RRL methods compound the difficulty in making any comparison. We evaluated LQ-RRL against two other RRL methods on **Blocks World**; however, due to its simplicity, many of our methods are not required nor used. In view of this, we used three benchmark domains from the field of automated planning and introduced three new domains in this dissertation. **Robot Fetch** was designed to demonstrate that RRL methods cannot deal with interdependent goals. **Robot Inspection** was designed to have many dead ends (unlike **Triangle Tireworld**) and to allow multiple goals to be achieved at the same time. We tried to assign negative rewards for reaching dead ends. However, this leads to poor performance in our experiments if the agent never achieves a goal; the resulting policy will avoid moving and simply choose to dock and undock till the end of the

time horizon. Thus, formulating the reward function is a critical design choice. We designed **Service Robot** with the intention of demonstrating it in Gazebo’s simulated environment. We introduced dynamic objects and TGs which we felt are interesting to consider in real world applications and have not been explored before in the context of RRL.

One type of problems which our RRL method (and possibly RRL in general) faced difficulty in solving is problems which require agents to move in a grid environment to more than one specific location to achieve a goal. One example is **Robot Inspection** with a grid environment instead of allowing the robot to move directly between any two locations. This difficulty is due to plateaus. While we have demonstrated that plateaus can be overcome with MQTE or an ensemble of ground and first-order approximations, both methods have their shortcomings. MQTE requires a model while the ground approximation in the ensemble cannot be generalised or transferred. Hierarchical RRL is one direction to look at to solve such problems and is complementary to our work. Nevertheless, in the context of robotic applications, navigation between two locations is determined by a motion planner; depending on the choice of a motion planner, it will discretise the environment to grids. Thus, decision making at the high-level typically do not need to consider grid environments. Because of this, in our three new domains, robots can move directly between any two locations.

**Understanding the learning process.** In our empirical results, we discovered the limitations of RRL in some types of relational problems. This is by no means a trivial process. We pinpointed specific states or stages of learning, examined the Q-values, grounding of first-order features, active and inactive features, and the actions executed. In particular, we ask ourselves the following questions. What are the necessary features? Are the necessary features not learned due to missing base features, inadequate observations, or poorly tuned hyperparameters? Can the problem be solved by RRL methods? Obtaining answers to these questions is not straightforward because a myriad of closely related factors are involved, especially in online RRL, making it difficult to distinguish which are the contributing factors and which are the consequences. To ease the analysis and facilitate algorithmic

design choices (e.g., to use an ensemble of approximations, MQTE, combinations of contextual knowledge, etc.), we can turn to the field of explainable reinforcement learning (XRL) [138] to aid us by answering questions such as:

1. Why did the policy select this action?
2. Why is there a plateau here?
3. Why is this feature active?
4. Why is this feature necessary or unnecessary to learn or represent the policy?
5. Why are the policies generated from different approximations suggesting different actions in this state?

XRL can aid in the formulation of domains and problems such as the introduction of symbolic state predicates which might be superfluous for representing states but are necessary for RRL to approximate the value function or Q-function relationally. This is required due to the limited representational capacity of RRL—only a limited number of objects are considered to compute the Q-values. For example, in our work, first-order features represent relations involving at most two objects of the same type using bound and free variables. In *Academic Advising*, this limited representational capacity can be overcome by introducing a symbolic state predicate which represents the recursive relation of the prerequisites of courses.

**Identifying the shortcoming of RRL.** The idea for contextual knowledge comes about when we encountered long computation times to run the experiments due to the large number of groundings for free variables, particularly in *Recon* which has many locations. We noticed that the existential grounding of free variables is usually uninformative as the resulting ground features are often true. For example, in *Recon*, “is there a location where there is an object?” is always true. With location context, this query changes to “is there a location adjacent to my destination where there is an object?” which is significantly more informative. Similarly, we observed that the policy often selects sequences of actions which attempts to achieve a goal which has already been achieved. Thus, we introduced goal context to remove such objects (or goals) from contextual grounding, noting the limitations in goal-conditioned Q-functions for symbolic problems. Contextual grounding has a significant impact on online RRL. It affects the Q-values, policy, observations

acquired, and online feature discovery. While we do not have theoretical proofs or properties for contextual knowledge applied to RRL, our empirical results for six domains showed that it can reduce sample complexity and computational cost.

The choice of contextual knowledge or the order in which combinations of contextual knowledge are applied might require domain knowledge or an ablation study. Fortunately, the order only matters when there is a conflict between two types of contextual knowledge. For transfer learning, the contextual knowledge must be the same in the source and target problems. Intuitively, since the first-order approximation is an abstraction of the state which now involves contextual knowledge, a different contextual knowledge gives a different state abstraction. Since the Q-function is learned in a different abstract state space, transfer learning will fail. It will be interesting to study the impact of state abstraction due to contextual knowledge in a theoretically grounded premise.

**Behind the scenes of hyperparameter tuning.** There is a multitude of hyperparameters in our work as is often the case with RL. Considering all combinations of hyperparameters is prohibitive. Thus, we used common values for hyperparameters of lower concern or interest (e.g.,  $\alpha$  and  $\epsilon$ ) and performed sensitivity analysis to tune some hyperparameters. This was not done for the time horizon, the number of episodes, and the hyperparameters for UCT. We shall briefly discuss them here. For each problem, the time horizon is set such that an (near-)optimal policy can be learned within 3000 episodes (or 500 episodes where UCT is used). Next, since some hyperparameters (e.g.,  $\alpha$  and  $\epsilon$ ) are decayed over episodes, the number of episodes can be considered as a hyperparameter. Thus, for simplicity, we used the same number of episodes for the small and large scale problems. Lastly, we implemented four ways in which a Q-function can be combined with UCT. Some of these significantly increase the computation time due to the computationally expensive operation of grounding first-order features. Since UCT is an anytime algorithm which terminates after timeout, a balance has to be made between more informative rollouts and more trials. We varied the length of the rollout ( $H_{rollout}$ ), the rollout policy ( $\epsilon$ -greedy, greedy, random, and a mixture of them), and the mixing parameters ( $\Lambda_{leaf}$ ,  $\Lambda_{root}$ , and even adaptive variants), noting the impact they had on the number of trials UCT

performed. We used the best combinations of hyperparameters from this tuning.

**What did not work.** We have tried many things which did not work. We discuss three of them here. First, we had hoped for synergy between **MBFS** and ground context where **MBFS** eliminates base features which in turn reduces the number of substitutions due to ground context. This combination induces a model-based contextual knowledge which is rather different from goal context and location context. Although we have seen that the combination of the two does indeed produce ground context as expected in **Academic Advising**, the performance did not improve. We later learned that the state abstraction which can represent a policy is not necessarily sufficient to learn the policy. We hope that this approach can be further developed under the framework of options which might resolve this issue.

Second, we implemented a counting operator for free variables: a first-order feature takes the value of the number of its groundings which is true. The resulting approximator has a higher representational capacity than one which uses binary features and can consider more than two objects of the same type. Unfortunately, our results showed that this does not give a significantly better performance than binary features. If the domain does not require them, it can increase the sample complexity as the first-order approximation now considers the number of objects for free variables as opposed to binary values. This also affects transfer learning when transferring between problems with different number of objects. Furthermore, since features are non-binary, this affects eligibility traces (replacing eligibility traces is only applicable for binary features), feature discovery (**iFDD+** only works for binary features), and TD learning. All of these complications are compounded when first-order features have multiple free variables. In [192], the number of free variables per first-order feature is limited to one. Further investigation is warranted to understand the impact of the counting operator in online RRL ([192] performs supervised learning).

Third, we implemented a method which selects a subset of the groundings of the first-order features such that a metric is maximised. This replaces the existential query of free variables and can be used with or without contextual grounding. We tried three metrics: Q-value, number of active features, and number of active com-

plex features. Empirical results showed that the performance is not robust across the six domains considered, and the best overall performance is obtained when no metric is maximised. However, we believe that reducing the number of groundings is crucial in reducing the granularity of the first-order approximation or any relational value function or Q-function approximation (e.g., relational decision trees). While this approach did not performed well, we hope it inspires other possible methods or maximisation metrics to reduce the granularity of relational function approximations where required.

**View on different approximators.** First-order approximation and relational decision trees have their respective pros and cons in approximating the Q-function relationally. While we have argued that decision trees are ill-suited to the incremental nature of online RL, decision trees could be ideal function approximators for some domains. Driessens and Džeroski [42] combine TG and RIB by using RIB in the leaf nodes of TG. Following the same concept, we can combine first-order approximation and relational decision trees as one approximator to obtain their respective strengths. Alternatively, we can use an ensemble of different function approximators.

**Vision of our work.** Our vision of RRL is to enable lifelong learning for robots where they learn continuously over long horizons to achieve multiple types of goals. Perhaps robots have to learn skills or options and build upon them to perform even more complex skills to achieve increasing complicated tasks. Perhaps multiple heterogeneous robots have to coordinate to achieve a joint task. In lifelong learning, different problems are attempted sequentially; this can be seen as a long-horizon or infinite horizon problem segmented into a sequence of smaller problems. We obtained promising results for Sim-to-Sim and zero-shot transfer with GK-RRL+, taking the first step towards lifelong learning. Potentially, RRL agents can be trained in simulated environments, thereby avoiding expensive data collection in the real world. On this note, we would have liked to test our work in the real world and demonstrate Sim-to-Real transfer. Unfortunately, in current times of a pandemic, this is not possible.

# Chapter 7

## Conclusions and Future Work

This dissertation makes several contributions which are theoretical, algorithmic, and empirical. We summarise them here.

In Chapter 2, we introduced three new domains for robotic applications following observations that prior RRL methods are tested in only a few types of domains which are rather simple. Our domains provide different types of challenges for RRL: interdependent goals, dead ends, and unpredictable state transitions.

In Chapter 3, we presented LQ-RRL, an online RRL method which learns a first-order approximation of the Q-function. We introduced the concepts of consistent abstraction, subsumption of problems, and abstract-equivalent problems. We proved that the first-order approximation performs consistent abstraction of the state-action space for abstract-equivalent problems. This allows transfer learning by directly transferring the first-order approximation. For non-abstract-equivalent problems, the set of features is augmented such that transfer learning is still possible if the target problem subsumes the source problem. We identified the limitations of RRL due to the relational abstraction of the state-action space and proposed three solutions: contextual knowledge, MQTE, and an ensemble of ground and first-order approximations. This extends the application of LQ-RRL to domains which RRL methods perform poorly in.

In Chapter 4, we presented GK-RRL which is an extension of LQ-RRL to utilise and learn different types of generalised knowledge. This generalised knowledge can be transferred to accelerate learning in another problem. First, a relational model can be learned from state transitions by any existing, appropriate model learner.

The model is used by UCT to provide better quality estimates for the Q-values. To compensate for model errors, we mix the Q-values estimated by UCT and Q-function approximations. Also, MBFS determines unnecessary base features from the structure of the (learned) transition function. Second, LDE learns from dead ends and influences the policy to avoid previously encountered dead ends. The efficacy of LDE is improved by generalising observations with a first-order representation and detecting dead end traps. Third, GK-RRL uses an ensemble of Q-function approximations to learn from extrinsic and multiple intrinsic rewards.

In Chapter 5, we introduced a new class of problems, TDORMDP, which models continuous time, action durations, TGs, and dynamic objects, to address some complexities inherent in real world applications. To solve TDORMDP problems, GK-RRL is extended to GK-RRL+ which handles dynamic objects, TGs, and the prediction of the transition of time. We can also use TGs to model weakly-coordinated actions between multiple agents; then, solving TDORMDP problems is akin to coordinating multiple agents to achieve a joint goal. To this end, we proposed two variants of a multi-agent decomposition framework which decomposes a multi-agent problem into single-agent problems.

The performance of our methods is demonstrated empirically in six domains. Table 7.1 summarises empirical results from Chapters 3 and 4. Evidently, our methods outperform the baseline most of the time. This is a significant progress from many prior RRL approaches which are typically tested on a small number of simplistic domains. We compared LQ-RRL against two prior RRL methods; empirical results showed that LQ-RRL performs comparably with RIB (which uses domain-specific knowledge) and outperforms TG. Three types of generalised knowledge, relational models, first-order intrinsic approximations, and first-order dead end situations, are utilised by GK-RRL to reduce the sample complexity when learning from scratch or in transfer learning. In particular, we have demonstrated the efficacy of combining different types of generalised knowledge. Lastly, we tested GK-RRL+ on Sim-to-Sim and zero-shot transfer where the source and target problems are different classes of problems and use different simulated environments. Results are promising and demonstrated that GK-RRL+ accelerates learning in the target problems despite significant domain shift.



Method	Desc.	Domain					
		AA	RC	TT	RF	RI	SR
$\tau$ -iFDD+	1	✓✓	—✓	— —	—✓	— —	—✓
Contextual knowledge	2	✓	✓	✓	✓	✓	✓
Ensemble	3	✓	✓	✓	✓		
MQTE	4	✓	✓	✓	✓		
MBFS	5	×✓	✓✓	✓✓	✓✓	✓✓	✓✓
LDE	6			✓		✓	
<i>Greedy-<math>\epsilon</math>-greedy</i> policy	7	✓	✓	—	✓	✓	✓
Intrinsic rewards	8	✓✓	✓✓	✓✓	✓✓	✓✓	✓✓
Transfer $\tilde{Q}^{fo}$	9		✓		✓	✓	✓
$\tilde{Q}^{gnd}$ + transfer $\tilde{Q}^{fo}$	10		✓		✓	×	×
Transfer $\tilde{Q}^{fo}$ and $\tilde{Q}_{int}^{fo}$	11		✓		✓	—	✓
$\tilde{Q}^{gnd}$ + transfer $\tilde{Q}^{fo}$ and $\tilde{Q}_{int}^{fo}$	12		✓		×	✓	×
RRL + Dyna	13		×		×	×	×
RRL + UCT	14		✓		✓	✓	×

Table 7.1: Summary of the performance of the methods introduced in this dissertation. ✓ indicates that the method improves performance relative to the baseline (refer to description [Desc.]), × indicates that it did not improve performance, and — indicates that it makes no difference. Empty cells indicate that the method is not applicable for the domain. If there are two symbols in a cell, the first pertains to the ground approximation and the second pertains to the first-order approximation. AA denotes Academic Advising, RC denotes Recon, TT denotes Triangle Tireworld, RF denotes Robot Fetch, RI denotes Robot Inspection, and SR denotes Service Robot.

<sup>1</sup>✓ denotes  $\tau$ -iFDD+ is more robust than iFDD+. See Section C.1.3 for results.

<sup>2</sup>✓ denotes the best contextual knowledge outperforms first-order approximation without contextual knowledge. See Section 3.7.2 for results and Table 3.2 for a summary of results.

<sup>3</sup>✓ denotes it outperforms the baseline which doesn't use any method to resolve plateaus. See Section 3.7.3 for results and Table 3.4 for a summary of results.

<sup>4</sup>✓ denotes it outperforms the baseline which doesn't use any method to resolve plateaus. See Section 3.7.3 for results and Table 3.4 for a summary of results.

<sup>5</sup>✓ denotes it performs at least comparably with MBFS but with fewer features. See Section 4.5.2 for results.

<sup>6</sup>✓ denotes LDE and all of its variants improve performance. See Section 4.5.3 for results.

<sup>7</sup>✓ denotes it has the best performance. See Section 4.5.4 for results.

<sup>8</sup>✓ denotes at least one type of intrinsic rewards we considered improves performance. See Section 4.5.4 for results and Table 4.2 for a summary of results.

<sup>9</sup>✓ denotes it can solve the problem. See Section 4.5.5 for results.

<sup>10</sup>✓ denotes it outperforms  $\tilde{Q}^{fo}$  which is transferred. See Section 4.5.5 for results.

<sup>11</sup>✓ denotes it outperforms  $\tilde{Q}^{fo}$  which is transferred. See Section 4.5.5 for results.

<sup>12</sup>✓ denotes it outperforms an ensemble of  $\tilde{Q}^{gnd}$  and  $\tilde{Q}^{fo}$  where  $\tilde{Q}^{fo}$  is transferred. See Section 4.5.5 for results.

<sup>13</sup>✓ denotes it outperforms the lower baseline, LQ-RRL. See Section 4.5.6 for results.

<sup>14</sup>✓ denotes it outperforms the lower baseline, LQ-RRL. See Section 4.5.6 for results.

## 7.1 Future Work

There are a number of open questions which we have not addressed in this dissertation. We discuss some of them here which future research can explore.

The grounding of first-order features is crucial for the performance of a first-order approximation. There remains possibilities for other types of contextual knowledge. For example, subgoals or landmarks [137] can be identified and used in the same manner as goal context. This could perform well in domains with subgoals such as *Academic Advising*. In domains where goals are interdependent or where there is an optimal sequence to achieve the goals (e.g, spatially-distributed goals in a grid environment), an order to achieve the goals can be learned. Instead of considering all active goals, goal context considers only one active goal at a time following this order. Lastly, domain-specific contextual knowledge can be defined with some background knowledge.

Richer forms of first-order features can be considered for RRL. Counting features (count the number of groundings which satisfy the feature in a state) has been used in [192] but for supervised learning. Other forms of features include ternary features and continuous features. A ternary feature has three possible values: true, false, or unknown. A ternary first-order feature evaluates to unknown if it has no grounding. One cause is when dynamic objects are not yet added to the set of objects. Continuous features are necessary to represent the elapsed time or real-valued sensor measurements. While these types of features are not new areas of research, under the context of online RRL, there remains unanswered questions to address their impact on generalisation. Also, what are the additional classes of problems that RRL methods can solve given this increased representational capacity?

In domains where RRL methods do not perform well, an ensemble of ground and first-order approximations is one possible solution. Instead of the average aggregation, the weights for each Q-function approximation in the ensemble can be learned and adapted such that the weight for an approximation which does not perform well is reduced. Alternatively, the policy is automatically generated from different approximations at different phases, perhaps taking into consideration their learning progress. For example, a first-order approximation might be preferred initially due to its generalisation while a ground approximation is preferred later due to its finer

granularity in approximating the Q-function.

Options could extend our work to address more complex domains where problems have hierarchies of subproblems. While relational options have been introduced in [30, 95], they are based on relational representations different from our first-order features. Since options function as partial policies for regions of a state space [169], this provides opportunities for varying levels of abstraction across options. **MBFS** can be used to determine and eliminate unnecessary base features for not just actions but also options. This could increase the efficiency of option learning. Another possibility is to introduce options to our framework for multi-agent coordination: a TG is associated with an option instead of an action. However, only the time bound of the option will be coordinated rather than the time bound of every action in the option.

The rather high computational cost of grounding a first-order approximation diminishes the efficacy of using the policy generated from it as the rollout policy in **UCT**. One solution is to map a first-order approximation to a ground approximation by enumerating all possible groundings of the first-order approximation. If the preciseness of Q-values is not important (because small changes to the Q-values might not induce any change to the policy), this can be done only at the start of each episode rather than in every time step.

Learning relational models which are close to the true models remain a challenge. Though model learning is beyond the scope of our work, the quality of the learned models has a direct impact on the performance of any model-based methods including ours. To mitigate the issue of poor approximate models, we augment the model prediction with the prediction from the maximum likelihood model. This is of a limited impact as the maximum likelihood model does not make predictions beyond what has been observed. A similar direction to take is to learn an ensemble of relational models and use the most common prediction. This was done in [69]; however, the models are not relational.

We identified three areas which warrant further empirical studies. First, our empirical results for combinations of intrinsic rewards do not show any dominant combination but it is possible that there are other domains which benefit from such combinations where different types of intrinsic rewards, not restricted to those pre-

sented in this dissertation, interact to provide effective, guided exploration over the entire state space. For example, a particular problem can have states in which a state novelty type of intrinsic reward works well and other states in which a salient type of intrinsic reward works better. Second, we did not evaluate the empirical sample complexity of **GK-RRL+** for Sim-to-Sim transfer due to resource limitations; running experiments in Gazebo is computationally expensive. The amount of observations required for **GK-RRL+** to adapt to new simulated environments provides insights on its potential performance in Sim-to-Real transfer. Lastly, demonstrating Sim-to-Real transfer is the next step forward following Sim-to-Sim transfer. Our **Service Robot** domain can be used for this purpose. It is straightforward to install ROS and the required ROS packages (including our ROS packages for the high-level actions in **Service Robot**) on a TIAGo robot.

RRL has been shown to enable efficient learning in relational problems. We investigated the limitations of RRL, explored several ways to incorporate it with existing and novel techniques, and discussed new classes of problems which it can perform well in. Moving forward, our hope is that this dissertation spurs interest in the field of RRL where many possibilities remain.

# Appendix A

## Proofs

### A.1 Proofs from Chapter 3

#### Theorem 3 (Consistent Abstract State Space)

$\bar{\mathcal{S}} = \wp(\hat{\mathcal{P}})$  is a consistent abstract state space for abstract-equivalent problems.

*Proof.* Since a state is a conjunction of  $|\mathbf{P}|$  literals in  $\mathbf{P}^\#$ ,  $\wp(\hat{\mathcal{P}})$  is a consistent abstraction of  $\mathcal{S}$  if  $\hat{\mathcal{P}}$  is a consistent abstraction of  $\mathbf{P}^\#$ . From Equations 3.3 and 3.4, we have:

$$\begin{aligned}\hat{\mathcal{P}}_{\hat{a}} &= \bigcup_{a \in \mathbf{A}_{\hat{a}}} \bigcup_{p \in \mathbf{P}^\#} \mathcal{L}(p, a) \\ &= \bigcup_{p \in \mathbf{P}^\#} \bigcup_{a \in \mathbf{A}_{\hat{a}}} \mathcal{L}(p, a) \\ &= \bigcup_{p \in \mathbf{P}^\#} \mathcal{P}(p, \hat{a}).\end{aligned}$$

There are  $2^n$  possible first-order base features that  $\mathcal{P}$  can map a literal  $p = p(o_1, \dots, o_n)$  to because each object is either lifted to a bound variable or to a free variable. Without loss of generality, consider the object  $o_1$  which has a type  $\mathcal{C}$ . There are three mutually exclusive scenarios which determine whether  $o_1$  is lifted to a bound variable or a free variable. (1) If  $\hat{a}$  does not have a term with the type  $\mathcal{C}$ , then  $o_1$  will not be in the terms of any actions in  $\mathbf{A}_{\hat{a}}$  and  $o_1$  will be lifted to a free variable. (2) If  $\hat{a}$  has a term with the type  $\mathcal{C}$  and there is only one object of

type  $\mathcal{C}$  in  $\mathbf{O}$ , then  $o_1$  is lifted to a bound variable. (3) If  $\hat{a}$  has a term with the type  $\mathcal{C}$  and there are at least two object of type  $\mathcal{C}$  in  $\mathbf{O}$ ,  $o_1$  is lifted to bound and free variables (i.e.,  $\mathcal{P}(p, \hat{a}) = \{\mathbf{p}(\mathcal{C}, \dots), \mathbf{p}(\star\mathcal{C}, \dots), \dots\}$ ) because there exists an action in  $\mathbf{A}_{\hat{a}}$  where  $o_1$  is in its terms ( $o_1$  is lifted to a bound variable) and another action where it is not ( $o_1$  is lifted to a free variable).

$\mathcal{P}$  is a consistent abstraction function for  $p$  if  $\mathcal{P}$  maps  $p$  to the same set of first-order base features for every problem  $P \in \mathbf{P}_D$ . In other words, the same scenario applies for each object in  $p$  regardless of the problem. If this condition is violated, then  $\mathcal{P}$  cannot be a consistent abstraction function as different scenarios are applied in different problems and  $p$  will be mapped to different sets of first-order base features. If the condition is satisfied for every literal and symbolic action in  $\mathbf{P}_D$ , then  $\hat{\mathcal{P}}$  is a consistent abstraction of  $\mathbf{P}^\#$ .

It is straightforward to see that the condition is satisfied in  $\mathbf{P}_D$  for scenario (1) since the scenario depends on  $\mathbf{A}$  which is the same in  $\mathbf{P}_D$ . Scenarios (2) and (3) depend on the set of objects in the problem. For both scenarios, the condition is only satisfied in abstract-equivalent problems. Thus,  $\mathcal{P}$  is a consistent abstraction function for abstract-equivalent problems only. It follows that  $\wp(\hat{\mathcal{P}})$  is a consistent abstract state space for abstract-equivalent problems.  $\square$

#### Theorem 4 (Subsumption of First-Order Base Features)

*If a problem  $P_i$  is subsumed by another problem  $P_j$ , then the set of first-order base features for  $P_i$  is a subset of that for  $P_j$ .*

*Proof.* If  $P_i$  is subsumed by  $P_j$ , then from Definition 19,  $\mathbf{R}_1(P_i, P_j) \wedge \mathbf{R}_2(P_i, P_j)$  is true. Let  $\mathcal{C}$  be the type that satisfies  $\mathbf{R}_2(P_i, P_j)$ : there is one object of type  $\mathcal{C}$  in  $P_i$  and multiple objects of type  $\mathcal{C}$  in  $P_j$ . Symbolic actions either have term(s) of the type  $\mathcal{C}$  or do not have term(s) of the type  $\mathcal{C}$ . We denote the former by  $\hat{a}_1$  and the latter by  $\hat{a}_2$ . First, we consider  $P_i$ . Every literal with an object of the type  $\mathcal{C}$  in its term must have the same object since there is no other objects of the type  $\mathcal{C}$ . For these literals,  $\mathcal{P}(\cdot, \hat{a}_1)$  substitutes the object with a bound variable and  $\mathcal{P}(\cdot, \hat{a}_2)$  with a free variable. We consider  $P_j$  now. For every literal which has an object of the type  $\mathcal{C}$  in its term,  $\mathcal{P}(\cdot, \hat{a}_1)$  substitutes the object with a bound variable and a free variable (i.e, one-to-many mapping) and  $\mathcal{P}(\cdot, \hat{a}_2)$  substitutes the object with a free variable. Since  $\mathbf{R}_1(P_i, P_j)$  is also true, the set of first-order base features for  $\hat{a}_1$

in  $P_j$  is a superset of that in  $P_i$ . It follows that the set of first-order base features for  $P_i$  is a subset of that for  $P_j$ .  $\square$

## A.2 Proofs from Chapter 4

### Theorem 6 (Soundness of LDE and its Variants)

*LDE, LDE-DT, LDE-FO, and LDE-DT-FO are sound.*

*Proof.* It is straightforward to see that LDE is sound as it only adds observed dead end situations to  $\chi$ . Subsumption finds a more general dead end situation among observed ones but does not generalise to unseen state-action pairs. Therefore, a state-action pair  $(s, a)$  is in  $\chi$  if and only if there is an observation  $(s_t, a_t, r_t, s_{t+1})$  where  $s = s_t$ ,  $a = a_t$ , and  $s_{t+1}$  is a dead end. The same reasoning applies for dead end traps; thus, LDE-DT is also sound.

Next, we prove that LDE-FO is sound. It uses first-order abstraction to generalise to unobserved state-action pairs. LDE-FO is sound if this generalisation does not cause it to erroneously determine state-action pairs lead to dead ends. Let  $(s_t, a_t, r_t, s_{t+1})$  be an observation where  $s_{t+1}$  is a dead end and  $(\bar{s}_t, \hat{a}_t, r_t, \bar{s}_{t+1})$  be its first-order representation. Any state-action pair which maps to  $(\bar{s}_t, \hat{a}_t)$  must lead to a next state, with non-zero probability, that maps to  $\bar{s}_{t+1}$ . Formally,  $\forall s \in \mathbf{S}, a \in \mathbf{A}$ , if  $s \mapsto \bar{s}_t$  and  $a \mapsto \hat{a}_t$ , then  $\exists s' \mathcal{T}(s'|s, a) > 0$  such that  $s' \mapsto \bar{s}_{t+1}$ . This is true for relational problems since  $\mathcal{T}(\bar{s}_{t+1}|\bar{s}_t, \hat{a}_t) = \mathcal{T}(s_{t+1}|s_t, a_t) > 0$ . We shall prove this next.

In relational problems, transition functions are parameterised and factored. First, we describe what a parameterised transition function entails. The parameterised transition of a symbolic state predicate  $\hat{p}$  is denoted by  $\mathcal{T}(\hat{p}|\hat{\mathcal{P}}, \hat{a})$  where  $\hat{\mathcal{P}}$  is the set of lifted state predicates. The transition of a state predicate  $p$  is the grounding of  $\mathcal{T}(\hat{p}|\hat{\mathcal{P}}, \hat{a})$  which gives  $\mathcal{T}(p|\mathbf{P}^-, a)$  where  $\mathbf{P}^- \subset \mathbf{P}$ . Each lifted state predicate in  $\hat{\mathcal{P}}$  is grounded by substituting variables with objects in  $p$  and  $a$  if they are of the same type. Remaining variables are substituted with objects in  $\mathbf{O}$  (this is a one-to-many substitution and the variables are analogous to existential or free variables).

Next, we describe what a parameterised and factored transition function entails.

A factored transition function is given by Equation 2.2. If it is also parameterised, then the transition of a state  $s$  is given as:

$$\begin{aligned}\mathcal{T}(s'|s, a) &= \prod_i \mathcal{T}(p'_i | \mathbf{P}^-, a) \\ &= \prod_i \mathcal{T}(\hat{p}'_i | \hat{\mathcal{P}}^-, \hat{a}),\end{aligned}\tag{A.1}$$

where  $p'_i$  is the value of  $p_i$  in  $s'$ ,  $\hat{p}_i$  is the symbolic state predicate of  $p_i$ ,  $\hat{a}$  is the symbolic action of  $a$ , and  $\hat{\mathcal{P}}^- \subset \hat{\mathcal{P}}$  results from the lifting of  $\mathbf{P}^-$ . Concretely, we lift each state predicate in  $\mathbf{P}^-$  as follows:

$$\hat{\mathcal{P}}^- = \bigcup_{p \in \mathbf{P}^-} \mathcal{L}(p, a).\tag{A.2}$$

Since all state predicates in  $\mathbf{P}^-$  are in  $s$ , it follows that all lifted state predicates in  $\hat{\mathcal{P}}^-$  are in  $\bar{s}$  where  $s \mapsto \bar{s}$  using Equation 4.6. Following Equation A.1, we have:

$$\begin{aligned}\mathcal{T}(s'|s, a) &= \prod_i \mathcal{T}(\hat{p}'_i | \hat{\mathcal{P}}^-, \hat{a}) \\ &= \mathcal{T}(\bar{s}' | \bar{s}, \hat{a})\end{aligned}\tag{A.3}$$

This assumes that deictic objects of the same type can be treated as a homogeneous entity in the parameterised and factored transition function. This completes the proof that LDE-FO is sound in relational problems. Again, the same reasoning applies for dead end traps; thus, LDE-DT-FO is sound.

□



# Appendix B

## Additional Examples

### B.1 Examples for MBFS

#### Example 40 (Constructing graphs and nodes for Recon)

We consider a particular problem for *Recon* where there is the following objects with their types in parentheses: *a1* (agent), *w1*, *l1*, *p1* (tool), *o1*, *o2* (obj), *wp1*, *wp2*, *wp3*, and *wp4* (wp). The objects *o1* and *o2* are at *wp2* and *wp4*, respectively. The DBN is shown in Figure B.1 where actions and state predicates involving *o2* are omitted for brevity. The parameterised reward function written in RDDDL is:

```
reward =
  [sum_{?OBJ : obj} (20 * [~pictureTaken(?OBJ) ^ lifeDetected(?OBJ) ^
    exists_{?AGENT: agent, ?TOOL: tool}
    [useToolOn(?AGENT, ?TOOL, ?OBJ) ^ CAMERA_TOOL(?TOOL) ^
    ~damaged(?TOOL)])]
] +
  [sum_{?OBJ : obj} -(20 * [~lifeDetected(?OBJ) ^
    exists_{?AGENT: agent, ?TOOL: tool}
    [useToolOn(?AGENT, ?TOOL, ?OBJ) ^ CAMERA_TOOL(?TOOL)])]
];
```

The reward function is parameterised with variables and is additive with the summation operator  $sum_{\{...\}}$  which sums every grounding of the logical formula enclosed by the square brackets. A true (false) formula has a value of 1 (0). Variables are prefixed with “?” and negative literals are prefixed with “~”. The parameterised reward function is grounded with objects such that the conditions in the arithmetic evaluates

to true. A graph is constructed from each grounding of an arithmetic. Consider the following arithmetic:

```
20 * [~pictureTaken(?OBJ) ^ lifeDetected(?OBJ) ^
exists_{?AGENT: agent, ?TOOL: tool}
[useToolOn(?AGENT, ?TOOL, ?OBJ) ^ CAMERA_TOOL(?TOOL) ^
~damaged(?TOOL)]]
```

which is ground to  $\wedge(\neg\text{pictureTaken}(o1), \text{lifeDetected}(o1), \neg\text{damaged}(p1), \text{CAMERA\_TOOL}(p1), \text{useToolOn}(a1, p1, o1))$  with the substitution  $\{AGENT/a1, TOOL/p1, OBJ/o1\}$ . The literals  $\text{pictureTaken}(o1)$ ,  $\text{lifeDetected}(o1)$ ,  $\text{damaged}(p1)$ , and  $\text{CAMERA\_TOOL}(p1)$  and the action  $\text{useToolOn}(a1, p1, o1)$  are grouped as a set of nodes,  $\mathbf{n}_1$ . The arithmetic can be grounded to other forms (e.g., substitute  $OBJ$  with  $o2$ ).

The parameterised precondition for  $\text{useToolOn}(AGENT, TOOL, OBJ)$  written in RDDL is:

```
forall_{?AGENT: agent, ?TOOL: tool, ?OBJ: obj}
[useToolOn(?AGENT, ?TOOL, ?OBJ) =>
(exists_{?WP : pos} [(agentAt(?AGENT, ?WP) ^ OBJECT_AT(?OBJ, ?WP))]]];
```

which states that the agent  $AGENT$  and the object  $OBJ$  must be at the same location for  $\text{useToolOn}(AGENT, TOOL, OBJ)$  to be applicable. The ground precondition for  $\text{useToolOn}(a1, p1, o1)$  is  $\text{agentAt}(a1, wp2) \wedge \text{OBJECT\_AT}(o1, wp2)$  ( $WP$  is substituted with  $wp2$  because  $o1$  is at  $wp2$ ). The literals  $\text{agentAt}(a1, wp2)$  and  $\text{OBJECT\_AT}(o1, wp2)$  and the action  $\text{useToolOn}(a1, p1, o1)$  are grouped as a set of nodes,  $\mathbf{n}_2$ .

The parameterised CPF for  $\text{pictureTaken}(OBJ)$  is written in RDDL as:

```
pictureTaken'(?OBJ) =
if (pictureTaken(?OBJ))
then true
else
(exists_{?AGENT: agent, ?TOOL: tool}
[CAMERA_TOOL(?TOOL) ^ useToolOn(?AGENT, ?TOOL, ?OBJ)
^ ~damaged(?TOOL)]);
```

where  $\text{pictureTaken}'(?OBJ)$  is the value of  $\text{pictureTaken}(OBJ)$  at the next time step. The CPF for  $\text{pictureTaken}(o1)$  is the ground form of the parameterised CPF with the substitution  $\{AGENT/a1, TOOL/p1, OBJ/o1\}$ . The literals  $\text{pictureTaken}(o1)$ ,  $\text{CAMERA\_TOOL}(p1)$ , and  $\text{damaged}(p1)$  and the action  $\text{useToolOn}(a1, p1, o1)$  are the parent nodes of the child node  $\text{pictureTaken}(o1)$  in the graph  $G$ . Note that  $\text{pictureTaken}(o1)$  is a parent of itself; this is rather common because the value of a state predicate often depends on its value at the previous time step. Since the action in  $G$  is the same as the action in  $\mathbf{n}_1$  and  $\mathbf{n}_2$ , they are added to  $G$  as parent nodes to  $\text{pictureTaken}(o1)$ .

#### Example 41 (Base Features for Recon Using MBFS)

Using the graph  $G$  constructed in Example 40, Algorithm 13 returns 8 literals for  $\nu = 2$ :  $\text{agentAt}(a1, wp2)$ ,  $\text{damaged}(p1)$ ,  $\text{lifeDetected}(o1)$ ,  $\text{pictureTaken}(o1)$ , and their negation. They are the base features for  $\text{useToolOn}(a1, p1, o1)$ . The agent receives a positive reward if it takes a good picture of  $o1$  by executing  $\text{useToolOn}(a1, p1, o1)$ . The states for which this can happen are when the agent is at the same location as  $o1$  ( $\text{agentAt}(a1, wp2)$ ), life is detected on  $o1$  ( $\text{lifeDetected}(o1)$ ), the agent has not taken a picture of  $o1$  ( $\neg\text{pictureTaken}(o1)$ ), and the camera tool is not damaged ( $\neg\text{damaged}(p1)$ ). Clearly, MBFS provides the necessary literals for approximating the  $Q$ -values of  $\text{useToolOn}(a1, p1, o1)$ .

For  $\nu = 3$ , Algorithm 13 returns the following literals:  $\text{agentAt}(a1, wp1)$ ,  $\text{agentAt}(a1, wp4)$ ,  $\text{agentAt}(a1, wp2)$ ,  $\text{agentAt}(a1, wp7)$ ,  $\text{lifeChecked2}(o1)$ ,  $\text{lifeChecked2}(o2)$ ,  $\text{lifeDetected}(o1)$ ,  $\text{lifeDetected}(o2)$ ,  $\text{pictureTaken}(o1)$ ,  $\text{pictureTaken}(o2)$ ,  $\text{waterChecked}(o1)$ ,  $\text{waterChecked}(o2)$ ,  $\text{waterDetected}(o1)$ ,  $\text{waterDetected}(o2)$ ,  $\text{damaged}(l1)$ ,  $\text{damaged}(p1)$ ,  $\text{damaged}(w1)$ , and their negation. Among them, there are some literals such as  $\text{lifeDetected}(o2)$  which are unnecessary base features for  $\text{useToolOn}(a1, p1, o1)$ . There are some literals which might seem necessary for approximating the  $Q$ -values but are unnecessary. Consider the literal  $\text{waterDetected}(o1)$ . Life can only be detected on an object after water has been detected—if  $\text{lifeDetected}(o1)$  is true, then  $\text{waterDetected}(o1)$  must be true. Therefore, we need only consider  $\text{lifeDetected}(o1)$  for  $\text{useToolOn}(a1, p1, o1)$ .

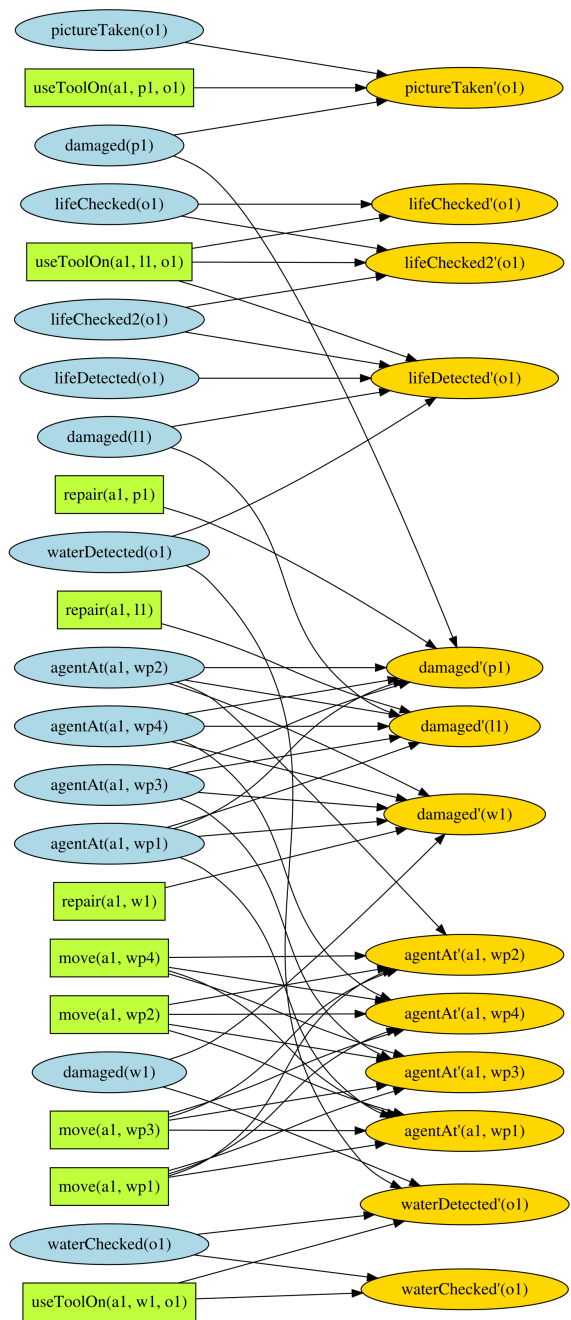


Figure B.1: DBN representing the transition function for a particular problem of Recon which is described in Example 40.

# Appendix C

## Additional Empirical Results

### C.1 Empirical Results for Chapter 3

#### C.1.1 Blocks World

We describe our version of the `Blocks World` domain which is written in RDDDL. In `Blocks World`, blocks are either on the table or stacked on another block. The RDDDL domain file is shown in Section D.5 of Appendix D. The set of types is  $\mathcal{C} = \{block\}$ . The symbolic state predicates  $\mathcal{P}$  are:

- `GOAL(BLOCK1, BLOCK2)`: the goal is to stack the block `BLOCK1` on `BLOCK2`,
- `clear(BLOCK)`: there is no other block on the block `BLOCK`,
- `on_table(BLOCK)`: the block `BLOCK` is on the table, and
- `on(BLOCK1, BLOCK2)`: the block `BLOCK1` is stacked directly on the block `BLOCK2`.

The symbolic actions  $\mathcal{A}$  are:

- `stack(BLOCK1, BLOCK2)`: unstack the block `BLOCK1` from another block or pick it from the table and stack it on the block `BLOCK2`, and
- `unstack(BLOCK1, BLOCK2)`: unstack the block `BLOCK1` from `BLOCK2` and put it on the table.

All actions are deterministic. There are three types of goals:

1. *Stack*: stack all blocks in one column.
2. *Unstack*: put all blocks on the table.

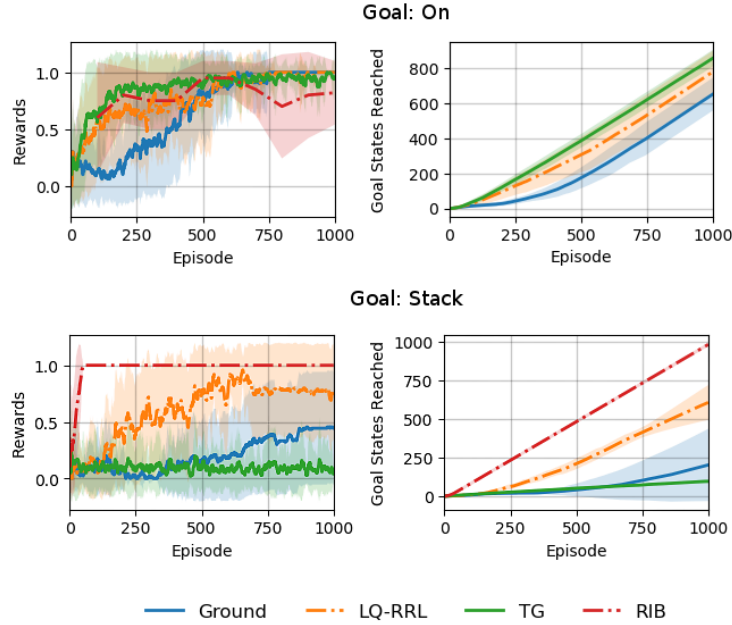


Figure C.1: Comparing other RRL methods with LQ-RRL on randomised problems of `Blocks World` which involve ten blocks. “Ground” denotes the ground approximation which serves as a point of reference.

3. *On*: stack a specific block on top of another specific block.

A problem can have either one of the three goals. The immediate reward is 0 in every time step that the goal state is not reached and 1 for reaching the goal state. We consider randomised problems with ten blocks. The size of the state-action space is  $2^{120} \times 201$  and the time horizon is 30. The problems are randomised in the initial states, and for *On*, the blocks involved in the goal.

### C.1.2 Comparison with Other RRL Methods

We compared our RRL method, LQ-RRL (see Algorithm 1), with state-of-the-art RRL methods TG [44] and RIB [45]. This work, hereafter denoted by SOTA methods, is tested extensively on `Blocks World` which is written in Prolog. The SOTA methods and the domain and problems for `Blocks World` are obtained from the publicly available ACE (A Combined Engine) data mining system [14]. We model `Blocks World` in RDDDL as described in the previous section, replicating as close as possible to the Prolog version. The SOTA methods are tested on the Prolog version while LQ-RRL is tested on the RDDDL version. For details on the Prolog version, we refer readers to [42]. One major difference is that we use lesser symbolic state pred-

icates and no handcrafted features. The SOTA methods consider a state variable to represent the number of blocks on a block and a state predicate to represent the relation that if a block is above another block (this is different from directly on the block).

**Experimental setup.** The hyperparameters for the SOTA methods use the default values in ACE [14] except that the time horizon is changed to 30. The hyperparameters for LQ-RRL are listed in Section 3.7.1. LQ-RRL uses model-free feature selection and ground context. For the goal *On*, the data points for RIB are at an interval of 100 episodes rather than 1 episode due to its high computational cost. For this reason, the cumulative number of goal states reached cannot be computed and is omitted in Figure C.1. The results are the aggregation of ten independent runs where each run uses a different randomised problem.

Figure C.1 shows the results. We exclude results for the goal *Unstack* since all methods cannot solve it in 10000 episodes. We focus on comparing the first-order approximation with the SOTA methods. For the goal *On*, TG performs best. The number of goal states reached, from highest to lowest, is  $857.8 \pm 46.6$  for TG,  $820.0 \pm 107.7$  for RIB (based on the percentages from ten data points),  $780.3 \pm 118.9$  for LQ-RRL, and  $652.9 \pm 93.0$  for “Ground”. For the goal *Stack*, RIB performs best followed by LQ-RRL while TG has the worst performance.

To conclude, RIB has the best overall performance followed by LQ-RRL. However, LQ-RRL did not utilise background knowledge unlike RIB’s domain-specific distance metric. Furthermore, LQ-RRL has been empirically demonstrated on other domains. The SOTA methods cannot solve problems with multiple goals with additive rewards or multiple types of goals (e.g., *Service Robot* has two types of goals) since they use only one goal predicate at the root node of the relational decision tree to substitute variables.

### C.1.3 Sensitivity Analysis for Online Feature Discovery

We perform the sensitivity analysis on the hyperparameters  $\xi$  for *iFDD+* and  $\tau$  for  $\tau$ -*iFDD+*. Both algorithms are discussed in Section 3.1.1.  $\xi$  is the discovery threshold for adding candidate features to  $\Phi$  and  $\tau$  controls the maximum number of features

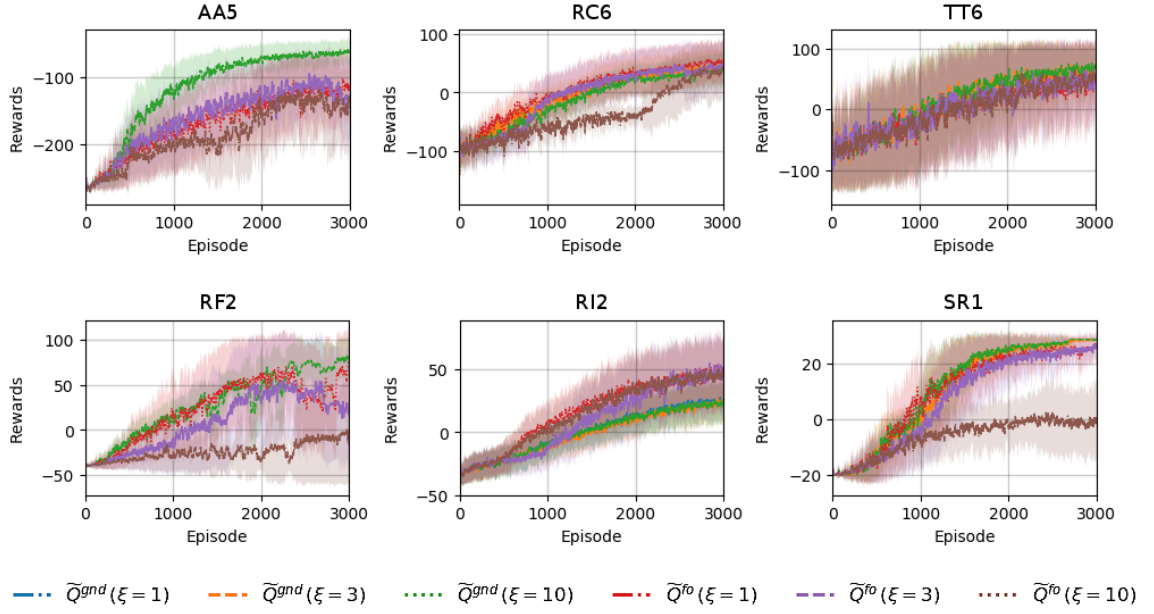


Figure C.2: Sensitivity analysis for the hyperparameter  $\xi$  for iFDD+. The total undiscounted rewards received in each episode is shown.

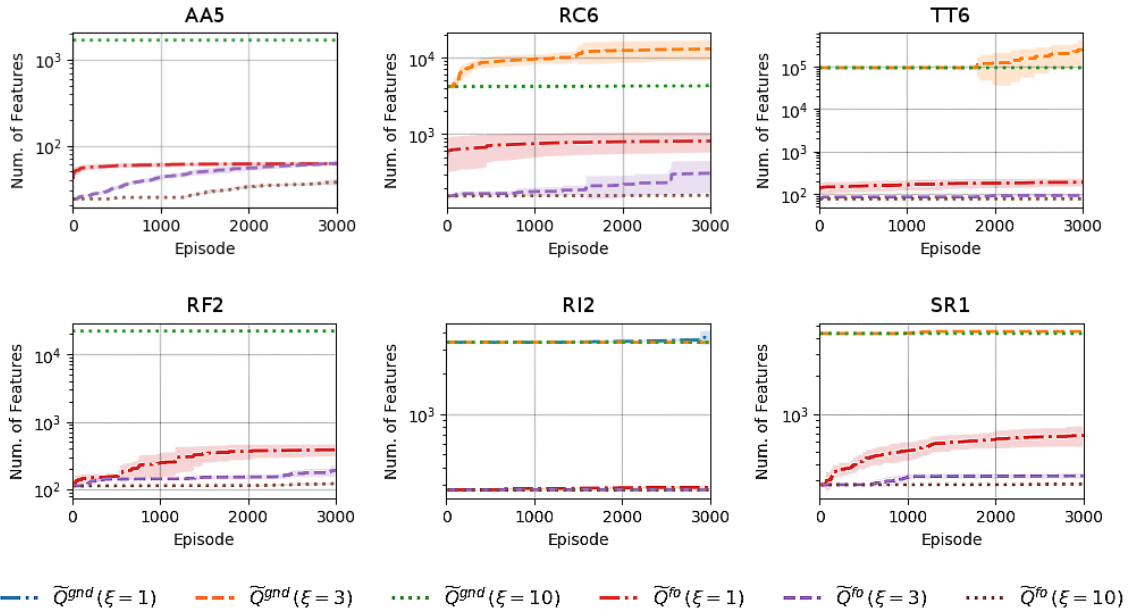


Figure C.3: Sensitivity analysis for the hyperparameter  $\xi$  for iFDD+. The total number of features added to  $\Phi$  is shown.

which can be added in each time step. Figures C.2 and C.3 show the results for  $\xi$ . The values 1, 3, and 10 are used. If  $\xi$  is too small, many candidate features are added which can be intractable. This is the case for  $\xi = 1$  which is tractable only for RI2. Results are omitted for the other problems due to intractability—the number of features added are of the order of  $10^6$ . In addition,  $\xi = 3$  is also intractable for AA5 if ground approximation is used. Since first-order features do not scale with the



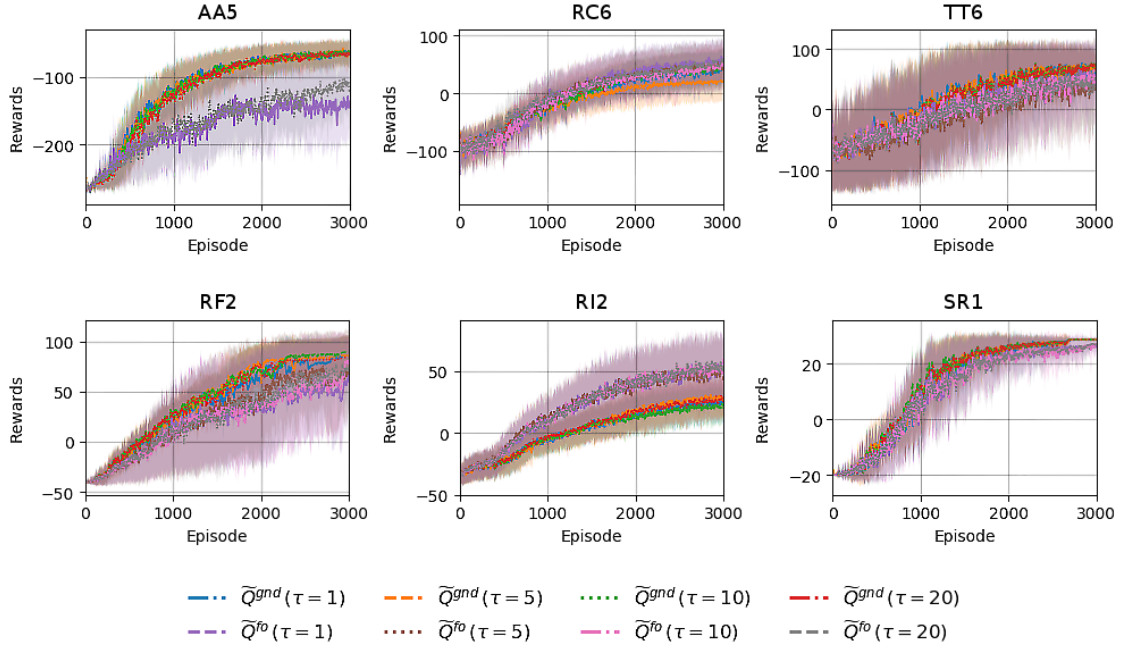


Figure C.4: Sensitivity analysis for the hyperparameter  $\tau$  for  $\tau$ -iFDD+. The total undiscounted rewards received in each episode is shown.

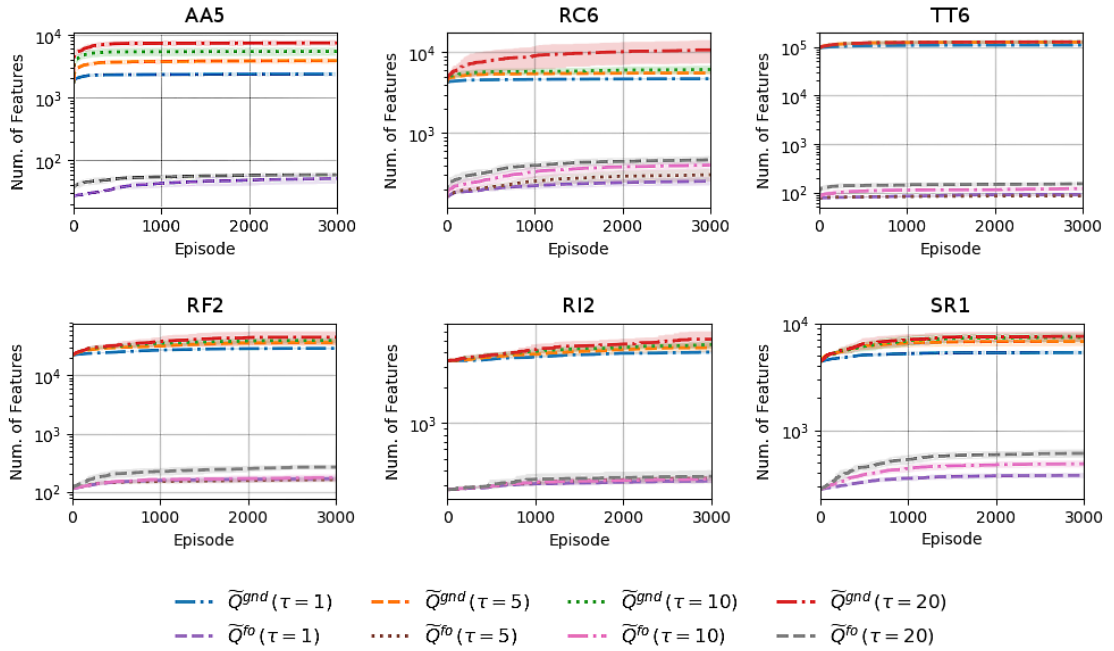


Figure C.5: Sensitivity analysis for the hyperparameter  $\tau$  for  $\tau$ -iFDD+. The total number of features added to  $\Phi$  is shown.

number of objects in the problem, it scales well with large scale problems. This is evident in Figure C.3 which shows that the number of features in a ground approximation is order of magnitudes larger than that of a first-order approximation. A large value for  $\xi$  slows down feature discovery which could increase sample complex-

ity. This is the case for  $\xi = 10$  in **AA5**, **RC6**, **RF2**, and **SR1** as shown in Figure C.2. In summary, the results show that the tuning of  $\xi$  is problem-specific and can result in large variance in performance or even intractability.

Next, we analyse the sensitivity of  $\tau$  for our proposed variant of **iFDD+**,  $\tau$ -**iFDD+**. Figures C.4 and C.5 show the results. The values 1, 5, 10, and 20 are used. In all of the runs, they are tractable. For **AA5**, results for  $\tau = 5$  and  $\tau = 10$  with a first-order approximation are omitted as they are equivalent to  $\tau = 1$ —at most one feature can be added per time step for all three of them. Figure C.4 shows that the performance for the different values of  $\tau$  are comparable in all problems. The number of features added to  $\Phi$  is consistent with the value of  $\tau$ —a larger value of  $\tau$  results in a larger number of features with the exception in **TT6** and **SR1** because the features necessary to approximate the Q-function have been added and no additional features are required. Despite starting off with a small value of  $\xi = 0.1$ ,  $\tau$ -**iFDD+** does not add too many features. The rate of feature addition is high initially, then decreases in later episodes. This is because more candidate features are generated as features are added to  $\Phi$  and  $\tau$ -**iFDD+** incrementally increases  $\xi$  whenever too many candidate features have relevances larger than  $\xi$ .

## C.2 Empirical Results for Chapter 4

### C.2.1 Tuning the Hyperparameter $\nu$ for MBFS

Although  $\nu$  seems like a hyperparameter that needs to be tuned with expert knowledge, we posit that it is less tedious to tune as it is an integer and ideal values typically range from 2 to 3. Figure C.6 shows the number of base features for different values of  $\nu$ .<sup>1</sup> This is compared with the number of base features given by **MFFS**. This information can be obtained from **MBFS** (Algorithm 12) without the need to run any experiments and can provide some guidance to tune  $\nu$ . In general, when  $\nu$  increases, the number of base features increases to a **fixed point**—the value of  $\nu$  where **MBFS** no longer adds new base features with further increases to  $\nu$ . The fixed point ranges from 3 to 4 in the six domains. For all domains except

---

<sup>1</sup>For brevity here, we refer to base features and first-order base features collectively as base features unless necessary to distinguish them.

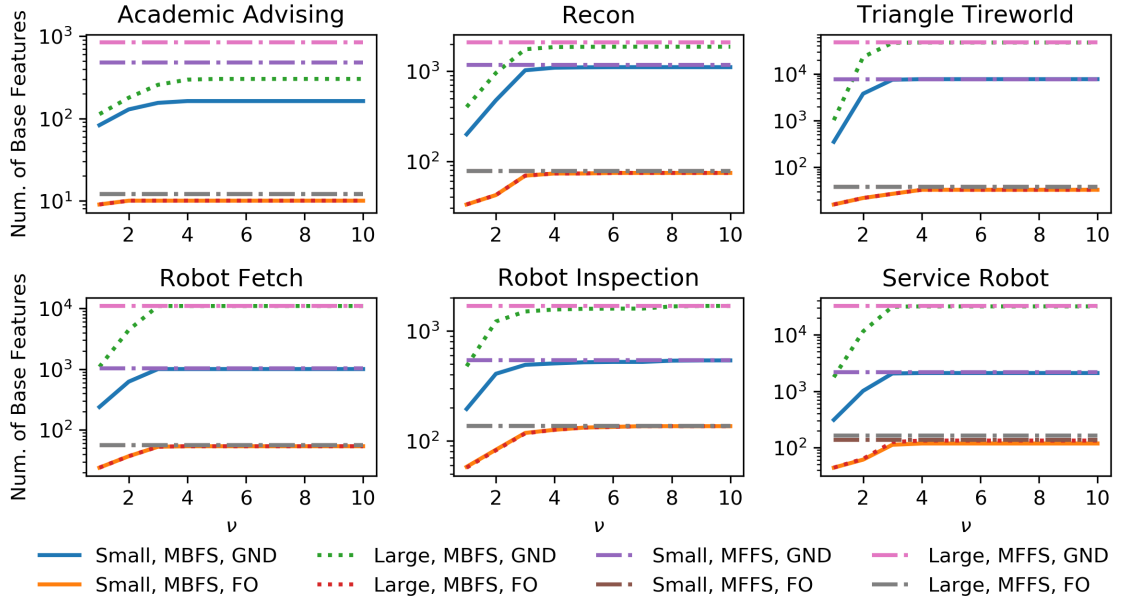


Figure C.6: The number of base features or first-order base features given by MFFS and MBFS for different values of  $\nu$ . Legend: *Small* (*Large*) stands for a small (large) scale problem of the domain, *GND* stands for ground approximation, and *FO* stands for first-order approximation.

Academic Advising, the number of base features at the fixed point is close to or equal to the number of base features from MFFS. That is, MBFS at the fixed point is equivalent to MFFS for some domains. For  $\nu = 1$ , the number of base features is too small; the granularity of the Q-function approximation is expected to be too coarse and this typically results in a poor performance. Therefore, the values to consider for  $\nu$  ranges from 2 to 3.

As a side note, the number of first-order base features is the same for the small scale and large scale problems except for Service Robot. This is because the first-order approximation uses the same set of base features for abstract-equivalent problems. Large scale problems of Service Robot (SR2 and SR3) have more first-order base features than SR1 because they are not abstract-equivalent problems and the set of first-order base features for large scale problems has to be augmented as described in Section 3.3.3.

## C.2.2 Sensitivity Analysis for $\beta$

To select a value for  $\beta$ , the coefficient for the intrinsic reward (see Equation 4.7), we tested the sensitivity of  $\beta$  using TDE as the intrinsic reward (see Section 4.4.2)

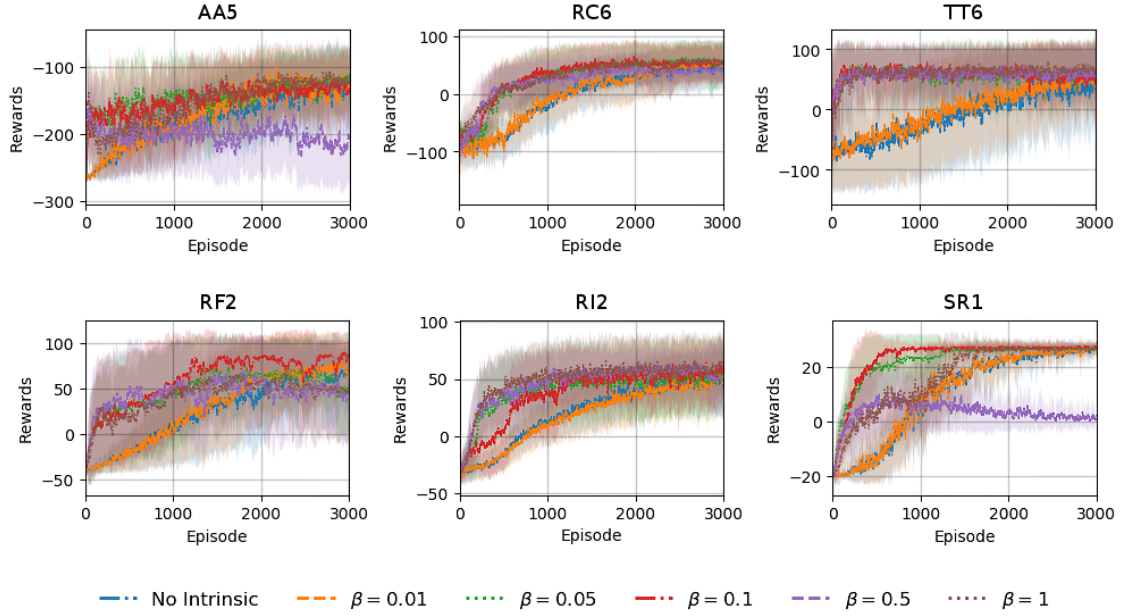


Figure C.7: Sensitivity analysis for the coefficient for intrinsic reward,  $\beta$ . The intrinsic reward used is TDE. Both the extrinsic and intrinsic approximations are first-order approximations.

and *greedy- $\epsilon$ -greedy* policy where  $\epsilon$  is initialised to 1 and decayed exponentially to 0 over the episodes. We use the values  $\{0.01, 0.05, 0.1, 0.5, 1\}$  for  $\beta$ . For  $\beta = 1$ ,  $\beta$  is exponentially decayed from 1 over episodes at the same rate as  $\epsilon$  for the  *$\epsilon$ -greedy* policy. Figure C.7 shows the results for the sensitivity analysis. The baseline is the first-order approximation without intrinsic rewards. The asymptotic performance for all values of  $\beta$  are comparable with the baseline. For  $\beta = 0.01$ , the performance is similar to the baseline. This is because the coefficient is too small and any intrinsic reward becomes insignificant. Increasing  $\beta$  to 0.05 and 0.1 gives the best overall performance.

When  $\beta$  is too large, as is the case for  $\beta = 0.5$ , there might be too much exploration rather than exploitation. This can deteriorate the performance as evident in AA5 and SR1. Even though the initial performance of  $\beta = 0.5$  is better than the baseline, the asymptotic performance is significantly worse. This can be mitigated by decaying  $\beta$ . As observed for  $\beta = 1$  in SR1, the asymptotic performance is comparable with the baseline unlike  $\beta = 0.5$ . Thus, one possible strategy to avoid having too large a value for  $\beta$  is to decay it.

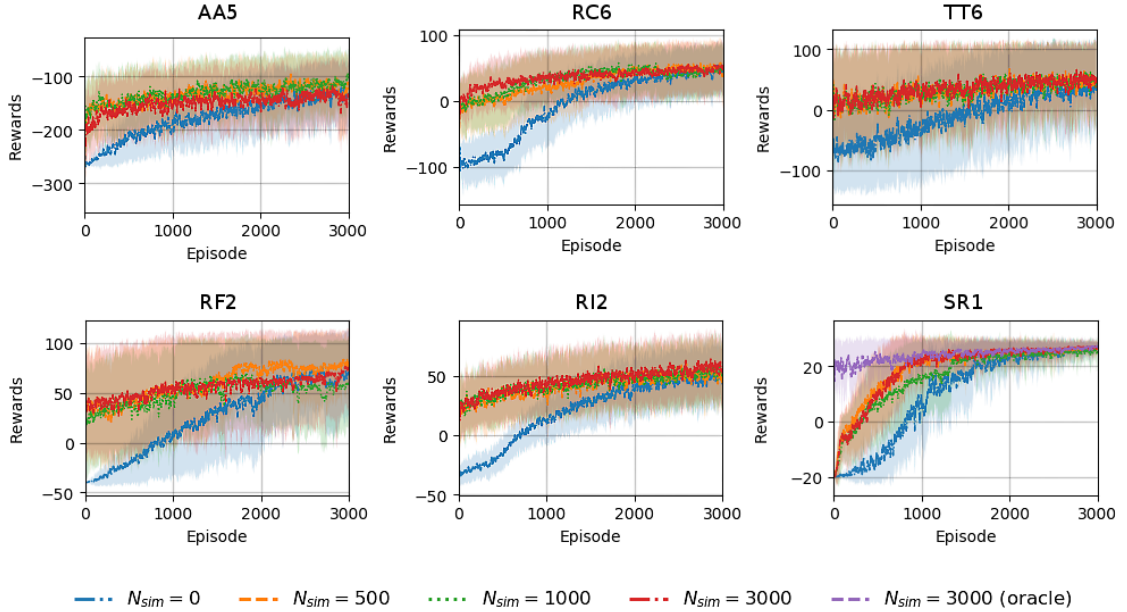


Figure C.8: Dyna uses the true models to generate imagined observations which an extrinsic first-order approximation learns from. Different number of rollouts,  $N_{sim}$ , are tested with a simulated horizon  $H_{sim} = 40$ . For  $N_{sim} = 0$ , Dyna is not used.

### C.2.3 Sensitivity Analysis for Number of Rollouts in Dyna

We performed a sensitivity analysis on the number of rollouts ( $N_{sim}$ ) for Dyna. For all runs, the simulated horizon  $H_{sim}$  is 40. Figure C.8 shows the results for the true model and Figure C.9 shows the results for the learned models where Dyna uses a different learned model for each run. All experiments use a first-order approximation. There is no intrinsic approximation; Dyna trains and initialises the extrinsic approximation. The baseline is the result for which Dyna is not used (denoted by  $N_{sim} = 0$  in the figure).

In general, varying the value of  $N_{sim}$  does not significantly impact the performance. In some cases, increasing  $N_{sim}$  gives a marginal improvement in performance while in other cases, this does not impact the performance. Dyna gives a significant jump start in all problems. The jump start is observed in SR1 only when the oracle model is used (for other domains, the oracle model and the true model are the same). The oracle model predicts unpredictable transitions of state predicates as described in Example 27 and exogenous events (i.e., a person needs assistance). This shows that unpredictable dynamics can affect model-based methods. Since some parts of the state space is not reached, learning is not possible in these states until they are observed. Nevertheless, it is encouraging that the true model and the

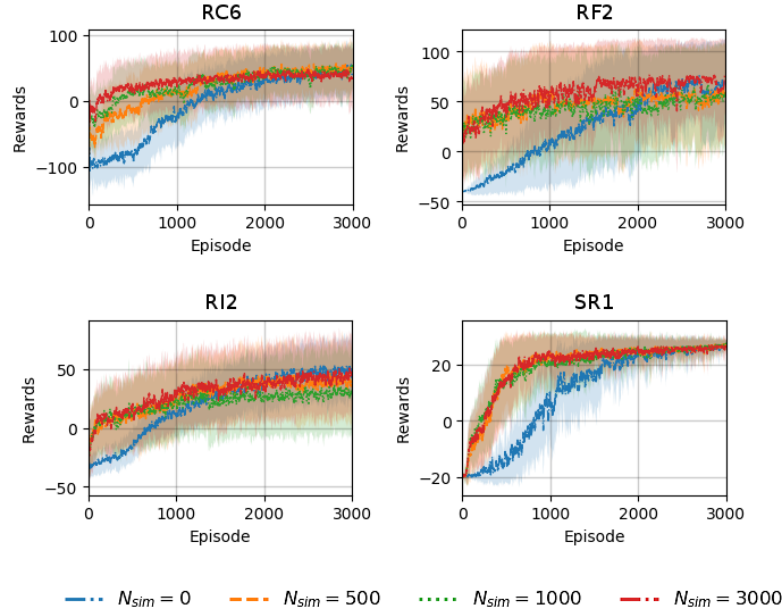


Figure C.9: Dyna uses the learned models to generate imagined observations which an extrinsic first-order approximation learns from. The models are learned from a set of training data which contains 500 state transitions per symbolic action. Different rollouts  $N_{sim}$  are tested with a simulated horizon  $H_{sim} = 40$ .  $N_{sim} = 0$  implies that Dyna is not used.

learned models give improved performance over the baseline in SR1. This is likely due to the addition of useful candidate features to  $\Phi$  by Dyna. For all problems, the asymptomatic performance are comparable with the baseline; despite errors in the learned models, Dyna did not worsen the asymptomatic performance.

## C.3 Empirical Results for Chapter 5

### C.3.1 Effect of $\Delta T$ on TDORMDP Problems

$\Delta T$  determines the time bounds for TGs (see Equations 5.2 and 5.3). Example 34 shows that  $\Delta T$  is the average time given for achieving each TG if the policy is optimal. We tested values of 150, 300, and 600 seconds for  $\Delta T$  in randomised TDORMDP problems of SR3. In all problems, the first-order approximation is learned from scratch. The results are shown in Figure C.10.  $\Delta T = \infty$  denotes RMDP problems (i.e., no dynamic objects and time-bounded goals); their results serve as a baseline. The performance when  $\Delta T = 600$  is close to the performance when  $\Delta T = \infty$ . On the other end of the spectrum, the performance is poor when

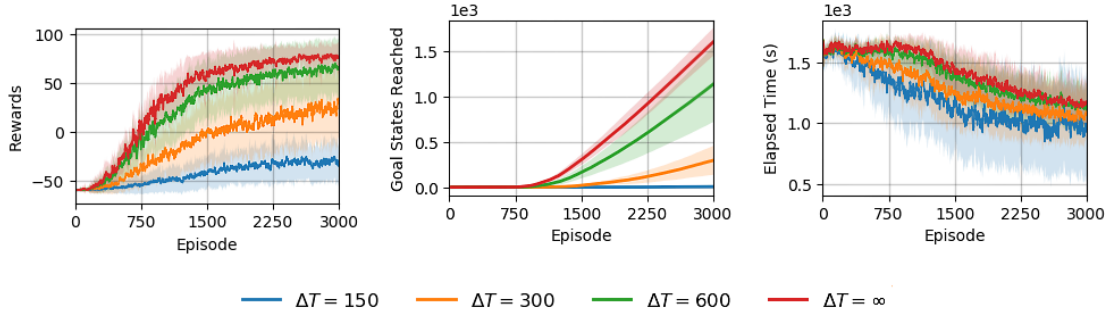


Figure C.10: Effect of  $\Delta T$  on performance in randomised problems of SR3. All problems have dynamic objects and time-bounded goals except for those with  $\Delta T = \infty$ . The values of  $\Delta T$  are in seconds.

$\Delta T = 150$ ; the goal state was never reached in 3000 episodes. As a middle ground, we use  $\Delta T = 300$ . We want a value for  $\Delta T$  that makes TDORMDP problems challenging but not (near) impossible to solve. 300 seconds is a reasonable value for  $\Delta T$  considering the action durations, probabilistic effects (e.g., loss of localisation, failure to grasp an item), and non-greedy behaviours due to exploration.

# Appendix D

## RDDL Domains

### D.1 Recon

```
domain recon_mdp {
  requirements = {
    reward-deterministic
  };

  types {
    wp : object;
    obj : object;
    agent: object;
    tool : object;
  };

  pvariables {
    COST: {non-fluent, real, default = -1.0};
    ADJACENT(wp, wp) : {non-fluent, bool, default = false};
    OBJECT_AT(obj, wp) : {non-fluent, bool, default = false};
    HAZARD(wp) : {non-fluent, bool, default = false};
    DAMAGE_PROB(tool) : {non-fluent, real, default = 0.0};
    DETECT_PROB : {non-fluent, real, default = 1.0};
    DETECT_PROB_DAMAGED : {non-fluent, real, default = 0.4};
    CAMERA_TOOL(tool) : {non-fluent, bool, default = false};
    LIFE_TOOL(tool) : {non-fluent, bool, default = false};
    WATER_TOOL(tool) : {non-fluent, bool, default = false};
    BASE(wp) : {non-fluent, bool, default = false};
  };
}
```



```

GOOD_PIC_WEIGHT : {non-fluent, real, default = 20.0};
BAD_PIC_WEIGHT : {non-fluent, real, default = 20.0};

damaged(tool) : {state-fluent, bool, default = false};
waterChecked(obj) : {state-fluent, bool, default = false};
waterDetected(obj) : {state-fluent, bool, default = false};
lifeChecked(obj) : {state-fluent, bool, default = false};
lifeChecked2(obj) : {state-fluent, bool, default = false};
lifeDetected(obj) : {state-fluent, bool, default = false};
pictureTaken(obj) : {state-fluent, bool, default = false};
agentAt(agent, wp) : {state-fluent, bool, default = false};

move(agent, wp) : {action-fluent, bool, default = false};
useToolOn(agent, tool, obj) : {action-fluent, bool, default = false};
repair(agent, tool) : {action-fluent, bool, default = false};
};

cpfs {
  damaged'(?t) =
    if (damaged(?t) ^ ~(exists_{?loc : wp, ?a: agent}
      [agentAt(?a, ?loc) ^ BASE(?loc) ^ repair(?a, ?t)]))
    then true
    else if (exists_{?loc : wp, ?a: agent}
      [agentAt(?a, ?loc) ^ ~BASE(?loc) ^ HAZARD(?loc)])
    then Bernoulli(DAMAGE_PROB(?t))
    else if (exists_{?loc : wp, ?a: agent, ?loc2 :wp}
      [agentAt(?a, ?loc) ^ ~BASE(?loc)
        ^ HAZARD(?loc2) ^ ADJACENT(?loc, ?loc2)])
    then Bernoulli(DAMAGE_PROB(?t) / 2.0)
    else false;

  waterChecked'(?o) =
    KronDelta(waterChecked(?o) | exists_{?a: agent, ?t: tool}
      [useToolOn(?a, ?t, ?o) ^ WATER_TOOL(?t)]);

  waterDetected'(?o) =
    if (waterDetected(?o)) then true
    else if (waterChecked(?o)) then false
    else if (exists_{?t : tool, ?a: agent}

```

```

        [WATER_TOOL(?t) ^ damaged(?t) ^ useToolOn(?a, ?t, ?o)])
    then Bernoulli(DETECT_PROB_DAMAGED)
    else if (exists_{?t : tool, ?a: agent}
        [WATER_TOOL(?t) ^ useToolOn(?a, ?t, ?o)])
    then Bernoulli(DETECT_PROB)
    else false;

lifeChecked'(?o) =
    KronDelta(lifeChecked(?o) | exists_{?a: agent, ?t: tool}
        [useToolOn(?a, ?t, ?o) ^ LIFE_TOOL(?t)]);

lifeChecked2'(?o) =
    KronDelta(lifeChecked2(?o) | lifeChecked(?o) ^
        exists_{?a: agent, ?t: tool}
        [useToolOn(?a, ?t, ?o) ^ LIFE_TOOL(?t)]);

lifeDetected'(?o) =
    if (lifeDetected(?o)) then true
    else if (lifeChecked2(?o) | ~waterDetected(?o)) then false
    else if (exists_{?t : tool, ?a: agent}
        [LIFE_TOOL(?t) ^ damaged(?t) ^ useToolOn(?a, ?t, ?o)])
    then Bernoulli(DETECT_PROB_DAMAGED)
    else if (exists_{?t : tool, ?a: agent}
        [LIFE_TOOL(?t) ^ useToolOn(?a, ?t, ?o)])
    then Bernoulli(DETECT_PROB)
    else false;

pictureTaken'(?o) =
    if (pictureTaken(?o)) then true
    else KronDelta(exists_{?a: agent, ?t: tool}
        [CAMERA_TOOL(?t) ^ useToolOn(?a, ?t, ?o) ^ ~damaged(?t)]);

agentAt'(?a, ?loc) =
    if (move(?a, ?loc)) then true
    else if (exists_{?loc1 : wp} (move(?a, ?loc1))) then false
    else agentAt(?a, ?loc);
};

reward =

```

```

[sum_{?a: agent, ?loc: wp} [COST * move(?a, ?loc)]] +
[sum_{?a: agent, ?t: tool, ?o : obj}
  [COST * useToolOn(?a, ?t, ?o)]] +
[sum_{?a: agent, ?t: tool} [COST * repair(?a, ?t)]] +
[sum_{?o : obj} (GOOD_PIC_WEIGHT *
  [~pictureTaken(?o) ^ lifeDetected(?o) ^ exists_{?a: agent, ?t: tool}
  [useToolOn(?a, ?t, ?o) ^ CAMERA_TOOL(?t) ^ ~damaged(?t)]])] +
[sum_{?o : obj} -(BAD_PIC_WEIGHT *
  [~lifeDetected(?o) ^ exists_{?a: agent, ?t: tool}
  [useToolOn(?a, ?t, ?o) ^ CAMERA_TOOL(?t)]])];

action-preconditions {
  forall_{?a: agent, ?to: wp} [move(?a, ?to) =>
    (exists_{?from : wp} [(agentAt(?a, ?from) ^
      (ADJACENT(?from, ?to) | ADJACENT(?to, ?from)))]);
  forall_{?a: agent, ?t: tool, ?o: obj} [useToolOn(?a, ?t, ?o) =>
    (exists_{?loc : wp} [(agentAt(?a, ?loc) ^ OBJECT_AT(?o, ?loc))]]);
};
}

```

## D.2 Robot Fetch

```

domain robot_fetch_mdp {
  types {
    obj : object;
    robot : object;
    wp : object;
  };

  pvariables {
    TASK_REWARD : {non-fluent, real, default = 20.0};
    COST : {non-fluent, real, default = -1.0};
    OBJECT_GOAL(obj, wp) : {non-fluent, bool, default = false};

    robot_at(robot, wp) : {state-fluent, bool, default = false};
    localised(robot) : {state-fluent, bool, default = false};
    emptyhand(robot) : {state-fluent, bool, default = false};
    holding(robot, obj) : {state-fluent, bool, default = false};
  };
}

```

```

object_at(obj, wp) : {state-fluent, bool, default = false};

move(robot, wp) : {action-fluent, bool, default = false};
localise(robot) : {action-fluent, bool, default = false};
pick_up(robot, obj, wp) : {action-fluent, bool, default = false};
put_down(robot, obj, wp) : {action-fluent, bool, default = false};
};

cpfs {
  robot_at'(?r, ?loc) =
    if (~robot_at(?r, ?loc) ^ move(?r, ?loc) ^ localised(?r)) then true
    else if (exists_{?loc1: wp}
             [(robot_at(?r, ?loc) ^ move(?r, ?loc1) ^ localised(?r))])
    then false
    else robot_at(?r, ?loc);

  localised'(?r) =
    if (localise(?r)) then true
    else if (exists_{?loc: wp} [move(?r, ?loc)]) then true
    else localised(?r);

  emptyhand'(?r) =
    if (exists_{?o: obj, ?loc: wp} [pick_up(?r, ?o, ?loc)])
    then false
    else if (exists_{?o: obj, ?loc: wp} [put_down(?r, ?o, ?loc)])
    then true
    else if (exists_{?o: obj} [holding(?r, ?o)]) then false
    else true;

  holding'(?r, ?o) =
    if (exists_{?loc: wp} [pick_up(?r, ?o, ?loc)]) then true
    else if (exists_{?loc: wp} [put_down(?r, ?o, ?loc)]) then false
    else holding(?r, ?o);

  object_at'(?o, ?loc) =
    if (exists_{?r: robot} [pick_up(?r, ?o, ?loc)]) then false
    else if (exists_{?r: robot} [put_down(?r, ?o, ?loc)]) then true
    else object_at(?o, ?loc);
};

```

```

reward =
  [sum_{?o: obj}
    [TASK_REWARD * (exists_{?r: robot, ?loc: wp}
      [(put_down(?r, ?o, ?loc) ^ OBJECT_GOAL(?o, ?loc))]) -
      TASK_REWARD * (exists_{?r: robot, ?loc: wp}
        [(pick_up(?r, ?o, ?loc) ^ OBJECT_GOAL(?o, ?loc))])] +
    [sum_{?r: robot, ?loc: wp} [COST * move(?r, ?loc)]] +
    [sum_{?r: robot} [COST * localise(?r)]] +
    [sum_{?r: robot, ?o: obj, ?loc: wp} [COST * pick_up(?r, ?o, ?loc)]] +
    [sum_{?r: robot, ?o: obj, ?loc: wp} [COST * put_down(?r, ?o, ?loc)]];

action-preconditions {
  forall_{?r: robot, ?loc: wp} [move(?r, ?loc) =>
    (localised(?r) ^ ~robot_at(?r, ?loc))];
  forall_{?r: robot, ?o: obj, ?loc: wp} [pick_up(?r, ?o, ?loc) =>
    (robot_at(?r, ?loc) ^ object_at(?o, ?loc) ^ emptyhand(?r))];
  forall_{?r: robot, ?o: obj, ?loc: wp} [put_down(?r, ?o, ?loc) =>
    (robot_at(?r, ?loc) ^ holding(?r, ?o) ^
      ~(exists_{?o1: obj} [object_at(?o1, ?loc)])]);
};
}

```

### D.3 Robot Inspection

```

domain robot_inspection_mdp {
  types {
    wp: object;
    robot: object;
    obj: object;
  };

  pvariables {
    COST_MOVE : {non-fluent, real, default = -1.0};
    COST_LOCALISE : {non-fluent, real, default = -1.0};
    COST_DOCK : {non-fluent, real, default = -1.0};
    COST_UNDOCK : {non-fluent, real, default = -1.0};
    COST_SURVEY : {non-fluent, real, default = -1.0};
  };
}

```

```

COST_INSPECTION : {non-fluent, real, default = -1.0};
COST_TRANSMIT : {non-fluent, real, default = -1.0};
COST_CALIBRATE : {non-fluent, real, default = -1.0};
TASK_REWARD : {non-fluent, real, default = 20.0};
PROB_LOSING_LOCALISATION : {non-fluent, real, default = 0.0};
PROB_POOR_CALIBRATION : {non-fluent, real, default = 0.0};
PROB_LOW_ENERGY : {non-fluent, real, default = 0.0};
PROB_SUCCESSFUL_SURVEY : {non-fluent, real, default = 1.0};
PROB_SUCCESSFUL_OBSERVATION : {non-fluent, real, default = 1.0};
PROB_SUCCESSFUL_SURVEY_DAMAGED : {non-fluent, real, default = 1.0};
PROB_SUCCESSFUL_OBSERVATION_DAMAGED : {non-fluent, real, default = 1.0};

DOCK_AT(wp) : {non-fluent, bool, default = false};
COMM_TOWER(wp) : {non-fluent, bool, default = false};
OBJECT_AT(obj, wp) : {non-fluent, bool, default = false};

robot_at(robot, wp) : {state-fluent, bool, default = false};
undocked(robot) : {state-fluent, bool, default = false};
docked(robot) : {state-fluent, bool, default = false};
localised(robot) : {state-fluent, bool, default = false};
object_found(robot, obj) : {state-fluent, bool, default = false};
object_inspected(robot, obj) : {state-fluent, bool, default = false};
object_info_received(obj) : {state-fluent, bool, default = false};
camera_calibrated(robot) : {state-fluent, bool, default = false};
has_energy(robot) : {state-fluent, bool, default = false};
low_energy(robot) : {state-fluent, bool, default = false};
reward_received(obj) : {state-fluent, bool, default = false};

move(robot, wp) : {action-fluent, bool, default = false};
localise(robot) : {action-fluent, bool, default = false};
dock(robot, wp) : {action-fluent, bool, default = false};
undock(robot, wp) : {action-fluent, bool, default = false};
survey(robot, wp) : {action-fluent, bool, default = false};
inspect_object(robot, obj) : {action-fluent, bool, default = false};
transmit_info(robot) : {action-fluent, bool, default = false};
calibrate_camera(robot) : {action-fluent, bool, default = false};
};

cpfs {

```

```

robot_at'(?r, ?loc) =
  if (~robot_at(?r, ?loc) ^ move(?r, ?loc) ^ localised(?r)) then true
  else if (exists_{?loc: wp}
           [(robot_at(?r, ?loc) ^ move(?r, ?loc1) ^ localised(?r))])
  then false
  else robot_at(?r, ?loc);

undocked'(?r) =
  undocked(?r) ^ ~(exists_{?loc: wp} (dock(?r, ?loc))) |
  docked(?r) ^ (exists_{?loc: wp} (undock(?r, ?loc)));

docked'(?r) =
  docked(?r) ^ ~(exists_{?loc: wp} (undock(?r, ?loc))) |
  undocked(?r) ^ (exists_{?loc: wp} (dock(?r, ?loc)));

localised'(?r) =
  if (localise(?r)) then true
  else if (exists_{?from: wp, ?loc: wp}
           [move(?r, ?loc) ^ localised(?r)])
  then Bernoulli (1-PROB_LOSING_LOCALISATION)
  else localised(?r);

object_found'(?r, ?o) =
  if (object_found(?r, ?o)) then true
  else if (exists_{?loc: wp}
           [localised(?r) ^ (survey(?r, ?loc) ^ OBJECT_AT(?o, ?loc))] ^
           camera_calibrated(?r))
  then Bernoulli (PROB_SUCCESSFUL_SURVEY)
  else if (exists_{?loc: wp} [localised(?r) ^ (survey(?r, ?loc) ^
           OBJECT_AT(?o, ?loc))] ^ ~camera_calibrated(?r))
  then Bernoulli (PROB_SUCCESSFUL_SURVEY_DAMAGED)
  else object_found(?r, ?o);

object_inspected'(?r, ?o) =
  if (object_inspected(?r, ?o)) then true
  else if (exists_{?loc: wp}
           [localised(?r) ^ inspect_object(?r, ?o) ^
           object_found(?r, ?o) ^ robot_at(?r, ?loc) ^
           OBJECT_AT(?o, ?loc) ^ camera_calibrated(?r)])

```

```

then Bernoulli (PROB_SUCCESSFUL_OBSERVATION)
else if (exists_{?loc: wp}
        [localised(?r) ^ inspect_object(?r, ?o) ^
         object_found(?r, ?o) ^ robot_at(?r, ?loc) ^
         OBJECT_AT(?o, ?loc) ^ ~camera_calibrated(?r)])
then Bernoulli (PROB_SUCCESSFUL_OBSERVATION_DAMAGED)
else object_inspected(?r, ?o);

object_info_received'(?o) =
  if (exists_{?r: robot, ?loc: wp}
      [transmit_info(?r) ^ object_inspected(?r, ?o) ^
       robot_at(?r, ?loc) ^ COMM_TOWER(?loc)])
  then true
  else object_info_received(?o);

camera_calibrated'(?r) =
  if (exists_{?loc: wp}
      [calibrate_camera(?r) ^ robot_at(?r, ?loc) ^ DOCK_AT(?loc)])
  then true
  else if (exists_{?o: obj}
           [inspect_object(?r, ?o) ^ camera_calibrated(?r)])
  then Bernoulli (1-PROB_POOR_CALIBRATION)
  else camera_calibrated(?r);

low_energy'(?r) =
  if (docked(?r)) then false
  else if (low_energy(?r)) then false
  else if (exists_{?loc: wp} [move(?r, ?loc) ^ localised(?r)])
  then Bernoulli (PROB_LOW_ENERGY)

has_energy'(?r) =
  if ((exists_{?loc: wp} [dock(?r, ?loc)]) | docked(?r))
  then true
  else if (low_energy(?r)) then false
  else has_energy(?r);

reward_received'(?o) =
  if (object_info_received(?o)) then true else reward_received(?o);
};

```



```

reward =
  [sum_{?r: robot, ?loc: wp} [COST_MOVE * move(?r, ?loc)]] +
  [sum_{?r: robot} [COST_LOCALISE * localise(?r)]] +
  [sum_{?r: robot, ?loc: wp} [COST_DOCK * dock(?r, ?loc)]] +
  [sum_{?r: robot, ?loc: wp} [COST_UNDOCK * undock(?r, ?loc)]] +
  [sum_{?r: robot, ?loc: wp} [COST_SURVEY * survey(?r, ?loc)]] +
  [sum_{?r: robot, ?o: obj} [COST_INSPECTION * inspect_object(?r, ?o)]] +
  [sum_{?r: robot} [COST_TRANSMIT * transmit_info(?r)]] +
  [sum_{?r: robot} [COST_CALIBRATE * calibrate_camera(?r)]] +
  [sum_{?o: obj}
    [TASK_REWARD * (~reward_received(?o) ^ object_info_received(?o))]];

action-preconditions {
  forall_{?r: robot, ?loc: wp} [move(?r, ?loc) =>
    (has_energy(?r) ^ undocked(?r) ^ ~robot_at(?r, ?loc))];
  forall_{?r: robot, ?loc1: wp, ?loc2: wp}
    [?loc1 == ?loc2 | (robot_at(?r, ?loc1) => ~robot_at(?r, ?loc2))];
  forall_{?r: robot} [localise(?r) => has_energy(?r) ^ undocked(?r)];
  forall_{?r: robot, ?loc: wp} [undock(?r, ?loc) =>
    (has_energy(?r) ^ DOCK_AT(?loc) ^ robot_at(?r, ?loc))];
  forall_{?r: robot, ?loc: wp} [survey(?r, ?loc) =>
    has_energy(?r) ^ undocked(?r) ^ robot_at(?r, ?loc)];
  forall_{?r: robot, ?o: obj} [inspect_object(?r, ?o) =>
    has_energy(?r) ^ undocked(?r) ^ (exists_{?loc: wp}
      [(robot_at(?r, ?loc) ^ OBJECT_AT(?o, ?loc))]);
  forall_{?r: robot} [transmit_info(?r) => has_energy(?r)];
  forall_{?r: robot} [calibrate_camera(?r) => has_energy(?r)];
};

state-invariants {
  forall_{?r: robot} [docked(?r) <=> ~undocked(?r)];
};
}

```

## D.4 Service Robot

```
domain service_robot_mdp {
```

```

types {
  obj : object;
  person : object;
  robot : object;
  wp : object;
};

pvariables {
  COST : {non-fluent, real, default = -1.0};
  TASK_REWARD : {non-fluent, real, default = 20};
  PROB_LOSING_LOCALISATION : {non-fluent, real, default = 0.0};
  PROB_NEED_ASSISTANCE(person) : {non-fluent, real, default = 0.1};
  TABLE_AT(wp) : {non-fluent, bool, default = false};
  PERSON_GOAL_OBJECT_AT(person, obj, wp) :
    {non-fluent, bool, default = false};
  PERSON_GOAL_OBJECT_WITH(person, obj, person) :
    {non-fluent, bool, default = false};
  PERSON_IS_AT(person, wp) : {non-fluent, bool, default = false};

  robot_at(robot, wp) : {state-fluent, bool, default = false};
  localised(robot) : {state-fluent, bool, default = false};
  emptyhand(robot) : {state-fluent, bool, default = false};
  holding(robot, obj) : {state-fluent, bool, default = false};
  object_at(obj, wp) : {state-fluent, bool, default = false};
  object_with(obj, person) : {state-fluent, bool, default = false};
  goal_object_at(obj, wp) : {state-fluent, bool, default = false};
  goal_object_with(obj, person) : {state-fluent, bool, default = false};
  person_at(person, wp) : {state-fluent, bool, default = false};
  need_assistance(person) : {state-fluent, bool, default = false};
  needed_assistance(person) : {state-fluent, bool, default = false};
  goal_attempted(obj) : {state-fluent, bool, default = false};
  reward_received(obj) : {state-fluent, bool, default = false};

  move(robot, wp, wp) : {action-fluent, bool, default = false};
  localise(robot) : {action-fluent, bool, default = false};
  find_person(robot, person) : {action-fluent, bool, default = false};
  talk_to_person(robot, person) : {action-fluent, bool, default = false};
  pick_up(robot, obj) : {action-fluent, bool, default = false};
  put_down(robot, obj) : {action-fluent, bool, default = false};

```

```

take(robot, obj, person) : {action-fluent, bool, default = false};
give(robot, obj, person) : {action-fluent, bool, default = false};
};

cpfs {
  robot_at'(?r, ?loc) =
    if (exists_{?loc1: wp} [(move(?r, ?loc1, ?loc))]) then true
    else if (exists_{?loc1: wp}
             [(robot_at(?r, ?loc) ^ move(?r, ?loc, ?loc1))])
    then false
    else robot_at(?r, ?loc);

  localised'(?r) =
    if (localise(?r)) then true
    else if (~localised(?r)) then false
    else if (exists_{?loc1 : wp, ?loc2 : wp} [move(?r, ?loc1, ?loc2)])
    then (Bernoulli (1 - PROB_LOSING_LOCALISATION))
    else if (exists_{?p: person} [find_person(?r, ?p)])
    then (Bernoulli (1 - PROB_LOSING_LOCALISATION))
    else localised(?r);

  emptyhand'(?r) =
    if (exists_{?o: obj} [pick_up(?r, ?o)]) then false
    else if (exists_{?o: obj} [put_down(?r, ?o)]) then true
    else if (exists_{?o: obj, ?p: person} [take(?r, ?o, ?p)]) then false
    else if (exists_{?o: obj, ?p: person} [give(?r, ?o, ?p)]) then true
    else emptyhand(?r);

  holding'(?r, ?o)=
    if (pick_up(?r, ?o)) then true
    else if (put_down(?r, ?o)) then false
    else if (exists_{?p: person} [take(?r, ?o, ?p)]) then true
    else if (exists_{?p: person} [give(?r, ?o, ?p)]) then false
    else holding(?r, ?o);

  object_at'(?o, ?loc) =
    if (exists_{?r: robot} [(pick_up(?r, ?o) ^ robot_at(?r, ?loc))])
    then false
    else if (exists_{?r: robot}

```

```

        [(put_down(?r, ?o) ^ robot_at(?r, ?loc))]
    then true
    else object_at(?o, ?loc);

object_with'(?o, ?p) =
    if (exists_{?r: robot} [(take(?r, ?o, ?p))]) then false
    else if (exists_{?r: robot} [(give(?r, ?o, ?p))]) then true
    else object_with(?o, ?p);

goal_object_at'(?o, ?loc) =
    if (exists_{?r: robot, ?p: person}
        [(need_assistance(?p) ^ talk_to_person(?r, ?p) ^
          PERSON_GOAL_OBJECT_AT(?p, ?o, ?loc))])
    then true
    else goal_object_at(?o, ?loc);

goal_object_with'(?o, ?p) =
    if (exists_{?r: robot, ?p2: person}
        [(need_assistance(?p2) ^ talk_to_person(?r, ?p2) ^
          PERSON_GOAL_OBJECT_WITH(?p2, ?o, ?p))])
    then true
    else goal_object_with(?o, ?p);

person_at'(?p, ?loc) =
    if (exists_{?r: robot}
        [(find_person(?r, ?p) ^ PERSON_IS_AT(?p, ?loc))])
    then true
    else person_at(?p, ?loc);

need_assistance'(?p) =
    if (~needed_assistance(?p) ^ ~need_assistance(?p))
    then (Bernoulli (PROB_NEED_ASSISTANCE(?p)))
    else if (exists_{?r: robot}
        [(need_assistance(?p) ^ talk_to_person(?r, ?p))])
    then false
    else need_assistance(?p);

needed_assistance'(?p) =
    if (exists_{?r: robot}

```

```

        [(need_assistance(?p) ^ talk_to_person(?r, ?p))]
    then true
    else needed_assistance(?p);

goal_attempted'(?o) =
    if (exists_{?r: robot, ?loc: wp}
        [(put_down(?r, ?o) ^ goal_object_at(?o, ?loc))])
    then true
    else if (exists_{?r: robot, ?p: person}
        [(give(?r, ?o, ?p) ^ goal_object_with(?o, ?p))])
    then true
    else goal_attempted(?o);

reward_received'(?o) =
    if (exists_{?loc: wp} [(goal_attempted(?o) ^
        object_at(?o, ?loc) ^ goal_object_at(?o, ?loc))])
    then true
    else if (exists_{?p: person} [(goal_attempted(?o) ^
        object_with(?o, ?p) ^ goal_object_with(?o, ?p))])
    then true
    else reward_received(?o);
};

reward =
[sum_{?r: robot} [COST]] +
[sum_{?o: obj}
    [TASK_REWARD * (exists_{?loc: wp}
        [(~reward_received(?o) ^ goal_attempted(?o) ^
        object_at(?o, ?loc) ^ goal_object_at(?o, ?loc))]) +
    TASK_REWARD * (exists_{?p: person, ?loc: wp}
        [(~reward_received(?o) ^ goal_attempted(?o) ^
        object_with(?o, ?p) ^ goal_object_with(?o, ?p))])]];

action-preconditions {
    forall_{?r: robot, ?loc1: wp, ?loc2: wp} [move(?r, ?loc1, ?loc2) =>
        (robot_at(?r, ?loc1) ^ localised(?r))];
    forall_{?r: robot} [localise(?r) => (~localised(?r))];
    forall_{?r: robot, ?p: person} [find_person(?r, ?p) =>
        (~(exists_{?loc: wp} [person_at(?p, ?loc)]) ^ localised(?r))];
}

```

```

forall_{?r: robot, ?p: person} [talk_to_person(?r, ?p) =>
    (exists_{?loc: wp} [(robot_at(?r, ?loc) ^ person_at(?p, ?loc))]);
forall_{?r: robot, ?o: obj} [pick_up(?r, ?o) =>
    (exists_{?loc: wp} [(robot_at(?r, ?loc) ^ TABLE_AT(?loc) ^
    object_at(?o, ?loc) ^ emptyhand(?r) ^ ~goal_attempted(?o))]);
forall_{?r: robot, ?o: obj} [put_down(?r, ?o) => (exists_{?loc: wp}
    [(robot_at(?r, ?loc) ^ TABLE_AT(?loc) ^ holding(?r, ?o))]);
forall_{?r: robot, ?o: obj, ?p: person} [take(?r, ?o, ?p) =>
    (exists_{?loc: wp} [(robot_at(?r, ?loc) ^ person_at(?p, ?loc) ^
    emptyhand(?r) ^ object_with(?o, ?p) ^ ~goal_attempted(?o))]);
forall_{?r: robot, ?o: obj, ?p: person} [give(?r, ?o, ?p) =>
    (exists_{?loc: wp} [(robot_at(?r, ?loc) ^
    person_at(?p, ?loc) ^ holding(?r, ?o))]);
};
}

```

## D.5 Blocks World

```

domain blocksworld_mdp {
    requirements = {reward-deterministic};

    types {
        block : object;
    };

    pvariables {
        GOAL_REWARD_ON : {non-fluent, real, default = 0.0};
        GOAL_REWARD_UNSTACK : {non-fluent, real, default = 0.0};
        GOAL_REWARD_STACK : {non-fluent, real, default = 0.0};
        COST : {non-fluent, real, default = 0.0};
        GOAL(block,block) : {non-fluent, bool, default = false};

        clear(block) : {state-fluent, bool, default = false};
        on_table(block) : {state-fluent, bool, default = false};
        on(block, block) : {state-fluent, bool, default = false};

        stack(block, block) : {action-fluent, bool, default = false};
        unstack(block, block) : {action-fluent, bool, default = false};
    };
}

```

```

};

cpfs {
  clear'(?block) =
    if (exists_{?block2 : block} [stack(?block2, ?block)]) then false
    else if (exists_{?block2: block} [unstack(?block2, ?block)])
    then true
    else if (exists_{?block2: block, ?block3: block}
             [stack(?block2, ?block3) ^ on(?block2, ?block)])
    then true
    else clear(?block);

  on_table'(?block) =
    if (exists_{?block2 : block} [stack(?block, ?block2)]) then false
    else if (exists_{?block2 : block} [unstack(?block, ?block2)])
    then true
    else on_table(?block);

  on'(?block1, ?block2) =
    if (unstack(?block1, ?block2)) then false
    else if (stack(?block1, ?block2)) then true
    else if (exists_{?block3: block} [stack(?block1, ?block3)])
    then false
    else on(?block1, ?block2);
};

reward =
  [sum_{?block1 : block, ?block2 : block}
    +(GOAL_REWARD_ON * [GOAL(?block1, ?block2) ^ on(?block1, ?block2)])
    -COST*stack(?block1, ?block2)
    -COST*unstack(?block1, ?block2)
  ]
+ (GOAL_REWARD_STACK * [exists_{?block1 : block} [on_table(?block1) ^
  ~exists_{?block2 : block} (?block1 ~= ?block2 ^ on_table(?block2))]])
+ (GOAL_REWARD_UNSTACK * [~exists_{?block1 : block, ?block2 : block}
  [on(?block1, ?block2)]]);

state-action-constraints {
  forall_{?block : block} [~on(?block, ?block)];
}

```

```
};

action-preconditions {
  forall_{?block1 : block, ?block2 : block} [stack(?block1, ?block2) =>
    (clear(?block1) ^ clear(?block2))];
  forall_{?block1 : block, ?block2 : block} [unstack(?block1, ?block2) =>
    (clear(?block1) ^ on(?block1, ?block2))];
};
}
```



# Bibliography

- [1] Abel, D.; Hershkowitz, D.; and Littman, M. 2016. Near optimal behavior via approximate state abstraction. In *Proceedings of the International Conference on Machine Learning*, 2915–2923.
- [2] Achiam, J.; and Sastry, S. 2017. Surprise-based intrinsic motivation for deep reinforcement learning. *arXiv preprint 1703.01732*.
- [3] Agarwal, A.; Kakade, S.; and Yang, L. F. 2020. Model-based reinforcement learning with a generative model is minimax optimal. In Abernethy, J.; and Agarwal, S., eds., *Proceedings of Conference on Learning Theory*, volume 125, 67–83.
- [4] Andrychowicz, M.; Wolski, F.; Ray, A.; Schneider, J.; Fong, R.; Welinder, P.; McGrew, B.; Tobin, J.; Pieter Abbeel, O.; and Zaremba, W. 2017. Hindsight experience replay. In *Proceedings of the International Conference on Machine Learning*, volume 30, 5048–5058.
- [5] Aubret, A.; Matignon, L.; and Hassas, S. 2019. A survey on intrinsic motivation in reinforcement learning. *arXiv preprint arXiv:1908.06976*.
- [6] Bai, A.; Srivastava, S.; and Russell, S. J. 2016. Markovian state and action abstractions for MDPs via hierarchical MCTS. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 3029–3039.
- [7] Barto, A. G.; and Mahadevan, S. 2003. Recent advances in hierarchical reinforcement learning. *Discrete Event Dynamic Systems*, 13(1): 41–77.
- [8] Bellemare, M.; Srinivasan, S.; Ostrovski, G.; Schaul, T.; Saxton, D.; and Munos, R. 2016. Unifying count-based exploration and intrinsic motivation.

- In *Proceedings of the Conference on Neural Information Processing Systems*, 1471–1479.
- [9] Bellman, R. 1957. *Dynamic programming*. Dover Publications.
- [10] Bengio, Y.; Louradour, J.; Collobert, R.; and Weston, J. 2009. Curriculum learning. In *Proceedings of the International Conference on Machine Learning*, 41–48.
- [11] Benson, S. S. 1996. *Learning action models for reactive autonomous agents*. Ph.D. thesis, Stanford University.
- [12] Benton, J.; Coles, A. J.; and Coles, A. 2012. Temporal planning with preferences and time-dependent continuous costs. In *Proceedings of International Conference on Automated Planning and Scheduling*.
- [13] Bertsekas, D. P. 1995. *Dynamic programming and optimal control*, volume 1. Athena scientific Belmont, MA.
- [14] Blockeel, H. 2009. The ACE datamining system. <https://dtai.cs.kuleuven.be/ACE/>. Accessed: 03.02.2022.
- [15] Boutilier, C.; Dearden, R.; and Goldszmidt, M. 2000. Stochastic dynamic programming with factored representations. *Artificial intelligence*, 121(1-2): 49–107.
- [16] Boutilier, C.; Reiter, R.; and Price, B. 2001. Symbolic dynamic programming for first-order MDPs. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 690–700.
- [17] Boyan, J. A.; and Littman, M. L. 2001. Exact solutions to time-dependent MDPs. In *Proceedings of the Conference on Neural Information Processing Systems*, 1026–1032.
- [18] Bradtke, S.; and Duff, M. 1994. Reinforcement learning methods for continuous-time Markov decision problems. *Advances in Neural Information Processing Systems*, 7.

- [19] Brafman, R. I.; and Tenenbholz, M. 2002. R-MAX - A general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research*, 3: 213–231.
- [20] Buckman, J.; Hafner, D.; Tucker, G.; Brevdo, E.; and Lee, H. 2018. Sample-efficient reinforcement learning with stochastic ensemble value expansion. In *Proceedings of the Conference on Neural Information Processing Systems*, 8224–8234.
- [21] Burda, Y.; Edwards, H.; Storkey, A.; and Klimov, O. 2018. Exploration by random network distillation. *arXiv preprint arXiv:1810.12894*.
- [22] Camacho, A.; Muise, C.; and McIlraith, S. A. 2016. From FOND to robust probabilistic planning: Computing compact policies that bypass avoidable deadends. In *Proceedings of the International Conference on International Conference on Automated Planning and Scheduling*, 65–69.
- [23] Canal, G.; Cashmore, M.; Krivic, S.; Alenyà, G.; Magazzeni, D.; and Torras, C. 2019. Probabilistic planning for robotics with ROSPlan. In *Proceedings of the TAROS*, 236–250.
- [24] Carreno, Y.; Ng, J. H. A.; Petillot, Y.; and Petrick, R. P. A. 2022. Planning, execution, and adaptation for multi-robot systems using probabilistic and temporal planning. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems*.
- [25] Cashmore, M.; Fox, M.; Long, D.; Magazzeni, D.; Ridder, B.; Carrera, A.; Palomeras, N.; Hurtos, N.; and Carreras, M. 2015. ROSPlan: Planning in the robot operating system. In *Proceedings of the International Conference on Automated Planning and Scheduling*.
- [26] Chapman, D.; and Kaelbling, L. P. 1991. Input generalization in delayed reinforcement learning: An algorithm and performance comparisons. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 726–731.
- [27] Christiano, P.; Shah, Z.; Mordatch, I.; Schneider, J.; Blackwell, T.; Tobin, J.; Abbeel, P.; and Zaremba, W. 2016. Transfer from simulation to

- real world through learning deep inverse dynamics model. *arXiv preprint arXiv:1610.03518*.
- [28] Coles, A. J.; Coles, A.; Fox, M.; and Long, D. 2010. Forward-chaining partial-order planning. In *Proceedings of International Conference on Automated Planning and Scheduling*, 42–49.
- [29] Cresswell, S. N.; McCluskey, T. L.; and West, M. M. 2013. Acquiring planning domain models using LOCM. *The Knowledge Engineering Review*, 28(2): 195–213.
- [30] Croonenborghs, T.; Driessens, K.; and Bruynooghe, M. 2007. Learning relational options for inductive transfer in relational reinforcement learning. In *Proceedings of the International Conference on Inductive Logic Programming*, 88–97.
- [31] Croonenborghs, T.; Ramon, J.; Blockeel, H.; and Bruynooghe, M. 2007. Online learning and exploiting relational models in reinforcement learning. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 726–731.
- [32] Cserna, B.; Doyle, W. J.; Ramsdell, J. S.; and Ruml, W. 2018. Avoiding dead ends in real-time heuristic search. In *Proceedings of the AAAI Conference on Artificial Intelligence*.
- [33] Cullen, J.; and Bryman, A. 1988. The knowledge acquisition bottleneck: Time for reassessment? *Expert Systems*, 5(3): 216–225.
- [34] Dabney, W.; and McGovern, A. 2007. Utile distinctions for relational reinforcement learning. In *Proceedings of the International Joint Conference on Artificial Intelligence*, volume 7, 738–743.
- [35] Dean, T.; and Kanazawa, K. 1989. A model for reasoning about persistence and causation. *Computational Intelligence*, 5(2): 142–150.
- [36] Deisenroth, M.; and Rasmussen, C. E. 2011. PILCO: A model-based and data-efficient approach to policy search. In *Proceedings of the International Conference on Machine Learning*, 465–472.

- [37] Dietterich, T. G. 2000. Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal of Artificial Intelligence Research*, 13: 227–303.
- [38] Diuk, C. 2010. *An object-oriented representation for efficient reinforcement learning*. Ph.D. thesis, Rutgers, The State University of New Jersey.
- [39] Diuk, C.; Li, L.; and Leffler, B. R. 2009. The adaptive k-meteorologists problem and its application to structure learning and feature selection in reinforcement learning. In *Proceedings of the International Conference on Machine Learning*, 249–256.
- [40] Doshi-Velez, F.; and Kim, B. 2017. Towards a rigorous science of interpretable machine learning. *arXiv preprint arXiv:1702.08608*.
- [41] Driessens, K.; and Džeroski, S. 2004. Integrating guidance into relational reinforcement learning. *Machine Learning*, 57: 271–304.
- [42] Driessens, K.; and Džeroski, S. 2005. Combining model-based and instance-based learning for first order regression. In *Proceedings of the International Conference on Machine Learning*, 193–200.
- [43] Driessens, K.; and Ramon, J. 2003. Relational instance based regression for relational reinforcement learning. In *Proceedings of the International Conference on Machine Learning*, 123–130.
- [44] Driessens, K.; Ramon, J.; and Blockeel, H. 2001. Speeding up relational reinforcement learning through the use of an incremental first order decision tree learner. In *Proceedings of the European Conference on Machine Learning*, 97–108.
- [45] Driessens, K.; Ramon, J.; and Gärtner, T. 2006. Graph kernels and Gaussian processes for relational reinforcement learning. *Machine Learning*, 64: 91–119.
- [46] Duckworth, P.; Lacerda, B.; and Hawes, N. 2021. Time-bounded mission planning in time-varying domains with semi-MDPs and Gaussian processes. In *Proceedings of the Conference on Robot Learning*, volume 155, 1654–1668.

- [47] Džeroski, S.; De Raedt, L.; and Driessens, K. 2001. Relational reinforcement learning. *Machine Learning*, 43(1-2): 7–52.
- [48] Feinberg, V.; Wan, A.; Stoica, I.; Jordan, M. I.; Gonzalez, J. E.; and Levine, S. 2018. Model-based value estimation for efficient model-free reinforcement learning. *arXiv preprint arXiv:1803.00101*.
- [49] Fern, A.; Yoon, S. W.; and Givan, R. 2003. Approximate policy iteration with a policy language bias. In *Proceedings of the International Conference on Machine Learning*, 847–854.
- [50] Fikes, R. E.; and Nilsson, N. J. 1971. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2(3-4): 189–208.
- [51] Finn, C.; Abbeel, P.; and Levine, S. 2017. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the International Conference on Machine Learning*.
- [52] Fox, M.; and Long, D. 2003. PDDL2.1: An extension to PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Research*, 20: 61–124.
- [53] Garg, S.; Bajpai, A.; and Mausam. 2020. Symbolic network: Generalized neural policies for relational MDPs. In *Proceedings of the International Conference on Machine Learning*, 3397–3407.
- [54] Geramifard, A.; Dann, C.; and How, J. P. 2013. Off-policy learning combined with automatic feature expansion for solving large MDPs. In *Proceedings of the Multidisciplinary Conference on Reinforcement Learning and Decision Making*, 29–33.
- [55] Geramifard, A.; Doshi, F.; Redding, J.; Roy, N.; and How, J. 2011. Online discovery of feature dependencies. In *Proceedings of the International Conference on Machine Learning*, 881–888.

- [56] Geramifard, A.; Walsh, T.; Roy, N.; and How, J. 2013. Batch iFDD: A scalable matching pursuit algorithm for solving MDPs. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*.
- [57] Geramifard, A.; Walsh, T. J.; Tellex, S.; Chowdhary, G.; Roy, N.; and How, J. P. 2013. *A tutorial on linear function approximators for dynamic programming and reinforcement learning*, volume 6. Now Foundations and Trends.
- [58] Ghallab, M.; Nau, D.; and Traverso, P. 2004. *Automated planning*. Elsevier. ISBN 9781558608566.
- [59] Gil, Y. 1994. Learning by experimentation: Incremental refinement of incomplete planning domains. In *Proceedings of the International Conference on Machine Learning*, 87–95.
- [60] Gordon, D.; Kadian, A.; Parikh, D.; Hoffman, J.; and Batra, D. 2019. Split-Net: Sim2Sim and Task2Task transfer for embodied visual navigation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*.
- [61] Grounds, M. J.; and Kudenko, D. 2007. Combining reinforcement learning with symbolic planning. In *Proceedings of the European Workshop on Adaptive Agents and Multi-Agent Systems*.
- [62] Grzes, M.; Hoey, J.; and Sanner, S. 2014. International Probabilistic Planning Competition (IPPC) 2014. In *Proceedings of the International Conference on Automated Planning and Scheduling*.
- [63] Guerin, J.; Hanna, J. P.; Ferland, L.; Mattei, N.; and Goldsmith, J. 2012. The Academic Advising planning domain. In *Proceedings of the ICAPS Workshop on the International Planning Competition*.
- [64] Guestrin, C.; Koller, D.; Gearhart, C.; and Kanodia, N. 2003. Generalizing plans to new environments in relational MDPs. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 1003–1010.
- [65] Guidotti, R.; Monreale, A.; Ruggieri, S.; Turini, F.; Giannotti, F.; and Pedreschi, D. 2018. A survey of methods for explaining black box models. *ACM Computing Surveys*, 51(5).

- [66] Guo, Z. D.; and Brunskill, E. 2017. Sample efficient feature selection for factored MDPs. *arXiv preprint arXiv:1703.03454*.
- [67] Guo, Z. D.; and Brunskill, E. 2019. Directed exploration for reinforcement learning. *arXiv preprint arXiv:1906.07805*.
- [68] Hasselt, H. v.; Guez, A.; and Silver, D. 2016. Deep reinforcement learning with Double Q-learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2094–2100.
- [69] Hester, T.; and Stone, P. 2010. Real time targeted exploration in large domains. In *Proceedings of the International Conference on Development and Learning*, 191–196.
- [70] Illanes, L.; Yan, X.; Icarte, R. T.; and McIlraith, S. A. 2020. Symbolic plans as high-level instructions for reinforcement learning. In *Proceedings of the International Conference on Automated Planning and Scheduling*.
- [71] Issakkimuthu, M.; Fern, A.; and Tadepalli, P. 2018. Training deep reactive policies for probabilistic planning problems. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 28.
- [72] James, S.; Davison, A. J.; and Johns, E. 2017. Transferring end-to-end visuomotor control from simulation to real world for a multi-stage task. In *Proceedings of the Conference on Robot Learning*.
- [73] James, S.; and Johns, E. 2016. 3D Simulation for robot arm control with deep Q-learning. In *Proceedings of the Workshop on Deep Learning for Action and Interaction*.
- [74] James, S.; Wohlhart, P.; Kalakrishnan, M.; Kalashnikov, D.; Irpan, A.; Ibarz, J.; Levine, S.; Hadsell, R.; and Bousmalis, K. 2019. Sim-To-Real via Sim-To-Sim: Data-efficient robotic grasping via randomized-to-canonical adaptation networks. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, 12619–12629.



- [75] Jiménez, S.; De la Rosa, T.; Fernández, S.; Fernández, F.; and Borrajo, D. 2012. A review of machine learning for automated planning. *The Knowledge Engineering Review*, 27(4): 433–467.
- [76] Jong, N. K.; and Stone, P. 2005. State abstraction discovery from irrelevant state variables. In *Proceedings of the International Joint Conference on Artificial Intelligence*, volume 8, 752–757.
- [77] Jong, N. K.; and Stone, P. 2007. Model-based function approximation in reinforcement learning. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems*, 1–8.
- [78] Jonsson, A.; and Barto, A. G. 2001. Automated state abstraction for options using the U-tree algorithm. *Proceedings of the International Conference on Machine Learning*, 1054–1060.
- [79] Joshi, S.; Khardon, R.; Tadepalli, P.; Raghavan, A.; and Fern, A. 2013. Solving relational MDPs with exogenous events and additive rewards. In *Proceedings of the Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, 178–193.
- [80] Jung, T.; and Stone, P. 2009. Feature selection for value function approximation using Bayesian model selection. In *Proceedings of the Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, 660–675.
- [81] Kakade, S. M. 2003. *On the sample complexity of reinforcement learning*. Ph.D. thesis, University College London.
- [82] Karia, R.; and Srivastava, S. 2020. Learning generalized relational heuristic networks for model-agnostic planning. *arXiv preprint arXiv:2007.06702*.
- [83] Kartal, B.; Nunes, E.; Godoy, J.; and Gini, M. 2016. Monte Carlo Tree Search with branch and bound for multi-robot task allocation. In *Proceedings of the IJCAI Workshop on Autonomous Mobile Service Robots*.

- [84] Kearns, M.; and Singh, S. 1998. Near-optimal reinforcement learning in polynomial time. In *Proceedings of the International Conference on Machine Learning*, 260–268.
- [85] Keller, P.; Mannor, S.; and Precup, D. 2006. Automatic basis function construction for approximate dynamic programming and reinforcement learning. In *Proceedings of the International Conference on Machine Learning*, volume 2006, 449–456.
- [86] Keller, T.; and Eyerich, P. 2012. PROST: Probabilistic planning based on UCT. In *Proceedings of the International Conference on Automated Planning and Scheduling*.
- [87] Keller, T.; and Helmert, M. 2013. Trial-based heuristic tree search for finite horizon MDPs. In *Proceedings of the International Conference on Automated Planning and Scheduling*, 135–143.
- [88] Kocsis, L.; and Szepesvári, C. 2006. Bandit based Monte-Carlo planning. In *Proceedings of the European Conference on Machine Learning*, volume 2006, 282–293.
- [89] Koenig, N.; and Howard, A. 2004. Design and use paradigms for Gazebo, an open-source multi-robot simulator. In *Proceedings of the International Conference on Intelligent Robots and Systems*, volume 3, 2149–2154.
- [90] Koenig, S.; Tovey, C.; Lagoudakis, M.; Markakis, V.; Kempe, D.; Keskinocak, P.; Kleywegt, A.; Meyerson, A.; and Jain, S. 2006. The power of sequential single-item auctions for agent coordination. In *Proceedings of the National Conference on Artificial Intelligence*, 1625–1629.
- [91] Kokel, H.; Manoharan, A.; Natarajan, S.; Ravindran, B.; and Tadepalli, P. 2021. RePreL: Integrating relational planning and reinforcement learning for effective abstraction. In *Proceedings of the International Conference on Automated Planning and Scheduling*, 533–541.
- [92] Kolobov, A.; Mausam; and Weld, D. S. 2010. SixthSense: Fast and reliable

- recognition of dead ends in MDPs. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 1108–1114.
- [93] Kolobov, A.; Mausam; and Weld, D. S. 2012. LRTDP vs. UCT for online probabilistic planning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 1786–1792.
- [94] Konidaris, G.; and Barto, A. 2006. Autonomous shaping: Knowledge transfer in reinforcement learning. In *Proceedings of the International Conference on Machine Learning*, 489–496.
- [95] Konidaris, G. D.; and Barto, A. G. 2007. Building portable options: Skill transfer in reinforcement learning. In *Proceedings of the International Joint Conference on Artificial Intelligence*, volume 7, 895–900.
- [96] Kroon, M.; and Whiteson, S. 2009. Automatic feature selection for model-based reinforcement learning in factored MDPs. In *Proceedings of the International Conference on Machine Learning and Applications*, 324–330.
- [97] Lacerda, B.; Parker, D.; and Hawes, N. 2017. Multi-objective policy generation for mobile robots under probabilistic time-bounded guarantees. In *Proceedings of the International Conference on Automated Planning and Scheduling*, 504–512.
- [98] Lang, T.; and Toussaint, M. 2010. Planning with noisy probabilistic relational rules. *Journal of Artificial Intelligence Research*.
- [99] Lang, T.; Toussaint, M.; and Kersting, K. 2012. Exploration in relational domains for model-based reinforcement learning. *Journal of Machine Learning Research*, 13: 3725–3768.
- [100] Li, L.; Walsh, T. J.; and Littman, M. L. 2006. Towards a unified theory of state abstraction for MDPs. *Proceedings of the International Symposium on Artificial Intelligence and Mathematics*, 4: 5.
- [101] Lin, L. J. 1992. *Reinforcement learning for robots using neural networks*. Ph.D. thesis, Carnegie Mellon University.

- [102] Lipovetzky, N.; Muise, C.; and Geffner, H. 2016. Traps, invariants, and dead-ends. In *Proceedings of the International Conference on Automated Planning and Scheduling*, 211–215.
- [103] Little, I.; and Thiebaux, S. 2007. Probabilistic planning vs. replanning. In *Proceedings of the ICAPS Workshop on the International Planning Competition: Past, Present and Future*.
- [104] Liu, L.; and Sukhatme, G. 2018. A solution to time-varying Markov decision processes. *IEEE Robotics and Automation Letters*.
- [105] Mahadevan, S.; and Maggioni, M. 2007. Proto-value functions: A Laplacian framework for learning representation and control in Markov decision processes. *Journal of Machine Learning Research*, 8(Oct): 2169–2231.
- [106] Martínez, D.; Alenya, G.; Ribeiro, T.; and Torras, C. 2017. Relational reinforcement learning for planning with exogenous effects. *Journal of Machine Learning Research*, 18(1): 2689–2732.
- [107] Martínez, D.; Alenya, G.; and Torras, C. 2017. Relational reinforcement learning with guided demonstrations. *Artificial Intelligence*, 247: 295–312.
- [108] Martínez, D.; Alenya, G.; Torras, C.; Ribeiro, T.; and Inoue, K. 2016. Learning relational dynamics of stochastic domains for planning. In *Proceedings of the International Conference on Automated Planning and Scheduling*, 235–243.
- [109] Mausam; and Weld, D. S. 2003. Solving relational MDPs with first-order machine learning. In *Proceedings of the ICAPS Workshop on Planning Under Uncertainty And Incomplete Information*.
- [110] McCallum, A. K. 1996. *Reinforcement learning with selective perception and hidden state*. Ph.D. thesis, University of Rochester.
- [111] McDermott, D.; Ghallab, M.; Howe, A.; Knoblock, C.; Ram, A.; Veloso, M.; Weld, D.; and Wilkins, D. 1998. PDDL - The Planning Domain Definition Language. Technical Report CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control.

- [112] Mnih, V.; Kavukcuoglu, K.; Silver, D.; Graves, A.; Antonoglou, I.; Wierstra, D.; and Riedmiller, M. 2013. Playing Atari with deep reinforcement learning. In *Proceedings of the NeurIPS Deep Learning Workshop*.
- [113] Morales, E. 2003. Scaling up reinforcement learning with a relational representation. In *Proceedings of the Workshop on Adaptability in Multi-agent System*.
- [114] Mourão, K.; Zettlemoyer, L. S.; Petrick, R. P. A.; and Steedman, M. 2012. Learning STRIPS operators from noisy and incomplete observations. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, 614–623.
- [115] Mourão, K. M. T. 2012. *Learning action representations using kernel perceptrons*. Ph.D. thesis, The University of Edinburgh.
- [116] Muggleton, S.; and de Raedt, L. 1994. Inductive Logic Programming: Theory and methods. *The Journal of Logic Programming*, 19-20: 629–679. Special issue: Ten years of logic programming.
- [117] Ng, J. H. A.; and Petrick, R. P. A. 2019. Incremental learning of action models for planning. In *Proceedings of the ICAPS Workshop on Knowledge Engineering for Planning and Scheduling (KEPS)*.
- [118] Ng, J. H. A.; and Petrick, R. P. A. 2019. Incremental learning of planning actions in model-based reinforcement learning. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 3195–3201.
- [119] Ng, J. H. A.; and Petrick, R. P. A. 2020. Practical feature selection for online reinforcement learning and planning in relational MDPs. In *Proceedings of the Workshop of the UK Planning and Scheduling Special Interest Group*.
- [120] Ng, J. H. A.; and Petrick, R. P. A. 2021. First-order function approximation for transfer learning in relational MDPs. In *Proceedings of the ICAPS Workshop on Bridging the Gap Between AI Planning and Reinforcement Learning (PRL)*.

- [121] Ng, J. H. A.; and Petrick, R. P. A. 2021. Generalised linear function approximation with first-order features. In *Proceedings of the IJCAI Workshop on Generalization in Planning (GenPlan)*.
- [122] Ng, J. H. A.; and Petrick, R. P. A. 2021. Generalised task planning with first-order function approximation. In *Proceedings of the Conference on Robot Learning*.
- [123] Ng, J. H. A.; and Petrick, R. P. A. 2022. First-Order Dead End Situations for Policy Guidance in Relational Problems. In *Proceedings of the IJCAI Workshop on Generalization in Planning (GenPlan)*.
- [124] Nguyen, T.; Li, Z.; Silander, T.; and Leong, T. Y. 2013. Online feature selection for model-based reinforcement learning. In *Proceedings of the International Conference on Machine Learning*, 498–506.
- [125] Nunes, E.; and Gini, M. 2015. Multi-robot auctions for allocation of tasks with temporal constraints. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2110–2216.
- [126] Oates, T.; and Cohen, P. R. 1996. Learning planning operators with conditional and probabilistic effects. In *Proceedings of the AAAI Spring Symposium on Planning with Incomplete Information for Robot Problems*, 86–94.
- [127] Ornik, M.; and Topcu, U. 2021. Learning and planning for time-varying MDPs using maximum likelihood estimation. *Journal of Machine Learning Research*, 22: 1–40.
- [128] Oudeyer, P.-Y.; Kaplan, F.; and Hafner, V. V. 2007. Intrinsic motivation systems for autonomous mental development. *IEEE Transactions on Evolutionary Computation*, 11(2): 265–286.
- [129] Painter-Wakefield, C.; and Parr, R. 2012. Greedy algorithms for sparse reinforcement learning. In *Proceedings of the International Conference on Machine Learning*.
- [130] Palmer, T. J. 2015. *Learning action-state representation forests for implicitly relational worlds*. Ph.D. thesis, University Of Oklahoma.

- [131] Pardo, F.; Tavakoli, A.; Levdik, V.; and Kormushev, P. 2018. Time limits in reinforcement learning. In *Proceedings of the International Conference on Machine Learning*.
- [132] Parr, R.; Li, L.; Taylor, G.; Painter-Wakefield, C.; and Littman, M. 2008. An analysis of linear models, linear value-function approximation, and feature selection for reinforcement learning. In *Proceedings of the International Conference on Machine Learning*, 752–759.
- [133] Parr, R.; Painter-Wakefield, C.; Li, L.; and Littman, M. 2007. Analyzing feature generation for value-function approximation. In *Proceedings of the International Conference on Machine Learning*, volume 227, 737–744.
- [134] Pasula, H.; Zettlemoyer, L. S.; and Kaelbling, L. P. 2004. Learning probabilistic relational planning rules. In *Proceedings of the International Conference on Automated Planning and Scheduling*, 73–82.
- [135] Pasula, H. M.; Zettlemoyer, L. S.; and Kaelbling, L. P. 2007. Learning symbolic models of stochastic domains. *Journal of Artificial Intelligence Research*, 29: 309–352.
- [136] Peng, J.; and Williams, R. 1998. Incremental multi-step Q-learning. *Machine Learning*, 22.
- [137] Pereira, R. F.; Oren, N.; and Meneguzzi, F. 2017. Landmark-based heuristics for goal recognition. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 3622–3628.
- [138] Puiutta, E.; and Veith, E. 2020. Explainable reinforcement learning: A survey. In *Machine Learning and Knowledge Extraction*, 77–95. Springer International Publishing. ISBN 978-3-030-57320-1.
- [139] Puterman, M. L. 2014. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons.
- [140] Quigley, M.; Gerkey, B.; Conley, K.; Faust, J.; Foote, T.; Leibs, J.; Berger, E.; Wheeler, R.; and Ng, A. Y. 2009. ROS: An open-source Robot Operating

- System. In *Proceedings of the ICRA Workshop on Open Source Software*, volume 3.
- [141] Ramon, J.; Driessens, K.; and Croonenborghs, T. 2007. Transfer learning in reinforcement learning problems through partial policy recycling. In *Proceedings of the European Conference on Machine Learning*, volume 4701, 699–707.
- [142] Rao, D.; and Jiang, Z. 2015. Learning planning domain descriptions in RDDL. *International Journal on Artificial Intelligence Tools*, 24(3).
- [143] Ribeiro, T.; and Inoue, K. 2014. Learning prime implicant conditions from interpretation transition. In *Proceedings of the International Conference on Inductive Logic Programming*.
- [144] Rodrigues, C.; Gérard, P.; and Rouveirol, C. 2008. On and off-policy relational reinforcement learning. *Inductive Logic Programming*, 99.
- [145] Roncagliolo, S.; and Tadepalli, P. 2004. Function approximation in hierarchical relational reinforcement learning. In *Proceedings of the ICML Workshop on Relational Reinforcement Learning*, 1–5.
- [146] Ryan, M. R. K. 2002. Using abstract models of behaviours to automatically generate reinforcement learning hierarchies. In *Proceedings of the International Conference on Machine Learning*, 522–529.
- [147] Sanner, S. 2006. Online feature discovery in relational reinforcement learning. In *Proceedings of the Open Problems in Statistical Relational Learning Workshop*.
- [148] Sanner, S. 2011. ICAPS 2011 International Probabilistic Planning Competition (IPPC). [http://users.cecs.anu.edu.au/~ssanner/IPPC\\_2011/](http://users.cecs.anu.edu.au/~ssanner/IPPC_2011/). Accessed: 18.12.2020.
- [149] Sanner, S. 2011. Relational Dynamic Influence Diagram Language (RDDL): Language description. [Http://users.cecs.anu.edu.au/ssanner/IPPC\\_2011/RDDL.pdf](Http://users.cecs.anu.edu.au/ssanner/IPPC_2011/RDDL.pdf).



- [150] Sanner, S.; and Boutilier, C. 2012. Practical linear value-approximation techniques for first-order MDPs. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, 409–417.
- [151] Schaul, T.; Horgan, D.; Gregor, K.; and Silver, D. 2015. Universal value function approximators. In *Proceedings of the International Conference on Machine Learning*, volume 37, 1312–1320.
- [152] Schillinger, P.; Bürger, M.; and Dimarogonas, D. V. 2018. Auctioning over probabilistic options for temporal logic-based multi-robot cooperation under uncertainty. In *Proceedings of the International Conference on Robotics and Automation*, 7330–7337.
- [153] Schmidhuber, J. 2010. Formal theory of creativity, fun, and intrinsic motivation (1990-2010). *IEEE Transactions on Autonomous Mental Development*, 2: 230– – 247.
- [154] Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; and Klimov, O. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.
- [155] Seurin, M.; Strub, F.; Preux, P.; and Pietquin, O. 2021. Don’t do what doesn’t matter: Intrinsic motivation with action usefulness. In *Proceedings of the International Joint Conference on Artificial Intelligence*.
- [156] Sharma, M.; Holmes, M.; Santamaría, J.; Irani, A.; Jr, C.; and Ram, A. 2007. Transfer learning in real-time strategy games using hybrid CBR/RL. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 1041–1046.
- [157] Silver, D.; Huang, A.; Maddison, C. J.; Guez, A.; Sifre, L.; van den Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; Dieleman, S.; Grewe, D.; Nham, J.; Kalchbrenner, N.; Sutskever, I.; Lillicrap, T.; Leach, M.; Kavukcuoglu, K.; Graepel, T.; and Hassabis, D. 2016. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529: 484–503.

- [158] Singh, S.; Barto, A. G.; and Chentanez, N. 2005. Intrinsically motivated reinforcement learning. In *Proceedings of the Conference on Neural Information Processing Systems*, 1281–1288.
- [159] Singh, S.; Sutton, R.; and Kaelbling, P. 1995. Reinforcement learning with replacing eligibility traces. *Machine Learning*, 22.
- [160] Srivastava, S.; Immerman, N.; and Zilberstein, S. 2008. Learning generalized plans using abstract counting. In *Proceedings of the National Conference on Artificial Intelligence*, 991–997.
- [161] Srivastava, S.; Russell, S.; Ruan, P.; and Cheng, X. 2014. First-order open-universe POMDPs. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, 742–751.
- [162] Steinmetz, M.; and Hoffmann, J. 2017. Search and learn: On dead-end detectors, the traps they set, and trap learning. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 4398–4404.
- [163] Strehl, A. L.; Diuk, C.; and Littman, M. L. 2007. Efficient structure learning in factored-state MDPs. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 645–650.
- [164] Strehl, A. L.; and Littman, M. L. 2008. An analysis of model-based interval estimation for Markov decision processes. *Journal of Computer and System Sciences*, 74(8): 1309–1331.
- [165] Ståhlberg, S.; Francès, G.; and Seipp, J. 2021. Learning generalized unsolvability heuristics for classical planning. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 4175–4181.
- [166] Sutton, R. S. 1984. *Temporal credit assignment in reinforcement learning*. Ph.D. thesis, University of Massachusetts Amherst.
- [167] Sutton, R. S. 1991. Dyna, an integrated architecture for learning, planning, and reacting. *SIGART Bull.*, 2(4): 160–163.

- [168] Sutton, R. S.; and Barto, A. G. 2018. *Reinforcement learning: An introduction*. MIT press.
- [169] Sutton, R. S.; Precup, D.; and Singh, S. 1999. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2): 181–211.
- [170] Szita, I.; and Lőrincz, A. 2008. The many faces of optimism: A unifying approach. In *Proceedings of the International Conference on Machine Learning*, 1048–1055.
- [171] Tadepalli, P.; Givan, R.; and Driessens, K. 2004. Relational reinforcement learning: An overview. In *Proceedings of the International Conference on Machine Learning*, volume 4, 1–9.
- [172] Taylor, M. E.; and Stone, P. 2007. Cross-domain transfer for reinforcement learning. In *Proceedings of the International Conference on Machine Learning*, 879–886.
- [173] Taylor, M. E.; and Stone, P. 2007. Representation transfer for reinforcement learning. In *Proceedings of the AAAI Fall Symposium: Computational Approaches to Representation Change during Learning and Development*, 78–85.
- [174] Taylor, M. E.; and Stone, P. 2009. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10(7).
- [175] Tobin, J.; Fong, R.; Ray, A.; Schneider, J.; Zaremba, W.; and Abbeel, P. 2017. Domain randomization for transferring deep neural networks from simulation to the real world. In *Proceedings of the International Conference on Intelligent Robots and Systems*, 23–30.
- [176] Torrey, L.; and Shavlik, J. 2010. Transfer learning. In *Handbook of Research on Machine Learning Applications and Trends: Algorithms, Methods, and Techniques*, 242–264. IGI Global.
- [177] Toyer, S.; Trevizan, F. W.; Thiébaux, S.; and Xie, L. 2020. ASNets: Deep learning for generalised planning. *Journal of Artificial Intelligence Research*, 68: 1–68.

- [178] Tsitsiklis, J.; and van Roy, B. 1997. An analysis of temporal-difference learning with function approximation. *IEEE Transactions on Automatic Control*, 42(5): 674–690.
- [179] van Otterlo, M. 2004. Reinforcement learning for relational MDPs. In *Proceedings of the Machine Learning Conference of Belgium and the Netherlands*.
- [180] van Otterlo, M. 2005. A survey of reinforcement learning in relational domains. Technical report, CTIT Technical Report Series.
- [181] van Otterlo, M. 2012. Solving relational and first-order logical Markov decision processes: A survey. In *Reinforcement Learning*, 253–292. Springer.
- [182] van Seijen, H.; Mahmood, A. R.; Pilarski, P. M.; Machado, M. C.; and Sutton, R. S. 2016. True online temporal-difference learning. *Journal of Machine Learning Research*, 17(145): 1–40.
- [183] Veeriah, V.; Oh, J.; and Singh, S. 2018. Many-goals reinforcement learning. *arXiv preprint arXiv:1806.09605*.
- [184] Walker, T. 2004. Relational reinforcement learning via sampling the space of first-order conjunctive features. In *Proceedings of the ICML Workshop on n Relational Reinforcement Learning*.
- [185] Walsh, T.; Li, L.; and Littman, M. 2006. Transferring state abstractions between MDPs. In *Proceedings of the ICML Workshop on Structural Knowledge Transfer for Machine Learning*.
- [186] Walsh, T. J. 2010. *Efficient learning of relational models for sequential decision making*. Ph.D. thesis, Rutgers, The State University of New Jersey.
- [187] Wang, C.; Joshi, S.; and Khardon, R. 2008. First order decision diagrams for relational MDPs. *Journal of Artificial Intelligence Research*, 31: 431–472.
- [188] Wang, X. 1995. Learning by observation and practice: An incremental approach for planning operator acquisition. In *Proceedings of the International Conference on Machine Learning*, 549–557.

- [189] Watkins, C. J. C. H. 1989. *Learning from Delayed Rewards*. Ph.D. thesis, King’s College, Cambridge, UK.
- [190] Wiering, M.; and van Otterlo, M. 2012. *Reinforcement learning*, volume 12 of *Adaptation, learning, and optimization*. Springer.
- [191] Wu, J.-H.; and Givan, R. 2005. Feature-discovering approximate value iteration methods. In Zucker, J.; and Saitta, L., eds., *Proceedings of the International Symposium on Abstraction, Reformulation and Approximation*, volume 3607 of *Lecture Notes in Computer Science*, 321–331.
- [192] Wu, J.-H.; and Givan, R. 2010. Automatic induction of Bellman-error features for probabilistic planning. *Journal of Artificial Intelligence Research*, 38: 687–755.
- [193] Yang, F.; Lyu, D.; Liu, B.; and Gustafson, S. 2018. PEORL: Integrating symbolic planning and hierarchical reinforcement learning for robust decision-making. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 4860–4866.
- [194] Yang, Q.; Wu, K.; and Jiang, Y. 2005. Learning actions models from plan examples with incomplete knowledge. In *Proceedings of the International Conference on Automated Planning and Scheduling*, 241–250.
- [195] Younes, H. L.; and Littman, M. L. 2004. PPDDL1.0: An extension to PDDL for expressing planning domains with probabilistic effects. Technical Report CMU-CS-04-162, Carnegie Mellon University.
- [196] Zambaldi, V.; Raposo, D.; Santoro, A.; Bapst, V.; Li, Y.; Babuschkin, I.; Tuyls, K.; Reichert, D.; Lillicrap, T.; Lockhart, E.; Shanahan, M.; Langston, V.; Pascanu, R.; Botvinick, M.; Vinyals, O.; and Battaglia, P. 2018. Relational deep reinforcement learning. *arXiv preprint arXiv:1806.01830*.
- [197] Zhang, K.; Yang, Z.; and Başar, T. 2019. Multi-agent reinforcement learning: A selective overview of theories and algorithms. *arXiv preprint arXiv:1911.10635*.

- [198] Zhang, S.; Jiang, Y.; Sharon, G.; and Stone, P. 2017. Multirobot symbolic planning under temporal uncertainty. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems*, 501–510.
- [199] Zhang, Y.; Tino, P.; Leonardis, A.; and Tang, K. 2021. A Survey on Neural Network Interpretability. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 5(5): 726–742.
- [200] Zhuo, H. H.; Yang, Q.; Hu, D. H.; and Li, L. 2010. Learning complex action models with quantifiers and logical implications. *Artificial Intelligence*, 174(18): 1540–1569.